# Programing in Nuprl

Vincent Rahli

May 30, 2017

# Table of contents

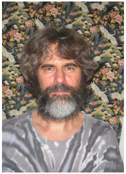
# PRL Group

**PRL group**

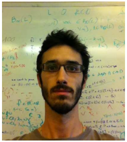
Robert L. Constable

Mark Bickford

Richard Eaton



Vincent Rahli



Abhishek Anand

**System group**

Robbert van Renesse

Nicolas Schiper

Ken Birman

# Goals

**Long term goal:** Develop provably correct code.

**Current Goals:**

- Ease the programming task (EventML).
- Domain specific programming (Logic of Events/EventML).
- Generate efficient code (Constructive domain theory).

Work done as part of the CRASH project (**Correct-by-Construction Attack-Tolerant Systems**) funded by DARPA (Defense Advanced Research Projects Agency).

# Table of Contents

# Nupri
## Computation System

A **constructive type theory**: CTT13 an evolution of CTT84 closely related to ITT82 [CAB+86, Kre02, ABC+06].

System **à la Curry** as opposed to Coq's system *à la* Church.

**Untyped**, **deterministic**, **lazy**, applied $\lambda$-calculus with: natural numbers, pairs, injections, fix operator, $\perp$, call-by-value operator,. . . .

# Nuprl

## Computation System

2 meta-relations defined on top of the evaluation function [How96]:

- approximation $\preccurlyeq$
- computation equivalence $\sim$ (a congruence).
  $a \sim b \triangleq a \preccurlyeq b \wedge b \preccurlyeq a$.

# Nuprl

## Computation System

2 meta-relations defined on top of the evaluation function [How96]:

- approximation $\preccurlyeq$
- computation equivalence $\sim$ (a congruence).
  $a \sim b \triangleq a \preccurlyeq b \wedge b \preccurlyeq a$.

```
Inductive approxn: nat -> term -> term -> Prop :=
| sql0 : forall t1 t2, approxn 0 t1 t2
| sqlS : forall n t1 t2,
           (forall op terms1 terms2,
                 computes_to t1 (CanonicalTerm op terms1)
              -> computes_to t2 (CanonicalTerm op terms2)
              -> forall a b,
                   In (a,b) (combine terms1 terms2)
                   -> approxn n a b)
           -> approxn (S n) t1 t2.

Definition approx t1 t2 := forall n, approxn n t1 t2.
```

For all terms $t$, $\bot \preccurlyeq t$.

$\langle \bot, 1 \rangle \preccurlyeq \langle 2, 1 \rangle$

$\texttt{halts}(t) \triangleq 0 \preccurlyeq (\texttt{let}\ x := t\ \texttt{in}\ 0)$

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

Type system built on top of the untyped computation system.

A type is a **partial equivalence relation** on $\lambda$-terms [All87a, All87b].

Computational semantics: applied $\lambda$-terms provide **evidence** for the truth of propositions.

A sequent $H \vdash C \; \lfloor_{\textbf{ext}} \; t \rfloor$ means that $C$ has computational evidence (extract) $t$ in context $H$.

# Nuprl
## Environment

Distributed.

Runs in the cloud.

Structured editor.

Shared library.

Tactic language: Classic ML.

# Nuprl

## ITT82 Types

**Equality**: $a = b \in T$
members: $\texttt{Ax}$.

**Dependent function**: $a{:}A \to B[a]$
members: $f$ such that $\forall a \in A,\ f(a) \in B[a]$
(Extensional function equality.)

