Workshop in Verification of Distributed Protocols

Mooly Sagiv, Oded Padon
08-March-2018

http://www.cs.tau.ac.il/~odedp/workshop18/
http://microsoft.github.io/ivy/
Administration

• Start-off meeting (today)
• Project teams:
  • 2-3 students
  • Each team will take different a project, and work independently during the semester
  • Meet with Oded / Mooly as needed
• If needed, we’ll have more workshop meeting during the semester
• 14/6 – project presentation meeting
  • Each team will present project
  • Project must be finished and approved by Oded / Mooly before
Possible Projects

• Use Ivy to verify any distributed / shared memory algorithm

• Paxos variants
  • Disk Paxos, Generalised Paxos, EPaxos (see http://paxos.systems/variants.html for ideas)
  • Prove reconfiguration / failure recovery / log truncation / liveness

• Mutual Exclusion Algorithms
  • Knuth’s Algorithm, Lamport’s Bakery, Patterson, ...
  • Prove safety and liveness

• Blockchain algorithms
  • Algorand, HoneyBadgerBFT, Bitcoin-NG, ...

• Improve Ivy
  • Experiment with other SMT solvers (e.g. iProver, CVC4, Vampire, SPASS)
Why verify distributed protocols?

- Distributed systems are everywhere
  - Safety-critical systems
  - Cloud infrastructure
  - Blockchain
- Distributed systems are notoriously hard to get right
  - Even small protocols can be tricky
  - Bugs occur on rare scenarios
  - Testing is costly and not sufficient
Why verify distributed protocols?

• Distributed systems are everywhere
  • Safety-critical systems
  • Cloud infrastructure
  • Blockchain

• Distributed systems are notoriously hard to get right

SIGCOMM’01

Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications

Jon Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Member, IEEE

Attractive features of Chord include its simplicity, provable correctness, and provable performance even in the face of concurrent node arrivals and departures. It continues to func-
Why verify distributed protocols?

- Distributed systems are everywhere
  - Safety-critical systems
  - Cloud infrastructure
  - Blockchain

- Distributed systems are notoriously hard to get right

SIGCOMM’01
Chord: A Scalable Peer-to-Peer System for Internet Applications
Jon Stoica, Robert Morris, David Libes, David R. Karp

CCR’12
Using Lightweight Modeling To Understand Chord
Pamela Zave
AT&T Laboratories—Research
Florham Park, New Jersey USA
pamela@research.att.com

Attractive features of Chord include correctness, concurrency and provable performance, but concurrent node arrivals and departures can lead to unpredictable behavior.

Under the same assumptions made in the Chord papers, the [SIGCOMM] version of the protocol is not correct, and not one of the properties claimed invariant in [PODC] is actually invariantly true of it. The [PODC] version satisfies one invariant, but is still not correct.
Zyzyva: Speculative Byzantine Fault Tolerance

Ramakrishna Kolla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong

ACM Transactions on Computer Systems '09

Zyzyva: Speculative Byzantine Fault Tolerance

RAMAKRISHNA KOTLA
Microsoft Research, Silicon Valley

LORENZO ALVISI, MIKE DAHLIN, ALLEN CLEMENT, and EDMUND WONG
The University of Texas at Austin

Best Paper Award

Zyzyva: Speculative Byzantine Fault Tolerance

Ramakrishna Kolla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong

Dept. of Computer Sciences
University of Texas at Austin

Zyzyva is a state machine replication protocol based on
three simple protocols: (1) agreement, (2) view change, and (3)
leave. Agreement protocol orders requests for execution.
View change protocol coordinates failures.


Revisiting Fast Practical Byzantine Fault Tolerance

Ittai Abraham, Guy Gura, Dahlia Malkhi
VMware Research

with:
Lorenzo Alvisi (Cornell),
Rama Kotla (Amazon),
Jean-Philippe Martin (Verily)

We now proceed to demonstrate that the view-change
timeout mechanism in Zyzyva does not guarantee safety.
Proving distributed systems is hard

• Amazon [CACM’15] uses TLA+ for testing protocols, but no proofs
• IronFleet [SOSP’15] – verification of Multi-Paxos in Dafny (3.7 person-years)
• Verdi [PLDI’15] – verification of Raft in Coq (50,000 lines of proofs)

Our goal: reduce human effort while maintaining flexibility

Our approach: decompose verification into decidable problems

Automatic verification of infinite-state systems

Verification
Is there a behavior of $S$ that violates $\varphi$?

