Nuprl's Inductive Logical Forms

Mark Bickford, Robert L. Constable, Rich Eaton and Vincent Rahli[*]

Cornell University

Abstract

For more than a decade, we have been working on specifying, verifying, and synthesizing asynchronous distributed protocols using the Nuprl proof assistant. Only recently we have been able to do so for complicated protocols such as Paxos. This has been made possible thanks to the level of abstraction of our specification language called the Logic of Events (LoE). This paper discusses our main automation tool, namely our Inductive Logical Forms (ILFs), which are first order formulas that characterize the responses of a system to events in terms of observations made at causally prior events.

Nuprl's type theory. The Nuprl proof assistant [12, 3] implements a type theory called *Constructive Type Theory* (CTT), which is an extensional dependent type theory à la Martin-Löf. It is based on an untyped functional programming language à la Curry. Nuprl has a rich type theory including dependent product and sum types, identity (or equality) types, a hierarchy of universes, disjoint union types, W types, quotient types [14], set types, union and (dependent) intersection types [15, 22], image types [25], PER types [4], approximation and computational equivalence types [19, 28], and partial types [16].

Type checking is undecidable but in practice this is mitigated by type inference and checking heuristics implemented as tactics. Moreover, to avoid proving well-formedness conditions (i.e., doing type checking) altogether, we often do untyped reasoning using Howe's computational equivalence relation (an observational congruence, which we write as $\sim$) [19, 28]. For example, we can prove that for all terms f and t (t need not be a list), $\mathtt{map}(f, t)$ `@` `nil` $\sim \mathtt{map}(f, t)$. We can then rewrite $\mathtt{map}(f, t)$ `@` `nil` into $\mathtt{map}(f, t)$ anywhere in a sequent without having to prove any well-formedness condition.

Allen developed a semantics for Nuprl where types are defined as partial equivalence relations (PERs) on closed terms [2, 1, 16]. We have implemented this semantics in Coq and verified a large number of Nuprl's inference rules [6, 7] (Nuprl's consistency follows from the fact that its inference rules are valid w.r.t. Allen's PER semantics and that False is not inhabited).

We have also recently turned Nuprl into a fully intuitionistic theory following Brouwer's principles. Using our Coq framework, we have proved Brouwer's continuity principle for numbers [27]. Following Dummett's "standard" classical proof [17, pp.55], we have also proved (in Prop and using classical axioms) the truth of several bar induction rules (see, e.g., [21, 11, 31]), such as bar induction on decidable bars for free choice sequences of numbers [26].

By default the Nuprl system is distributed and runs in the cloud. Alternatively, Nuprl can run locally using one of our virtual machines available at `http://www.nuprl.org/vms/`. Nuprl is composed of several processes: *database* and *library* processes to store and access definitions, lemmas, and proofs; *refiner* processes to apply inference rules; and *editor* processes, which are user interfaces. It also allows several users to simultaneously edit a proof, and users to simultaneously *refine* several subgoals in a proof.

Nuprl led to other similar proof assistants such as MetaPRL [18] and JonPRL [20]. It has been used over the years to both formalize mathematical results and develop verified software.

[*]This work was partially supported by the DARPA CRASH project, award number FA8750-10-2-0238, by the SnT, and by the National Research Fund Luxembourg (FNR), through PEARL grant FNR/P14/8149128.

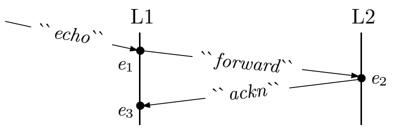
For example, we have recently proved a completeness result of intuitionistic first-order logic [13], and we have built distributed systems including a verified ordered broadcast service [33, 30, 32].

The Logic of Events. To specify, verify, and synthesize asynchronous distributed protocols, we use two languages, both implemented in Nuprl: (1) a high-level specification language called the Logic of Events (LoE) [8, 10] to specify and reason about the information flow of distributed program runs; and (2) a low-level programming language, the General Process Model (GPM) [9], to implement these information flows.

LoE, related to Lamport's notion of causal order [23], was developed to reason about events occurring in the execution of a distributed system, where an event is an abstract entity corresponding to the receipt of a message; the message is called the *primitive information* of the event. An event happens at a specific point in space/time. The space coordinate of an event is called its location, and the time coordinate is given by a well-founded causal ordering on events that totally orders all events at the same location. Using LoE one can describe systems in terms of the causal relations among events and (ultimately) their primitive information. LoE has been used among other things to verify consensus protocols [33] and cyber-physical systems [5].

