

mypyvy: A DSL for verifying infinite state systems

James Wilcox



EPR++

- But what can you possibly express in such a restricted logic?
 - Transitive closure over deterministic paths
 - Set cardinalities
 - Avoiding quantifier alternations
 - Encoding liveness and LTL

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
- Active research and extensive industry use

Sets and cardinalities in EPR

- Consensus algorithms use set cardinalities
 - Wait for messages from **more than $N / 2$ nodes**
- Set Cardinalities + Arithmetic + Uninterpreted \gg EPR ?
- **Insight: set cardinalities are used to get a simple effect**
Can be modeled in first-order logic and EPR!
- Solution: axiomatize quorums in first-order logic
sort `Quorum`
relation `member` (Node, Quorum)
 - set membership (2^{nd} -order logic in first-order)**axiom** $\forall q_1, q_2: \text{Quorum}. \exists n: \text{Node}. \text{member}(n, q_1) \wedge \text{member}(n, q_2)$

```
action propose(r:Round) {  
  require ">N/2 join_msg's"  
  ...  
}
```



```
action propose(r:Round) {  
  require  $\exists q. \forall n. \text{member}(n, q) \rightarrow$   
     $\exists r', v'. \text{join\_msg}(n, r, r', v')$   
  ...  
}
```

EPR++

- But what can you possibly express in such a restricted logic?
 - Transitive closure over deterministic paths
 - Set cardinalities
 - Avoiding quantifier alternations
 - Encoding liveness and LTL

Quantifier alternation cycles

- Axiom

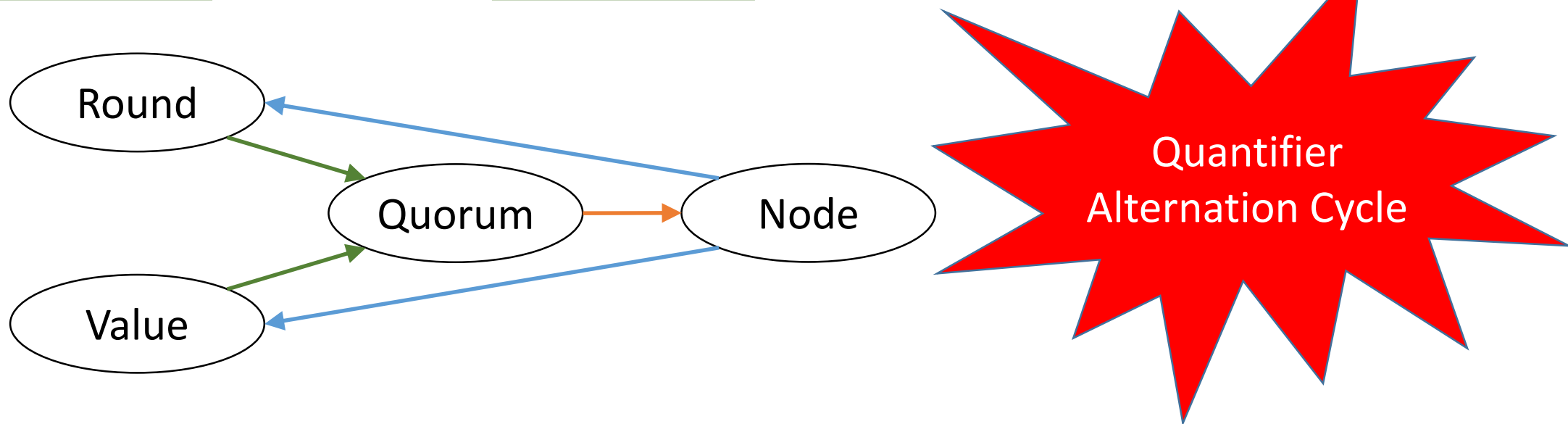
$\forall q_1, q_2: \text{Quorum}. \exists n: \text{Node}. \text{member}(n, q_1) \wedge \text{member}(n, q_2)$

- 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

$\forall r: \text{Round}, v: \text{Value}. \text{decision}(r, v) \rightarrow \exists q: \text{Quorum}. \forall n: \text{Node}. \text{member}(n, q) \rightarrow \text{vote_msg}(n, r, v)$

