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

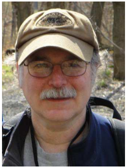
Richard Eaton

Vincent Rahli

ATC-NY

David Guaspari

Matt Stillerman



System group

Robbert van Renesse

Nicolas Schiper

Nuprl Environment

Distributed

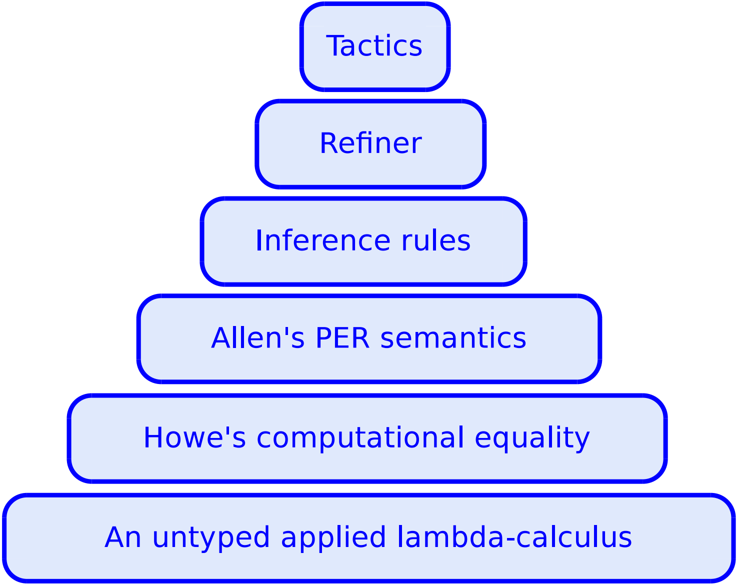
Runs in the cloud

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: $\mathbb{U}_i$

Nupr1 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: $\texttt{Base}$

Simulation: $t_1 \preccurlyeq t_2$

Bisimulation: $t_1 \sim t_2$

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

PER: $\texttt{per}(R)$

Nuprl Types

Image type (Nogin & Kopylov)

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

Union: $\cup a{:}A.B[a] \triangleq \texttt{Img}(a{:}A \times B[a], \pi_2)$

Nuprl Types

PER type (extensional)

$$\texttt{Void} = \texttt{per}(\lambda\_, \_.1 \preccurlyeq 0)$$

$$\texttt{Top} = \texttt{per}(\lambda\_, \_.0 \preccurlyeq 0)$$

Nuprl Types

PER type (extensional)

$$\texttt{Void} = \texttt{per}(\lambda\_, \_.1 \preccurlyeq 0)$$

$$\texttt{Top} = \texttt{per}(\lambda\_, \_.0 \preccurlyeq 0)$$

$$\texttt{halts}(t) = \texttt{Ax} \preccurlyeq (\texttt{let } x := t \texttt{ in Ax})$$

$$A \sqcap B = \cap x{:}\texttt{Base}. \cap y{:}\texttt{halts}(x).\texttt{isaxiom}(x, A, B)$$

$$T//E = \texttt{per}(\lambda x, y.(x \in T) \sqcap (y \in T) \sqcap (E\ x\ y))$$

Nuprl Types

Squashing

$\{\texttt{Unit} \mid T\}$

$\texttt{Img}(T, \lambda\_.\texttt{Ax})$

$\texttt{per}(\lambda x.\lambda y.\texttt{Ax} \preccurlyeq x \sqcap \texttt{Ax} \preccurlyeq y \sqcap T)$

$T//\texttt{True}$ $\quad$ $\texttt{per}(\lambda x.\lambda y.x \in T \sqcap y \in T)$

$\cap x{:}\neg T.\texttt{Void}$ $\quad$ $\texttt{per}(\lambda\_.\lambda\_.T)$

Nuprl Types

Recursive types

- Used to have **Mendler's recursive types**.
- Still consistent?
- **Indexed W types** from **bar induction**.

