Specifying and Testing Distributed Protocols with Action Temporal Logic

José João Alves dos Santos Ferreira - 99259

Instituto Superior Técnico,
University of Lisbon, Portugal

Dissertation - MSc in Computer Science and Engineering
Advisors: José Fragoso Santos, Alessandro Gianola

Introduction

The importance of distributed systems and protocols

Introduction

The importance of distributed systems and protocols

- large-scale social networks

Introduction

The importance of distributed systems and protocols

- large-scale social networks
- financial infrastructures

Introduction

The importance of distributed systems and protocols

- large-scale social networks
- financial infrastructures
- cloud computing and storage platforms

Introduction

The importance of distributed systems and protocols

- large-scale social networks
- financial infrastructures
- cloud computing and storage platforms
- blockchain networks

Introduction

The importance of distributed systems and protocols

- large-scale social networks
- financial infrastructures
- cloud computing and storage platforms
- blockchain networks

. . .

The need for correctness: from algorithms to implementations

Examples of bugs found in implementations

- FaB Paxos [1]
- Raft [2]
- Tendermint [3]
- Gasper [4]

[1] I. Abraham, G. Gueta, D. Malkhi, L. Alvisi, R. Kotla, and J. Martin, "Revisiting fast practical byzantine fault tolerance," *CoRR*, vol. abs/1712.01367, 2017.

[2] C. Jensen, H. Howard, and R. Mortier, "Examining Raft's behaviour during partial network failures," in *Proceedings of the 1st Workshop on High Availability and Observability of Cloud Systems*, ACM, 2021, pp. 11–17.

[3] C. Cachin and M. Vukolic, "Blockchain consensus protocols in the wild," *CoRR*, vol. abs/1707.01873, 2017.

[4] J. Neu, E. N. Tas, and D. Tse, "Ebb-and-flow protocols: A resolution of the availability-finality dilemma," in *42nd IEEE Symposium on Security and Privacy*, IEEE, 2021, pp. 446–465.

How can we validate distributed protocols?

Approaches for validating distributed protocols

- **deductive verification**
  - TLAPS for Paxos or Raft [1]
  - Coq for Raft [2]
  - EPR for Paxos [3]

[1] K. Chaudhuri, D. Doligez, L. Lamport, and S. Merz, "Verifying safety properties with the TLA+ proof system," in *Proceedings of the 5th International Joint Conference on Automated Reasoning*, Springer, 2010, pp. 142–148.

[2] J. R. Wilcox et al., "Verdi: A framework for implementing and formally verifying distributed systems," in *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ACM, 2015, pp. 357–368.

[3] O. Padon, G. Losa, M. Sagiv, and S. Shoham, "Paxos made EPR: decidable reasoning about distributed protocols," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, 108:1–108:31, 2017.

Approaches for validating distributed protocols

- **model checking**
  - Alloy for Chord [1]
  - TLC for Pastry [2]

[1] P. Zave, "Using lightweight modeling to understand Chord," *Comput. Commun. Rev.*, vol. 42, no. 2, pp. 49–57, 2012.

[2] N. Azmy, S. Merz, and C. Weidenbach, "A machine-checked correctness proof for Pastry," *Science of Computer Programming*, vol. 158, pp. 64–80, 2018.

Approaches for validating distributed protocols

- **fuzzing**
  - Mallory for Raft [1]
  - Chronos for Zab and GHOST [2]

[1] R. Meng, G. Pîrlea, A. Roychoudhury, and I. Sergey, "Greybox fuzzing of distributed systems," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2023, pp. 1615–1629.

[2] Y. Chen, F. Ma, Y. Zhou, M. Gu, Q. Liao, and Y. Jiang, "Chronos: Finding timeout bugs in practical distributed systems by deep-priority fuzzing with transient delay," in *IEEE Symposium on Security and Privacy*, 2024, pp. 1939–1955.

Approaches for validating distributed protocols

- **monitoring**
  - TLA+ for 2PC [1]
  - TLA+ for Raft [2]
  - ATL for Chord [3]

[1] H. Cirstea, M. A. Kuppe, B. Loillier, and S. Merz, "Validating traces of distributed programs against TLA+ specifications," in *Proceedings of the 22nd Int. Conf. on Software Eng. and Formal Methods*, Springer, 2024, pp. 126–143.

[2] H. Howard, M. A. Kuppe, E. Ashton, A. Chamayou, and N. Crooks, "Smart casual verification of the confidential consortium framework," in *22nd USENIX Symposium on Networked Systems Design and Impl.*, 2025, pp. 259–276.