System $S$

Property $\varphi$

Counterexample

Proof
Automatic verification of infinite-state systems

**Verification**

Is there a behavior of $S$ that violates $\varphi$?

System $S$  \quad Property $\varphi$

- **Counterexample**
- **Unknown**
- **Proof**

Rice’s Theorem ➔ I can’t decide!

Automatic verification of infinite-state systems
Automatic verification of infinite-state systems

Is there a behavior of $S$ that violates $\varphi$?

Counterexample

Proof

Rice’s Theorem

“I can’t decide!”

Chet Murthy

“Program verification is the holy grail of computer science; always was; always will be”
Inductive invariants

System S is **safe** if all the reachable states satisfy the property $P = \neg \text{Bad}$.
System S is safe if all the reachable states satisfy the property $P = \neg \text{Bad}$.

System S is safe iff there exists an inductive invariant $\text{Inv}$:

\[
\text{Inv} \cap \text{Bad} = \emptyset \quad \text{(Safety)}
\]

\[
\text{Init} \subseteq \text{Inv} \quad \text{(Initiation)}
\]

\[
\text{if } \sigma \in \text{Inv} \text{ and } \sigma \rightarrow \sigma' \text{ then } \sigma' \in \text{Inv} \quad \text{(Consecution)}
\]
Counterexample To Induction (CTI)

• States $\sigma, \sigma'$ are a CTI of Inv if:
  • $\sigma \in \text{Inv}$
  • $\sigma' \notin \text{Inv}$
  • $\sigma \rightarrow \sigma'$

• A CTI may indicate:
  • A bug in the system
  • A bug in the safety property
  • A bug in the inductive invariant
    • Too weak
    • Too strong
Strengthening & weakening from CTI
Simple example: loop invariants

x := 1;
y := 2;
while * do {
  assert ¬even[x];
  x := x + y;
  y := y + 2;
}

Even values of $x$:
- $x = 1, y = 2$
- $x = 2, y = 4$
- $x = 3, y = 2$
- $x = 4, y = 5$
- $x = 5, y = 4$

Odd values of $x$:
- $x = 1, y = 0$
- $x = 2, y = 3$
- $x = 3, y = 1$
- $x = 4, y = 5$
- $x = 2, y = 4$
- $x = 2, y = 3$

$x$ and $y$ at each loop iteration:
- Iteration 1: $x = 1, y = 2$
- Iteration 2: $x = 3, y = 4$
- Iteration 3: $x = 5, y = 4$
- Iteration 4: $x = 7, y = 6$
- Iteration 5: $x = 3, y = 2$
- Iteration 6: $x = 7, y = 6$
- Iteration 7: $x = 3, y = 2$
- Iteration 8: $x = 7, y = 6$
- Iteration 9: $x = 3, y = 2$
- Iteration 10: $x = 7, y = 6$
Simple example: loop invariants

```plaintext
x := 1;
y := 2;
while * do {
    assert ¬even[x];
    x := x + y;
y := y + 2;
}
```

Counterexample to induction (CTI)
Simple example: loop invariants

```plaintext
x := 1;
y := 2;
while * do {
  assert ¬even[x];
  x := x + y;
  y := y + 2;
}
```

**Inv = ¬even[x] ∧ even[y]**
Simple example: loop invariants

\[
\text{Inv} = \neg \text{even}[x] \land \text{even}[y]
\]

\[
x := 1;
y := 2;
\text{while} \; * \; \text{do} \{
\text{assert} \; \neg \text{even}[x];
x := \frac{x \times x - y \times y}{x - y};
y := y + 2;
\}
\]
Challenges in Deductive Verification

1. **Formal specification:** formalizing infinite-state systems
   - Modeling the system and property (TR, Init, Bad)

2. **Deduction:** checking inductiveness
   - Undecidability of implication checking
     - Unbounded state (threads, messages), arithmetic, quantifier alternation

