Reef

Fast Succinct Non-Interactive Zero-Knowledge Regex Proofs

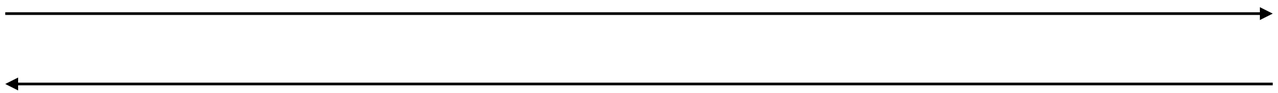
Sebastian Angel[+], Eleftherios Ioannidis[+], Eli Margolin[+], Srinath Setty[*], Jess Woods[+]

[+]University of Pennsylvania, *Microsoft Research

Scenario - Clinical Trials

Patty registers online for Vikki's clinical trial and is automatically accepted

Patty



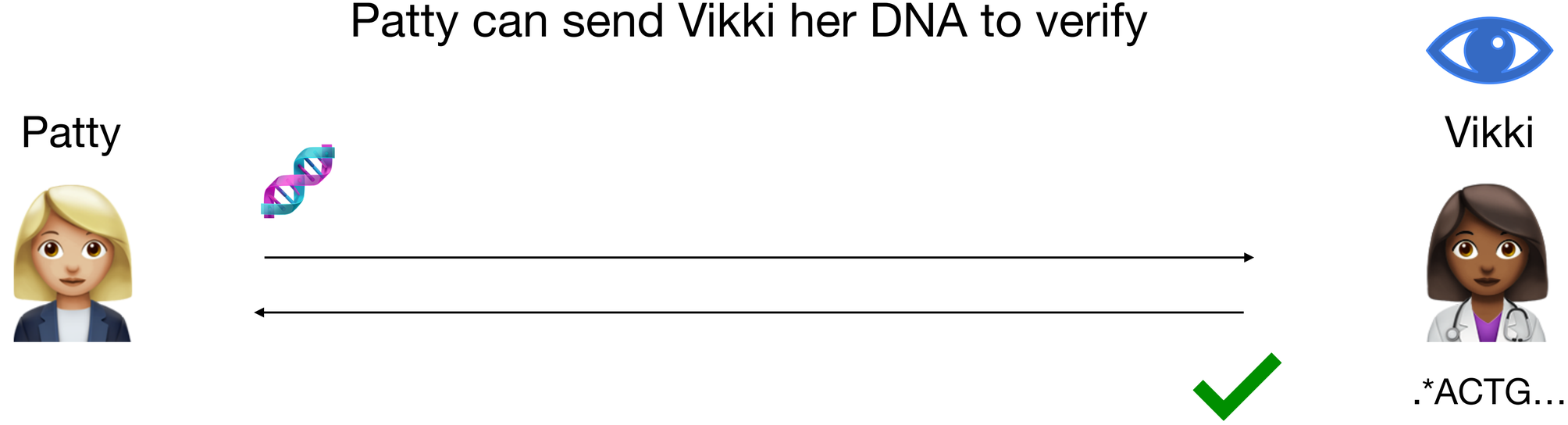
Vikki

.*ACTG…

Vikki can't ensure Patty actually qualifies

Have Patty send her DNA

Patty can send Vikki her DNA to verify



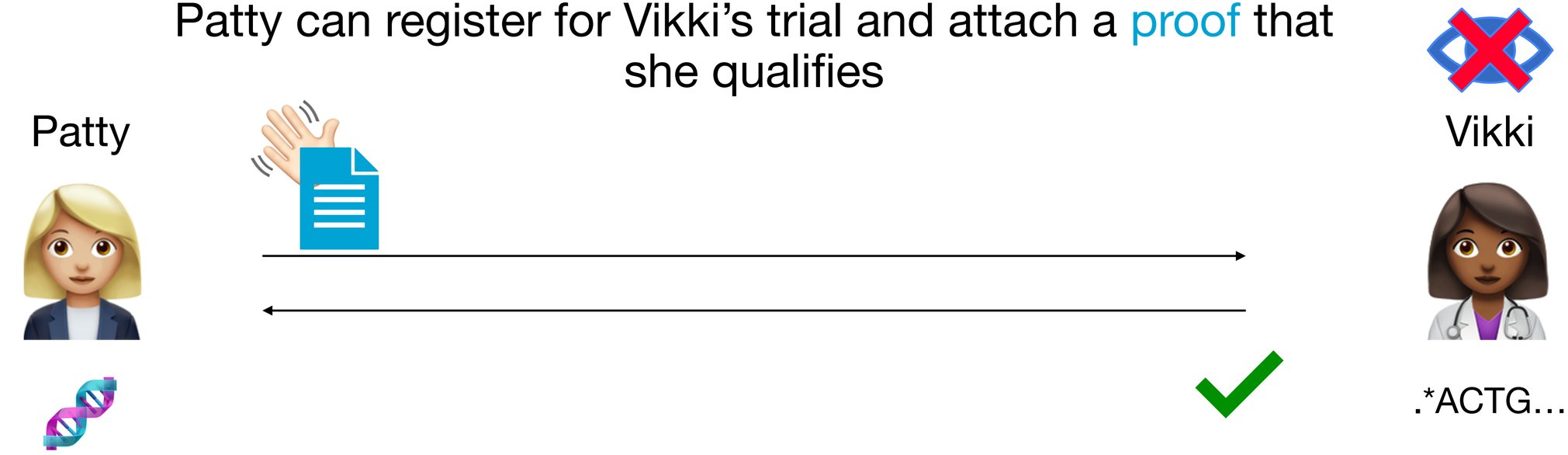
Vikki has Patty's DNA in plain text, even if Patty isn't accepted into the trial

We want to prove Patty qualifies for Vikki's trial

Without Vikki learning Patty's DNA before she's enrolled

Zero Knowledge Proofs

Patty can register for Vikki's trial and attach a proof that she qualifies