[3] N. Policarpo, J. F. Santos, A. Cunha, J. Leitão, and P. Á. Costa, "Specifying distributed hash tables with allen temporal logic," in *13th IEEE/ACM International Conference on Formal Methods in Software Engineering*, IEEE, 2025, pp. 63–73.

Online vs offline monitoring

Offline monitoring difficulties

- **specification**
  - properties involve complex temporal and data dependencies
  - existing formalisms make specification cumbersome and hard to read

Offline monitoring difficulties

- **specification**
  - properties involve complex temporal and data dependencies
  - existing formalisms make specification cumbersome and hard to read

challenge:

how can we describe properties in an expressive yet intuitive way?

Offline monitoring difficulties

- **logging**
  - instrumentation is hard and system-specific
  - events must be correlated across nodes despite delays and failures

Offline monitoring difficulties

- **logging**
  - instrumentation is hard and system-specific
  - events must be correlated across nodes despite delays and failures

challenge:

how can we obtain and interpret execution logs of protocols?

Offline monitoring difficulties

- **validation**
  - existing tools are not purpose-built and thus suboptimal
  - tools face scalability and efficiency limits on large traces

Offline monitoring difficulties

- **validation**
  - existing tools are not purpose-built and thus suboptimal
  - tools face scalability and efficiency limits on large traces

challenge:

how can we efficiently verify traces against specifications at scale?

Contributions

- **ACTL: definition of a new DSL based on actions**
  - combines the strengths of ATL and FOLTL
  - formal syntax, semantics and proofs
  - efficient trace validation algorithm

Contributions

- **specification of properties of protocols using logics**
  - distributed protocols: 2PC and DHT
  - temporal logics: FOLTL, dFOATL and ACTL

Contributions

- **ACTLChecker: implementation of an efficient monitoring tool**
  - parses a formula, producing an AST
  - parses a log, producing a trace
  - validates the trace on the AST, outputting whether the trace satisfies the property

Contributions

- **evaluation: monitoring of DHTs using ACTLChecker**
  - functionality
  - efficiency and scalability
  - using pre-collected logs of OpenChord and specifications

ACTL

Syntax

$$a(x_1, \ldots, x_n) \overset{i}{\rightsquigarrow} (y_1, \ldots, y_m)$$

Syntax

$$a(x_1, \ldots, x_n) \overset{i}{\rightsquigarrow} (y_1, \ldots, y_m)$$

$$\begin{aligned} \varphi ::= {} & \neg \varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \Rightarrow \varphi_2 \\ & \mid \boxed{\forall a(\overline{x}) \overset{i}{\rightsquigarrow} (\overline{y}) \,.\, \varphi} \mid \boxed{\exists a(\overline{x}) \overset{i}{\rightsquigarrow} (\overline{y}) \,.\, \varphi} \\ & \mid x_1 = x_2 \mid r(i_1, i_2) \end{aligned}$$

Syntax

$$a(x_1, \ldots, x_n) \overset{i}{\rightsquigarrow} (y_1, \ldots, y_m)$$

$$\begin{aligned} \varphi ::= & \neg \varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \Rightarrow \varphi_2 \\ & \mid \boxed{\forall a(\overline{x}) \overset{i}{\rightsquigarrow} (\overline{y}) \,.\, \varphi} \mid \boxed{\exists a(\overline{x}) \overset{i}{\rightsquigarrow} (\overline{y}) \,.\, \varphi} \\ & \mid x_1 = x_2 \mid r(i_1, i_2) \end{aligned}$$

$$\begin{aligned} r(i_1, i_2) ::= & \mathit{Before}(i_1, i_2) \mid \mathit{Meets}(i_1, i_2) \mid \mathit{Overlaps}(i_1, i_2) \\ & \mid \mathit{Starts}(i_1, i_2) \mid \mathit{During}(i_1, i_2) \mid \mathit{Finishes}(i_1, i_2) \\ & \mid \mathit{Equals}(i_1, i_2) \end{aligned}$$

Semantics

$$\sigma : (\mathcal{V} \to D) \cup (\mathcal{I} \to \mathbb{N} \times \mathbb{N})$$

Semantics

$$\sigma : (\mathcal{V} \to D) \cup (\mathcal{I} \to \mathbb{N} \times \mathbb{N})$$

$$\varepsilon ::= B\langle a \rangle_k(\overline{v}) \mid E\langle a \rangle_k(\overline{w})$$

Semantics

$$\sigma : (\mathcal{V} \to D) \cup (\mathcal{I} \to \mathbb{N} \times \mathbb{N})$$

$$\varepsilon ::= B\langle a \rangle_k(\overline{v}) \mid E\langle a \rangle_k(\overline{w})$$

$$\tau = [\{\varepsilon_1, \ldots, \varepsilon_p\}, \ldots, \{\varepsilon_q, \ldots, \varepsilon_r\}]$$