3. **Inference:** inferring inductive invariants (Inv)
   - Hard to specify
   - Hard to infer
     - Undecidable even when deduction is decidable
Ultimately limited by human

“the proofs consisted of about 5000 lines and assumed several nontrivial invariants of the Raft protocol. This paper discusses the verification of Raft as a whole, including all the invariants from the original Raft paper [32]. These new proofs consist of about 45000 additional lines” [Verdi, CPP’16]
State of the art in formal verification

“but our input language cannot compete in generality with mechanized proof methods that rely heavily on human expertise, e.g., IVY [55], Verdi [68], IronFleet [38], TLAPS [16]” [Konnov et al, POPL’17]
IVy’s Principles

• Specify systems and properties in decidable fragment of first-order logic (EPR)
  • Allows quantifiers to reason about unbounded sets
  • Decidable to check inductiveness
  • Finite counterexamples to induction, display graphically
  • Logic is mostly hidden

• Interact with the user to find inductive invariants

• Challenge: use restricted logic to verify interesting systems
  • Paxos, Reconfiguration, Byzantine Fault Tolerance
  • Liveness and Temporal Properties
Example: Leader Election in a Ring

- Nodes are organized in a ring
- Each node has a unique numeric id
- Protocol:
  - Each node sends its id to the next
  - A node that receives a message passes it (to the next) if the id in the message is higher than the node’s own id
  - A node that receives its own id becomes the leader
- Theorem:
  - The protocol selects at most one leader

Example: Leader Election in a Ring

• Nodes are organized in a ring
• Each node has a unique numeric id

Protocol:
• Each node sends its id to the next
• A node that receives a message passes it (to the next) if the id in the message is higher than the node’s own id
• A node that receives its own id becomes the leader

Theorem:
• The protocol selects at most one leader

Proposition: This algorithm detects one and only one highest number.

Argument: By the circular nature of the configuration and the consistent direction of messages, any message must meet all other processes before it comes back to its initiator. Only one message, that with the highest number, will not encounter a higher number on its way around. Thus, the only process getting its own message back is the one with the highest number.

Leader Election Protocol (IVy)

- \(\leq\) (ID, ID) – total order on node id's
- btw (Node, Node, Node) – the ring topology
- id: Node \(\to\) ID – relate a node to its unique id
- pending(ID, Node) – pending messages
- leader(Node) – leader(n) means n is the leader

Axiomatized in first-order logic
Leader Election Protocol (IVy)

- \( \preceq (\text{ID}, \text{ID}) \) – total order on node id’s
- btw (Node, Node, Node) – the ring topology
- id: Node \( \rightarrow \) ID – relate a node to its unique id
- pending(ID, Node) – pending messages
- leader(Node) – leader(n) means n is the leader

\[
\exists n,s : \text{Node}. \; "s := \text{next}(n)" \land \forall x:\text{ID}, y: \text{Node}. \; \text{pending}'(x,y) \leftrightarrow (\text{pending}(x,y) \lor (x = \text{id}(n) \land y = s))
\]

protocol = (send | receive)*

assert I\( \emptyset \) = \( \forall x,y : \text{Node}. \; \text{leader}(x) \land \text{leader}(y) \rightarrow x = y \)
Specify and verify the protocol for any number of nodes in the ring.
Inductive Invariant for Leader Election

- $\leq (\text{ID}, \text{ID})$ – total order on node id’s
- $\text{btw} \ (\text{Node}, \text{Node}, \text{Node})$ – the ring topology
- $\text{id}: \text{Node} \rightarrow \text{ID}$ – relate a node to its id
- $\text{pending}(\text{ID}, \text{Node})$ – pending messages
- $\text{leader}(\text{Node})$ – leader(n) means n is the leader

**Safety property:** $I_0$

$I_0 = \forall x, y: \text{Node}. \ \text{leader}(x) \land \text{leader}(y) \Rightarrow x = y$

**Inductive invariant:** $\text{Inv} = I_0 \land I_1 \land I_2 \land I_3$

$I_1 = \forall n_1, n_2: \text{Node}. \ \text{leader}(n_2) \Rightarrow \text{id}[n_1] \leq \text{id}[n_2]$

$I_2 = \forall n_1, n_2: \text{Node}. \ \text{pending}(\text{id}[n_2], n_2) \Rightarrow \text{id}[n_1] \leq \text{id}[n_2]$

$I_3 = \forall n_1, n_2, n_3: \text{Node}. \ \text{btw}(n_1, n_2, n_3) \land \text{pending}(\text{id}[n_2], n_1) \Rightarrow \text{id}[n_3] \leq \text{id}[n_2]$

How can we find an inductive invariant without knowing it?