Vikki only learns if Patty qualifies for her trial

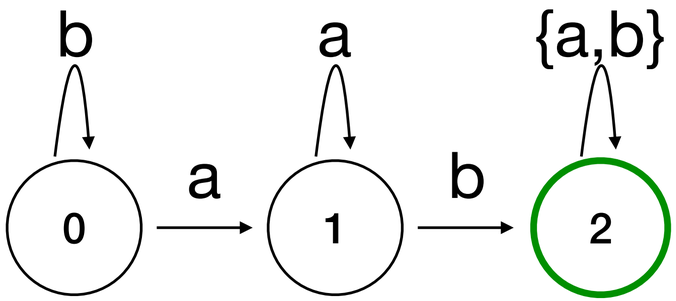
How to make a ZKP in 4 easy steps…

1. Vikki expresses her regex matching statement as a circuit satisfiability instance
2. Vikki publishes her circuit
3. Patty finds a satisfying witness
4. Patty proves to Vikki that she knows the satisfying witness

Naive Solution

`a+b.*`

(Q: {0,1,2}, Σ: {a,b}, δ, q_0:{0}, F:{2})

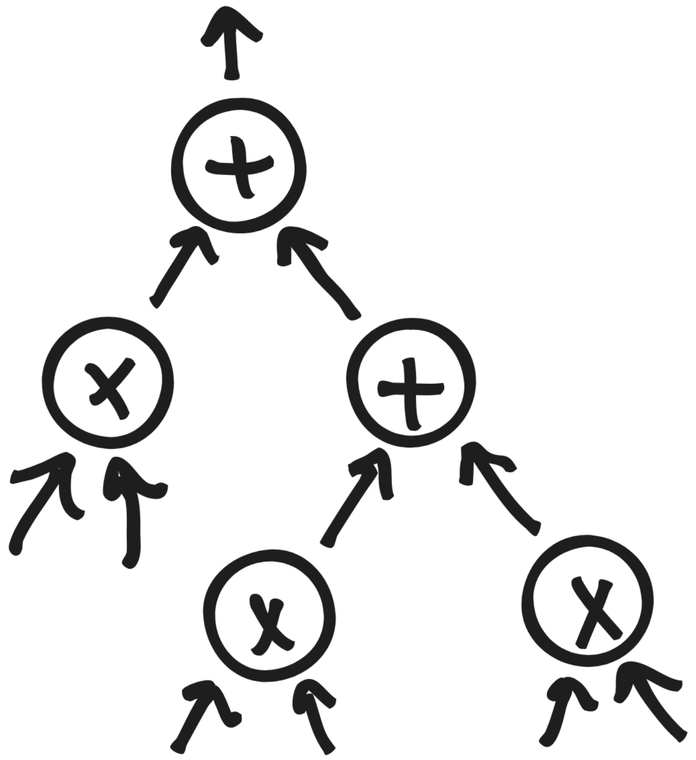


Limited Expressivity

```
field match(field commit, field blind) {
  field[SIZE] document = open(commit,
                              blind);
  field state = 0; // initial state
  for (i = 0; i < SIZE; i++) {
    state = delta(state, document[i]);
  }
  if (state == 2) { // accepting state
    return 1; // match
  } else {
    return 0; // no match
  }
}
```

