Academic software reuse, an experience report

Edward Z. Yang <ezyang@mit.edu>

May 14, 2012

The objective of this 6.UAP project was to build an online, educational proof assistant for classical first-order logic, which allowed users to directly manipulate the Gentzen tree of a sequent calculus derivation. The goal was to make it easy for students of logic to explore symbolic manipulation of terms without having to use a heavy-duty, command-oriented interface.

That system is not really what this report is about. As a piece of functionality, it has been done to death, and there is nothing really exceptional about Logitext in its feature set to distinguish it from the rest. Instead, the focus of this report will be on a variety of topics whose unifying theme is that they all came out of the explicit choice to reuse as much pre-existing academic software as possible, in the construction of this system. While software reuse is a time-honored tradition in the open source community, the ethos in the research community is different. However, I believe letting development proceed this way was very beneficial for this project, though not without its own surprises.

Prior work

There are many existing proof assistants geared towards educational uses; too many to list here. One might cite this as a case of academics *not* reusing pre-existent research software. Even when we limit ourselves to proof assistants which utilize the Gentzen tree as their primary user interface presentation, we still have many pieces of software to review. The unifying themes of these pieces of software are as follows:

- Displayed proofs look like those which might be found in a textbook (Gentzen trees, Fitch-style deduction, etc),
- The software is easily installable by nontechnical users (usually by implementing the system on a portable runtime such as Java),
- Various deductive styles are supported, the most common of which are natural deduction and sequent calculus,
- Various logics are supported, though usually not going beyond propositional logic and first order logic (which are the usual logics taught to university students), and
- The system was used by the researchers in a real pedagogical context, usually to assist them in teaching introductory logic classes.

We now review a selection of this software in chronological order of introduction. Many of these tools were developed concurrently with the introduction of the Macintosh operating system, which introduced GUI programming to desktop computers.

The Openproof project[1] is one of the oldest projects, spanning Tarski's World, Hyperproof and OpenBox. This software was probably the most sophisticated in the arena; for example, the software associated with Tarski's World had a sophisticated interface for exploring the meanings of first-order sentences in various worlds. Unfortunately, uptake of the project was severely hampered by the fact that the software was proprietary and distributed with their associated textbooks; however the Grade Grinder service has been online for more than a decade now.

WinKE/MacKE[2] by Jeremy Pitt and Ulle Endriss is based off of the KE calculus. It is written in MacProlog. In [Pitt95] they cite LogicWorks, Tableau, MacLogic and Hyperproof. Work on the system was continued by Endriss until 2000; his project page[3] offers a detailed timeline of papers.

Jape[4], by Richard Bornat and Bernard Sufrin, is exceptional in the sense that it makes no commitment to metatheory, allowing users to program in their own logics and deductive systems. Its core is written in Standard ML,

[1] http://ggweb.stanford.edu/openproof/
[2] http://staff.science.uva.nl/~ulle/WinKE/papers.html
[3] http://staff.science.uva.nl/~ulle/WinKE/papers.html
[4] http://www.cs.ox.ac.uk/people/bernard.sufrin/personal/jape.org/

and it has user interfaces written in Tcl/Tk, Java and C. This is one of the earliest and most well regarded systems of this type, including an implementation of "proof by pointing" although the interface is somewhat dated at this point. In [Bornat99], they cite the CMU Proof Tutor (an early, text-only educational proof assistant) as a predecessor. The system was in use at Oxford and QMW.

Pandora[5] is by Krysia Broda, Jiefei Ma, Gabrielle Sinnadurai, and Alexander Summers. It supports natural deduction in first order logic, and is written in Java. Besides [Broda04], they have a refereed article, but it was behind a paywall; the technical report contained no citations. The system is in use at Imperial College.

Proof Lab[6], part of the AProS project, is a system in the Fitch deductive style written in Java. It is part of the courseware of the Logic and Proofs course at CMU. They have a refereed article, but it was behind a paywall.

Panda[7], by Oliver Gasquet, François Schwartzentruber and Martin Strecker, is a recent system which notably supports composition of small proofs into larger proofs. It is written in Java and only supports classical propositional logic and first order logic. In [Gasquet11] they cite Pandora, Jape, Proof Lab, Hyperproof and Bonsai. The system is in use at Universitéde Toulouse.

Yoda[8], by Benjamín Machín and Luis Sierra, is one of the few systems written in JavaScript for display in the browser. It only supports propositional logic and first order logic. In [Machin11], they cite Etps, Tarski's World, WinKE, ProofWeb, Papuq and Logic Daemon as predecessors. The system is in use at the Universidad de la República, Uruguay.