The leader has the highest ID

Only the leader can be self-pending

Cannot bypass higher nodes
Invariant Inference in IVy

- Model
- Candidate Inductive Invariant

Inductive Invariant Found

Inductive?

Yes

No

Find "minimal" CTI

Generalize from CTI

Modify candidate invariant

User

Automation

User Automation

Inductive Invariant Found
**IVy: Check Inductiveness**

**Leader Protocol**

\[ \text{Inv} = I_0 \land I_1 \land I_2 \]

**Check Inductiveness**

- \( \leq \)
- \( \leftarrow \)
- \( \rightarrow \)
- \( \land \)
- \( \lor \)
- \( \neg \)

- \( rcv(1, id(2)) \)

- \( \text{id} \)
- \( \text{next} \)
- \( \text{pnd} \)
IVy: Generalize from CTI

Cannot bypass nodes with higher ids

This looks good, add to the invariant as $I_3$

$I_3 = \neg \exists n_1, n_2, n_3 : \text{Node.} \text{ btw}(n_1, n_2, n_3) \land \text{pnd}(\text{id}[n_2], n_1) \land \text{id}[n_2] \leq \text{id}[n_3]$
\( \text{IVy: Check Inductiveness} \)

Leader Protocol \( \iff \text{Inv} = I_0 \land I_1 \land I_2 \land I_3 \)

Bad = \( \neg I_0 \)

**VC Generator**

\( \text{Init} \land \neg \text{Inv} \)
\( \text{Inv}(V) \land \text{TR}(V,V') \land \neg \text{Inv}(V') \)
\( \text{Inv}(V) \land \text{Bad}(V) \)

**EPR Solver**

\( I_0 \land I_1 \land I_2 \land I_3 \) is an inductive invariant for the leader protocol, which proves the protocol is safe
\[ \text{Init} \subseteq \text{Inv} \text{ (Initiation)} \]

if \( \sigma \in \text{Inv} \) and \( \sigma \rightarrow \sigma' \) then \( \sigma' \in \text{Inv} \) \text{ (Consecution)}

\[ \text{Inv} \cap \text{Bad} = \emptyset \text{ (Safety)} \]
Leader Election Protocol (axioms)

- \( \preceq (ID, ID) \) – total order on node id’s
- \( \text{btw} (a: \text{Node}, b: \text{Node}, c: \text{Node}) \) – the ring topology
- \( \text{id}: \text{Node} \rightarrow \text{ID} \) – relate a node to its unique id
- \( \text{pending}(\text{ID}, \text{Node}) \) – pending messages
- \( \text{leader}(\text{Node}) \) – leader(n) means n is the leader