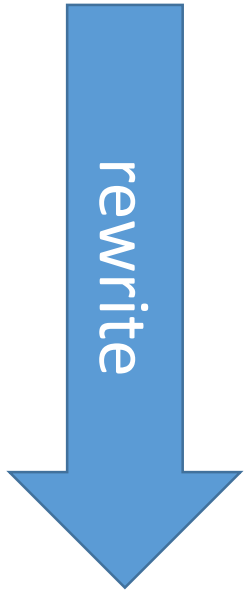


Derived relations

$\exists q:\text{quorum}. \forall n:\text{node}. \text{member}(n,q) \rightarrow \exists r':\text{round}, v':\text{value}. \text{join_msg}(n,r,r',v')$

Derived relations

$\exists q:\text{quorum}. \forall n:\text{node}. \text{member}(n,q) \rightarrow \exists r':\text{round}, v':\text{value}. \text{join_msg}(n,r,r',v')$



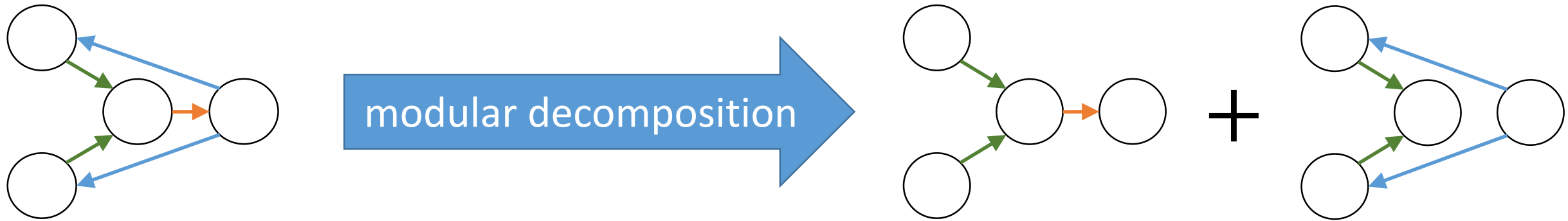
new relation: $\text{joined}(n:\text{node}, r:\text{round}) \equiv \exists r':\text{round}, v':\text{value}. \text{join_msg}(n,r,r',v')$

update code:

```
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
}
```

$\exists q:\text{quorum}. \forall n:\text{node}. \text{member}(n,q) \rightarrow \text{joined}(n,r)$

Modularity



Proof Length

Protocol	System/Project	LOC	# manual proof	Ratio
RAFT	Coq/Verdi	530	50,000	94
	Ivy	560	200	0.36
MULTIPAXOS	Dafny/IronFleet	3000	12,000	4
	Ivy	330	266	0.8

Verification Effort

Protocol	System/Project	Human Effort	Verification Time
RAFT	Coq/Verdi	3.7 years	-
	Ivy	3 months (from ground up)	Few min
MULTIPAXOS	Dafny/IronFleet	Several years	6hr in cloud
	Ivy	1 month (pre-verified model)	few minutes on laptop

EPR++

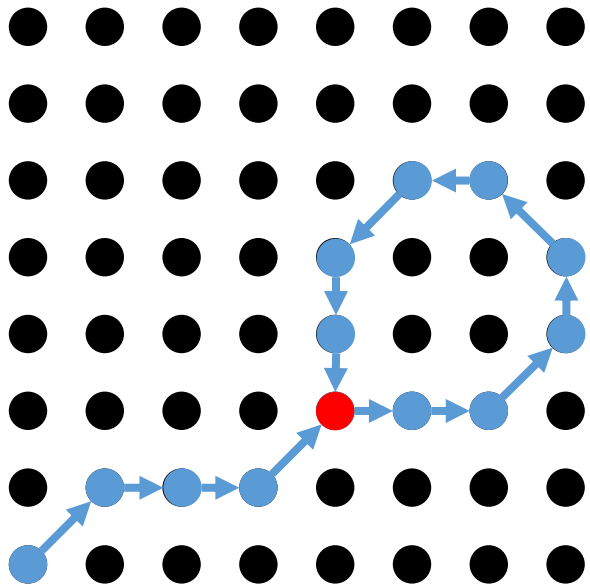
- But what can you possibly express in such a restricted logic?
 - Transitive closure over deterministic paths
 - Set cardinalities
 - Avoiding quantifier alternations
 - Encoding liveness and LTL

Liveness properties

- Liveness property: “something good eventually happens”
- Often depend on fairness assumptions
- Typically proven by ranking functions, well-founded relations
 - Well beyond EPR, or not?