We note that all of these systems have chosen to implement their own theorem provers, rather than defer to the larger, more fully-fledged proof assistants. The authors of Jape justify this as follows:

> Our approach is in contrast to the tradition of user interface work in theorem proving... in which a user interface is bolted on to an otherwise unmodified theorem prover. That tradition starts with a proof engine and fits the user interface to it: we feel that the user interface should be at the centre of the design, supporting

[5] http://www.doc.ic.ac.uk/pandora/
[6] http://www.phil.cmu.edu/projects/apros/index.php?page=prooflab
[7] http://www.irit.fr/panda/
[8] http://www.fing.edu.uy/cgi-bin/logica/Yoda/indexPropH.pl

> only as much proof machinery as it will bear. The proof display in Jape is constantly available and not... extracted from a proof term once the proof is complete.

This author believes that modern proof assistants are sophisticated and powerful enough to support interesting alternative interface paradigms with only a small amount of modification.

Working with research software

Logitext is written in three programming languages: Ur/Web, Haskell (compiled by GHC) and Coq. The compilers of all of these languages constitute "research software" to varying degrees, and during the development of this software I often found I needed patches for the compilers themselves, for various reasons.

In Coq, I discovered that the interchange format between coqtop and Proof General was undocumented, ad hoc, and didn't have enough metadata. In Ur/Web, we ran into a variety of infelicities spanning all parts of the system, from the frontend to the compiler. However, none of these issues ended up being large problems. Having the author of one of these compilers be your advisor is certainly helpful! Additionally, it is surprisingly easy to make modifications to the frontends of compilers, since there is not very much deep algorithmic content and the typechecker helps ensure any changes you make get propagated everywhere necessary. I wrote new command line options for Ur/Web and Coq in comparatively small amounts of time. Maintainers of active research software tend to be quite receptive to these "engineering" patches, which serve no direct research purpose, and I consider these contributes to be a vital component of being a good citizen of the open source community.

While I was not well versed enough with OCaml (esp. its FFI) to feel comfortable using it for this particular project, the IDE slave interface `Ide_slave` utilized by coqide in Coq 8.4 and later is probably the correct mechanism to interact with Coq interactively. (It was not pointed out to me until I had already done the bulk of work parsing Coq terms in Haskell.) This is made somewhat more difficult by the fact that the interchange format is undocumented; however, I've filed a bug[9]. With any luck, it will hopefully

[9] https://coq.inria.fr/bugs/show_bug.cgi?id=2777

do better than my patch, which added the beginnings of PGIP support to Coq. My patch, while sufficient for my needs (and likely any other similar applications) is unlikely to make it into the Coq mainline any time soon, as the PGIP project is inactive, and the other user, Isabelle, has discontinued using their PGIP backend.

Needless to say, the lack of compelling interchange formats is probably impeding the progress on novel interface paradigms for proof assistants. I hear the pastures may be greener for other proof assistants, e.g. Isabelle.

The tactical library in Coq

The primary mechanism by which support for new logics may be added in Logitext is by the development of tactical libraries in Coq. This is because the ordinary mechanism by which users interact with Coq is driven by practical concerns; by limiting ourselves to tactics in our tactic library, we can present the stylized forms of proof taught in logic proofs, rather than the rough and ready Coq tactic script style. Here is a simple proof written in our tactic language:

```
Goal denote ( nil |= [ (exists x, P x) -> ~ (forall x, ~ P x) ] ).
  sequent.
    rImp Con0.
    lExists Hyp0.
    rNot Con0.
    lForall Hyp0 x.
    lNot Hyp0.
    myExact Hyp0.
Qed.
```

A nontrivial amount of support code was necessary to support classical logic (see below), but the tactics for intuitionistic logic were quite simple. In general, the better supported the reasoning style by the theorem prover, the easier the tactical library is to build.

Duality in classical logic

Coq is first and foremost a theorem prover for *intuitionistic* logics; while there is tactical support for classical logic, this support is less well devel-

oped than tactics which also work in intuitionistic settings. For example, in classical sequent calculus, sequents can take an arbitrary number of clauses to the right of the turnstile; it is well known that by limiting the number of clauses to one, you end up with intuitionistic sequent calculus. There is no native representation for when many goals are possible, comparable to named hypotheses, which live on the left side of the turnstile.

During the development of the tactic library for the sequent inference rules in Coq, I discovered a nice trick that allows us to abuse Coq's named hypotheses to implement named conclusions. The basic idea is to use the contrapositive rule and deMorgan laws to swap the hypotheses and conclusions. In particular:

$$A, B \vdash C, D$$

$$A \wedge B \vdash C \vee D$$

$$\neg(C \vee D) \vdash \neg(A \wedge B)$$

$$\neg C \wedge \neg D \vdash \neg A \vee \neg B$$

$$\neg C, \neg D \vdash \neg A, \neg B$$

Now implementing right-elimination rules simply reduces to implementing left-elimination rules for *negated* terms, and we can take advantage of Coq's named hypotheses to make the job easy. The rest is just a lot of Ltac coding.

