Nuprl's Inductive Logical Forms

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

http://www.nuprl.org

October 7, 2015

Nuprl Environment

Distributed

Runs in the cloud

Structure editor

Tactic language: Classic ML

Shared library

Database based

Nuprl & Friends

Getting access to Nuprl:
`http://www.nuprl.org/html/NuprlSystem.html`

Virtual Machines: `http://www.nuprl.org/vms/`

MetaPRL: `http://metaprl.org` (dead?)

JonPRL: `http://www.jonprl.org/`

Nuprl Stack

Tactics

Refiner

Inference rules

Allen's PER semantics

Howe's computational equality

An untyped applied lambda-calculus

Howe's Computational Equality

$\preccurlyeq$ is a simulation relation

Greatest fixpoint of the following relation: $t\ [R]\ u$ if whenever t computes to a value $\theta(\overline{b})$, then u also computes to a value $\theta(\overline{b'})$ such that $\overline{b}\ R\ \overline{b'}$.

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

Purely by computation:

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

Howe's Computational Equality

Used for automated program optimization

$\preccurlyeq$ and $\sim$ are congruences

Restricts the computation system

Howe's Computational Equality

Type checking and type inference are undecidable

Proving that terms are well-formed can sometimes be cumbersome

Howe's untyped equality saves us from having to prove well-formedness

It turned out that many equalities could be stated using Howe's untyped equality

Constructive Domain Theory

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

Constructive Domain Theory

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

Least element

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

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 $n \in \mathbb{N}$

Constructive Domain Theory

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

Least element

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

Least upper bound principle

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

Compactness

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

Nuprl Types

Based on Martin-Löf's extensional type theory

Equality: $a = b \in T$

Dependent product: $a{:}A \to B[a]$

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: $\texttt{Base}$

Simulation: $t_1 \preccurlyeq t_2$

Bisimulation: $t_1 \sim t_2$

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

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

Nupri Types

Image type (Nogin & Kopylov)

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

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

Nuprl Types

PER type

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

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

Nuprl Types

PER type

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

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

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

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

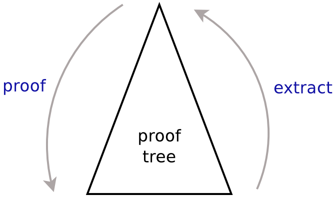
$$T//E = \mathtt{per}(\lambda x, y.(x \in T) \sqcap (y \in T) \sqcap (E\ x\ y))$$

Nuprl Refinements

Nuprl's proof engine is called a refiner (TB)

A generic goal directed reasoner:

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



Example of a rule

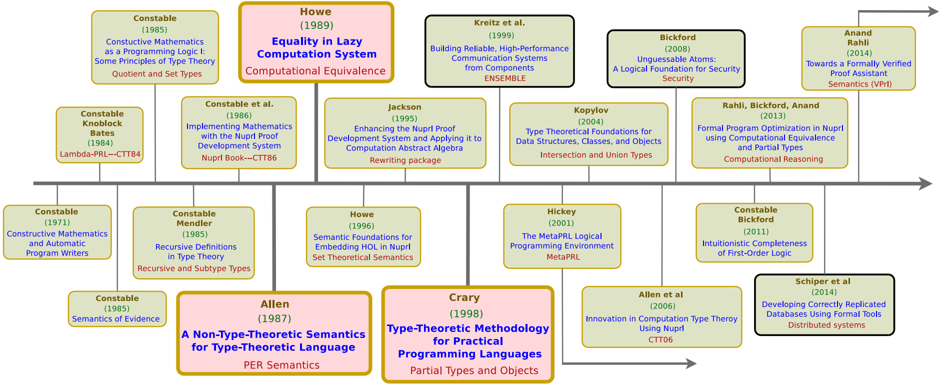
$$H \vdash a{:}A \to B[a] \ \lfloor \mathbf{ext}\ \lambda x.b \rfloor$$

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

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

$$H \vdash A \in \mathbb{U}_i \ \lfloor \mathbf{ext}\ \mathtt{Ax} \rfloor$$

Nuprl PER Semantics Implemented in Coq



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.