Lasso & Dynamic Abstraction

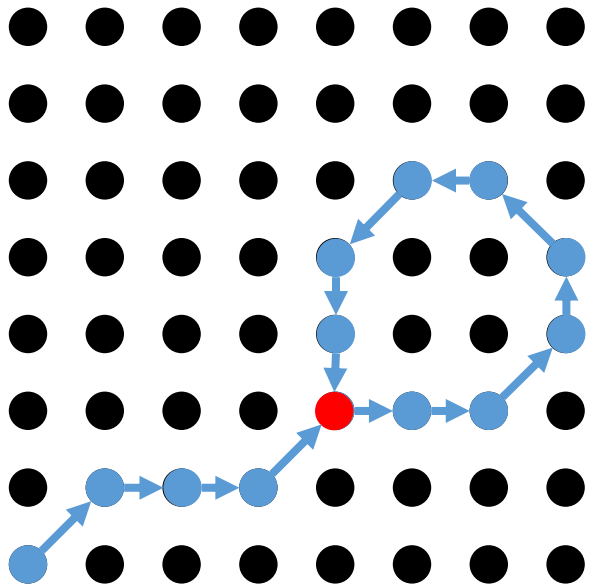
Finite State
Parameterized



Liveness \Leftrightarrow No Lasso

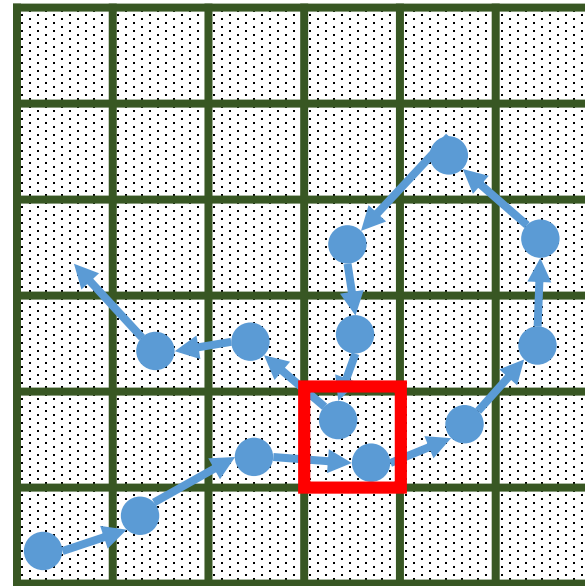
Lasso & Dynamic Abstraction

Finite State
Parameterized



Liveness \Leftrightarrow No Lasso

Infinite State
Finite Abstraction



Liveness \Leftarrow No Lasso
Problem: Spurious Lasso

Summary

- Verification in EPR
 - Deduction is decidable
 - Finite counterexamples
 - Transparent failures
- Powerful encodings
 - Transitive closure for deterministic paths
 - Set cardinalities
 - Derived relations and modularity to battle $\forall\exists$ cycles
 - Liveness and temporal properties

Open Questions

- What cannot be proven with EPR?
 - Proof theory
- Are solvers stable on EPR?
 - Can it be generalized?
 - Part of why CDCL and works
- Can it be used beyond a small group of fans?
 - Transitive closure 😊, LTL 😊, Cardinality 😊, Breaking $\forall\exists$ cycles 😞
- Can you get the benefit of EPR without the costs?
 - Bounded quantifier instantiations
 - Resolution
 - Special case of “simple proofs”?

Mypyvy: A DSL for Transition Relations

- A low level representation of transition relations
- An algorithm for checking inductiveness
 - Does not enforce EPR
 - May diverge when the transition relation is beyond EPR
- An algorithm for inferring inductive invariants
 - UPDR
 - May diverge when even when the transition relation is EPR
 - But the user can always make progress

[CAV12] Aaron R. Bradley: IC3 and beyond: Incremental, Inductive Verification. CAV 2012: 4