**Dependent product**: $a{:}A \times B[a]$
members: $\langle a, b \rangle$

**Disjoint union**: $A{+}B$
members: $\texttt{inl}(a)$, $\texttt{inr}(b)$

**Universe**: $\mathbb{U}_i$
A hierarchy of universes to avoid Girard's paradox

# Nuprl

## Types

**Subtype**: $A \sqsubseteq B$

**Quotient**: $T//E$

**Intersection**: $\cap a{:}A.B[a]$

$\star$**Image**: $\texttt{Img}(T, f)$

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

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

**Recursive type**: $\texttt{rec}(F)$
where $F$ is a monotone function on types [Men88].

# Nupral

## Types

Constructive domain theory:

**Domain**: `Base`
closed terms of the computation system quotiented by $\sim$

$\star$**Approximation**: $a \preccurlyeq b$
members: `Ax`

**Computational equivalence**: $a \sim b$
members: `Ax`

$\star$**Partial types**: $\overline{T}$
contains all members of $T$ as well as all divergent terms

# Nuprl
## Types

$\texttt{True} \triangleq 0 \preccurlyeq 0$

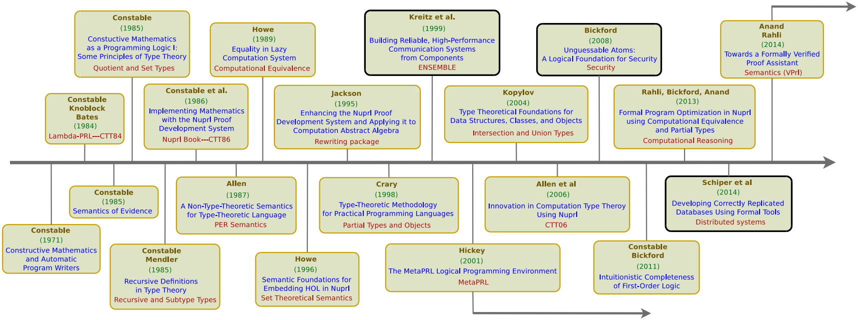
$\texttt{Void} \triangleq \texttt{False} \triangleq 0 \preccurlyeq 1$

$\texttt{Top} \triangleq \cap a{:}\texttt{Void}.\texttt{Void}$

$(\texttt{Type}, \sqsubseteq, \cap, \cup, \texttt{Top}, \texttt{Void})$ is a complete bounded lattice.

# Nuprl

## Timeline



# Table of Contents

# EventML

➲ **Formal Specification, Verification, and Implementation of Fault-Tolerant Systems**.

➲ **We built a developing tool**.

**ML-like**: an extension of ML.

**Domain-specific**: asynchronous distributed programs.

**Relies on a logical framework**: interacts with Nuprl.

# EventML
## Semantics

An EventML program corresponds to both:

- a **Logic of Events** (LoE) formula
- a **General Process Model** (GPM) program.

LoE formulas are used to reason about processes.

LoE formulas are implementable by GPM programs.

