UNPRL: NUPRL PROOF RE-EVALUATION LOGIC

RYAN KAVANAGH

Abstract. The Unprl proof assistant is introduced. Unprl generalises the Nuprl proof assistant's client-server architecture, thereby increasing the system's fault tolerance. Unprl's proof generation system far surpasses the current state of the art in artificial intelligence, *a fortiori* surpassing the naïve proof search provided by other proof assistants. Implementing Unprl is discussed.

1. Introduction

Proof assistants have become an important tool in the programming language theorist's toolbox. Recently, they have found applications beyond the study of formal systems and the formalisation of mathematics. For example, they have been applied to verifying the safety of cyber-physical systems [Pla10] and in creating verified software toolchains [Con16]. Though we find this trend encouraging, we believe the difficulty and tedium associated with using proof assistants impedes their widespread adoption. To allay these burdens, we propose several improvements to the Nuprl proof assistant described in [Con+85].

Nuprl is an influential proof assistant originally developed in the 1980s and it is still under development today. Despite its longevity and success, it suffers from using a peculiar closed-source client-server architecture. A single server instance is known to exist and is located at Cornell, and one gains access to it by e-mailing a project member. Though this client-server paradigm may on the surface appear unusual, it is perfectly reasonable: operating Nuprl utilises scarce resources and is inherently Oz-like. Indeed, it is widely believed[1] that whenever a user connects to the server, an alarm goes off at Cornell and the first author of the Nuprl book [Con+85] (hereinafter referred to as "the man behind the curtain") rushes to the terminal in the server room. Then, as the remote user enters his proofs, the man behind the curtain quickly checks them and informs the user of their correctness. Though the mental prowess underlying Nuprl is indisputable, the astute reader will immediately see the unsustainability of the Nuprl architecture.

We introduce a new proof assistant called Unprl that generalises Nuprl's client-server-proof checker model, thereby building on the strengths of Nuprl while addressing its aforementioned Achilles heel. Moreover, Unprl provides a proof generation mechanism far surpassing the state of the art in artificial intelligence, *a fortiori* surpassing the naïve proof search provided by other proof assistants. We examine

2010 *Mathematics Subject Classification.* Primary: 68T15; Secondary: 03B35, 03B70.
Key words and phrases. Proof assistant; Nuprl; The Wizard of Oz; Artificial artificial intelligence.
[1]Private communications at POPL'16.

these improvements in section 2 and discuss several techniques that may lead to performance increases in section 3. Implementation details are discussed in section 4.

2. Work Distribution and Proof Generation

Rather than assume a single "man behind the curtain", Unprl generalises Nuprl's client-server-proof checker model to support multiple "people behind the curtain". In particular, we assume an odd number of undergraduate students $u_1, \dots, u_n$ to be behind the curtain (the reader is referred to Appendix A for assistance in determining the parity of a given n). Unprl's client-server-multiple proof checkers architecture is depicted in Figure 1.

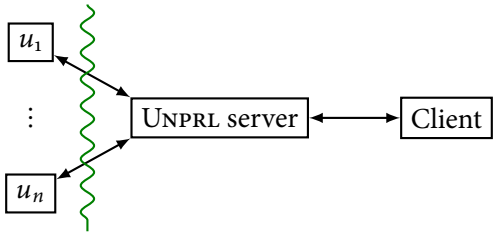

Figure 1. Unprl's client-server-multiple proof checkers architecture.

We now pass to study the benefits of this simple and obvious generalisation. These benefits are captured by Unprl's two main *modes* of operation.

