Fractal

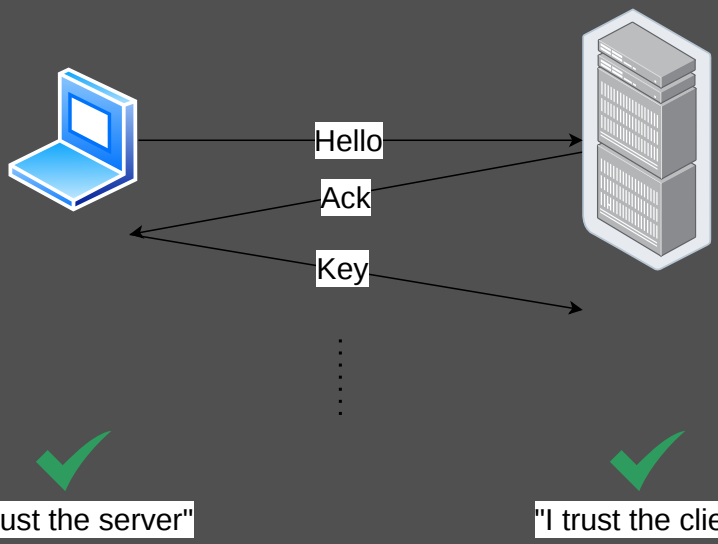
Certified Learning and Compilation of Finite State Automata

Charles Averill, Temitayo Ojo, Ben Kallus, William Doan, Kevin Hamlen

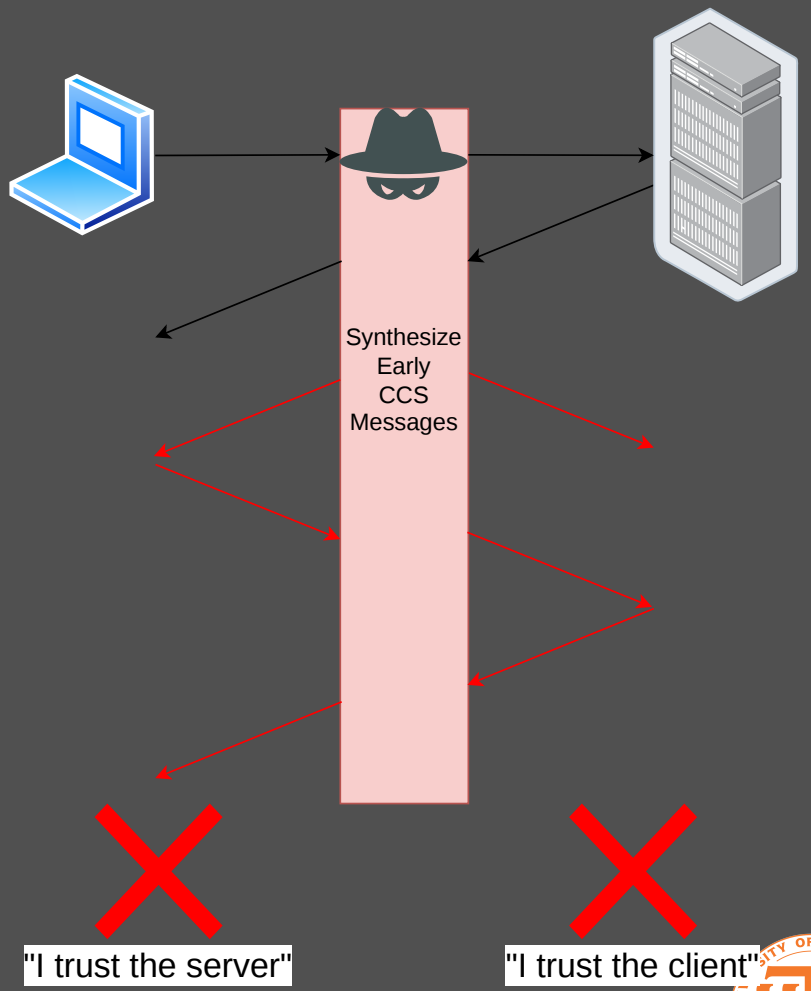
The University of Texas at Dallas
Dartmouth College