Nupri Types

Rich type language facilitates specification

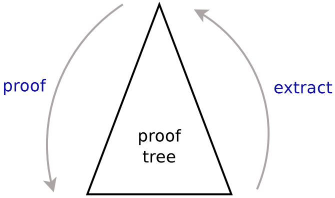
Makes type-checking harder

Refinements

Nuprl's proof engine is called a refiner

A generic goal directed reasoner:

- **a rule interpreter**
- **a proof manager**



Example of a rule

$$H \vdash a{:}A \rightarrow B[a] \ \lfloor \textbf{ext}\ \lambda x.b \rfloor$$

$$\text{BY } \texttt{[lambdaFormation]}$$

$$H, x : A \vdash B[x] \ \lfloor \textbf{ext}\ b \rfloor$$

$$H \vdash A \in \mathbb{U}_i \ \lfloor \textbf{ext}\ \texttt{Ax} \rfloor$$

Recent projects

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

What evidence do we have that our proofs are correct?

Recent projects

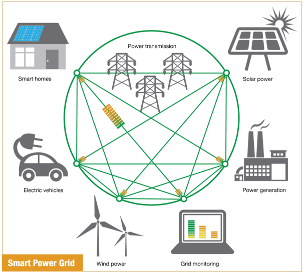
What evidence do we have that (distributed) systems are correct?

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?

Distributed Systems

What evidence do we have that these systems are correct?

Type checking

Testing

Distributed Systems

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**

Distributed Systems

A **logic of events** implemented in Nuprl.

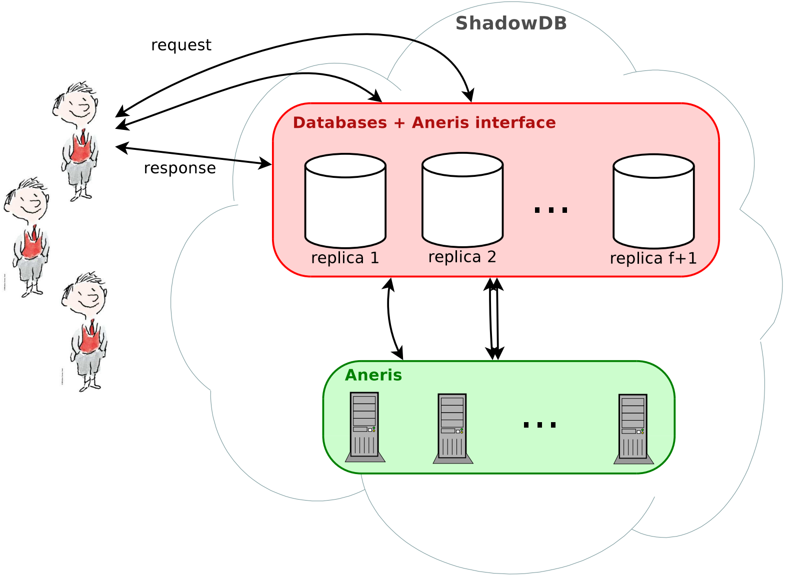
Specified, verified, and generated **consensus protocols** (e.g., Paxos) using **EventML**.

Aneris: a total ordered broadcast service.

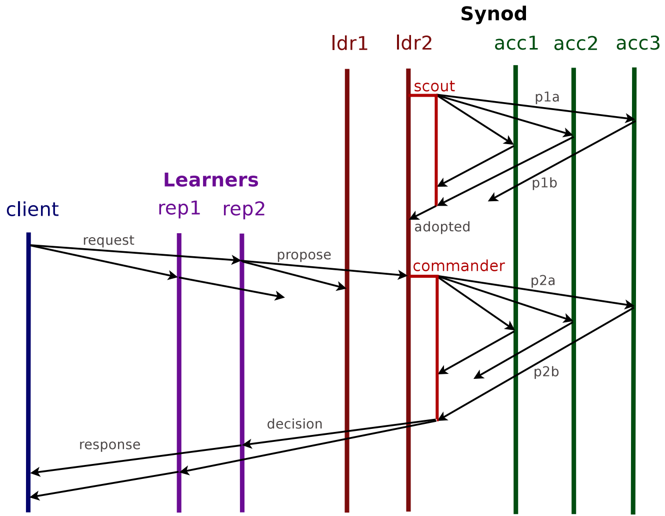
ShadowDB: a replicated database with 2 parametrizable replication protocols (PBR & SMR) built on top of Aneris.

Improved performance without introducing bugs.
We get **decent performance**.