Intuitionistic Type Theory

We've proved these rules correct using our Coq model:

Bar induction

- On free choice sequences of closed terms without atoms
- We can build indexed W types

Brouwer's Continuity Principle for numbers

$$\mathbf{\Pi} F{:}\mathcal{B} \to \mathbb{N}.\mathbf{\Pi} f{:}\mathcal{B}.{\downarrow}\mathbf{\Sigma} n{:}\mathbb{N}.\mathbf{\Pi} g{:}\mathcal{B}.f =_{\mathbb{N}^{\mathbb{N}_n}} g \to F(f) =_{\mathbb{N}} F(g)$$

Verification of Distributed Systems

Verification of Distributed Systems

A **logic of events (LoE)** and a **general process model (GPM)** implemented in Nuprl.

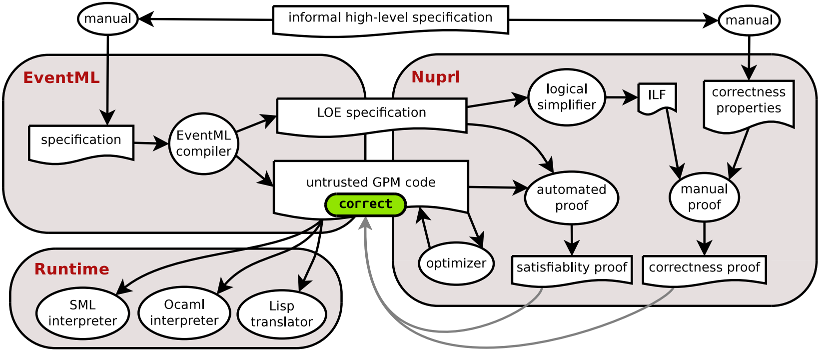
Specified, verified, and generated **consensus protocols** (e.g., 2/3-Consensus & Paxos) using **EventML**.

Aneris: a total ordered broadcast service.

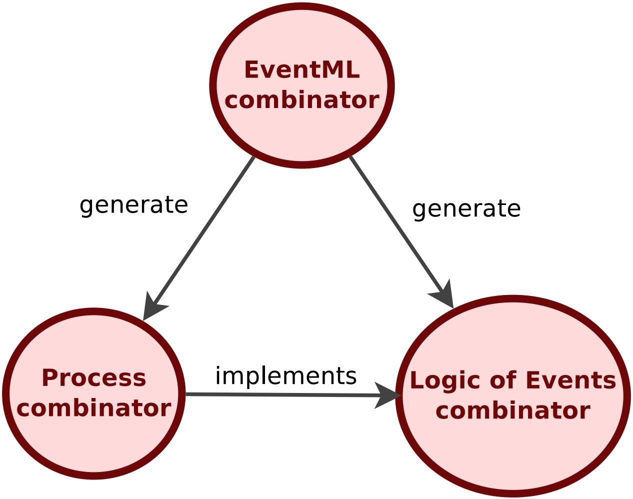
ShadowDB: a replicated database with 2 parametrizable replication protocols (PBR & SMR) built on top of Aneris.

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

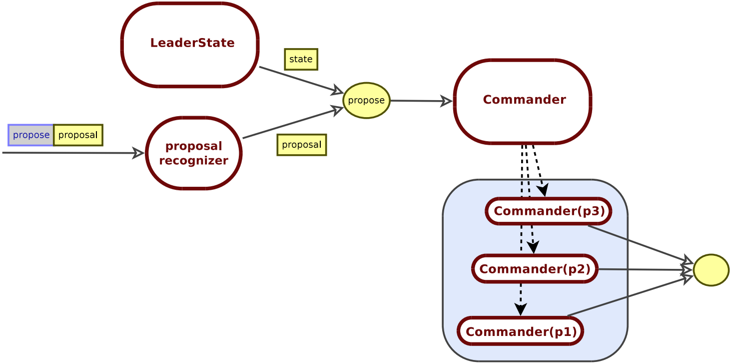
Our Methodology



Combinators



Combinators



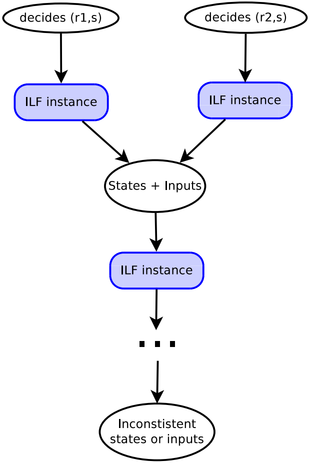
EventML for Paxos Synod:

