
Vincent Rahli, David Guaspari, Mark Bickford and Robert L. Constable
http://www.nuprl.org
Distributed Systems are Ubiquitous
Correctness

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

What evidence do we have that these systems are correct?

- Type checking
- Testing
Correctness

What evidence do we have that these systems are correct?

- Type checking
- Testing
- Model checking
Correctness

What evidence do we have that these systems are correct?

Type checking

Testing

Model checking

Theorem proving
New Challenges

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.
Contributions

We use Nuprl as a specification, programming and verification language for asynchronous distributed systems.

Programming interface:
- a constructive specification language called **EventML**

Verification **methodology**
Nuprl?

Similar to Coq and Agda

Extensional Intuitionistic Type Theory for partial functions

Consistency proof in Coq

Cloud based & virtual machines: http://www.nuprl.org

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

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

EventML

specification
EventML compiler
untrusted GPM code

Nuprl

LOE specification
logical simplifier
ILF
correctness properties
automated proof
satisfiability proof
correctness proof
hand proof

Runtime

SML interpreter
Ocaml interpreter
Lisp translator

manual
informal high-level specification

Our Methodology

EventML combinator

generate

Process combinator

implements

Logic of Events combinator
Event Orderings (or Message Sequence Diagrams)
Event Orderings

A dependent record

EO = \{ loc : Event \rightarrow Loc (e.g., N),
      info : Event \rightarrow Info (e.g., input message),
      pred : Event \rightarrow Event \leq P \}

plus some axioms

E.g., \leq \text{ is well-founded}

Vincent Rahli
EventML
October 7, 2015
Processes and Observers

Process (GPM)

\[
\text{corec}(\lambda P. (A \rightarrow P \times \text{Bag}(B)) + \text{Unit})
\]

(Programmable) Observer (LoE)

\[
eo: EO \rightarrow e: \text{Event}(eo) \rightarrow \text{Bag}(B)
\]
Observers

parallel

buffer

Vincent Rahli  
EventML  
October 7, 2015
Observers

return

Delegation/bind

A

B

B(x)

B(x2)

B(x1)

x

b1 U ...
Observers in EventML

(* -------------- Quorum: a state machine -------------- *)

(* --- filter --- *)
let new_vote (n,r) (((n',r'),cmd),sender) (cmds,locs) =
(n,r) = (n',r') & !(deq-member (op =) sender locs);

(* --- update --- *)
let upd_quorum (n,r) loc ((nr,c),sndr) (cmds,locs) =
if new_vote (n,r) ((nr,c),sndr) (cmds,locs)
then (c.cmds, sndr.locs)
else (cmds,locs);;

(* --- output --- *)
let roundout loc (((n,r),cmd),sender) (cmds,locs) =
if length cmds = 2 * F
then let (k,cmd') = poss-maj cmdeq (cmd.cmds) cmd in
  if k = 2 * F + 1 then decided'bcast reps(n, cmd')
  else {}
else {}

let when_quorum (n,r) loc vt state =
if new_vote (n,r) vt state then roundout loc vt state else {} ;;

(* --- state machine --- *)
observer QuorumState (n,r) =
  Memory(loc.([]),[], upd_quorum (n,r), vote'base) ;;
observer Quorum (n,r) =
  (when_quorum (n,r)) o (vote'base, QuorumState (n,r)) ;;
Observer Relation

\[ v \in (X \text{ eo } e) \quad \text{written as} \quad v \in X(e) \]

\[ v \in X \upharpoonright\downarrow Y(e) \iff \downarrow (v \in X(e) \lor v \in Y(e)) \]

\[ v \in X \gg= Y(e) \iff \\
\downarrow \exists e' : \{e' : E \mid e' \leq_{\text{loc}} e\}.
\exists u : A.
\quad u \in X(e') \land v \in (Y u \text{ eo}\cdot e' e) \]
Automated Verification

We use causal induction + inductive logical forms (ILFs) + state machine invariants + our brain
import no_repeats length

invariant quorum_inv on (cmds,locs) in (QuorumState ni) == no_repeats :: Loc locs \ length(cmds) = length(locs);;

import fseg
ordering quorum_fseg on (cmds1,locs1) then (cmds2,locs2) in QuorumState ni == fseg :: Cmd cmds1 cmds2 \ fseg :: Loc locs1 locs2 ;;

progress rounds_strict_inc on round1 then round2 in (NewRoundsState n) with ((n',round'),cmd) in RoundInfo and round \= n' = n \ round < round' == round1 < round2 ;;

memory rounds_mem on round1 then round2 in (NewRoundsState n) with ((n',round'),cmd) in RoundInfo == (n = n') \= round' \= round2 ;;

\[
\begin{align*}
&\langle d, i, \text{make-Msg(''vote'';<\langle n, r \rangle, c\rangle, sender\rangle) > \in \text{main}(Cmd;clients;cmdeq;F;reps;f)(e) \\
\iff [\text{loc}(e) \downarrow \in \text{reps} \land i \downarrow \in \text{reps} \land (d = 0) \\
\land (\exists n':Z. \exists c':Cmd. \exists e':E | e' \leq \text{loc} e ).
&((\langle\text{header}(e') = 'propose'\rangle \land \langle n', c'\rangle = \text{body}(e')) \\
\lor (\text{has-es-info-type}(es,e';f;Z \times Z \times Cmd \times Id) \\
\land (\text{header}(e') = 'vote') \\
\land (n' = (\text{fst}(\text{fst}(\text{msgval}(e'))))) \\
\land (c' = (\text{snd}(\text{msgval}(e'))))) \\
\land ((\text{fst}(\text{ReplicaStateFun}(Cmd;f;es;e'))) < n') \\
\land (n' \notin \text{snd}(\text{ReplicaStateFun}(Cmd;f;es;e'))) \\
\land (\text{no Notify}(Cmd;clients;f) n' \text{ between } e' \text{ and } e) \\
\land ((\langle\langle n, r \rangle, c\rangle, sender\rangle = \langle\langle n', 0 \rangle, c'\rangle, \text{loc}(e)) \land (e = e')) \\
\lor (\exists r':Z. \exists c'':Cmd. ((\langle\langle n, r \rangle, c\rangle, sender\rangle = \langle\langle n', r'\rangle, c''\rangle, \text{loc}(e)) \\
\land (\exists e1:{e1:E | e1 \leq \text{loc} e } \\
&((\langle\text{header}(e1) = 'retry'\rangle \land \langle n', r', c''\rangle = \text{body}(e1)) \\
\lor (\text{has-es-info-type}(es,e';e1;f;Z \times Z \times Cmd \times Id) \\
\land (\text{header}(e1) = 'vote') \\
\land (n' = (\text{fst}(\text{fst}(\text{msgval}(e1)))))) \\
\land (r' = (\text{snd}(\text{fst}(\text{msgval}(e1)))))) \\
\land (c'' = (\text{snd}(\text{msgval}(e1)))))) \\
\land (\text{NewRoundsStateFun}(Cmd;f;n';es.e';e1) < r') \land (e = e1)))))))))))
\end{align*}
\]
What next

Crash-tolerant → Byzantine fault-tolerant → Nysiad → probabilistic systems

EventML

- specification
- EventML compiler
- LOE specification
- untrusted GPM code
- optimizer
- SML interpreter
- Ocaml interpreter
- Lisp translator

Nuprl

- logical simplifier
- automated proof
- satisfiability proof
- correctness proof
- ILF
- correctness properties
- manual proof
- manual

Runtime

Scala interface? Complexity