Distributed Systems — Big picture

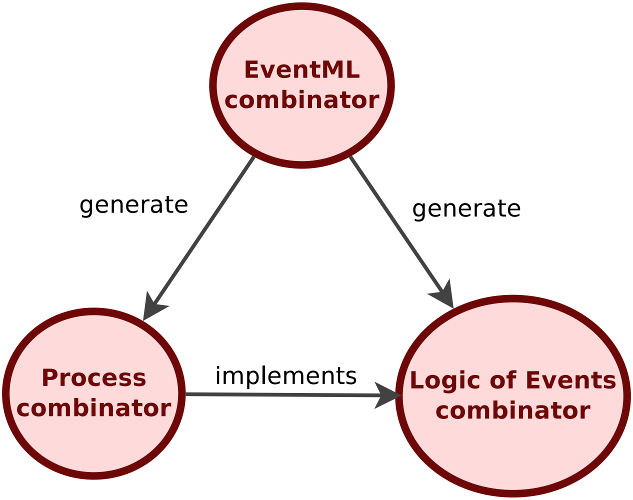


Distributed Systems — Message sequence diagram

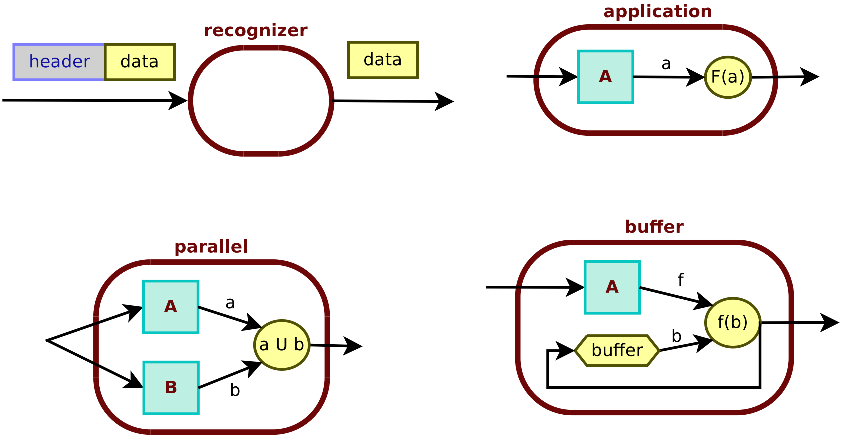


See: Paxos Made Moderately Complex

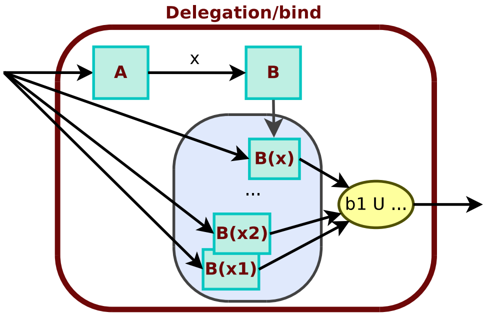
Distributed Systems — Combinators

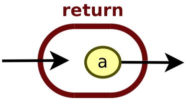


Distributed Systems — Combinators

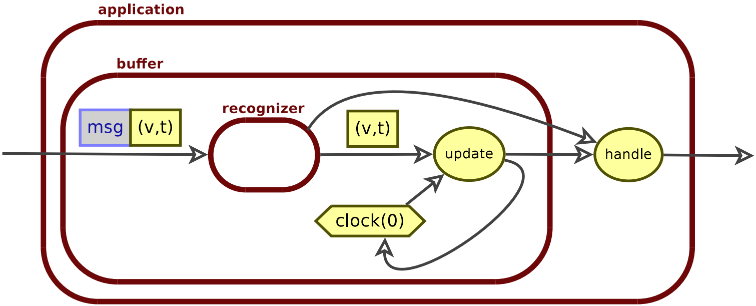


Distributed Systems — Combinators

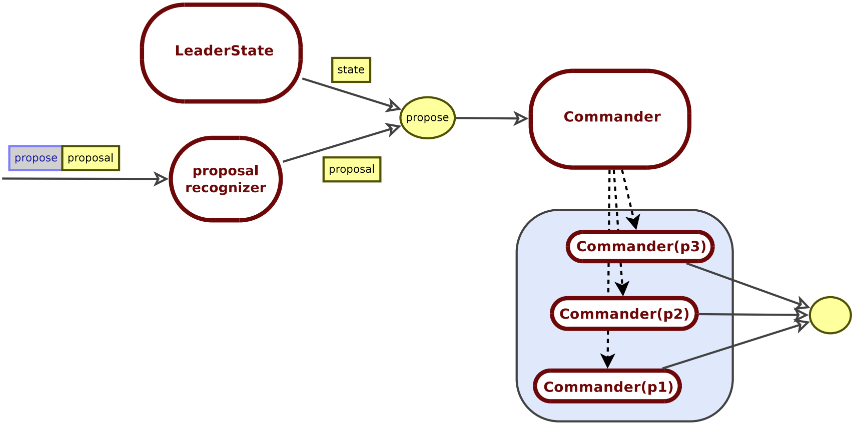




Distributed Systems — Combinators

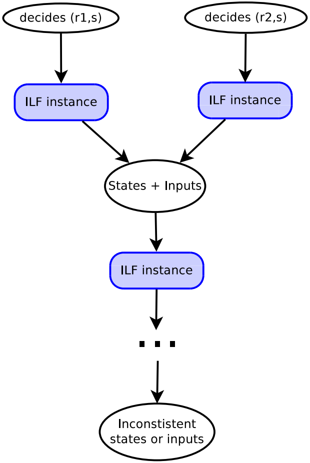


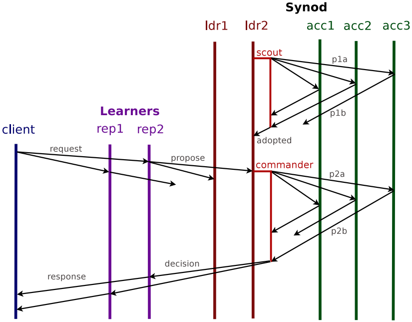