Semantics

$$\sigma : (\mathcal{V} \to D) \cup (\mathcal{I} \to \mathbb{N} \times \mathbb{N})$$

$$\varepsilon ::= B\langle a \rangle_k(\overline{v}) \mid E\langle a \rangle_k(\overline{w})$$

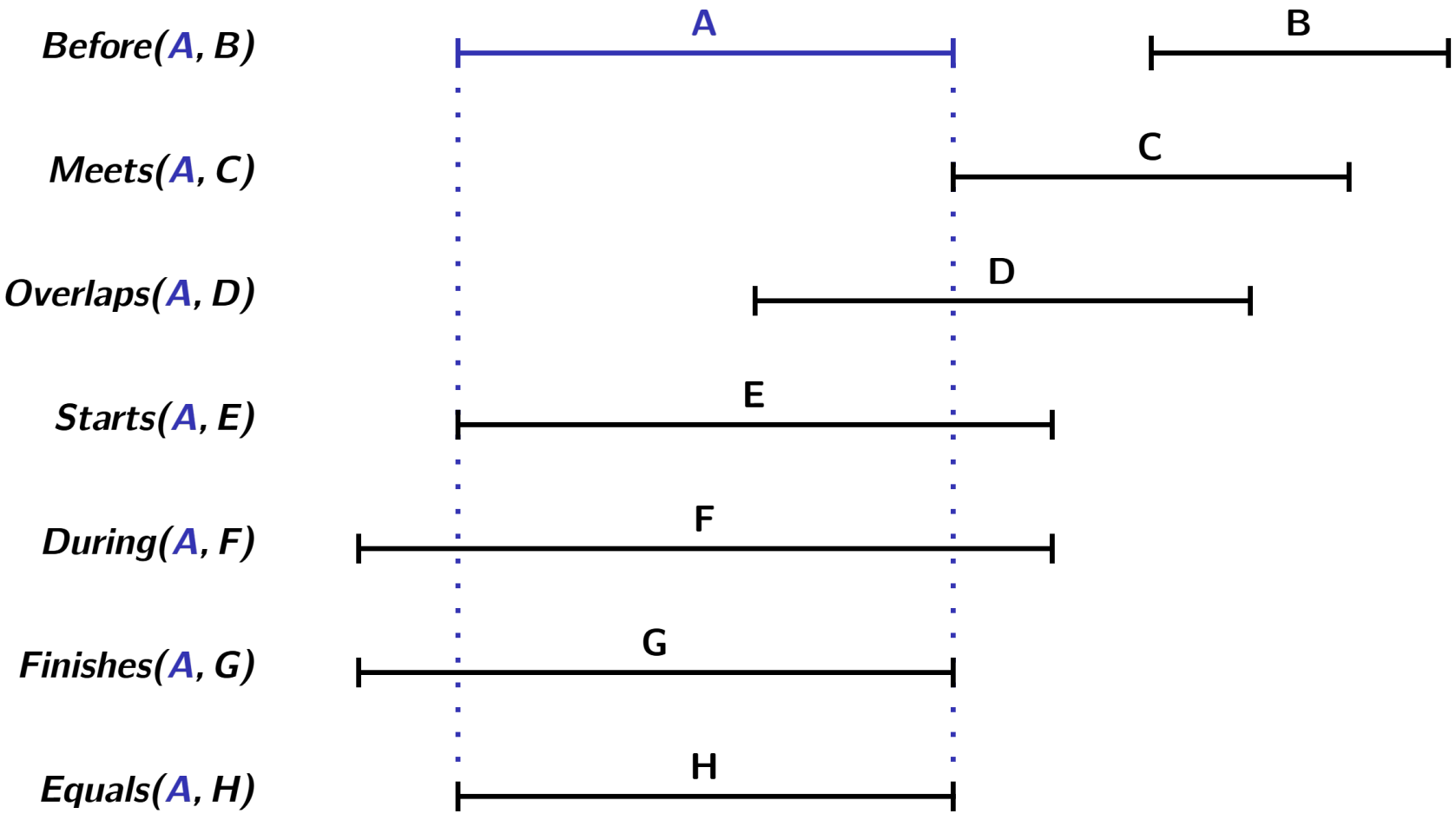
$$\tau = [\{\varepsilon_1, \ldots, \varepsilon_p\}, \ldots, \{\varepsilon_q, \ldots, \varepsilon_r\}]$$

$$\mathit{occurrences}(\tau, a) = \{(t_b, t_e, \overline{v}, \overline{w}) \mid \textbf{exists } k \,.\, \tau[t_b] \ni B\langle a \rangle_k(\overline{v}) \textbf{ and } \tau[t_e] \ni E\langle a \rangle_k(\overline{w})\}$$