[JACM] Aleksandr Karbyshev, Nikolaj Bjørner, Shachar Itzhaky, Noam Rinetzky, Sharon Shoham: Property-Directed Inference of Universal Invariants or Proving Their Absence. J. ACM 64(1): 7:1-7:33 (2017)

markt_ring.pyv (1)

sort node

sort identifier

immutable relation le(identifier, identifier)

axiom le(X,X)

axiom le(X, Y) & le(Y, Z) -> le(X, Z)

axiom le(X, Y) & le(Y, X) -> X = Y

axiom le(X, Y) | le(Y, X)

immutable relation btw(node, node, node)

immutable function id(node) : identifier

axiom [unique_ids] id(N1) = id(N2) -> N1 = N2

axiom btw(W,X,Y) & btw(W,Y,Z) -> btw(W,X,Z)

axiom btw(W,X,Y) -> !btw(W,Y,X)

axiom btw(W,X,Y) | btw(W,Y,X) | W=X | W=Y | X=Y

axiom btw(X,Y,Z) -> btw(Y,Z,X)

mutable relation leader(node)

mutable relation pending(node, node)

markt_ring.pyv (2)

init !leader(N)

init !pending(N1, N2)

markt_ring.pyv (3)

transition send(n: node, next: node)

modifies pending

(**forall** Z. n != next & ((Z != n & Z != next) -> btw(n,next,Z))) &

(**forall** N1, N2. pending(N1, N2) <-> old(pending(N1, N2)) | N1 = n & N2 = next)

markt_ring.pyv (4)

```
transition recv(sender: node, n: node, next: node)
modifies leader, pending
(forall Z. n != next & ((Z != n & Z != next) -> btw(n,next,Z))) & old(pending(sender, n)) &
if sender = n
  then
    (forall N. leader(N) <-> old(leader(N)) | N = n) &
    (forall N1, N2. !(N1 = sender & N2 = n) -> # message may or may not be removed
    (pending(N1, N2) <-> old(pending(N1, N2))))
  else (forall N. leader(N) <-> old(leader(N))) &
  if le(id(n), id(sender))
    then
      forall N1, N2. !(N1 = sender & N2 = n) -> # message may or may not be removed
      (pending(N1, N2) <-> old(pending(N1, N2)) | N1 = sender & N2 = next)
    else forall N1, N2. !(N1 = sender & N2 = n) -> # message may or may not be removed
    (pending(N1, N2) <-> old(pending(N1, N2)))
```

markt_ring.pyv (5)

safety [leader_highest_id] leader(L) \rightarrow le(id(N), id(L))

markt_ring.pyv (6)

invariant [self_pending_highest_id] pending(L, L) \rightarrow le(id(N), id(L))

invariant [no_bypass] pending(S, D) & btw(S, N, D) \rightarrow le(id(N), id(S))

Automatically Inferred Invariants

!(exists identifier0:identifier, identifier1:identifier, node0:node, node1:node. leader(node0) & !le(identifier0, identifier1) & id(node0) = identifier1 & id(node1) = identifier0)

!(exists identifier0:identifier, identifier1:identifier, node0:node, node1:node. pending(node1, node1) & !le(identifier0, identifier1) & id(node0) = identifier0 & id(node1) = identifier1)

!(exists identifier0:identifier, identifier1:identifier, node0:node, node1:node, node2:node. pending(node1, node2) & btw(node0, node2, node1) & !le(identifier0, identifier1) & id(node0) = identifier0 & id(node1) = identifier1)

Summary

- A useful tool for graduate students
- Possible Extensions
 - UI
 - Theories
 - Recursion
 - Scope
 - Rely/Gurantee
 - Different invariant inference
- Can be compiled from different languages (IVY)

Why do people hate First Order Logic?

- Hard to understand and error prone
 - Nested quantifiers and negations
- Too weak: Cannot express
 - Parity
 - Numeric
 - Quorums
 - Finiteness
 - Paths in a graph
- Hard for automation
 - Satisfiability is undecidable
 - NP-complete for fixed size

Our Solution

- Limited EPR formulas $\exists^*\forall^*$
 - Finite model guaranteed to exit
 - Display graphs graphically
- Define axioms per of domain of programs
 - Total orders
 - Paths in deterministic graphs
- Use a Turing complete programming language for updates
- *Satisfiability is NEXPTIME complete/ Σ_2*
 - *Support from Z3, lprover, Vampire*
- Rely on user provided loop invariants or sound inferred invariants