Past, Present and Future of Nuprl

Vincent Rahli
http://www.nuprl.org
http://www.cs.cornell.edu/~rahli/

May 30, 2017
My collaborators

**PRL group**
- Abhishek Anand
- Mark Bickford
- Robert L. Constable
- Richard Eaton
- Vincent Rahli

**ATC-NY**
- David Guaspari
- Matt Stillerman

**System group**
- Robbert van Renesse
- Nicolas Schiper
Nuprl Environment

- Distributed
- Runs in the cloud
- Structure editor
- Tactic language: Classic ML
- Shared library
Nuprl Stack

- Tactics
- Refiner
- Inference rules
- Allen's PER semantics
- Howe's computational equality
- An untyped applied lambda-calculus
Nuprl Types

Based on Martin-Löf’s extensional type theory

**Equality**: \( a = b \in T \)

**Dependent product**: \( a: A \rightarrow B[a] \)

**Dependent sum**: \( a: A \times B[a] \)

**Universe**: \( \mathbb{U}_i \)
Nuprl Types

Less “conventional types”

Partial: $\overline{A}$

Disjoint union: $A + B$

Intersection: $\cap a:A.B[a]$

Union: $\cup a:A.B[a]$

Subset: $\{ a : A \mid B[a] \}$

Quotient: $T // E$

Domain: Base

Simulation: $t_1 \preceq t_2$

Bisimulation: $t_1 \sim t_2$

Image: $\text{Img}(A, f)$

PER: $\text{per}(R)$
Nuprl Types

Image type (Nogin & Kopylov)

Subset: \( \{ a : A \mid B[a] \} \triangleq \text{Img}(a:A \times B[a], \pi_1) \)

Union: \( \bigcup a:A.B[a] \triangleq \text{Img}(a:A \times B[a], \pi_2) \)
PER type (extensional)

\[
\text{Void} = \text{per}(\lambda_\_, \_ . 1 \lessdot 0)
\]

\[
\text{Top} = \text{per}(\lambda_\_, \_ . 0 \lessdot 0)
\]
PER type (extensional)

\[
\begin{align*}
\text{Void} & = \text{per}(\lambda _, _1 \preceq 0) \\
\text{Top} & = \text{per}(\lambda _, _0 \preceq 0) \\
\text{halts}(t) & = \text{Ax} \preceq (\text{let } x := t \text{ in Ax}) \\
A \sqcap B & = \sqcap x:\text{Base.} \sqcap y:\text{halts}(x).\text{isaxiom}(x, A, B) \\
T \sqcap E & = \text{per}(\lambda x, y. (x \in T) \sqcap (y \in T) \sqcap (E x y))
\end{align*}
\]
Nuprl Types

Squashing

\{Unit | T\}

\text{Img}(T, \lambda_.Ax)

T//\text{True}

\cap x:\neg T.\text{Void}

\text{per}(\lambda x.\lambda y.Ax \preceq x \sqcap Ax \preceq y \sqcap T)

\text{per}(\lambda x.\lambda y.x \in T \sqcap y \in T)

\text{per}(\lambda_.\lambda_.T)
Recursive types

_used to have Mendler’s recursive types._

_used still consistent?_

_used indexed W types from bar induction._
Nuprl Types

Rich type language facilitates specification

Makes type-checking harder
Nuprl’s proof engine is called a refiner

A generic goal directed reasoner:

- a rule interpreter
- a proof manager

Example of a rule

\[ H \vdash a : A \to B[a] \ [\text{ext } \lambda x. b] \]

BY [lambdaFormation]

\[ H, x : A \vdash B[x] \ [\text{ext } b] \]

\[ H \vdash A \in \mathbb{U}; \ [\text{ext } Ax] \]
Recent projects

What evidence do we have that (distributed) systems are correct?

What evidence do we have that our proofs are correct?
Recent projects

What evidence do we have that (distributed) systems are correct?

Platform to develop and reason about distributed systems.

What evidence do we have that our proofs are correct?

Building and verifying Nuprl in Coq.
Distributed systems are ubiquitous
Distributed Systems

What evidence do we have that these systems are correct?
What evidence do we have that these systems are correct?

Type checking

Testing
What evidence do we have that these systems are correct?

- Type checking
- Testing
- Model checking
Distributed Systems

What evidence do we have that these systems are correct?

- Type checking
- Testing
- Model checking
- Theorem proving
Distributed Systems

Distributed systems are hard to specify, implement and verify.

We need to tolerate failures.

It is hard to test all possible scenarios.

State space explosion using model checking.