Distributed Systems — Combinators



Distributed Systems — Verification

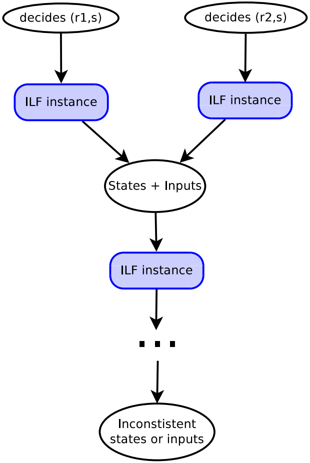
We use causal induction + inductive logical forms (ILFs) + state machine invariants

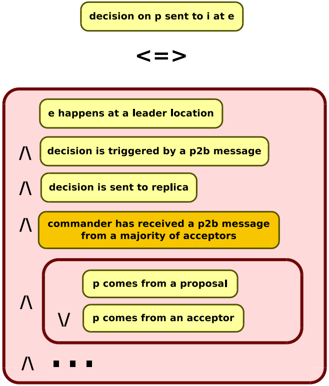




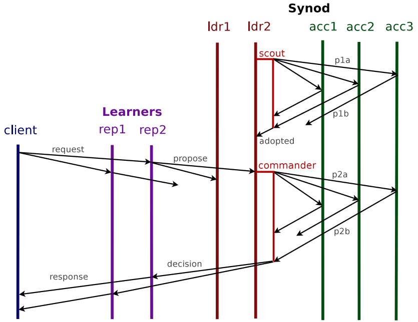
Distributed Systems — Verification

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





Distributed Systems — EventML



EventML for Paxos Synod:

```
...
agent Leader = SpawnFirstScout
            || ((LeaderPropose || LeaderAdopted) >>= Commander)
            || (LeaderPreempted >>= Scout) ;;
main Leader @ ldrs || Acceptor @ accpts
```