# EventML

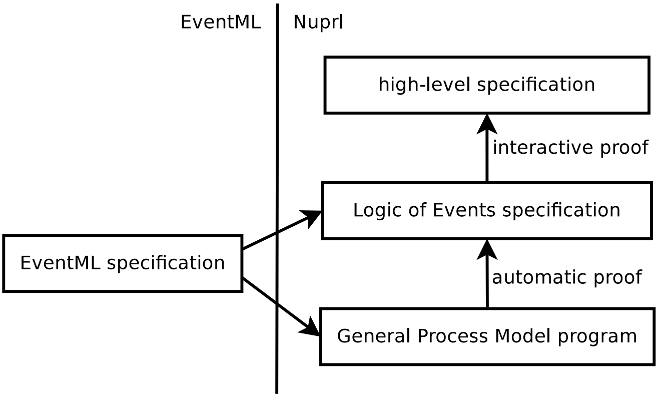
## Syntax

Similar to existing process calculi.

```
(* -- Leader -- *)

class Leader =
      (HigherLeader >>= LeaderStart)
   || (Once(start'base) >>= (\_.SpawnFirstScout))
   || ((LeaderPropose || LeaderAdopted) >>= Commander)
   || (LeaderPreempted >>= Scout)
   || ((\_.\loc.{pong'send loc ()}) o ping'base) ;;

(* ------ MAIN ------ *)

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

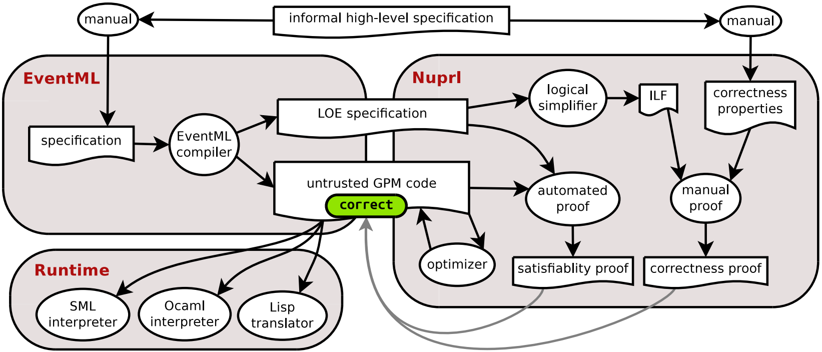
# EventML

## Workflow



# EventML

## Workflow



# EventML

## Verification

After spending several years on developing LoE, GPM, and EventML...

it took us:

- 2 days to prove the safety properties of 2/3-consensus,
- 3 weeks for Paxos Synod,
- 1 additional week to prove full Paxos (Synod + learners)

(Paxos is a widely used protocol to achieve software based replication which has many industrial implementations: Chubby, Megastore, . . . .)

➲ Used in ShadowDB (implemented by Nicolas Schiper).

# EventML

➲ **How efficient is our generated code?**

# Table of Contents

# Computational equivalence

**Motivation:**

Formal program optimization in an untyped setting [RBA13].

- ➲ More general
- ➲ More efficient

# Computational equivalence

**A simple example:**

$$\mathtt{let}\ x, y = \bot\ \mathtt{in}\ x \sim \bot?$$

# Computational equivalence

**A simple example:**

$$\texttt{let}\ x, y = \bot\ \texttt{in}\ x \sim \bot?$$

They have the same observable behavior.

How can we prove this equivalence?

# Computational equivalence

**A simple example:**

$$\texttt{let } x, y = \bot \texttt{ in } x \sim \bot?$$

They have the same observable behavior.

How can we prove this equivalence?

We have to prove:

$$\texttt{let } x, y = \bot \texttt{ in } x \preccurlyeq \bot$$

$$\bot \preccurlyeq \texttt{let } x, y = \bot \texttt{ in } x$$

# Computational equivalence

$\bot \preccurlyeq \texttt{let}\ x, y = \bot\ \texttt{in}\ x$ is trivial.

How about:

$$\texttt{let}\ x, y = \bot\ \texttt{in}\ x \preccurlyeq \bot$$

By definition of $\preccurlyeq$ we can assume:

$$\texttt{halts}(\texttt{let}\ x, y = \bot\ \texttt{in}\ x)$$

**We added a rule that says**:

$$\text{if } \texttt{halts}(\texttt{let}\ x, y = t\ \texttt{in}\ F) \text{ then } t \sim \langle \pi_1(t), \pi_2(t) \rangle$$

(And similarly for all destructors.)

# Computational equivalence

➲ **We added rules to reason about the computation system**

# Computational equivalence

## A more complicated example:

$$\forall t:\texttt{Top}.\ \texttt{map}(f,\texttt{map}(g,t)) \sim \texttt{map}(f \circ g, t)?$$

$$\begin{aligned}&\texttt{map}(f,t)\\ &= \texttt{fix}\left(\lambda R.\lambda t.\texttt{ispair}\left(\begin{array}{l} t,\\ \texttt{let}\ x,y = t\ \texttt{in}\ (f\ x) \bullet R\ y,\\ \texttt{isaxiom}(t,\texttt{nil},\bot)\end{array}\right)\right)\ t\end{aligned}$$

a list: $\langle 1, \langle 2, \langle 3, \texttt{Ax}\rangle\rangle\rangle$

# Computational equivalence

➲ **We added the following least upper bound property [Cra98]**