Going from `match` to circuit

```
field delta(field state, field cur_char) {
  if (state == 0 && cur_char == 0) return 1;
  if (state == 0 && cur_char == 1) return 0;
                    ...
  if (state == 2 && cur_char == 1) return 2;
  return -1; // invalid state or character
}
```



Unfold `delta` |document| times

Key Insight 1 - Skipping Alternating Finite Automata

Alternating Finite Automata give us greater expressivity

We can extend AFA to skip irrelevant parts of the document

Key Insight 2 - Lookup Arguments

We can represent the cascading `if` statements as (start state, character, end state) lookup arguments in the circuit

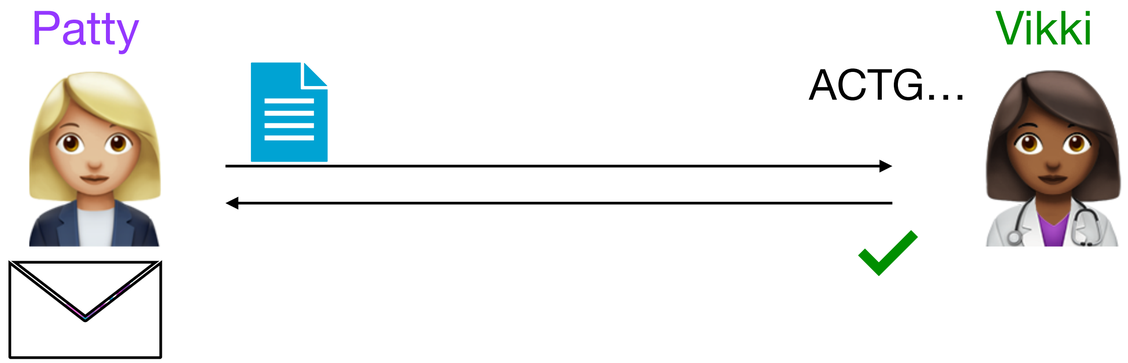
Key Insight 3 - Recursion

We can make iterating through the document much faster using a recursive proof system

Reef is able to decouple Prover running time from document size

Reef - Zero Knowledge Regex Proofs

- Patty commits to her DNA
- Vikki publishes a regex of the genetic variant required to participate in her trial
- Patty proves (in zero knowledge) that her committed DNA matches the public regex
- Vikki verifies Patty's proof



Roadmap

Background

Reef

- Skipping Alternating Finite Automata
- Lookup Arguments
- Adding Recursion
- Optimizations

Evaluation

Future Work

Summary

Background - Zero Knowledge Proofs

- Protocol that allows a Prover *P* to prove some statement to a Verifier *V*
- Proofs are…
  - Complete - *V* is always convinced of a true statement
  - Sound - *P* cannot convince *V* of a false statement
  - Zero Knowledge - *V* learns nothing except the truth of the statement
- Proofs are arithmetized using rank-1 constraint satisfiability (R1CS)
  - 1 constraint = 1 multiplication in circuit
  - More constraints → more complex proof