<table>
<thead>
<tr>
<th>Natural Interpretation</th>
<th>EPR Modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Node ID’s</strong></td>
<td>Integers</td>
</tr>
<tr>
<td>( \forall i: \text{ID}. \ i \preceq i ) Reflexive</td>
<td></td>
</tr>
<tr>
<td>( \forall i, j, k: \text{ID}. \ i \preceq j \land j \preceq k \Rightarrow i \preceq k ) Transitive</td>
<td></td>
</tr>
<tr>
<td>( \forall i, j: \text{ID}. \ i \preceq j \land j \preceq i \Rightarrow i = j ) Anti-Symmetric</td>
<td></td>
</tr>
<tr>
<td>( \forall i, j: \text{ID}. \ i \preceq j \lor j \preceq i ) Total</td>
<td></td>
</tr>
<tr>
<td>( \forall x, y: \text{Node}. \ \text{id}(x) = \text{id}(y) \Rightarrow x = y ) Injective</td>
<td></td>
</tr>
<tr>
<td><strong>Ring Topology</strong></td>
<td>Next edges + Transitive closure</td>
</tr>
<tr>
<td>( \forall x, y, z: \text{Node}. \ \text{btw}(x, y, z) \Rightarrow \text{btw}(y, z, x) ) Circular shifts</td>
<td></td>
</tr>
<tr>
<td>( \forall x, y, z, w: \text{Node}. \ \text{btw}(w, x, y) \land \text{btw}(w, y, z) \Rightarrow \text{btw}(w, x, z) ) Transitive</td>
<td></td>
</tr>
<tr>
<td>( \forall x, y, w: \text{Node}. \ \text{btw}(w, x, y) \Rightarrow \neg \text{btw}(w, y, x) ) A-Symmetric</td>
<td></td>
</tr>
<tr>
<td>( \forall x, y, z, w: \text{Node}. \ \text{distinct}(x, y, z) \Rightarrow \text{btw}(w, x, y) \lor \text{btw}(w, y, x) )</td>
<td></td>
</tr>
<tr>
<td>“next(a)=b” ( \equiv \ \forall x: \text{Node}. X = a \lor X = b \lor \text{btw}(a,b,x) )</td>
<td></td>
</tr>
</tbody>
</table>
Challenge: How to use restricted first-order logic to verify interesting systems?

- Expressing transitive closure
  - Linked lists
  - Ring protocols
- Expressing Consensus
  - Paxos, Multi-Paxos
  - Reconfiguration
  - Byzantine Fault Tolerance
- Liveness and temporal Properties
Key idea: representing deterministic paths
[Itzhaky SIGPLAN Dissertation Award 2016]

Alternative 1: maintain $n$
- $n^*$ defined by transitive closure of $n$
- not definable in first-order logic

Alternative 2: maintain $n^*$
- $n$ defined by transitive reduction of $n^*$
- Unique due to outdegree $\leq 1$
- Definable in first order logic (for roots)
  - $n^+(a, b) \equiv n^*(a, b) \land a \neq b$
  - $n(a, b) \equiv n^+(a, b) \land \forall z: n^+(a, z) \rightarrow n^*(b, z)$
Paxos made EPR

Methodology for decidable verification of infinite-state systems

Modeling

1

Formal specification
in first-order logic

Abstraction
Domain knowledge

Transforming

2

Formal specification with decidable VC

Z3
Paxos

- **Single decree Paxos – consensus**
  lets nodes make a common decision despite node crashes and packet loss

- **Paxos family of protocols – state machine replication**
  variants for different tradeoffs, e.g., Fast Paxos is optimized for low contention, Vertical Paxos is reconfigurable, etc.

- **Pervasive approach to fault-tolerant distributed computing**
  - Google Chubby
  - VMware NSX
  - AWS
  - Many more...
Challenge: reasoning about Paxos in FOL

- Consensus algorithms use set cardinalities
  - Wait for messages from more than \( N/2 \) nodes
- Insight: set cardinalities are used to get a simple effect
  Can be modeled in first-order logic!
- Solution: axiomatize quorums in first-order logic

**Sort** `quorum`  
**Relation** `member` (node, quorum)
- set membership (2\(^{nd}\)-order logic in first-order)

**Axiom**
\[
\forall q_1, q_2: \text{quorum}. \exists n: \text{node. } \text{member}(n, q_1) \land \text{member}(n, q_2)
\]

```plaintext
action propose(r:round) {
    requires ">N/2 join_msg's"
    ...
}
```

```plaintext
action propose(r:round) {
    requires \( \exists q. \forall n. \text{member}(n, q) \rightarrow \exists r', v'. \text{join_msg}(n, r, r', v') \)
    ...
}
```
<table>
<thead>
<tr>
<th>Concept</th>
<th>Intention</th>
<th>First-order abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quorums</td>
<td>Majority sets</td>
<td>relation member (node, quorum)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>axiom ( \forall q_1, q_2. \exists n: \text{node}. \text{member}(n, q_1) \land \text{member}(n, q_2) )</td>
</tr>
<tr>
<td>Rounds</td>
<td>Natural numbers</td>
<td>relation ( \leq ) (round, round)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>axiom ( \forall x: \text{round}. \ x \leq x ) reflexive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>axiom ( \forall x, y, z: \text{round}. x \leq y \land y \leq z \rightarrow x \leq z ) transitive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>axiom ( \forall x, y: \text{round}. x \leq y \land y \leq x \rightarrow x=y ) anti-symmetric</td>
</tr>
<tr>
<td></td>
<td></td>
<td>axiom ( \forall x, y: \text{round}. x \leq y \lor y \leq x ) total</td>
</tr>
<tr>
<td>Messages</td>
<td>Network with: dropping</td>
<td>relation start_msg(round)</td>
</tr>
<tr>
<td></td>
<td>duplication reordering</td>
<td>relation join_msg(node, round, round, value)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>relation propose_msg(round, value)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>relation vote_msg(node, round, value)</td>
</tr>
</tbody>
</table>