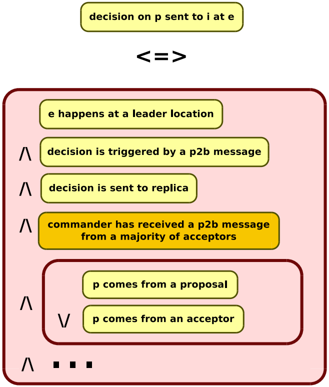
```
...
agent Leader = SpawnFirstScout
          || ((LeaderPropose || LeaderAdopted) >>= Commander)
          || (LeaderPreempted >>= Scout) ;;

main Leader @ ldrs || Acceptor @ accpts
```

Inductive Logical Forms

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





Inductive Logical Forms

E.g., logical explanation of why decisions are made by Paxos:

```
∀[Cmd:{T:Type| valueall-type(T)} ]. ∀[accpts,ldrs:bag(Id)]. ∀[ldrs_uid:Id ⟶ ℤ]. ∀[reps:bag(Id)].
∀[es:EO']. ∀[e:E]. ∀[i:Id]. ∀[p:Proposal].
 (decision'send(Cmd) i p ∈ pax_mb_main(Cmd;accpts;ldrs;ldrs_uid;reps)(e)
 ⟺ loc(e) ↓∈ ldrs
    ∧ (header(e) = ``pax_mb p2b``)
    ∧ (msgtype(e) = P2b)
    ∧ i ↓∈ reps
    ∧ (↓∃e':{e':E| e' ≤loc e }
        ∃z:PValue
         (((header(e') = [propose])
          ∧ (msgtype(e') = Proposal)
          ∧ ((↑ (proposal_slot (proposal_cmd LeaderStateFun(e'))))
            ∧ (¬↑ (in_domain (proposal_slot msgval(e')) (proposal_cmd (proposal_cmd LeaderStateFun(e'))))))
          ∧ (z = (mk_pvalue (proposal_slot LeaderStateFun(e')) msgval(e'))))
          ∨ ((header(e') = ``pax_mb adopted``)
          ∧ (msgtype(e') = pax_mb_AState(Cmd))
          ∧ ((astate_ballot msgval(e')) = (proposal_slot LeaderStateFun(e')))
          ∧ z ↓∈ map(λsp.(mk_pvalue (astate_ballot msgval(e')) sp);
                    update_proposals (proposal_cmd (proposal_cmd LeaderStateFun(e')))
                                     (pmax(ldrs_uid) (astate_pvals msgval(e'))))))
         ∧ (no commander_output(accpts;reps) z@Loc
              o (Loc,p2b'base(), CommanderState(accpts) (pval_ballot z) (proposal_slot (pval_proposal z)))
         between e' and e)
         ∧ ((pval_ballot z) = (bl_ballot (p2b_bl msgval(e))))
         ∧ ((proposal_slot (pval_proposal z)) = (p2b_slot msgval(e)))
         ∧ ((pval_ballot z) = (p2b_ballot msgval(e)))
         ∧ (#(CommanderStateFun(pval_ballot z;proposal_slot (pval_proposal z);es.e';e)) < threshold(accpts))
         ∧ (p = (pval_proposal z)))))
```

decision of p sent to i at e

e happens at a leader location

the decision is triggered by a p2b message

the recipient of the decision message is a replica

proposal p is extracted from a pvalue z

either pvalue z is made from a proposal and current ballot

or either pvalue z received in an adopted message or in leader state

this decision is the first output of the commander

the acceptor that sent the p2b message has accepted pvalue z

the commander has received a p2b messages from a majority of acceptors

Inductive Logical Forms

We found bugs using our ILFS

Could be used for blame tracking

Translate to English explanations?