Distributed Systems — Code generation

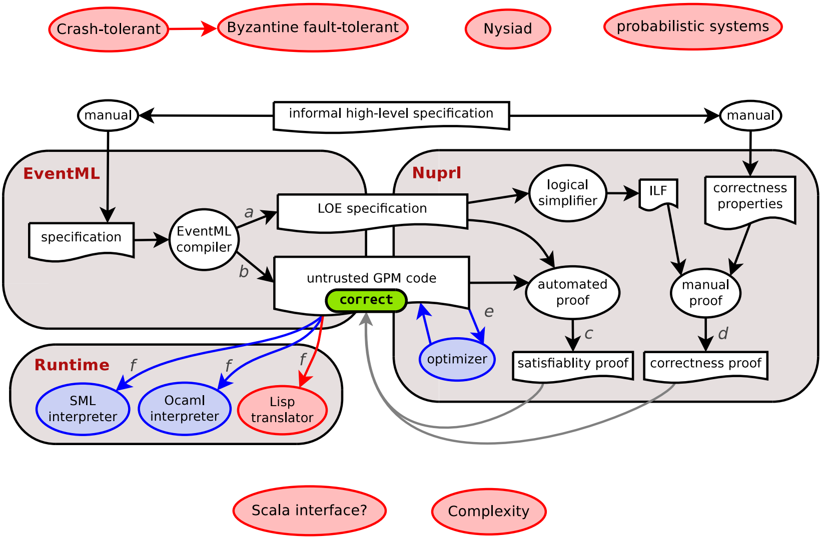
Efficiency?

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?



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.

Nuprl in Coq — Our initial motivation

We build theorem provers to prove programs' correctness

Nuprl in Coq — Our initial motivation

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

- 2013/2014: bug in their termination checker

Nuprl

- 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!

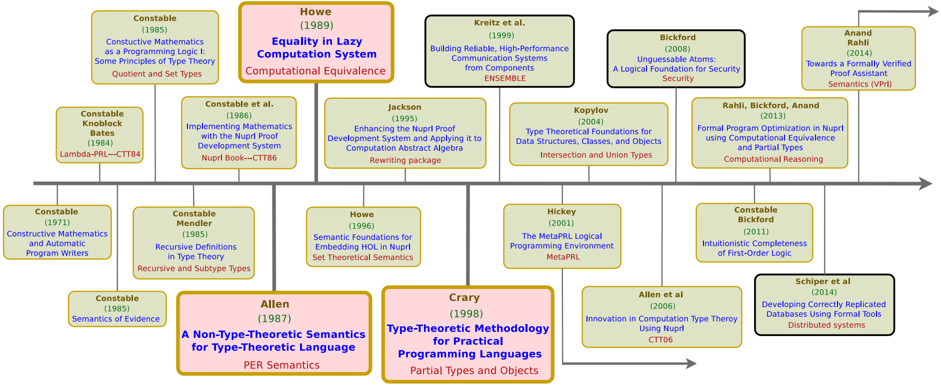
Mechanization

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

Experimentation

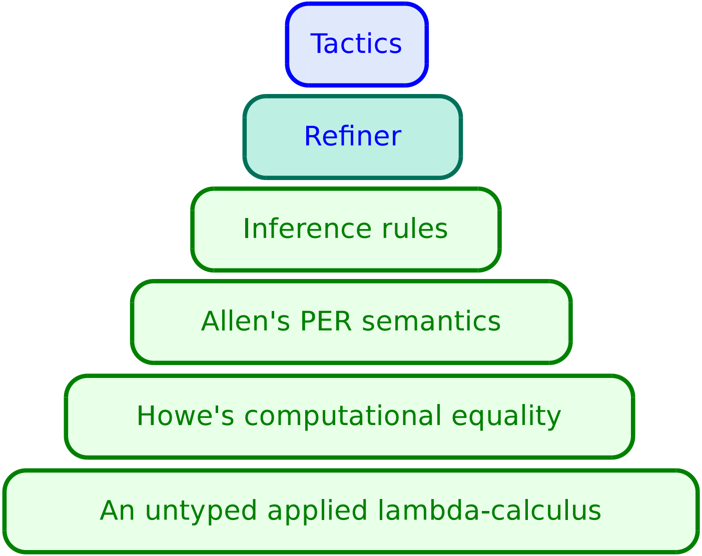
- Adding new computations
- Adding new types
- Exploring type theory
- Changing the theory

Nuprl in Coq — What do we cover?