We have also developed a programming language called EventML, to automatically generate both LoE specifications and GPM programs from EventML specifications [29]. Once we have extracted the semantic meaning of an EventML specification in terms of a LoE formula F and a GPM program P, we automatically prove that P satisfies F. It remains to interactively prove that the LoE formula F satisfies the desired correctness properties.

To reason about a protocol in LoE, we reason about its possible runs. An *event ordering* is an abstract representation of one run of a distributed system; it provides a formal definition of a *message sequence diagram* as used by systems designers. It is a structure consisting of: (1) a set of events; (2) a function `loc` that associates a *location* with each event; (3) a function `info` that associates primitive information with each event; and (4) a well-founded *causal ordering* relation, $<$, on events [23]. We express system properties as predicates on event orderings. A system satisfies such a property if every execution satisfies the predicate.



The message sequence diagram on the right depicts a simple event ordering with events e_1 and e_3 happening at location L1, and e_2 at location L2. Event e_1 happens causally before e_2, which happens causally before e_3. We write $e_1 < e_2$, $e_2 < e_3$, and $e_1 <_{\texttt{loc}} e_3$.

In LoE, we specify systems by defining and combining *event observers* [8] (which can be regarded as the combinations of *event recognizers* and *event handlers*). An event observer is a function that assigns to any event ordering eo and event e in that event ordering eo, an unordered bag of outputs observed (or produced) at e. For example, the following observer recognizes every event and observes its location: $\lambda eo.\lambda e.\{\texttt{loc}(e)\}$. We also have *primitive observers* to, e.g., run two processes in parallel or to build state machines.

We reason about event observers in terms of the *event observer relation*, which relates events, observers, and observations: we say that the observer X observes v at event e (in an event ordering eo), and write $v \in X(e)$, if v is a member of the bag $(X\ \ eo\ \ e)$.

Formally verifying distributed protocols is not trivial and can be time consuming. Our main automation tool to assist us in this task is called an Inductive Logical Form (ILF).

Inductive Logical Forms. An ILF is a first order formula that characterizes the responses of a system to events in terms of observations made at causally prior events. For example, in Paxos [24, 34]—a state machine replication protocol, if a leader L decides that slot n is to be filled with command c, then that means that c was proposed to the leader L at an earlier event.

ILFs are automatically generated from observers using logical simplifications, and characterizations of the LoE combinators. For example, one of the simplest but subtle such characterizations is the one for our parallel combinator "$\_ \,||\, \_$", which allows one to run two processes in parallel: $v \in X\,||\,Y(e) \iff \downarrow(v \in X(e) \;\vee\; v \in Y(e))$. This says that v is produced by $X\,||\,Y$ iff it is produced by either of its components (see [29] for more details).

Given an observer X, i.e. an LoE specification, we wrote a program that starts with a formula of the form $v \in X(e)$, and keeps on rewriting it using equivalences such as the one presented above (and also applies various logical simplifications), to finally generate a formula of the form $v \in X(e) \iff C$, where C is a complete declarative characterization of X's outputs. Finally, we have built a proof tactic that automatically proves such double implications.

Rewriting using such formulas we can easily trace back the outputs of a distributed system to the states of its state machines and to its inputs. It turns out that when proving safety properties of distributed systems, much of the effort is spent tracing back outputs to inputs. ILFs helped us automate this reasoning process.

References

[1] Stuart F. Allen. A non-type-theoretic definition of Martin-Löf's types. In *LICS*, pages 215–221. IEEE Computer Society, 1987.

[2] Stuart F. Allen. *A Non-Type-Theoretic Semantics for Type-Theoretic Language.* PhD thesis, Cornell University, 1987.

[3] Stuart F. Allen, Mark Bickford, Robert L. Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, and Evan Moran. Innovations in computational type theory using Nuprl. *J. Applied Logic*, 4(4):428–469, 2006. `http://www.nuprl.org/`.

[4] Abhishek Anand, Mark Bickford, Robert L. Constable, and Vincent Rahli. A type theory with partial equivalence relations as types. Presented at TYPES 2014, 2014.

[5] Abhishek Anand and Ross Knepper. ROSCoq: Robots powered by constructive reals. ITP 2015 (`http://www.cs.cornell.edu/~aa755/ROSCoq/`), 2015.

[6] Abhishek Anand and Vincent Rahli. Towards a formally verified proof assistant. In *ITP 2014*, volume 8558 of *LNCS*, pages 27–44. Springer, 2014.

[7] Abhishek Anand and Vincent Rahli. Towards a formally verified proof assistant. Technical report, Cornell University, 2014. `http://www.nuprl.org/html/Nuprl2Coq/`.

[8] Mark Bickford. Component specification using event classes. In *Component-Based Software Engineering, 12th Int'l Symp.*, volume 5582 of *LNCS*, pages 140–155. Springer, 2009.