September 2026

TLS EarlyCCS Vulnerability

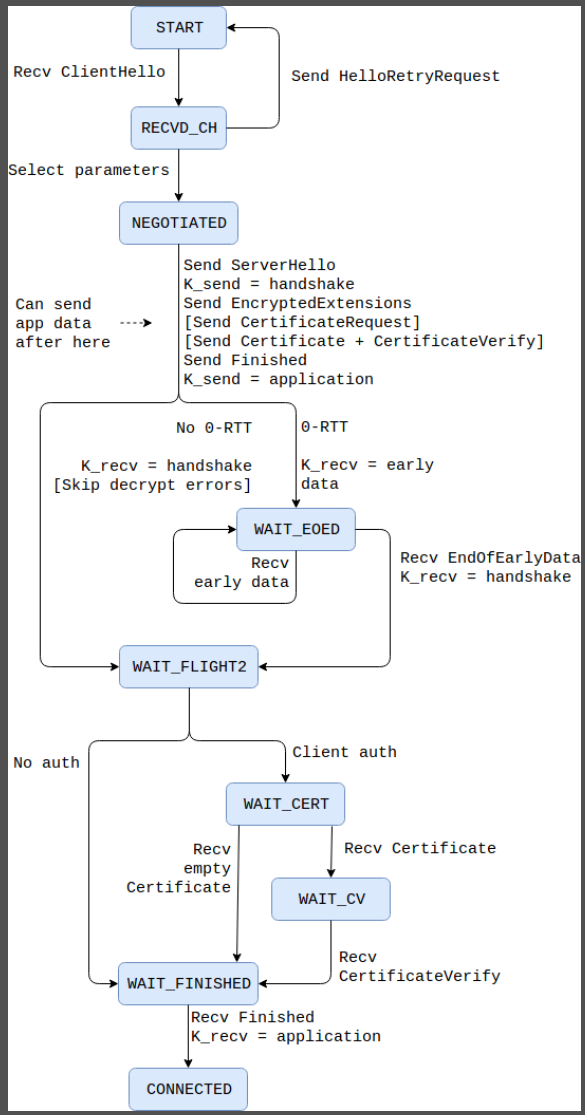


2-year vulnerability window for complete traffic decryption attacks.