$$\begin{array}{l} H \vdash G[\texttt{fix}(f)/x] \preccurlyeq t \\ \quad \text{BY } \texttt{[least-upper-bound]} \\ \quad H, n : \mathbb{N} \vdash G[f^n(\bot)/x] \preccurlyeq t \end{array}$$

We prove

$$\texttt{map}(f \circ g, t) \preccurlyeq \texttt{map}(f, \texttt{map}(g, t))$$

using `[least-upper-bound]` and then by induction on $n$.

# Computational equivalence

In the induction case, we end up with:

$$\mathtt{ispair}\begin{pmatrix} t, \\ \mathtt{let}\ x, y = t\ \mathtt{in}\ (f\ \ x) \bullet R\ \ y, \\ \mathtt{isaxiom}(t, \mathtt{nil}, \bot) \end{pmatrix} \preccurlyeq X$$

➲ **We added the following rule:**

$$\begin{array}{l} H \vdash C\ \lfloor_{\mathbf{ext}}\ \mathtt{ispair}(t, a, b)[x \backslash \mathtt{Ax}] \rfloor \\ \quad \mathrm{BY}\ [\mathtt{ispairCases}] \\ \quad H \vdash \mathtt{halts}(t) \\ \quad H \vdash t \in \mathtt{Base} \\ \quad H, x : t \sim \langle \pi_1(t), \pi_2(t) \rangle \vdash C\ \lfloor_{\mathbf{ext}}\ a \rfloor \\ \quad H, x : (\forall [u, v : \mathtt{Base}].\ \mathtt{ispair}(z, u, v) \sim v)[z \backslash t] \vdash C\ \lfloor_{\mathbf{ext}}\ b \rfloor \end{array}$$

# Computational equivalence

Process type:

$$\texttt{corec}(\lambda P.A \rightarrow P \times \texttt{Bag}(B))$$

where

$$\texttt{corec}(G) = \cap n{:}\mathbb{N}.\texttt{fix}\left(\begin{array}{r}\lambda P.\lambda n.\texttt{if}\ n =_{\mathbb{Z}} 0\ \texttt{then Top} \\ \texttt{else}\ G\ (P\ (n-1))\end{array}\right)\ n$$

$$P = \texttt{buffer}((\lambda n.\lambda buf.\{n + buf\})\ \texttt{o}\ \texttt{base}(\lambda m.\{m\}), \{0\})$$

$$\Downarrow$$

$$P' = \texttt{fix}(\lambda F.\lambda s.\lambda m.\texttt{let}\ x ::= m + s\ \texttt{in}\ \langle F\ x, \{x\}\rangle)\ 0$$

# Computational equivalence

- $P$ vs. $P'$:
  - 100/200 computation steps for $P$
  - less than 10 computation steps for $P'$

- ShadowDB:
  - non-optimized code: 127 milliseconds
  - optimized code: 60 milliseconds
  - Lisp code: 5 milliseconds
  - reference implementation: 1 millisecond

# Current and future work

- Performance
  - Identify more optimizations.
  - Prove that our optimizations improve the runtime.
  - Translators and compiler?
- ShadowDB
  - Deal with Byzantine faults.
  - Synthesize more of ShadowDB.
- Nuprl
  - Prove that our new types and rules are valid.

# References I


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/.

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

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

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.

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

Douglas J. Howe.
Proving congruence of bisimulation in functional programming languages.
*Inf. Comput.*, 124(2):103–112, 1996.

# References II


Christoph Kreitz.
*The Nuprl Proof Development System, Version 5, Reference Manual and User's Guide.*
Cornell University, Ithaca, NY, 2002.
www.nuprl.org/html/02cucs-NuprlManual.pdf.

Paul F. Mendler.
*Inductive Definition in Type Theory.*
PhD thesis, Cornell University, Ithaca, NY, 1988.

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.