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 — What we've implemented in Coq



Nuprl in Coq — An untyped λ-calculus

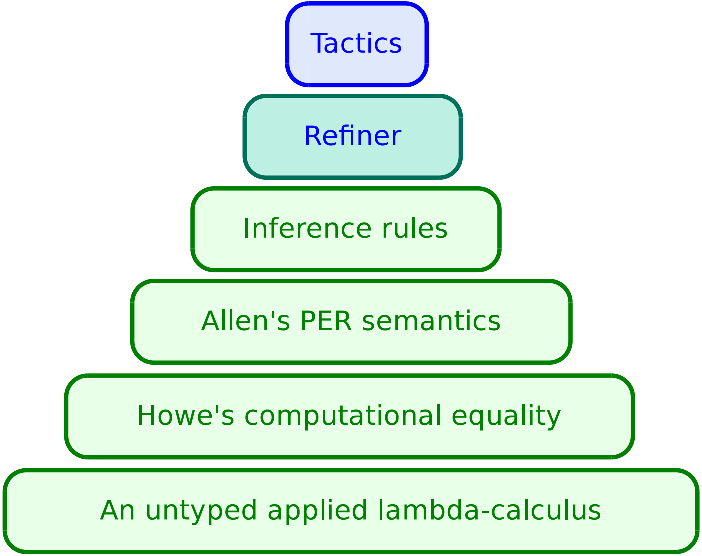
Parameterized by a library of definitions

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



Nuprl in Coq — Howe's computational equality

$\preccurlyeq$ is a simulation relation

$\sim$ is a bisimulation relation ($a \sim b = a \preccurlyeq b \;\wedge\; b \preccurlyeq a$)

Nuprl in Coq — Howe's computational equality

$\preccurlyeq$ is a simulation relation

$\sim$ is a bisimulation relation ($a \sim b = a \preccurlyeq b \ \wedge \ b \preccurlyeq a$)

Purely by computation:

$$\texttt{map}(f\,,\texttt{map}(g\,,l)) \ \sim \ \texttt{map}(f \circ g\,,l)$$

Used for program optimization

Nuprl in Coq — Howe's computational equality

$\preccurlyeq$ is a simulation relation

$\sim$ is a bisimulation relation ($a \sim b = a \preccurlyeq b \;\wedge\; b \preccurlyeq a$)

Purely by computation:

$$\mathtt{map}(f,\mathtt{map}(g,l)) \sim \mathtt{map}(f \circ g, l)$$

Used for program optimization

$\preccurlyeq$ and $\sim$ are congruences

Restricts the computation system

Nuprl in Coq — Constructive domain theory

Let $\bot$ be $\texttt{fix}(\lambda x.x)$.

Nuprl in Coq — Constructive domain theory

Let $\bot$ be $\texttt{fix}(\lambda x.x)$.

Least element

$$\forall t.\bot \preccurlyeq t$$

Nuprl in Coq — Constructive domain theory

Let $\perp$ be $\texttt{fix}(\lambda x.x)$.

Least element

$$\forall t.\perp \preccurlyeq t$$

Least upper bound principle

$G(\texttt{fix}(f))$ is the lub of the $\preccurlyeq$ chain $G(f^n(\perp))$ for $n \in \mathbb{N}$

Nuprl in Coq — Constructive domain theory

Let $\bot$ be $\texttt{fix}(\lambda x.x)$.