[9] Mark Bickford, Robert Constable, and David Guaspari. Generating event logics with higher-order processes as realizers. Technical report, Cornell University, 2010.

[10] Mark Bickford, Robert L. Constable, and Vincent Rahli. Logic of events, a framework to reason about distributed systems. In *Languages for Distributed Algorithms Workshop*, 2012.

[11] Douglas Bridges and Fred Richman. *Varieties of Constructive Mathematics.* London Mathematical Society Lecture Notes Series. Cambridge University Press, 1987.

[12] R.L. Constable, S.F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W. Harper, D.J. Howe, T.B. Knoblock, N.P. Mendler, P.Panangaden, J.T. Sasaki, and S.F. Smith. *Implementing mathematics with the Nuprl proof development system.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986.

[13] Robert Constable and Mark Bickford. Intuitionistic completeness of first-order logic. *Annals of Pure and Applied Logic*, 165(1):164–198, January 2014.

[14] Robert L. Constable. Constructive mathematics as a programming logic I: some principles of theory. In *Fundamentals of Computation Theory, Proceedings of the 1983 International*, volume 158 of *LNCS*, pages 64–77. Springer, 1983.

[15] Robert L. Constable and Jason Hickey. Nuprl's class theory and its applications. In *Foundations of Secure Computation*, NATO ASI Series, Series F: Computer & System Sciences, pages 91–116. IOS Press, 2000.

[16] Karl Crary. *Type-Theoretic Methodology for Practical Programming Languages.* PhD thesis, Cornell University, Ithaca, NY, August 1998.

[17] Michael A. E. Dummett. *Elements of Intuitionism.* Clarendon Press, second edition edition, 2000.

[18] Jason J. Hickey. *The MetaPRL Logical Programming Environment.* PhD thesis, Cornell University, Ithaca, NY, January 2001.

[19] Douglas J. Howe. Equality in lazy computation systems. In *Proceedings of Fourth IEEE Symposium on Logic in Computer Science*, pages 198–203. IEEE Computer Society, 1989.

[20] JonPRL. `http://www.jonprl.org/`.

[21] S.C. Kleene and R.E. Vesley. *The Foundations of Intuitionistic Mathematics, especially in relation to recursive functions.* North-Holland Publishing Company, 1965.

[22] Alexei Kopylov. *Type Theoretical Foundations for Data Structures, Classes, and Objects.* PhD thesis, Cornell University, Ithaca, NY, 2004.

[23] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[24] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[25] Aleksey Nogin and Alexei Kopylov. Formalizing type operations using the "image" type constructor. *Electr. Notes Theor. Comput. Sci.*, 165:121–132, 2006.

[26] Vincent Rahli and Mark Bickford. Coq as a metatheory for Nuprl with bar induction. CCC 2015 (`http://www.nuprl.org/html/Nuprl2Coq/`), 2015.

[27] Vincent Rahli and Mark Bickford. A nominal exploration of intuitionism. Extended version avaible at `http://www.nuprl.org/html/Nuprl2Coq/`, 2015.

[28] Vincent Rahli, Mark Bickford, and Abhishek Anand. Formal program optimization in Nuprl using computational equivalence and partial types. In *ITP'13*, volume 7998 of *LNCS*, pages 261–278. Springer, 2013.

[29] Vincent Rahli, David Guaspari, Mark Bickford, and Robert L. Constable. Formal specification, verification, and implementation of fault-tolerant systems. Accepted to AVoCS 2015, 2015.

[30] Vincent Rahli, Nicolas Schiper, Robbert Van Renesse, Mark Bickford, and Robert L. Constable. A diversified and correct-by-construction broadcast service. In *The 2nd Int'l Workshop on Rigorous Protocol Engineering (WRiPE)*, October 2012.

[31] Michael Rathjen. A note on bar induction in constructive set theory. *Math. Log. Q.*, 52(3):253–258, 2006.

[32] Nicolas Schiper, Vincent Rahli, Robbert Van Renesse, Mark Bickford, and Robert L. Constable. ShadowDB: A replicated database on a synthesized consensus core. In *Eighth Workshop on Hot Topics in System Dependability*, HotDep'12, 2012.

[33] Nicolas Schiper, Vincent Rahli, Robbert van Renesse, Mark Bickford, and Robert L. Constable. Developing correctly replicated databases using formal tools. In *DSN 2014*, pages 395–406. IEEE, 2014.

[34] Robbert van Renesse and Deniz Altinbuken. Paxos made moderately complex. *ACM Comput. Surv.*, 47(3):5:1–5:36, 2015.