Semantics Action Quantifiers

- $\tau, \sigma \models \forall a(\overline{x}) \overset{i}{\rightsquigarrow} (\overline{y}) \,.\, \varphi \iff$
  forall $(\mathrm{t}_b, t_e, \overline{v}, \overline{w}) \in \mathit{occurrences}(\tau, a)$.
  $\tau, \sigma[\overline{x} \mapsto \overline{v}, \overline{y} \mapsto \overline{w}, i \mapsto (t_b, t_e)] \models \varphi$
- $\tau, \sigma \models \exists a(\overline{x}) \overset{i}{\rightsquigarrow} (\overline{y}) \,.\, \varphi \iff$
  exists $(\mathrm{t}_b, t_e, \overline{v}, \overline{w}) \in \mathit{occurrences}(\tau, a)$.
  $\tau, \sigma[\overline{x} \mapsto \overline{v}, \overline{y} \mapsto \overline{w}, i \mapsto (t_b, t_e)] \models \varphi$

Semantics Interval Relations



Distributed Hash Table

DHT operations

- **store**$(n, k, v) \overset{i}{\rightsquigarrow} (n')$: during interval i, the client requests node n to store the key-value pair (k, v); the operation completes at node n'

DHT operations

- **store**$(n, k, v) \overset{i}{\rightsquigarrow} (n')$: during interval i, the client requests node n to store the key-value pair (k, v); the operation completes at node n'
- **lookup**$(n, k) \overset{i}{\rightsquigarrow} (n', v)$: during i, the client requests from n the value for k; the operation completes at n', which stores (k, v) and returns v

DHT operations

- **store**$(n, k, v) \overset{i}{\rightsquigarrow} (n')$: during interval i, the client requests node n to store the key-value pair (k, v); the operation completes at node n'
- **lookup**$(n, k) \overset{i}{\rightsquigarrow} (n', v)$: during i, the client requests from n the value for k; the operation completes at n', which stores (k, v) and returns v
- **findnode**$(n, k) \overset{i}{\rightsquigarrow} (n', n'')$: during i, the client requests from n which node is responsible for k; n' notifies the client that node n'' is responsible

DHT operations

- **store**$(n, k, v) \overset{i}{\rightsquigarrow} (n')$: during interval i, the client requests node n to store the key-value pair (k, v); the operation completes at node n'
- **lookup**$(n, k) \overset{i}{\rightsquigarrow} (n', v)$: during i, the client requests from n the value for k; the operation completes at n', which stores (k, v) and returns v
- **findnode**$(n, k) \overset{i}{\rightsquigarrow} (n', n'')$: during i, the client requests from n which node is responsible for k; n' notifies the client that node n'' is responsible
- **member**$(n)^{\,i}$: during interval i, node n is a member of the network

DHT operations

- **store**$(n, k, v) \overset{i}{\rightsquigarrow} (n')$: during interval i, the client requests node n to store the key-value pair (k, v); the operation completes at node n'
- **lookup**$(n, k) \overset{i}{\rightsquigarrow} (n', v)$: during i, the client requests from n the value for k; the operation completes at n', which stores (k, v) and returns v
- **findnode**$(n, k) \overset{i}{\rightsquigarrow} (n', n'')$: during i, the client requests from n which node is responsible for k; n' notifies the client that node n'' is responsible
- **member**(n) ^{i}: during interval i, node n is a member of the network
- **ideal** ^{i}: during i, the network is in an *ideal* state (i.e. every node holds sufficient information to forward messages to every other node correctly)

DHT properties

(VALUE) **Lookup consistency**: if a lookup for key k reads a value v, then that value must have been previously assigned to that key by a store operation

$$\forall\, \mathbf{lookup}(-, k) \overset{l}{\rightsquigarrow} (-, v)\,.\, \exists\, \mathbf{store}(-, k, v) \overset{s}{\rightsquigarrow} (-)\,.\, \neg \mathit{Before}(l, s) \wedge \neg \mathit{Meets}(l, s)$$

DHT properties

(STRUCTURAL) **Reachability**: if any node n is a member during an *ideal* state, then all *findnode* operations of the key k with the same identifier as the node must return the node

$$\forall\, \mathbf{findnode}(-, k) \overset{f}{\rightsquigarrow} (-, n)\,.\, \forall\, \mathbf{ideal}^{\,i}\,.\, \forall\, \mathbf{member}(k)^{\,m}\,.$$
$$\Big(\big(\mathit{In}(f, m) \vee \mathit{Equals}(f, m)\big) \wedge \big(\mathit{In}(m, i) \vee \mathit{Equals}(m, i)\big)\Big) \Rightarrow k = n$$