Background - Recursive Proof Systems

- Produce a proof π for each `do x`

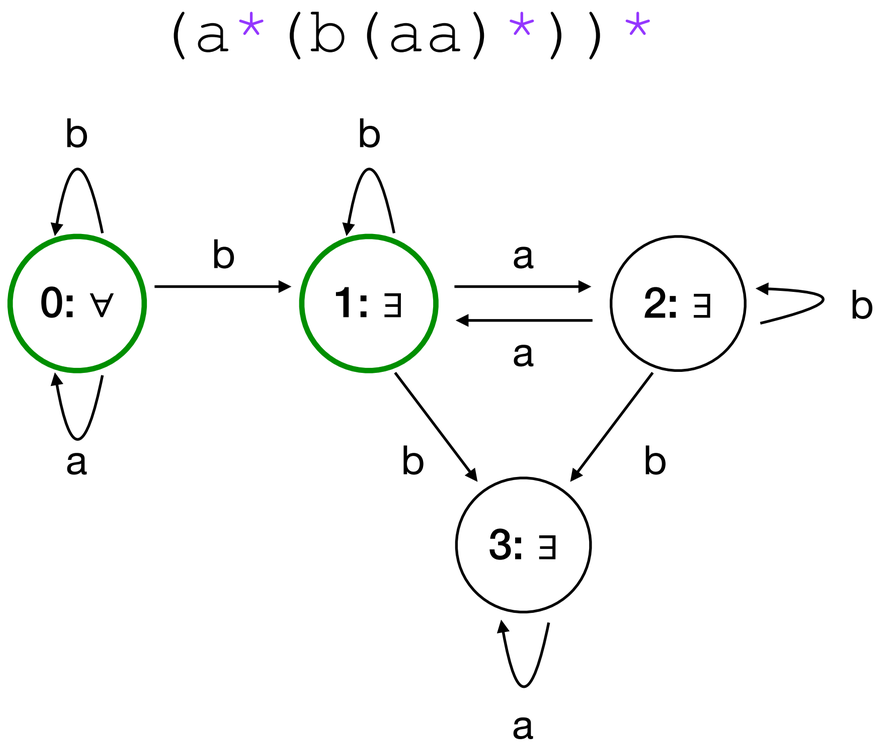
```
for (i = 0; i < j; i++) {
    do x;
}
```

$$\pi_0 \longrightarrow \pi_1+V(\pi_0) \longrightarrow \ldots \longrightarrow \pi_{j-1}+V(\pi_{j-2}) \longrightarrow \pi_j+V(\pi_{j-1})$$

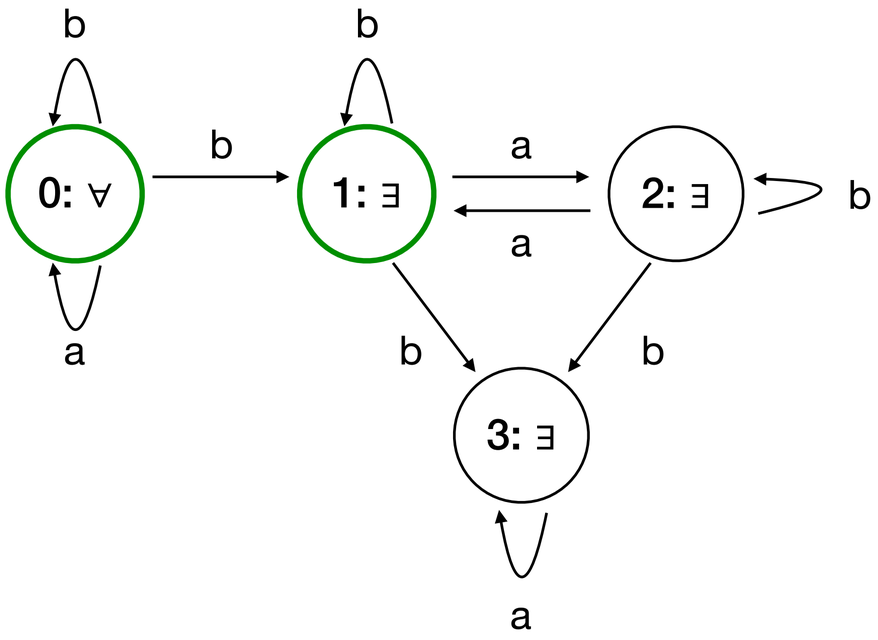
- Circuit size is parameterized by just one iteration