Unprl's main mode is its *proof checking mode*. In this mode, the server idles until it receives a theorem statement T, a proposed proof P for the theorem, and an acceptance threshold $0 \leq A \leq 1$. Upon their receipt, the Unprl server transmits a copy of T and P to each student behind the curtain. Each student then either replies to the server that they accept the proof, or they reject it and provide the server with a proof of $\neg T$. If at least An students accept the proof, then the server marks the proof as correct and notifies the client that P is a proof of T. Otherwise, there was disagreement and we must re-evaluate the proof. We proceed by breadth-first search to reach consensus as follows. The existence of disagreement implies the server received counter-proofs $P_1, \dots, P_s$ for some $s \geq 1$. The server enqueues the triples $(\neg T, P_i, A)$ for $i = 1, \dots, s$. We dequeue a triple (τ, π, A) and reenter proof checking mode, checking theorem statement τ with proof π and threshold A. If π is accepted for τ, we apply double-negation elimination to τ to determine if we should accept T. If we accept T, we inform the client that $\hat{\pi}$ is a proof of T, where $\hat{\pi}$ is π plus any double-negation eliminations required to transform π from a proof of τ to a proof of T. If we reject T, we inform that $\hat{\pi}$ is a proof of $\neg T$, where $\hat{\pi}$ is π plus any double-negation eliminations required to transform π from a proof of τ to a proof of $\neg T$. Otherwise, we received counter-proofs $P'_1, \dots, P'_t$ to τ, so we enqueue the triples $(\neg\tau, P'_i, A)$ and continue with our search. Upon termination, the client will have a verified proof of either T or $\neg T$.

Unprl's second mode is its *proof generation mode*. In this mode, the server idles until it receives a theorem statement T and acceptance threshold $0 \leq A \leq 1$. Upon

receipt, the UNPRL server switches into proof checking mode and attempts to check the proof triple $(T, \textbf{true}, A)$, where **true** is any tautology or always provable sentence. When the proof checking mode terminates, it will have returned a theorem-proof pair, providing either a verified proof of T or a verified proof of $\neg T$. These results are conveyed to the client. The deluxe version of UNPRL provides the option to perform proof search by problem set, though the latency is obviously increased.

3. Analysis and Incentivisation

Though a marked improvement on Nuprl, UNPRL suffers from several deficiencies that we address in this section.

3.1. **Proof Checking Mode.** Though UNPRL's run-time *bus factor* can be made arbitrarily large, its accuracy may not increase correlatedly. Moreover, it is not possible in general to estimate the likelihood that an arbitrary group **U** of undergraduate students u_i accepts an incorrect proof. Indeed, assuming **U** is composed predominantly of freshmen, the probability that the theorem statement and proof pair given in Figure 2 is accepted is believed to be arbitrarily close to 1, also independently of the acceptance threshold specified. However, the theorem is clearly false of the classical reals and the proof should have been rejected. Thankfully, this does not mean we should abandon UNPRL. Indeed, if **U** is composed predominantly of students having taken a course in analysis, the likelihood of the proof pair being rejected is close to 1, independently of the acceptance threshold specified. This example illustrates the difficulty in quantifying the probability that proofs accepted by UNPRL are correct. However, we believe that this example demonstrates the need to ensure that the u_i be uniformly sampled from the undergraduate student population.

To encourage "correct" results, we propose rewarding the proof checkers when they provide "correct" responses.

We assume the client can allocate $r > 0$ reward resources[2] to verifying k theorems using n undergraduates. For each theorem T_i, $i = 1, \ldots, k$, let c_{ij} be the number of rounds in which undergraduate u_j's response to the server agreed with the final result. In other words, let c_{ij} be the number of times u_j accepted a proof of a τ double-negation equivalent to T_i if T_i was accepted or the number of times u_j accepted a proof of a τ double-negation equivalent to $\neg T_j$ if T_j was rejected. Then, at the end of the theorem proving session, each undergraduate u_j receives

$$\frac{\sum_{i=1}^{k} c_{ij}}{\sum_{i=1}^{k}\sum_{j=1}^{n} c_{ij}}$$

of the reward r. Key to our strategy is that the reward received corresponds to a fraction of all correct answers in the system, and so the more a given undergraduate surpasses his peers, the greater his reward. It is thus against the undergraduates' individual best interests to collude and secretly agree to accept all proofs.