∀n₁, n₂ : node, r₁, r₂ : round, v₁, v₂ : value. decision(n₁, r₁, v₁) ∧ decision(n₂, r₂, v₂) → v₁ = v₂

∀r : round, v₁, v₂ : value. propose_msg(r, v₁) ∧ propose_msg(r, v₂) → v₁ = v₂

∀n : node, r : round, v : value. vote_msg(n, r, v) → propose_msg(r, v)

∀r : round, v : value. (∃n : node. decision(n, r, v)) → ∃q : quorum. ∀n : node. member(n, q) → vote_msg(n, r, v)

∀n : node, r, r' : round, v, v' : value. join_ack_msg(n, r, r', v) ∧ r' < r → ¬vote_msg(n, r', v')

∀n : node, r, r' : round, v : value. join_ack_msg(n, r, r', v) ∧ r' ≠ ⊥ → r' < r ∧ vote_msg(n, r', v)

∀n : node, r, r'', r''' : round, v, v' : value. join_ack_msg(n, r, r'', v) ∧ r'' ≠ ⊥ ∧ r'' < r'' → ¬vote_msg(n, r'', v')

∀n : node, v : value. ¬vote_msg(n, ⊥, v)

∀r₁, r₂ : round, v₁, v₂ : value, q : quorum. propose_msg(r₂, v₂) ∧ r₁ < r₂ ∧ ∃n : node. member(n, q) ∧ r₁ > r₁ ∧ join_ack_msg(n, r₁, v₁)

∀n : node, r', r'' : round, v : value. member(n, q) ∧ ¬vote_msg(n, r₁, v₁) ∧ r'' > r₁ ∧ join_ack_msg(n, r', r'', v)
Step 2: Obtaining decidable VC’s

Challenge: quantifier alternation cycles

• Axiom
  \[ \forall q_1,q_2: \text{quorum}. \exists n: \text{node}. \text{member}(n, q_1) \land \text{member}(n, q_2) \]

• Propose action precondition
  \[ \exists q: \text{quorum}. \forall n: \text{node}. \text{member}(n, q) \rightarrow \exists r': \text{round}, v': \text{value}. \text{join_msg}(n, r, r', v') \]

• Inductive invariant
  \[ \forall r: \text{round}, v: \text{value}. \text{decision}(r, v) \rightarrow \exists q: \text{quorum}. \forall n: \text{node}. \text{member}(n, q) \rightarrow \text{vote_msg}(n, r, v) \]
Solution: derived relations and rewrites

\[ \exists q: \text{quorum. } \forall n: \text{node. } \text{member}(n,q) \rightarrow \exists r': \text{round}, v': \text{value. } \text{join}_\text{msg}(n,r',r',v') \]
Solution: derived relations and rewrites

\[ \forall q: \text{quorum}. \exists n: \text{node}. \text{member}(n,q) \rightarrow \exists r': \text{round}, v': \text{value}. \text{join}_{-}\text{msg}(n,r,r',v') \]

new relation: \( \text{joined}(n: \text{node}, r: \text{round}) \equiv \exists r': \text{round}, v': \text{value}. \text{join}_{-}\text{msg}(n,r,r',v') \)

rewrite

update code:

```plaintext
action join(n:node, r:round) {
    requires start_round_msg(r)
    let maxr, v := ... 
    join_{-}msg(n,r,maxr,v) := true 
    joined(n,r) := true
}
```

\[ \forall q: \text{quorum}. \forall n: \text{node}. \text{member}(n,q) \rightarrow \text{joined}(n,r) \]
Solution: derived relations and rewrites

\[
\begin{align*}
\text{joined}(n:node, r:round) & \equiv \exists r':\text{round}, v':\text{value. join_msg}(n, r, r', v') \\
\text{left}(n:node, r:round) & \equiv \exists r', r'': \text{round}, v':\text{value. join_msg}(n, r', r'', v') \land r' > r
\end{align*}
\]