DHT properties

(STRUCTURAL) **Reachability**: if any node n is a member during an *ideal* state, then all *findnode* operations of the key k with the same identifier as the node must return the node

$$\forall\, \mathbf{findnode}(-, k) \overset{f}{\rightsquigarrow} (-, n)\,.\, \forall\, \mathbf{ideal}^{\,i}\,.\, \forall\, \mathbf{member}(k)^{\,m}\,.$$
$$\Big(\big(\mathit{In}(f, m) \vee \mathit{Equals}(f, m)\big) \wedge \big(\mathit{In}(m, i) \vee \mathit{Equals}(m, i)\big)\Big) \Rightarrow k = n$$

$$\mathit{In}(i_1, i_2) \iff \mathit{Starts}(i_1, i_2) \vee \mathit{During}(i_1, i_2) \vee \mathit{Finishes}(i_1, i_2)$$

DHT properties Reachability

$$\forall\, \mathbf{findnode}(-, k) \overset{f}{\rightsquigarrow} (-, n)\,.\, \forall\, \mathbf{ideal}\ i\ .\, \forall\, \mathbf{member}(k)\ m\ .$$
$$\Big(\big(\mathit{In}(f, m) \vee \mathit{Equals}(f, m)\big) \wedge \big(\mathit{In}(m, i) \vee \mathit{Equals}(m, i)\big)\Big) \Rightarrow k = n$$

DHT properties Reachability

$$\forall\, \textbf{findnode}(-, k) \overset{f}{\rightsquigarrow} (-, n)\,.\, \forall\, \textbf{ideal}\ i\ .\, \forall\, \textbf{member}(k)\ m\ .$$
$$\Big(\big(\mathit{In}(f, m) \vee \mathit{Equals}(f, m)\big) \wedge \big(\mathit{In}(m, i) \vee \mathit{Equals}(m, i)\big)\Big) \Rightarrow k = n$$

	τ	
32	$B\langle\textbf{ideal}\rangle()$	
35	$B\langle\textbf{member}\rangle_{a93}(488\mathrm{D})$	
36	$B\langle\textbf{findnode}\rangle_{b67}(\mathrm{N254}, 488\mathrm{D})$	
39	$E\langle\textbf{findnode}\rangle_{b67}(\mathrm{O1F5}, 488\mathrm{D})$	
43	$E\langle\textbf{ideal}\rangle()$	$E\langle\textbf{member}\rangle_{a93}()$

DHT properties Reachability

$$\forall\, \textbf{findnode}(-, k) \overset{f}{\rightsquigarrow} (-, n)\,.\, \forall\, \textbf{ideal}^{\,i}\,.\, \forall\, \textbf{member}(k)^{\,m}\,.$$
$$\Big(\big(\mathit{In}(f, m) \vee \mathit{Equals}(f, m)\big) \wedge \big(\mathit{In}(m, i) \vee \mathit{Equals}(m, i)\big)\Big) \Rightarrow k = n$$

τ		
32	$B\langle\textbf{ideal}\rangle()$	
35	$B\langle\textbf{member}\rangle_{a93}(488\mathrm{D})$	
36	$B\langle\textbf{findnode}\rangle_{b67}(\mathrm{N254}, 488\mathrm{D})$	
39	$E\langle\textbf{findnode}\rangle_{b67}(\mathrm{O1F5}, \mathrm{D32C})$	
43	$E\langle\textbf{ideal}\rangle()$	$E\langle\textbf{member}\rangle_{a93}()$

Decision Procedure

Decision Procedure Pseudocode

Function CHECK(τ, σ, φ)**:**
- **switch** φ **do**
  - **case** $\neg\varphi'$ **do**
    - **return not** CHECK(τ, σ, φ')
  - **case** $\varphi' \vee \varphi''$ **do**
    - **return** CHECK(τ, σ, φ') **or** CHECK$(\tau, \sigma, \varphi'')$
  - **case** $\varphi' \wedge \varphi''$ **do**
    - **return** CHECK(τ, σ, φ') **and** CHECK$(\tau, \sigma, \varphi'')$
  - **case** $\varphi' \Rightarrow \varphi''$ **do**
    - **return not** CHECK(τ, σ, φ') **or** CHECK$(\tau, \sigma, \varphi'')$
  - **case** $\forall a(\overline{x}) \overset{i}{\rightsquigarrow} (\overline{y})\,.\,\varphi'$ **do**
    - **foreach** $(t_b, t_e, \overline{v}, \overline{w}) \in \mathcal{M}_\tau[a]$ **do**
      - $c \leftarrow$ CHECK$(\tau, \sigma[\overline{x} \mapsto \overline{v}, \overline{y} \mapsto \overline{w}, i \mapsto (t_b, t_e)], \varphi')$
      - **if not** c **then return** $false$
    - **return** $true$
  - **case** $\exists a(\overline{x}) \overset{i}{\rightsquigarrow} (\overline{y})\,.\,\varphi'$ **do**
    - **foreach** $(t_b, t_e, \overline{v}, \overline{w}) \in \mathcal{M}_\tau[a]$ **do**
      - $c \leftarrow$ CHECK$(\tau, \sigma[\overline{x} \mapsto \overline{v}, \overline{y} \mapsto \overline{w}, i \mapsto (t_b, t_e)], \varphi')$
      - **if** c **then return** $true$
    - **return** $false$
  - **case** $x_1 = x_2$ **do**
    - **return** $\sigma(x_1) = \sigma(x_2)$
  - **case** $r(i_1, i_2)$ **do**
    - **return** $[\![r]\!](\sigma(i_1), \sigma(i_2))$