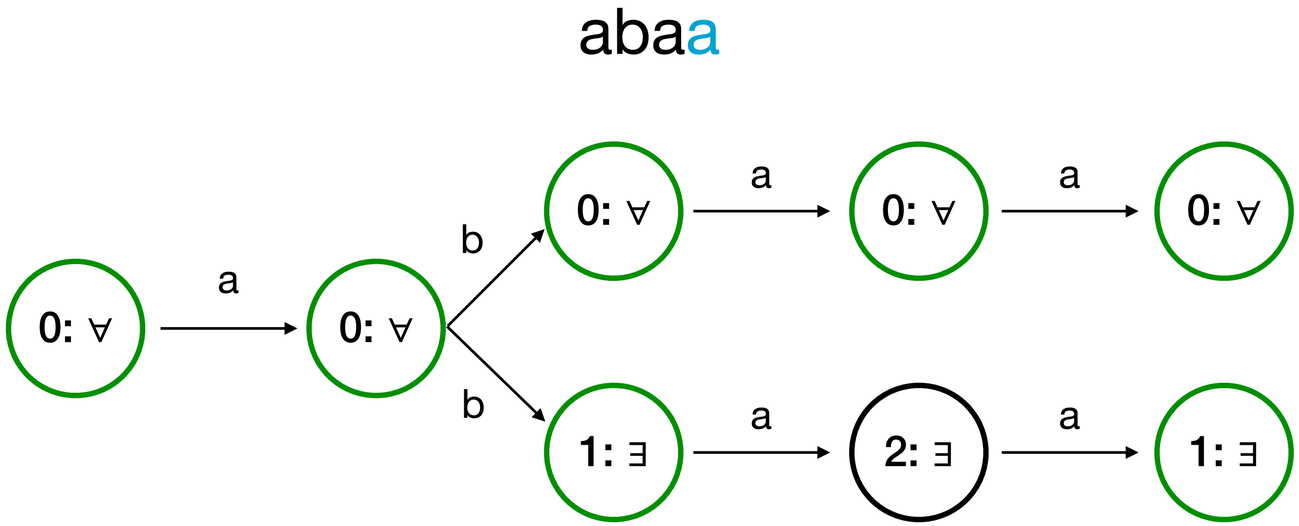
Background - Alternating Finite Automata

- Generalization of NFA with states labeled as ∃ or ∀
- ∃ same as normal NFA
- ∀ takes all transitions in parallel
- All ∀ transition paths must accept
- ∀ transitions support lookarounds