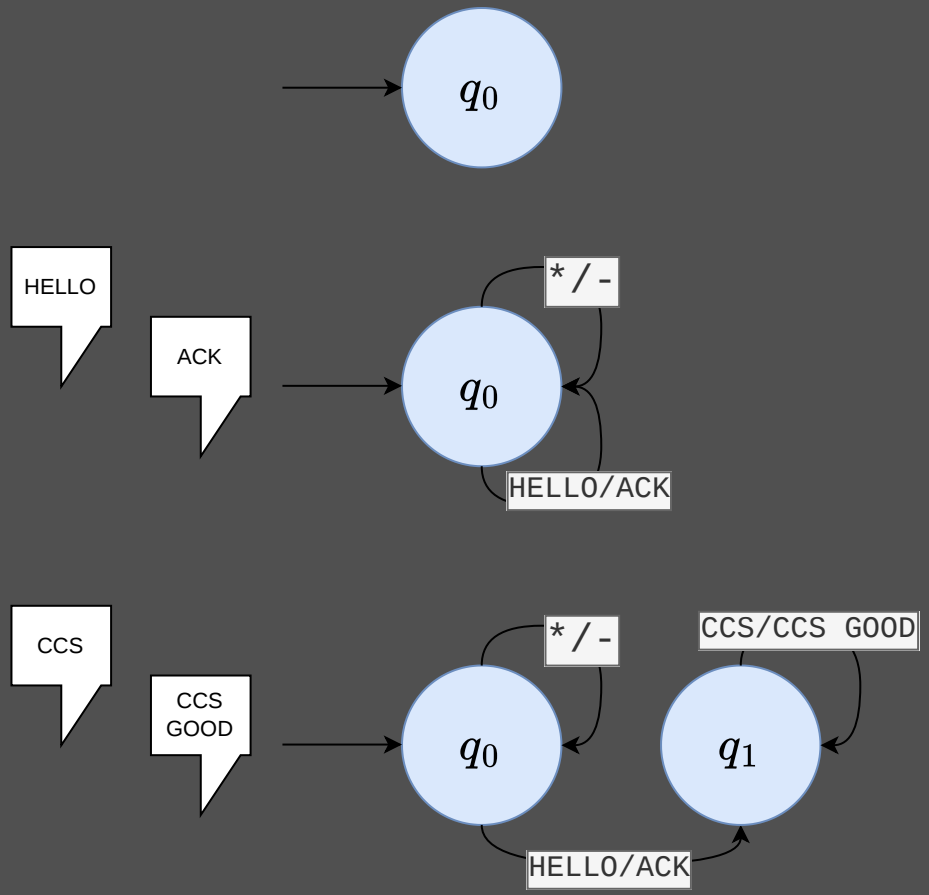
State Machines in Critical Systems

- **Finite State Machines** (FSMs) are a part of critical networking protocols, ICS, and hardware synthesis
- Very easy to implement and analyze state machines because of their limitations
- Because state machines are used in critical systems, we need to **verify** their **implementations** are accurate to **specifications**



Active Automata Learning

- **Learner** constructs state machines by asking the **Teacher** for **counterexamples**
- **Counterexample analysis** yields new states, transitions
- Iteratively improve system approximation until it is **relatively complete**
- Algorithms exist for DFAs, NFAs, Moore and Mealy machines, etc.

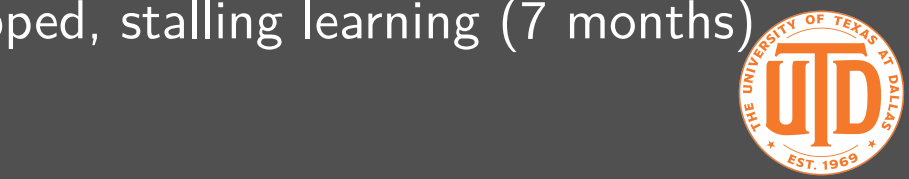


SOTA AAL Libraries

- **LearnLib** (Java, TU Dortmund)
  - L*, KV, TTT, NL*, and others for DFA, Mealy, and Moore machines
  - Used for state machine inference of TLS, SSH, TCP, and DTLS implementations, EMV bank cards, and industrial printer controllers
- **AALpy** (Python, TU Graz)
  - Active and passive learners for deterministic, non-deterministic, and stochastic automata
  - Used for Bluetooth LE and MQTT learning and device fingerprinting
- **LibALF** (C++, RWTH Aachen)
  - Active and passive learners (Angluin L*, KV, NL*, RPNI, Biermann)
  - Used for learning-based verification, e.g., inferring regular invariants

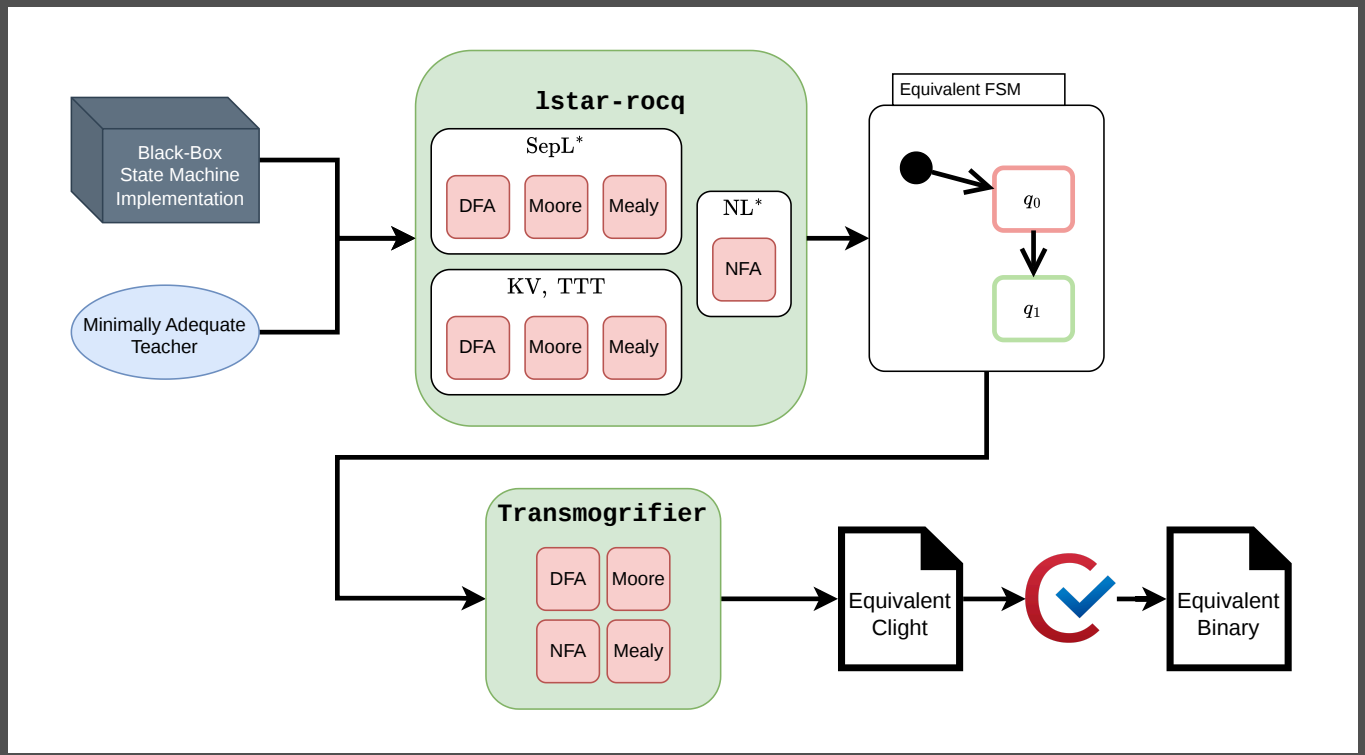
SOTA - Can you trust it?