To further counteract any chance of collusion, we suggest running two instances of UNPRL in parallel, each receiving $\frac{r}{2}$ resources. Then, for each T_i, randomly assign

[2]Experience has shown food to be the most effective form of reward.

Theorem 1. *The set of (classical) reals $\mathbb{R}$ is a singleton set.*

Proof. First observe first that in $\mathbb{N}$,

$$0 = 0^2 = (1-1)^2 = (1+(-1))^2 = 1^2 + (-1)^2 = 1 + 1 = 2.$$

We proceed by induction on n to show that either $n = 0$ or $n = 1$. Indeed, the base case is trivial, so assume the claim holds for a given k and consider $n = k + 1$. By the induction hypothesis, either $k = 0$ or $k = 1$. In the first case, we have $n = k+1 = 0+1 = 1$, and we're done. Otherwise, $k = 1$ and $n = k + 1 = 1 + 1 = 2 = 0$, and we're done.

Now, consider the fractions $\mathbf{0} := \frac{0}{1}$ and $\mathbf{1} := \frac{1}{1}$. We wish to show that they are equivalent, written $\mathbf{0} \sim \mathbf{1}$. Observe that $\mathbf{0} \sim \frac{2}{2}$, since $0 * 2 = 2 * 1$ (recall, $2 = 0$). Moreover, $\frac{2}{2} \sim \mathbf{1}$, since $2 * 1 = 1 * 2$. So by transitivity of $\sim$ (see [Lan09, Theorem 39]), we have $\mathbf{0} \sim \mathbf{1}$. We now show that for any arbitrary fraction $\frac{m}{n}$, $\frac{m}{n} \sim \mathbf{0}$. By the above, either $m = 0$ or $m = 1$. If $m = 0$, then $\frac{m}{n} \sim \mathbf{0}$ because $m * 1 = 0 = 0 * n$. In the second case, we must have either $n = 0$ or $n = 1$. If $n = 1$, then $\frac{m}{n} \sim \mathbf{1}$, because $m * 1 = 1 * 1 = 1 * n$, and so by symmetry [Lan09, Theorem 38] and transitivity of $\sim$, we have $\frac{m}{n} \sim \mathbf{0}$. Otherwise, we fall into the case of $m = 1$ and $n = 0$. Then by [Lan09, Theorem 40], $\frac{m}{n} \sim \frac{m}{n} \times \frac{n}{n} = \frac{0}{n}$. Then by the above $\frac{0}{n} \sim \mathbf{0}$, and we're again done by transitivity. Following Landau [Lan09, Definition 16], we thus conclude that there is a unique rational number: the set of fractions equivalent to $\mathbf{0}$.

This implies that there is a unique Cauchy sequence of rational numbers, namely the constant sequence of $\mathbf{0}$s. But its limit with respect to the standard metric is $\mathbf{0}$, and so the rationals are a complete metric space. But $\mathbb{R}$ is defined to be the completion of $\mathbb{Q}$, and so we conclude $\mathbb{Q} = \mathbb{R}$. But $\mathbb{Q}$ is a singleton set, so so is $\mathbb{R}$. □

FIGURE 2. A proof freshmen dream about.

T_i to one and $\neg T_i$ to the other, and withhold compensation for the T_i on which the two instances disagree. Not knowing if their instance received the true version of a theorem but knowing that they must reach the same conclusion as the other instance, the checkers will have no choice but to actually check the proof. Recent advances in friendship logic appearing at this conference may shed light on whether this overhead is truly necessary.

3.2. **Proof Generation Mode.** Given that the proof generation mode is a special case of the proof checking mode, it is sufficient to adapt the above incentivisation scheme. Suppose the client has r reward resources to allocate and k theorems to prove. Then allocate $\frac{r}{2}$ resources to the incentivisation of the k top-level proof checking

calls. Additionally, reward each u_i with $\frac{p_i}{2k}$ of the reward, where p_i is the number of final accepted proofs that u_i provided.