Least element

$$\forall t.\bot \preccurlyeq t$$

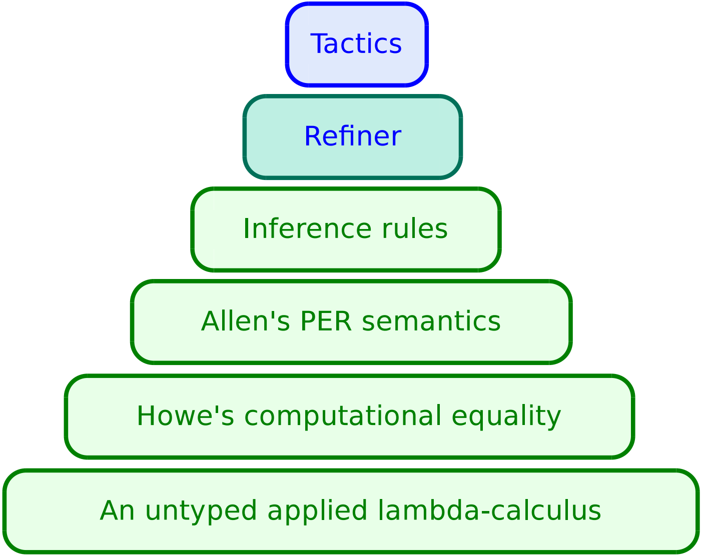
Least upper bound principle

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

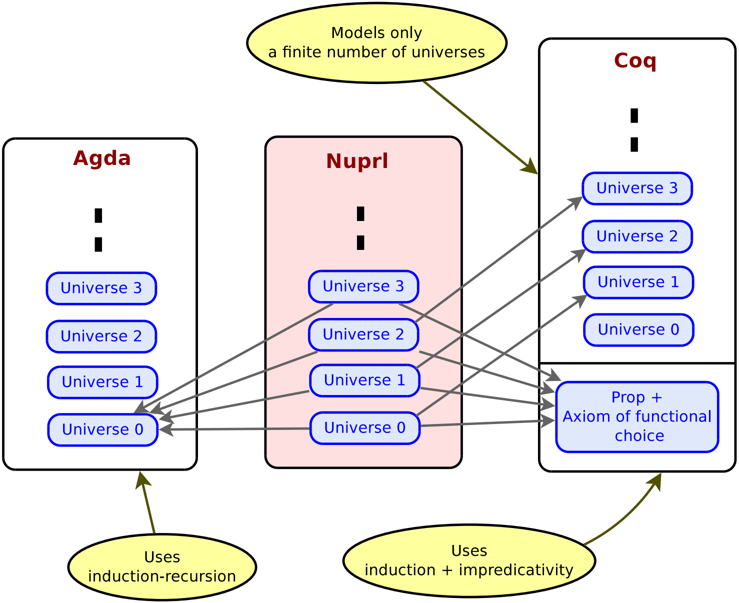
Compactness

if $G(\texttt{fix}(f))$ converges, then there exists a natural number n such that $G(f^n(\bot))$ converges

Nuprl in Coq — What we've implemented in Coq



Nuprl in Coq — Allen's PER semantics



Allen's PER semantics

$$f_1 \equiv f_2 \in x{:}A \rightarrow B$$

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

Allen's PER semantics

$f_1 \equiv f_2 \in x{:}A \to B$

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

$t_1 \equiv t_2 \in \texttt{Base}$

$$t_1 \sim t_2$$

$\texttt{Ax} \equiv \texttt{Ax} \in (a = b \in A)$

$$\texttt{type}((a = b \in A)) \land a \equiv b \in A$$

$t_1 \equiv t_2 \in \overline{A}$

$$\texttt{type}((\overline{A})) \land (t_1 \Downarrow \iff t_2 \Downarrow) \land (t_1 \Downarrow \Rightarrow t_1 \equiv t_2 \in A)$$

Allen's PER semantics

$$x_1{:}A_1 \rightarrow B_1 \equiv x_2{:}A_2 \rightarrow B_2$$

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

Allen's PER semantics

$$x_1{:}A_1 \rightarrow B_1 \equiv x_2{:}A_2 \rightarrow B_2$$

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

$$\mathtt{Base} \equiv \mathtt{Base}$$

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

$$A \equiv B \wedge (a_1 \equiv b_1 \in A \vee a_1 \sim b_1) \wedge (a_2 \equiv b_2 \in A \vee a_2 \sim b_2)$$

$$\overline{A} \equiv \overline{B}$$

$$A \equiv B \wedge (\forall a.\ a \in A \Rightarrow a \Downarrow)$$

Allen's PER semantics

Ternary relations

candidate type systems:

$$\mathtt{cts} = \mathrm{CTerm} \to \mathrm{CTerm} \to \mathrm{per} \to \mathtt{Univ}$$

where $\mathrm{per} = \mathrm{CTerm} \to \mathrm{CTerm} \to \mathtt{Univ}$

Allen's PER semantics

Ternary relations

candidate type systems:

$$\texttt{cts} = \mathrm{CTerm} \to \mathrm{CTerm} \to \mathrm{per} \to \texttt{Univ}$$

where $\mathrm{per} = \mathrm{CTerm} \to \mathrm{CTerm} \to \texttt{Univ}$

Type constructors

$\texttt{Definition}\ \mathrm{per\_function}\ (\mathit{ts} : \texttt{cts}) : \texttt{cts} := \ldots$