Decision Procedure Pseudocode

$$\mathcal{M}_\tau[a] \equiv \mathit{occurrences}(\tau, a)$$

Function PREPROCESSTRACE(τ):
- $\mathcal{M}_\tau \leftarrow \emptyset$; $\mathcal{X} \leftarrow \emptyset$
- **for** $t \leftarrow 0$ **to** $|\tau| - 1$ **do**
  - **foreach** $\varepsilon \in \tau[t]$ **do**
    - **switch** ε **do**
      - **case** $B\langle a \rangle_k(\overline{v})$ **do**
        - $\mathcal{X}[k] \leftarrow (t, \overline{v})$
      - **case** $E\langle a \rangle_k(\overline{w})$ **do**
        - $(t_b, \overline{v}) \leftarrow \mathcal{X}[k]$
        - $\mathcal{M}_\tau[a] \leftarrow \mathcal{M}_\tau[a] \cup (t_b, t, \overline{v}, \overline{w})$
- **return** $\mathcal{M}_\tau$

Decision Procedure Complexity

- $\textsc{PreprocessTrace}(\tau) \in \mathcal{O}(e \cdot |\tau|)$
  - traverse τ to build $\mathcal{M}_\tau$, pairing begin/end events in one pass
  - use auxiliary map $\mathcal{X}$ to track ongoing events and match identifiers k
  - $|\tau|$ is the number of time points, and e the max events per point
  - after building $\mathcal{M}_\tau$, occurrences of any action are retrieved in $\mathcal{O}(1)$

Decision Procedure Complexity

- $\textsc{Check}(\tau, \sigma, \varphi) \in \mathcal{O}(|\tau|^{|\varphi|})$
  - each action quantifier is $\mathcal{O}(1)$, using the precomputed $\mathcal{M}_\tau$
  - for each occurrence found, we recursively checks a subformula
  - the number of occurrences per action is of the order of magnitude of $|\tau|$
  - in the worst-case, the total sum of nested action quantifiers $= |\varphi|$

Implementation

Specification Language for ACTL

- ACTL syntax is concise and readable for humans, but not practical for machines
- prefix notation with nested parentheses is employed, which preserves readability while enabling machine parsing and evaluation

Specification Language for ACTL Lookup consistency

$$\forall\, \mathbf{lookup}(-, k) \overset{l}{\rightsquigarrow} (-, v)\,.\, \exists\, \mathbf{store}(-, k, v) \overset{s}{\rightsquigarrow} (-)\,.$$
$$\neg \mathit{Before}(l, s) \wedge \neg \mathit{Meets}(l, s)$$

```
(forall lookup l (- k) (- v)
    (exists store s (- k v) (-)
        (and
            (not (before l s))
            (not (meets l s))
)   )   )
```

Specification Language for ACTL Reachability

$$\forall\, \textbf{findnode}(-, k) \overset{f}{\rightsquigarrow} (-, n)\,.\, \forall\, \textbf{ideal}^{\,i}\,.\, \forall\, \textbf{member}(k)^{\,m}\,.$$
$$\Big(\big(\mathit{In}(f, m) \vee \mathit{Equals}(f, m)\big) \wedge \big(\mathit{In}(m, i) \vee \mathit{Equals}(m, i)\big)\Big) \Rightarrow k = n$$

```
(forall findnode f (- k) (- n)
    (forall ideal i () ()
        (forall member m (k) ()
            (implies
                (and
                    (or (in f m) (equals f m))
                    (or (in m i) (equals m i))
                )
                (k = n)
)   )   )   )
```