4. Implementation

Implementation is left as an exercise for the reader. Readers are especially encouraged to verify their implementation using Unprl.

5. Related Works

Many proof assistants have been introduced over the years, ranging from Edinburgh LCF in 1979 [GMW79] to Nuprl [Con+85] to Coq [The04]. These have lead to remarkable achievements of verified mathematics. For example, the Feit-Thompson Odd Order Theorem in Coq was recently verified in Coq [Gon+13]. While not seeking to diminish the works on which Unprl builds, we believe Unprl is a significant improvement on its predecessors.

First, Unprl's architecture does not shackle the client to any given set of axioms, assuming the axioms permit double-negation elimination.[3] In other words, we provide clients with the freedom to specify the axiomatic framework in which they wish to verify their proofs and prove their theorems. Our legal counsel has urged us, however, to caution users of Unprl against using the patented axiomatic system underlying Estatis Inc.'s *Falso HyperVerifier* [Est].

Finally, we allay concerns that may have arisen following section 3 regarding the risk of accepting an incorrect proof. We believe that this risk is no greater than the one inherent in other state-of-the-art proof systems. For example, it is possible to prove False in Coq 8.4.5 by exploiting a bug in the `vm_compute` command when there is a type with more than 255 constructors [DPC15]. Moreover, we believe that the incentivisation scheme we proposed is sufficient to counteract the risks.

Acknowledgements

The author gratefully acknowledges various insightful conversations on the mechanics of Nuprl with Olivier Savary Bélanger and other POPL'16 attendees.

Appendix A. Odd Integers

Determining the parity of integers is a difficult task. However, as a service to the reader, we provide the following functions. The function `returnTrueIfOdd`, implemented in Standard ML, returns true if applied to an odd integer. As a supplement, the function `returnTrueIfNotOdd`, also implemented in Standard ML, returns true if applied to an integer that isn't odd. The author has formally verified that these functions satisfy their specifications.

```
fun returnTrueIfOdd    (n : int) = true
fun returnTrueIfNotOdd (n : int) = true
```

[3] A simple work-around to the (not not absurd) objections[4] to double-negation elimination involves simply looping until a direct proof to T accepted by *An* of the students is produced.

[4] Those who object to double-negation elimination *cannot* be offended by this remark if they are at all consistent.

References

[Con+85] R. L. Constable et al. *Implementing Mathematics with The Nuprl Proof Development System*. Prentice-Hall, 1985.

[Con16] DeepSpec Consortium. *DeepSpec: The Science of Deep Specification*. http://deepspec.org/. 2016.

[DPC15] Maxime Dénès, Pierre-Marie Pédrot and Guillaume Claret. *clarus/falso: A proof of false.* 2015. URL: https://github.com/clarus/falso.

[Est] Estatis Inc. *Falso HyperVerifier and HyperProver*. URL: http://inutile.club/estatis/falso/.

[GMW79] Michael J. Gordon, Arthur J. Milner and Christopher P. Wadsworth. *Edinburgh LCF*. Vol. 78. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1979. ISBN: 978-3-540-09724-2. DOI: 10.1007/3-540-09724-4.

[Gon+13] Georges Gonthier et al. 'A machine-checked proof of the odd order theorem'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7998 LNCS (2013), pp. 163–179. ISSN: 03029743. DOI: 10.1007/978-3-642-39634-2_14.

[Lan09] Edmund Landau. *Foundations of Analysis*. Trans. from the German by F. Steinhardt. 3rd ed. Reprint of the 1966 edition. Providence, Rhode Island: AMS Chelsea Publishing, 2009. ISBN: 0-8218-2693-X.

[Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Heidelberg: Springer, 2010. ISBN: 978-3-642-14508-7. DOI: 10.1007/978-3-642-14509-4.

[The04] The Coq development team. *The Coq proof assistant reference manual*. Version 8.0. LogiCal Project. 2004. URL: http://coq.inria.fr.

E-mail address: rkavanagh@cs.cmu.edu

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213-3891