Background - Alternating Finite Automata

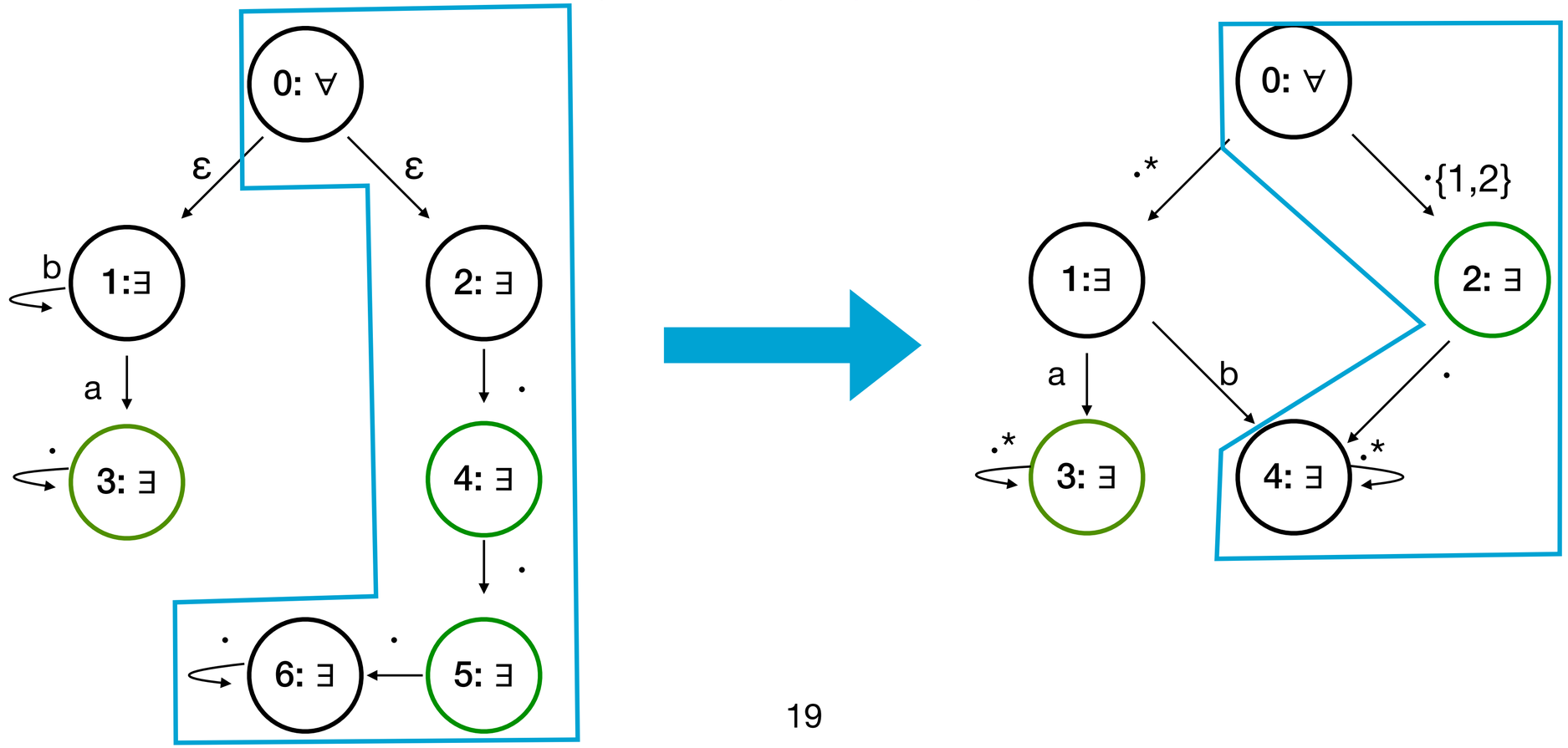




Skipping Alternating Finite Automata

- Compress multiple wildcards (`.{n}`,`.{m,n}`,`.*`) into a single transition

`(?=.*a).{1,2}`

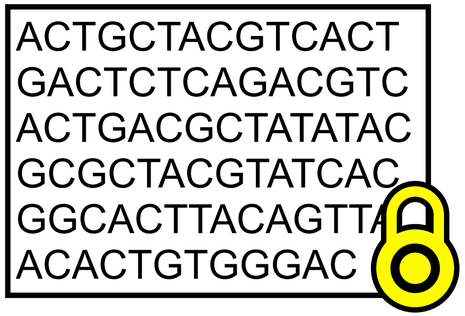


SAFA

- Designed to work with NP-Checkers
    - Reef traverses the SAFA with non-deterministic hints
    - Prover provides pre and post skip cursor
    - Only need to *check* that the hints are correct
- Keep track of finished branches with a minimal stack
- But how do we represent SAFA?

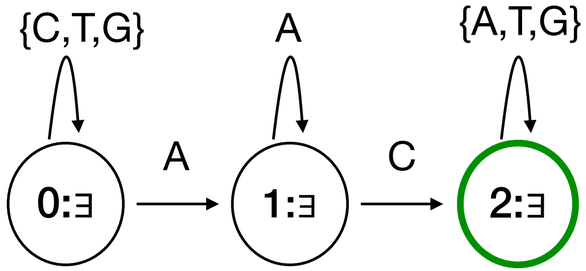
Lookup Arguments

- Prove that some values $\{v_0,\ldots,v_{m-1}\}$ are in a table T
- 2 tables



0	A
1	C
2	T
3	G
…	…

Commitment - ties characters to indices



0	G	0
0	T	0
0	C	0
0	A	1
…	…	…

Replaces the `delta` function

Everything's Better with Recursion

- Naive lookup argument → m·(log(n) + n)
- With recursion maintain a running claim to check at the end
- As the number of lookups increases, cost-per-lookup decreases

$$|D|\cdot(\log(|D|) + |D|)$$
$$|D|\cdot(\log(|SAFA|) + |SAFA|)$$

Check lookup inclusion at each step

$$|D|\cdot(\log(|D|)) + |D|$$
$$|D|\cdot(\log(|SAFA|)) + |SAFA|$$

Optimizations

- Hybrid tables
  - Public SAFA Table + Private Document Table → $m\log(|\text{Document}|\cdot|\text{SAFA}|)$ constraints
  - Single Hybrid Public/Private Table → $m\log(|\text{Document}|+|\text{SAFA}|)$ constraints
- Document Projections
  - Run lookup over subsets of the larger document table
  - Works for regexs with prescribed offsets
    - .{10}abc.* ✓
    - .*abc ✗

Roadmap

- Background
- Reef
  - Skipping Alternating Finite Automata
  - Lookup Arguments
  - Adding Recursion
  - Optimizations
- Evaluation
- Future Work
- Summary

Evaluation

- Can Reef support a variety of regexs?
- Can Reef support a variety of document sizes?
- Is Reef efficient for the prover?
- Is Reef efficient for the verifier?
- How does Reef compare to existing/alternative solution?
- Do SAFA meaningfully reduce the size of the automata?
- What impact do Reef's additional optimizations have?