- **LearnLib**
  - NL*: learned models contain spurious states (5.8 years)
  - TTT: counterexample handling deviates from the algorithm, stalling refinement (1.9 years)
- **AALpy**
  - KV: counterexamples analyzed from the wrong system state, corrupting refinement (2.9 years)
  - RPNI: incompatible states merged, producing wrong models (2.0 years)
- **LibALF**
  - KV: hypotheses built with out-of-range states and misplaced accepting states (7 months)
  - KV: rejecting states left without an output, yielding malformed hypotheses (8 months)
  - L*: counterexamples silently dropped, stalling learning (7 months)



FRACTAL[1] - Certified AAL Implementations

- **First formally-verified** implementations of AAL algorithms
- Includes 4 algorithms supporting 4 FSM classes
- **Guaranteed** to return **correct**, **minimal** FSMs given a Teacher
- Extracts to performant **OCaml**



[1] **F**ramework in **R**ocq for **ACT**ive **A**utomata **L**earning

Challenges in Certified AAL

- AAL algorithms are usually given in an **imperative** style, which is usually much more difficult to represent and verify than a **functional**-style program
- Imperative programs require **termination arguments**, which are occasionally underspecified in AAL papers
- Extracted OCaml code is often **not performant** by default

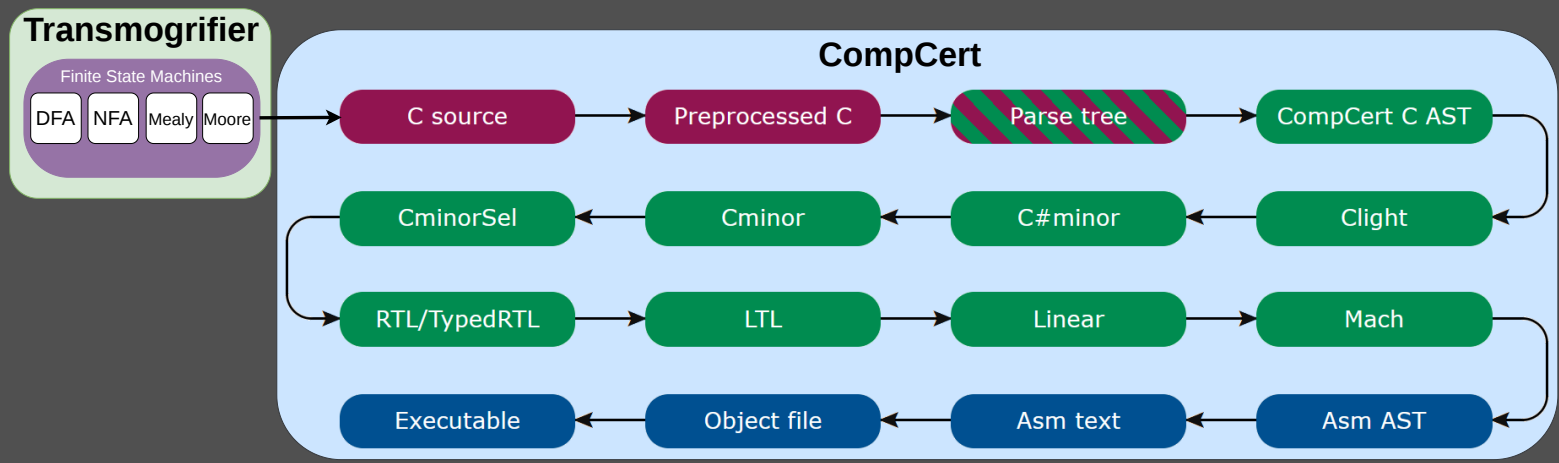
So the goal is: implement and verify **imperative-style** algorithms consisting of **complex termination arguments** that we can extract to **performant native code**. Sounds like a lot of work!