Instrumentation of a DHT implementation

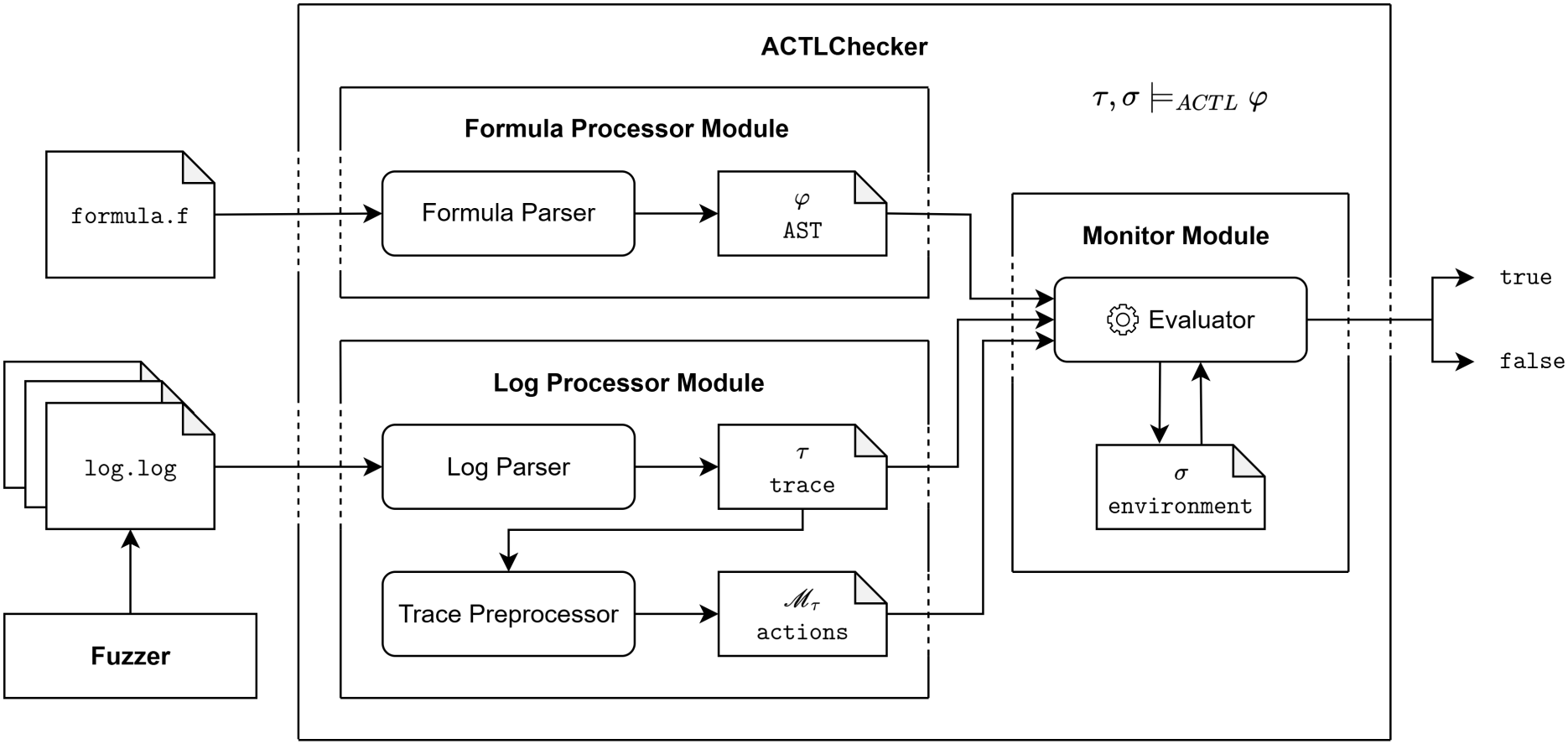
- logs are raw system data, traces are processed records for analysis
- instrumentation is required to collect logs and must correctly correlate events across concurrent nodes
- fuzzers generate diverse inputs to produce broader behavior coverage
- we reuse the OpenChord logs instrumented by Policarpo et al [1]

[1] N. Policarpo, J. F. Santos, A. Cunha, J. Leitão, and P. Á. Costa, "Specifying distributed hash tables with allen temporal logic," in *13th IEEE/ACM International Conference on Formal Methods in Software Engineering*, IEEE, 2025, pp. 63–73.