Model checking often done on abstractions of the code rather than on the code itself.
Distributed Systems

We use Nuprl as a specification, programming and verification language.

Programming interface: a constructive specification language called EventML

Verification methodology
A logic of events implemented in Nuprl.

Specified, verified, and generated consensus protocols (e.g., Paxos) using EventML.

Aneris: a total ordered broadcast service.

ShadowDB: a replicated database with 2 parametrizable replication protocols (PBR & SMR) built on top of Aneris.

Improved performance without introducing bugs. We get decent performance.
Distributed Systems — Big picture

request

response

ShadowDB

replica 1
replica 2
replica f+1

replica f+1

replica 1
replica 2

Databases + Aneris interface

Aneris

...
Distributed Systems — Message sequence diagram

See: Paxos Made Moderately Complex
Distributed Systems — Combinators

EventML combinator

generate

Process combinator

implies

Logic of Events combinator

generate
Distributed Systems — Combinators

Delegation/bind

A → B

B(x)

B(x2)

B(x1)

b1 U ...

return

a
Distributed Systems — Combinators

- Application
- Buffer
- Recognizer
- (v,t)
- Update
- Handle
- Clock(0)
Distributed Systems — Verification

We use causal induction + inductive logical forms (ILFs) + state machine invariants

Vincent Rahli

Past, Present and Future of Nuprl

May 30, 2017
Distributed Systems — Verification

We use causal induction + inductive logical forms (ILFs) + state machine invariants

Vincent Rahli

Past, Present and Future of Nuprl
May 30, 2017
EventML for Paxos Synod:

```
agent Leader = SpawnFirstScout
  || ((LeaderPropose || Leader Adopted) >>= Commander)
  || (LeaderPreempted >>= Scout)
main Leader @ ldrs || Acceptor @ accpts
```
Distributed Systems — Code generation

Efficiency?

January 2012: 2 seconds per transaction

Revamped the whole system.
June 2012: 500 milliseconds per transaction

Optimization/compilation to Lisp.
End of 2012: 60 milliseconds per transaction (interpreted), 9 milliseconds per transaction (compiled)
Distributed Systems — What next?

- Crash-tolerant
- Byzantine fault-tolerant
- Nysiad
- Probabilistic systems

EventML
- Specification
  - EventML compiler
- LOE specification
  - Untrusted GPM code
  - Correct
  - Optimizer
  - Satisfiability proof
  - Correctness proof
- Runtime
  - SML interpreter
  - Ocaml interpreter
  - Lisp translator
- Scala interface?
- Complexity

Nuprl
- Logical simplifier
- ILF
- Correctness properties
- Manual proof
- Correctness proof

Manual
- Informal high-level specification
Correctness

What evidence do we have that these distributed systems are correct?

What evidence do we have that our proofs are correct?
Correctness

What evidence do we have that these distributed systems are correct?

Platform to develop and reason about distributed systems.

What evidence do we have that our proofs are correct?

Building and verifying Nuprl in Coq.
We build theorem provers to prove programs’ correctness
We build theorem provers to prove programs’ correctness

... but don’t use them to prove their own correctness
Nuprl in Coq — Our initial motivation

How do we know that our systems are sound?
How do we safely extend them?

- Proofs mostly carried out on paper.
- Not carried out in full detail.
- Spread over several papers/PhD theses.
- Precise metatheory, precise account of Nuprl.
Nuprl in Coq — Our initial motivation

Agda & Coq

- 2013/2014: bug in their termination checker

Nuprl

- Invalid rules
Nuprl in Coq — Our initial motivation

Agda & Coq

 رائع 2013/2014: بيت في تصميمهم checker

Nuprl

 رائع Invalid rules

How can we be sure that these rules are valid?

Nuprl’s PER semantics (where types are defined as partial equivalence relations on terms) in Coq and Agda.
Nuprl in Coq — Mechanization and Experimentation!

Mechanization

- Less error prone
- Easier to propagate changes
- Positive feedback loop
- Additive

Experimentation

- Adding new computations
- Adding new types
- Exploring type theory
- Changing the theory
Stuart Allen had his own meta-theory that was meant to be meaningful on its own and needs not be framed into type theory. We chose to use Coq and Agda.
Nuprl in Coq — An untyped $\lambda$-calculus

Parameterized by a library of definitions

Nominal features

Lazy exceptions

Provides a generic framework for defining and reasoning about programming languages using a “nominal” style
Nuprl in Coq — What we’ve implemented in Coq

- Tactics
- Refiner
- Inference rules
- Allen's PER semantics
- Howe's computational equality
- An untyped applied lambda-calculus
Nuprl in Coq — Howe’s computational equality