Reef supports more robust Regexs

More expressivity with fewer constraints

	Fixed String Matching	Wildcards	Kleene stars	Negation	Alternation	Lookarounds	# Constraints
Reef	✓	✓	✓	✓	✓	✓	$O(\alpha \log(D+Q_{SAFA}+\lvert\Sigma\rvert))$
Zombie	✓	✓	✓	✓	✓		$O(D \cdot Q_{TNFA})$
ZKRegex	✓	✓	✓	✓			$O(D \cdot Q_{TNFA} \cdot \log(\lvert\Sigma\rvert))$
zkreg	✓	✓					$O(D+Q_{AC\text{-}DFA})$

D - Document Size, Q - # Transitions in Automata, Σ - Alphabet, α - Number of lookups

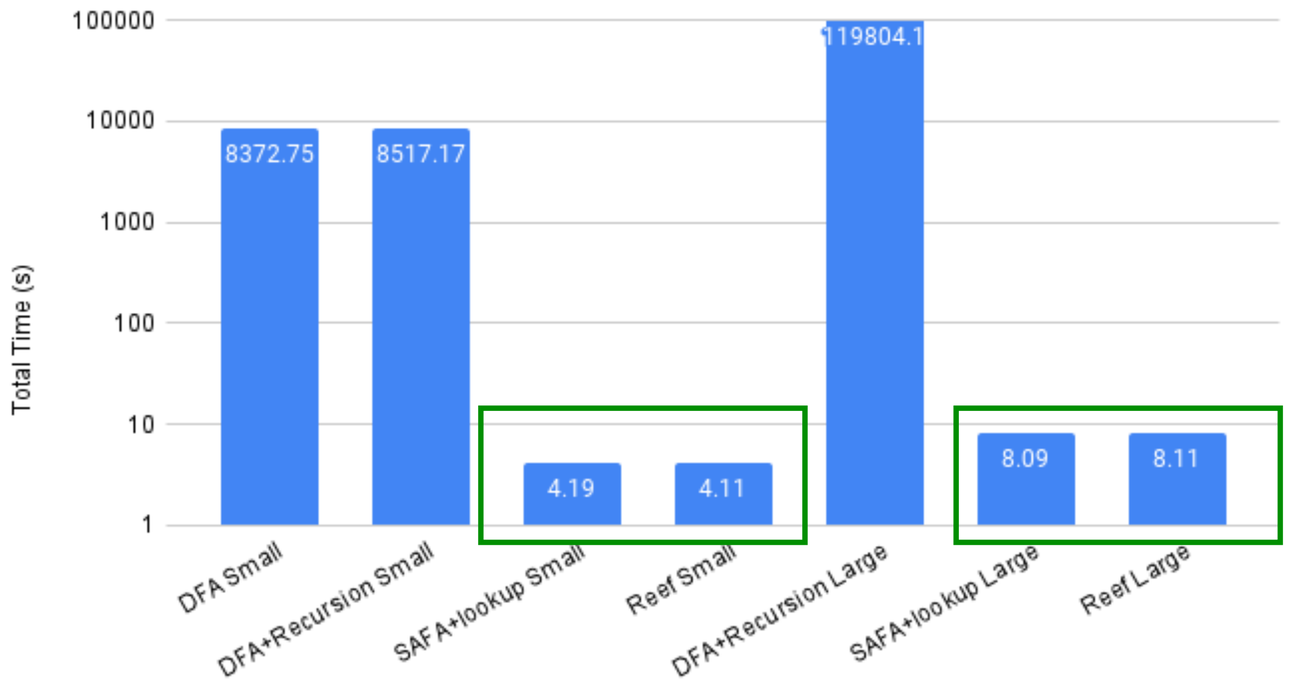
Reef Supports a Variety of Applications

	Document Size	Average SAFA Size - States	Average SAFA Size - Transitions
Document Redaction (Small)	415	331	42,318
Document Redaction (Large)	1,000	908	116,751
ODoH	128	16	1,958
Strong Password Match/Non-Match	12/6-9	21	1,188
DNA Match/Non-Match	32 million - 43 million	546	29,832