Instrumentation of a DHT implementation

```
2025-03-23 17:34:26.649, EndStable
2025-03-23 17:34:26.650, Join, a900fe36, 488D
2025-03-23 17:34:27.016, FindNode, be67d936, 7BB3, 488D
2025-03-23 17:34:27.017, ReplyFindNode, be67d936, 7BB3, 7BB3
2025-03-23 17:34:27.397, ReplyJoin, a900fe36
2025-03-23 17:34:27.398, Member, 488D
2025-03-23 17:34:27.398, Stable
2025-03-23 17:36:30.113, EndReadOnly
2025-03-23 17:36:30.114, Store, f95852e0, 7BB3, 8C1E, 44E6
2025-03-23 17:36:30.163, ReplyStore, f95852e0, ADC0
2025-03-23 17:36:30.164, ReadOnly
```

ACTLChecker



https://github.com/jjasferreira/actl-checker

Evaluation

Evaluation

Case study: DHT

We evaluate ACTLChecker using DHT traces, because of:

Evaluation

Case study: DHT

We evaluate ACTLChecker using DHT traces, because of:

- existing generic DHT properties already defined in ACTL for use

Evaluation

Case study: DHT

We evaluate ACTLChecker using DHT traces, because of:

- existing generic DHT properties already defined in ACTL for use
- pre-collected OpenChord logs available from prior work, avoiding extra instrumentation

Overview

We assess ACTLChecker from two complementary perspectives:

Evaluation

Overview

We assess ACTLChecker from two complementary perspectives:

- **functionality:** ability to verify or falsify ACTL properties on real DHT traces

Evaluation

Overview

We assess ACTLChecker from two complementary perspectives:

- **functionality:** ability to verify or falsify ACTL properties on real DHT traces
- **efficiency:** performance and scalability compared with prior Alloy-based evaluation

Evaluation

Functionality

Class	Property	ACTLChecker
VALUE	Lookup consistency	`true`
	Value consistency	`true`
	Value freshness	`true`
KEY	Key consistency	`true`
	Findnode lookup consistency	`true`
	Responsibility transfer	`false`
STRUCTURAL	Membership guarantee	`true`
	Reachability	`true`

Evaluation

Functionality

- all properties held except responsibility transfer, as expected from Chord's algorithm
- all specifications for the 8 properties were ran on multiple non-vacuous traces of different lengths

Functionality Vacuous tests

Property	Trace Length
Lookup consistency	150
Value consistency	vacuous
Value freshness	150
Key consistency	25
Findnode lookup consistency	25
Responsibility transfer	vacuous
Membership guarantee	25
Reachability	50

Evaluation

Efficiency Lookup consistency and Reachability

	Lookup consistency		Reachability	
Trace Length	Alloy	**ACTL**	Alloy	**ACTL**
50	0.618	**0.014**	0.662	**0.017**
125	2.820	**0.016**	2.800	**0.019**
250	77.375	**0.020**	16.876	**0.022**

Evaluation

Efficiency Average of all 8 properties

Trace Length	Alloy	**ACTLChecker**
50	2.883	**0.020**
125	125.827	**0.022**
250	1591.132	**0.027**

Evaluation

Efficiency

- 5-node networks and around 10 traces for each increasing length
- ACTLChecker times measured with Python's time module
- Alloy handled traces up to 250 events but showed clear scalability limits and ran out of memory beyond that
- some traces remain vacuous for certain properties under these settings

Conclusions and Future Work

Conclusions

- modern systems rely on distributed protocols, whose implementations can still fail
- we focus on offline monitoring, since tools like Alloy and TLA+ do not scale

Conclusions

- modern systems rely on distributed protocols, whose implementations can still fail
- we focus on offline monitoring, since tools like Alloy and TLA+ do not scale
- **contributions**
  - ACTL: a new action-based temporal logic
  - formal specification of distributed protocols
  - ACTLChecker: an efficient monitoring tool
  - evaluation on OpenChord traces

Future Work

Future Work

- evaluation with larger logs would further confirm ACTLChecker's scalability

Future Work

- evaluation with larger logs would further confirm ACTLChecker's scalability
- testing networks with more nodes and keys would help identify performance limits and match real industrial conditions

Future Work

- evaluation with larger logs would further confirm ACTLChecker's scalability
- testing networks with more nodes and keys would help identify performance limits and match real industrial conditions
- apply ACTLChecker to other consensus and blockchain protocols

Future Work

- evaluation with larger logs would further confirm ACTLChecker's scalability
- testing networks with more nodes and keys would help identify performance limits and match real industrial conditions
- apply ACTLChecker to other consensus and blockchain protocols
- underlying techniques can extend to areas like security, authentication, IoT communication, and public administration protocols

Conclusions and Future Work

Thank you

Specifying and Testing Distributed Protocols with Action Temporal Logic

José João Alves dos Santos Ferreira - 99259

Instituto Superior Técnico,
University of Lisbon, Portugal

Dissertation - MSc in Computer Science and Engineering
Advisors: José Fragoso Santos, Alessandro Gianola