VC’s are decidable!
Principle: decomposing into decidable checks

- User defines:
  - Derived relations
  - Rewrites
  - Inductive invariants
- Decidable checks:

\[
\forall \text{Spec in FOL} \implies \neg \exists \text{Inv}_{\text{aux}}
\]

\[
\neg \exists \text{Inv}_{\text{aux}} \iff \neg \exists \text{Inv}
\]

Formal specification in first-order logic
Formal specification with decidable VC
Inductive Invariant of Paxos

# safety property
\[ \text{conjecture } \text{decision}(N1,R1,V1) \land \text{decision}(N2,R2,V2) \rightarrow V1 = V2 \]

# proposals are unique per round
\[ \text{conjecture } \text{proposal}(R,V1) \land \text{proposal}(R,V2) \rightarrow V1 = V2 \]

# only vote for proposed values
\[ \text{conjecture } \text{vote}(N,R,V) \rightarrow \text{proposal}(R,V) \]

# decisions come from quorums of votes:
\[ \text{conjecture } \forall R,V. (\exists N. \text{decision}(N,R,V)) \rightarrow \exists Q. \forall N. \text{member}(N,Q) \rightarrow \text{vote}(N,R,V) \]

# properties of one_b_max_vote
\[ \text{conjecture } \text{one_b_max_vote}(N,R2,\text{none},V1) \land \neg \text{le}(R2,R1) \rightarrow \neg \text{vote}(N,R1,V2) \]
\[ \text{conjecture } \text{one_b_max_vote}(N,R,RM,V) \land RM \neq \text{none} \rightarrow \neg \text{le}(R,RM) \land \text{vote}(N,RM,V) \]
\[ \text{conjecture } \text{one_b_max_vote}(N,R,RM,V) \land RM \neq \text{none} \land \neg \text{le}(R,RO) \land \neg \text{le}(RO,RM) \rightarrow \neg \text{vote}(N,RO,VO) \]

# property of choosable and proposal
\[ \text{conjecture } \neg \text{le}(R2,R1) \land \text{proposal}(R2,V2) \land V1 \neq V2 \rightarrow \exists N. \text{member}(N,Q) \land \text{left_rnd}(N,R1) \land \neg \text{vote}(N,R1,V1) \]

# property of one_b, left_rnd
\[ \text{conjecture } \text{one_b}(N,R2) \land \neg \text{le}(R2,R1) \rightarrow \text{left_rnd}(N,R1) \]
## Experimental Evaluation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Paxos</td>
<td>85</td>
<td>11</td>
<td>1.0</td>
<td>0.1</td>
</tr>
<tr>
<td>Multi-Paxos</td>
<td>98</td>
<td>12</td>
<td>1.2</td>
<td>0.1</td>
</tr>
<tr>
<td>Vertical Paxos*</td>
<td>123</td>
<td>18</td>
<td>2.2</td>
<td>0.2</td>
</tr>
<tr>
<td>Fast Paxos*</td>
<td>117</td>
<td>17</td>
<td>4.7</td>
<td>1.6</td>
</tr>
<tr>
<td>Flexible Paxos</td>
<td>88</td>
<td>11</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>Stoppable Paxos*</td>
<td>132</td>
<td>16</td>
<td>3.8</td>
<td>0.9</td>
</tr>
</tbody>
</table>

*first mechanized verification
Transformation to EPR reusable across all variants!
have been chosen as the $j^{th}$ command for some $j < i$. Although the basic idea of the algorithm is not complicated, getting the details right was not easy.
(1.7) NoneChoosableAfter\((i, b, v)\)

**Proof:** We assume \(v \in \text{StopCmd}, j > i, c < b\), and \(w\) any command and we prove NotChoosable\((j, c, w)\). By Lemma 1.7, it suffices to prove NotChoosable\((j, c, w)\). We split the proof into two cases.