A lot of work - Learning

- Built in the **Rocq** theorem prover environment
- Uses a common **fuel pattern** to represent imperative programs as functional, cleanly folding into explicit **termination arguments**
- Supports L^*, TTT, KV, and NL^* for DFAs, NFAs, Moore machines, and Mealy machines
- Implementations are optimized to approach the performance of equivalent C code
- Can attain even better performance by combining with...

A lot more work - Compilation

- Extracted OCaml FSM code is pretty fast... but machine code compiled from C is usually a lot faster
- **Transmogrifier** is our **formally-verified FSM compiler**
- Converts FSMs into **Clight** code, a subset of C for the **CompCert** certified C compiler
- Guaranteed to generate functionally-equivalent machine code



Transmogrifier Results

```
const state_t  table[|Q| · |Σ|] = {(δ(q_i, Σ_j))_(|Q|,|Σ|)};
const output_t    atable[|Q|] = {(q_i ∈ F)_|Q|};
const state_t             q0 = q_0;

state_t delta(state_t q, symbol_t a) { // δ(q, a)
  if (q < |Q| & a < |Σ|)
    return table[q * |Σ| + a];
  else
    return |Q|; // Invalid state index
}

output_t accept(state_t q) { // q ∈? F
  if (q < |Q|)
    return atable[q];
  else return 2; // Neither true nor false
}

state_t run(word_t w, size_t len) { // δ*(q_0, w)
  size_t i = 0; state_t q = q0;
  while (i < len) {
    q = delta(q, w[i++]);
  }
  return q;
}
```

Figure 4. Shape of generated Clight code for DFAs

Table 3. Comparison of FSMs in OCaml (OC) and CompCert-compiled C on strings of length 10^6 averaged over 100 runs. Memory usage is the maximum Resident Set Size throughout all runs. Improvement (Imp.) is given by $\frac{\text{OCaml}}{\text{C}}$.

	Runtime (ms)			Mem. Usage (MB)		
Test	**OC**	**C**	**Imp.**	**OC**	**C**	**Imp.**
dfa.alternating	42.9	2.8	15.2×	30.7	9.5	3.23×
dfa.mod3	42.7	2.8	15.3×	30.8	9.5	3.24×
dfa.div7	134.7	3.1	44.0×	31.1	9.3	3.34×
nfa.suffix	191.0	2.8	68.4×	43.8	9.3	4.71×
moore.traffic	41.2	2.5	16.4×	30.6	9.4	3.26×
mealy.vending	301.6	3.0	100.6×	109.4	17.2	6.36×