Allen's PER semantics

Ternary relations

candidate type systems:

$$\texttt{cts} = \mathrm{CTerm} \to \mathrm{CTerm} \to \mathrm{per} \to \texttt{Univ}$$

where $\mathrm{per} = \mathrm{CTerm} \to \mathrm{CTerm} \to \texttt{Univ}$

Type constructors

$\texttt{Definition}\ \mathrm{per\_function}\ (\mathit{ts} : \texttt{cts}) : \texttt{cts} := \ldots$

Closure

$\texttt{Inductive}\ \mathsf{close}\ (\mathit{ts} : \texttt{cts}) : \texttt{cts} := \ldots$

Allen's PER semantics

Ternary relations

candidate type systems:

$$\texttt{cts} = \mathrm{CTerm} \to \mathrm{CTerm} \to \mathrm{per} \to \texttt{Univ}$$

where $\mathrm{per} = \mathrm{CTerm} \to \mathrm{CTerm} \to \texttt{Univ}$

Type constructors

$\texttt{Definition}\ \mathrm{per\_function}\ (\mathit{ts} : \texttt{cts}) : \texttt{cts} := \ldots$

Closure

$\texttt{Inductive}\ \mathsf{close}\ (\mathit{ts} : \texttt{cts}) : \texttt{cts} := \ldots$

Universes

$\texttt{Fixpoint}\ \mathrm{univi}\ (i : \mathsf{nat}) : \texttt{cts} := \ldots$

Allen's PER semantics

```
Fixpoint univi (i : nat) (T T' : CTerm) (eq : per) : Prop :=
  match i with
  | 0 ⇒ False
  | S n ⇒
    ...
    eq ⇐2⇒ (fun A A' ⇒ {eqa : per , close (univi n) A A' eqa})
    ...
end.
```

Has to be in Prop, otherwise we can only define a finite number of universes

Allen's PER semantics

Definition univ $T\ T'\ eq := \{i : \mathsf{nat}\ ,\ \text{univi}\ i\ T\ T'\ eq\}$.

Definition nuprl := close univ.

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

$$T \equiv T' \;=\; \{eq : \text{per}\ ,\ \text{nuprl}\ T\ T'\ eq\}$$

Nuprl in Coq — Allen's PER semantics

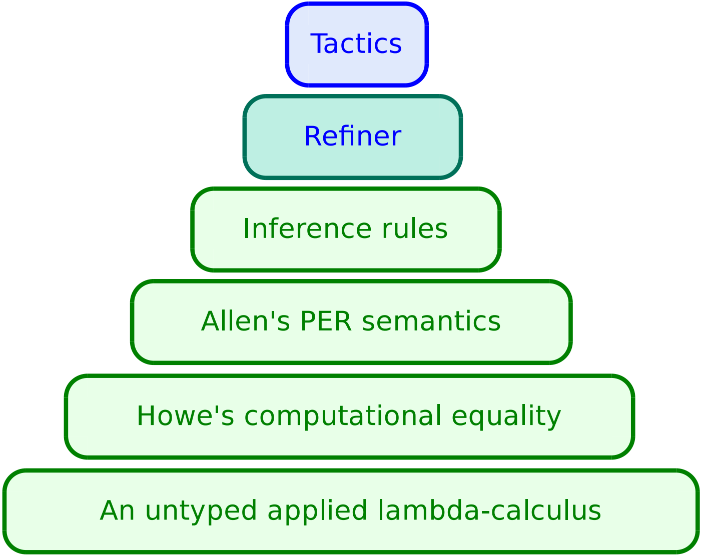
Interesting fact: $n{:}\mathbb{N} \to \mathbb{U}(n)$ is a Nuprl type

Nupirl in Coq — Allen's PER semantics

Interesting fact: $n{:}\mathbb{N} \to \mathbb{U}(n)$ is a Nuprl type

...but it's not in any universe

Nuprl in Coq — What we've implemented in Coq



Nuprl in Coq — Inference rules

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

Nuprl in Coq — What next?

Write a parser

Build a verified refiner

Type checker/type inferencer?

Build a proof assistant