(2.1) **Case:** \(sval2a(i, b, Q) = \top\)

**Proof:** Assumption (1.1.3) implies \(E4(i, b, Q, v)\), so the assumption \(v \in \text{StopCmd}\) implies \(E4b(i, b, Q, v)\). The case assumption, the assumption \(j > i\), and \(E4b(i, b, Q, v)\) imply \(sval2a(j, b, Q) = \top\). The assumption \(c < b\) and step (1.1.4) then imply NotChoosable\((j, c, w)\).

(2.2) **Case:** \(sval2a(i, b, Q) \neq \top\)

(3.1) \(sval2a(i, b, Q) = val2a(i, b, Q) = v\)

**Proof:** Assumption (1.1.3) implies \(E3(i, b, Q, v)\), which implies \(sval2a(i, b, Q) = v\). The case assumption and the definition of \(sval2a\) then implies \(val2a(i, b, Q) = v\).

(3.2) **Done2a(i, mba2a(i, b, Q), v)\)

**Proof:** (3.1), assumption (1.1.4), and the definition of \(val2a\) imply \(vote_i[a][mba2a(i, b, Q)] = v\) for some acceptor \(a\) in \(Q\), which by Lemma 1.3 implies \(Done2a(i, mba2a(i, b, Q), v)\).

By the assumption \(c < b\), it suffices to consider the following two cases.

(3.3) **Case:** \(c < mba2a(i, b, Q)\)

**Proof:** Step (3.2) and assumption (1.1.1) imply NoneChoosableAfter\((i, \text{mba2a}(i, b, Q), v)\). By the case assumption and the assumptions \(v \in \text{StopCmd}\) and \(j > i\), this implies NotChoosable\((j, c, w)\).

(3.4) **Case:** \(mba2a(i, b, Q) \leq c < b\)

(4.1) \(mba2a(j, b, Q) < mba2a(i, b, Q)\)

**Proof:** The assumption \(v \in \text{StopCmd}\) and (3.1) imply \(sval2a(i, b, Q) \in \text{StopCmd}\). Case assumption (2.2) and the definition of \(sval2a\) then imply \(mba2a(k, b, Q) < mba2a(i, b, Q)\) for all \(k > i\).

(4.2) **NotChoosable\((j, c, w)\)**

**Proof:** (4.1) and case assumption (3.4) imply \(mba2a(j, b, Q) < c < b\). By assumption (1.1.4), Lemma 3 implies NotChoosable\((j, c, w)\). \(\square\)
Verification of Temporal Properties

```haskell
global nat s, n
local nat m
1: while (true) {
    m=n++; // Acquire a ticket
    skip;
}  // Critical section
2: while (m>s) {  // Busy wait
    m=n++;
}  // Critical section
3: s++; // Exit critical
```

<table>
<thead>
<tr>
<th>Liveness Property</th>
<th>( \forall x : \text{thread. } \Box (pc_2(x) \rightarrow \Diamond pc_3(x)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fairness Assumption</td>
<td>( \forall x : \text{thread. } \Box \Diamond \text{scheduled}(x) )</td>
</tr>
<tr>
<td>Temporal Spec. ((\text{spec}))</td>
<td>((\forall x : \text{thread. } \Box \Diamond \text{scheduled}(x)) \rightarrow \forall x : \text{thread. } \Box (pc_2(x) \rightarrow \Diamond pc_3(x)))</td>
</tr>
</tbody>
</table>
Possible Projects

- Verify any distributed / shared memory algorithm
- Paxos variants
  - Disk Paxos, Generalised Paxos, EPaxos (see http://paxos.systems/variants.html for ideas)
  - Prove reconfiguration / failure recovery / log truncation / liveness
- Mutual Exclusion Algorithms
  - Knuth’s Algorithm, Lamport’s Bakery, Patterson, ...
  - Prove safety and liveness
- Blockchain algorithms
  - Algorand, HoneyBadgerBFT, Bitcoin-NG, ...
- Improve Ivy
  - Experiment with other SMT solvers (e.g. iProver, CVC4, Vampire, SPASS)