DSLs should have an FFI

One crucial feature in the Ur/Web compiler that made it possible to use it for this project was its FFI. Ur/Web is a domain specific language for writing web applications, and among other things it does not include robust systems libraries (e.g. executing external processes and interfacing with them). Indeed, Ur/Web's design makes this sort of support difficult: all side-effectful transactions need to also be able to be rolled back, and this is rather hard to achieve for general input-output. However, with an FFI, we can implement any code which needs this library support in a more suitable language, wrap it up in an API which gives the appropriate transactional guarantees, and let Ur/Web use it. Without it, we would not have been able to use Ur/Web. An FFI constitutes an extremely flexible escape hatch, and should be early on the checklist of anyone writing a new programming language.

However, because functions which manipulate C pointers are non-transactional, Ur/Web is limited to FFI functions which handle basic C types, e.g. integers and strings. Thus the question of parsing becomes one of utmost importance for Ur/Web, as strings are the preferred interchange format for complex structures. This question is dealt with in more detail in the following section.

Metaprogramming in Ur/Web and Haskell

In order to pass data between Ur/Web and Haskell, we utilized strings containing JSON encoded data. This was primarily motivated by the fact that Ur's metaprogramming library already had support for parsing basic JSON (the only features we needed to add were support for polymorphic variants and recursive datatypes, to deal with ordinary algebraic data types on the Haskell side). Because I had to make modifications on both the Haskell side and the Ur/Web side, JSON serialization offers an interesting case study for the different metaprogramming styles offered by the two languages.

In Haskell, the Scrap Your Boilerplate library allows us to do metaprogramming directly over any algebraic datatype defined in Haskell. Originally, we simply used an existing generic algorithm defined in the `aeson` library; however, for interoperability reasons we ended up having to make a few tweaks to better match up our representation with the Ur/Web data model.

In Ur, there are no metaprogramming facilities for dealing with traditional algebraic datatypes (defined using the `datatype` keyword). Instead, metaprogramming is done over type-level records, which can be instantiated into ordinary records and polymorphic variants. Fortunately, it is a relatively simple task to translate non-recursive algebraic datatypes to their record-based kin:

```
datatype foo = Bar | Baz of int * int
```

translates into:

```
type foo = variant [Bar = unit, Baz = int * int]
```

Recursive data types are more problematic. Currently, Logitext is using a solution using the module system to implement a Mu combinator and the JSON typeclass, but the solution is rather unsatisfactory. We suspect, however, that a better story for recursive polymorphic variants may require some changes to Ur/Web's type system.

Conclusion

We have shown some of the issues that arise when taking a well understood specification of software, and then implementing it with pre-existing academic software, and in the process, covered a wide variety of programming languages topics.

With the recent renewed interest in online learning, including the Coursera, Udacity and MITx initiatives, I believe educational proof assistants will have an important role to play in the development of these courses (or already are, in the case of the DeduceIt system at Stanford). However, as we have seen with prior work in this area, very little of these systems gets reused from project to project. As the scope of these systems and the complexity of the underlying theories increases, this approach may soon become untenable.

References

[Bertot94] Y. Bertot, G. Kahn, and L. Théry, "Proof by pointing[10]," Theoretical Aspects of Computer Software, 1994.

[Broda04] K. Broda, J. Ma, G. Sinnadurai, and A. Summers, "Pandora: Making Natural Deduction Easy[11]," 2004.

[Bornat99] R. Bornat and B. Sufrin, "Animating formal proof at the surface: the Jape proof calculator[12]," The Computer Journal, vol. 42, no. 3, 1999.

[Gasquet11] O. Gasquet, F. Schwarzentruber, and M. Strecker, "Panda: A Proof Assistant in Natural Deduction for All. A Gentzen Style Proof Assistant for Undergraduate Students[13]," Tools for Teaching Logic, pp. 85-92, 2011.

[Machin11] B. Machín and L. Sierra, "Yoda: a simple tool for natural

[10] http://www.springerlink.com/content/f0443836661r1x37

[11] http://people.inf.ethz.ch/summersa/wiki/lib/exe/fetch.php?media=papers:pandora.pdf

[12] http://comjnl.oxfordjournals.org/content/42/3/177

[13] http://www.springerlink.com/content/036825021556t540/

deduction[14],” Tools for Teaching Logic, Third International Congress, TICTTL, 2011.

[Pitt95] J. Pitt, “MacKE: Yet another proof assistant & automated pedagogic tool[15],” Theorem Proving with Analytic Tableaux and Related, no. Esprit 6471, 1995.

[14] http://logicae.usal.es/TICTTL/actas/MachinSierra.pdf
[15] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.1048