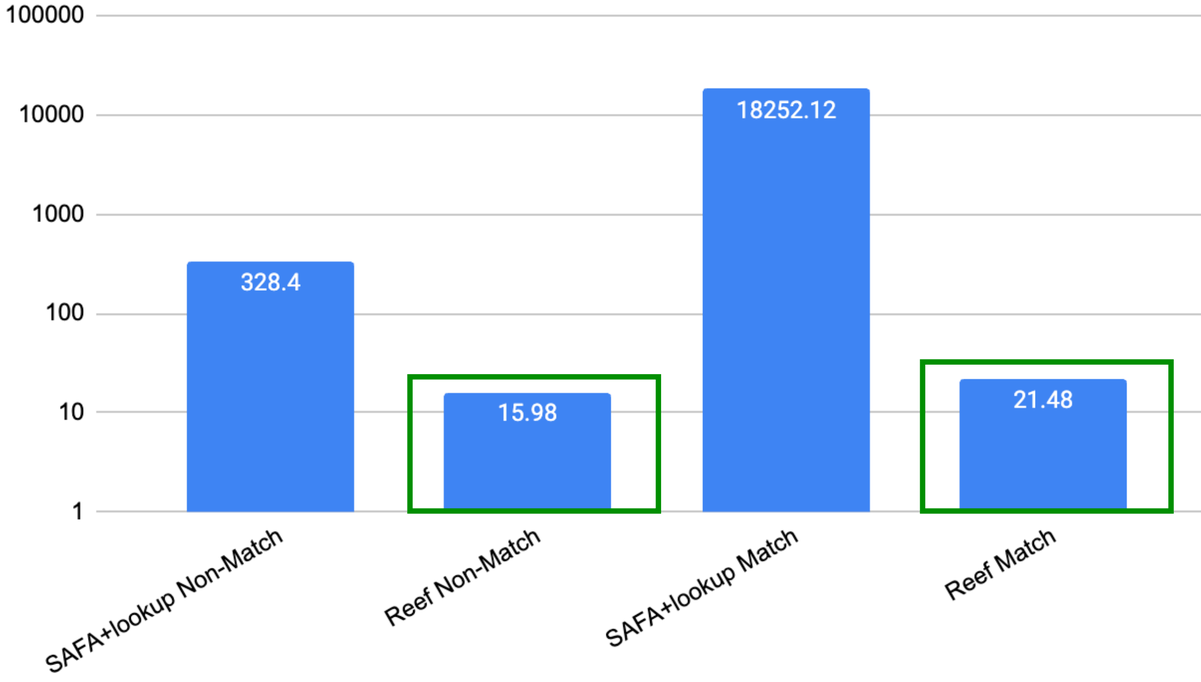
And It's More Efficient - Fewer Constraints

	DFA	DFA+Recursion	SAFA+lookup	Reef
Document Redaction (Small)	23,041,771	67,472	54,679	52,631
Document Redaction (Large)	X	141,712	57,628	54,636
ODoH	1,552,754	24,131	22,573	18,437
Password Match/ Non-Match	X	X	21,002/21,721	19,982/20,725
DNA Match/Non-Match	X	X	96,296/107,184	85,352/95,916

And It's More Efficient - Faster Total Prover Time



Document Redactions (Small and Large)



DNA Non-Match/Match

Even without Reef's additional optimizations, SAFA+lookup is orders of magnitude faster

Future Work

- Extending Reef to Context Free Grammars
    - JSON Validation
- Malware detection via YARA rules
    - Static Analysis = Regex + Propositional Logic
    - How to scale to checking hundreds of regexs at once?
- Zero Knowledge Proof of Compilation
    - Use Reef for parsing phase

Summary

Thank you!

Contact: ecmargo@seas.upenn.edu

We want…

- Succinct Zero Knowledge Proofs
- For a variety of regexs
- That scale well for large documents

We can…

- Use SAFA to get better expressivity
- And skip irrelevant parts of the document
- Use lookup tables for document commitment and SAFA transitions

Reef!

- Support for a variety of applications
- Fast Prover and Verifier times
- Fewer constraints