≼ is a simulation relation

∼ is a bisimulation relation \( (a ∼ b = a ≼ b \land b ≼ a) \)
Nuprl in Coq — Howe’s computational equality

\preceq is a simulation relation

\sim is a bisimulation relation \( (a \sim b = a \preceq b \land b \preceq a) \)

Purely by computation:

\[
\text{map}(f, \text{map}(g, l)) \sim \text{map}(f \circ g, l)
\]

Used for program optimization
Nuprl in Coq — Howe’s computational equality

≺ is a simulation relation

∼ is a bisimulation relation (a ∼ b = a ≺ b ∧ b ≺ a)

Purely by computation:

\[ \text{map}(f, \text{map}(g, l)) \sim \text{map}(f \circ g, l) \]

Used for program optimization

≺ and ∼ are congruences

Restricts the computation system
Let $\bot$ be $\text{fix}(\lambda x.x)$. 
Let $\bot$ be $\text{fix}(\lambda x.x)$.

**Least element**

$\forall t. \bot \preceq t$
Let $\bot$ be $\text{fix}(\lambda x.x)$.

**Least element**

$$\forall t. \bot \preceq t$$

**Least upper bound principle**

$G(\text{fix}(f))$ is the lub of the $\preceq$ chain $G(f^n(\bot))$ for $n \in \mathbb{N}$
Let \( \bot \) be \( \text{fix}(\lambda x.x) \).

**Least element**

\[ \forall t. \bot \preceq t \]

**Least upper bound principle**

\( G(\text{fix}(f)) \) is the lub of the \( \preceq \) chain \( G(f^n(\bot)) \) for \( n \in \mathbb{N} \)

**Compactness**

If \( G(\text{fix}(f)) \) converges, then there exists a natural number \( n \) such that \( G(f^n(\bot)) \) converges.
Nuprl in Coq — What we’ve implemented in Coq

- Tactics
- Refiner
- Inference rules
- Allen's PER semantics
- Howe's computational equality
- An untyped applied lambda-calculus
Nuprl in Coq — Allen’s PER semantics

Agda

Nuprl

Coq

Models only a finite number of universes

Universe 3
Universe 2
Universe 1
Universe 0

Universe 3
Universe 2
Universe 1
Universe 0

Prop + Axiom of functional choice

Uses induction-recursion

Uses induction + impredicativity

Vincent Rahli

Past, Present and Future of Nuprl

May 30, 2017
Allen’s PER semantics

\[ f_1 \equiv f_2 \in x : A \rightarrow B \]

\[ \text{type}((x : A \rightarrow B)) \land \forall a_1, a_2. \ a_1 \equiv a_2 \in A \Rightarrow \]
\[ f_1(a_1) \equiv f_2(a_2) \in B[x \setminus a_1] \]
Allen’s PER semantics

\[ f_1 \equiv f_2 \in x : A \rightarrow B \]

\[ \text{type}((x : A \rightarrow B)) \land \forall a_1, a_2. \ a_1 \equiv a_2 \in A \Rightarrow f_1(a_1) \equiv f_2(a_2) \in B[x \backslash a_1] \]

\[ t_1 \equiv t_2 \in \text{Base} \]

\[ t_1 \sim t_2 \]

\[ \text{Ax} \equiv \text{Ax} \in (a = b \in A) \]

\[ \text{type}((a = b \in A)) \land a \equiv b \in A \]

\[ t_1 \equiv t_2 \in \overline{A} \]

\[ \text{type}((\overline{A})) \land (t_1 \Downarrow \iff t_2 \Downarrow) \land (t_1 \Downarrow \Rightarrow t_1 \equiv t_2 \in A) \]
Allen’s PER semantics

\[ x_1 : A_1 \rightarrow B_1 \equiv x_2 : A_2 \rightarrow B_2 \]

\[ A_1 \equiv A_2 \land \forall a_1, a_2. a_1 \equiv a_2 \in A_1 \Rightarrow B_1[x_1 \backslash a_1] \equiv B_2[x_2 \backslash a_2] \]
Allen’s PER semantics

\[ x_1 : A_1 \to B_1 \equiv x_2 : A_2 \to B_2 \]

\[ A_1 \equiv A_2 \land \forall a_1, a_2. \ a_1 \equiv a_2 \in A_1 \Rightarrow B_1[x_1 \backslash a_1] \equiv B_2[x_2 \backslash a_2] \]

Base \equiv Base

\[ (a_1 = a_2 \in A) \equiv (b_1 = b_2 \in B) \]

\[ A \equiv B \land (a_1 \equiv b_1 \in A \lor a_1 \sim b_1) \land (a_2 \equiv b_2 \in A \lor a_2 \sim b_2) \]

\[ \overline{A} \equiv \overline{B} \]

\[ A \equiv B \land (\forall a. \ a \in A \Rightarrow a \downarrow) \]
Allen’s PER semantics

candidate type systems:

\[ \text{cts} = \text{CTerm} \rightarrow \text{CTerm} \rightarrow \text{per} \rightarrow \text{Univ} \]

where \[ \text{per} = \text{CTerm} \rightarrow \text{CTerm} \rightarrow \text{Univ} \]
Allen’s PER semantics

Ternary relations

candidate type systems:

\[ \text{cts} = \text{CTerm} \rightarrow \text{CTerm} \rightarrow \text{per} \rightarrow \text{Univ} \]

where \( \text{per} = \text{CTerm} \rightarrow \text{CTerm} \rightarrow \text{Univ} \)

Type constructors

Definition \( \text{per\_function} \) (\( \text{ts} : \text{cts} \)) : \text{cts} := \ldots \)
Allen’s PER semantics

Ternary relations

candidate type systems:

\[
cts = \text{CTerm} \rightarrow \text{CTerm} \rightarrow \text{per} \rightarrow \text{Univ}
\]

where \(\text{per} = \text{CTerm} \rightarrow \text{CTerm} \rightarrow \text{Univ}\)

Type constructors

Definition \(\text{per\_function}(ts : cts) : cts := \ldots\)

Closure

Inductive \(\text{close}(ts : cts) : cts := \ldots\)
Allen’s PER semantics

Ternary relations

candidate type systems:

\[ \text{cts} = \text{CTerm} \rightarrow \text{CTerm} \rightarrow \text{per} \rightarrow \text{Univ} \]

where \( \text{per} = \text{CTerm} \rightarrow \text{CTerm} \rightarrow \text{Univ} \)

Type constructors

Definition \text{per}_\text{function} (\text{ts} : \text{cts}) : \text{cts} := \ldots

Closure

Inductive \text{close} (\text{ts} : \text{cts}) : \text{cts} := \ldots

Universes

Fixpoint \text{univi} (i : \text{nat}) : \text{cts} := \ldots
Allen’s PER semantics

\[
\text{Fixpoint } \text{univi} \ (i : \text{nat}) \ (T \ T' : \text{CTerm}) \ (eq : \text{per}) : \text{Prop} \ := \\
\text{match } i \text{ with} \\
\mid 0 \Rightarrow \text{False} \\
\mid S \ n \Rightarrow \\
\ldots \\
\text{eq } \leftrightarrow 2 \Rightarrow (\text{fun } A \ A' \Rightarrow \{eqa : \text{per}, \text{close} (\text{univi} \ n) A A' eqa\}) \\
\ldots \\
\text{end.}
\]

Has to be in \text{Prop}, otherwise we can only define a finite number of universes
Allen’s PER semantics

**Definition**  
univ  \( T \ T' \) eq := \( \{ i : \text{nat}, \text{univi} i \ T \ T' \ \text{eq}\} \).

**Definition**  
nuprl := close univ.

\( t_1 \equiv t_2 \in T \ = \ \{ \text{eq} : \text{per}, \text{nuprl} \ T \ T \ \text{eq} \times \text{eq} \ t_1 \ t_2\} \)

\( T \equiv T' \ = \ \{ \text{eq} : \text{per}, \text{nuprl} \ T \ T' \ \text{eq}\} \)
Interesting fact: \( n: \mathbb{N} \rightarrow \mathbb{U}(n) \) is a Nuprl type
Interesting fact: \( n: \mathbb{N} \rightarrow \mathbb{U}(n) \) is a Nuprl type

...but it's not in any universe
Nuprl in Coq — What we’ve implemented in Coq

1. Tactics
2. Refiner
3. Inference rules
4. Allen's PER semantics
5. Howe's computational equality
6. An untyped applied lambda-calculus
The more (verified) rules the better

Expose more of the metatheory

Encode Mathematical knowledge

We have verified over 70 rules

Gives us the basis for a formally verified Nuprl
Nuprl in Coq — What now?

- Support for a library of definitions
- Experimenting with new types (e.g., PER types)
- Mendler’s recursive types?
- Experimenting with new computations
- Nominal type theory
- Continuity
- Bar induction
Write a parser

Build a verified refiner

Type checker/type inferencer?

Build a proof assistant