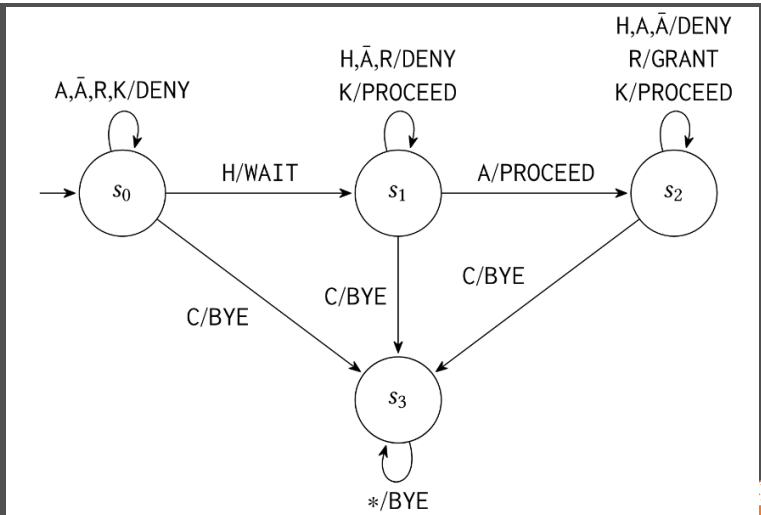
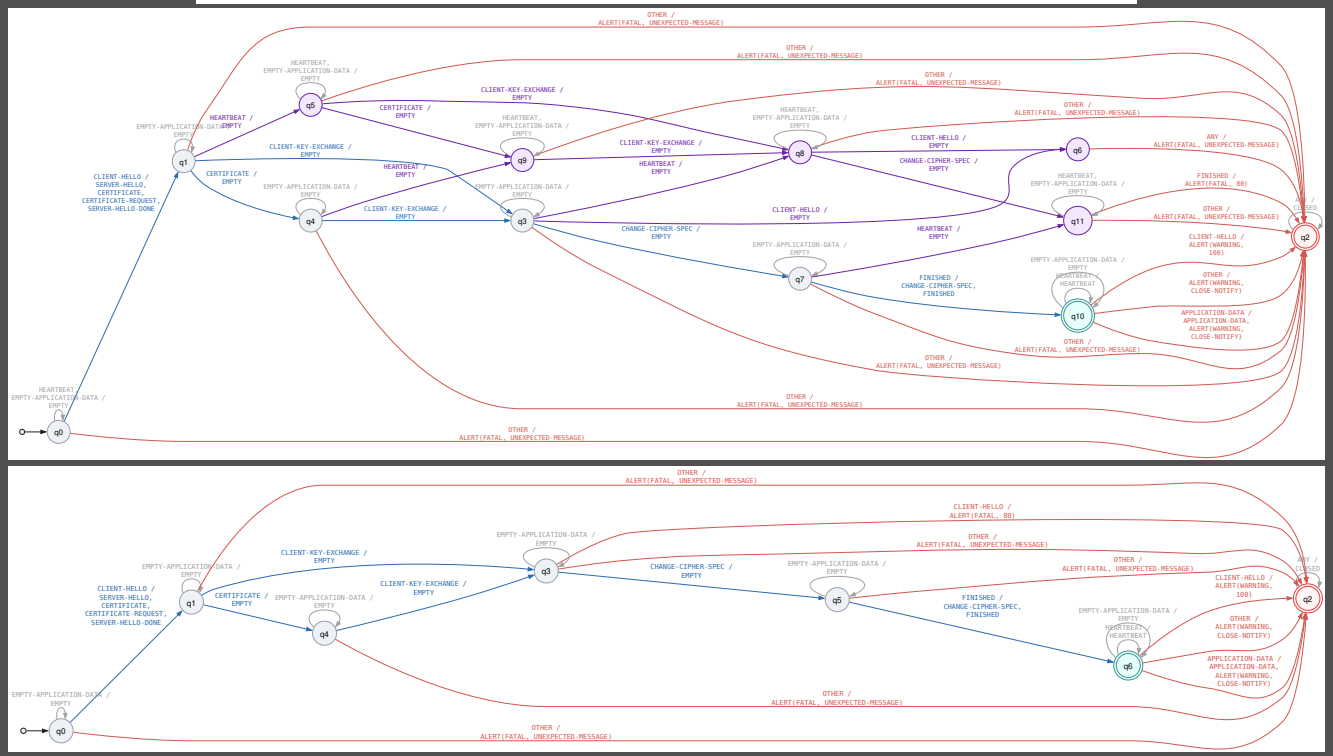


Figure 5. The learned session handshake machine. s_0 is initial, s_1 is reached once HELLO is seen, s_2 once AUTH succeeds, and s_3 absorbs after CLOSE. Edges are labelled *input*/*output*.

FRACTAL Results

Table 2. Membership query comparisons between FRACTAL and LearnLib (LL) on the LearnLib integration test suite with query caching

Algorithm	FRACTAL	LL	Overhead
L*	6760	6018	12%
KV	2681	2335	15%
TTT	3805	3494	9%
NL*	268	258	4%



Future work

- **More support** - **Buchi automata** via coinduction, register automata, more algorithm variants, etc.
- **Optimization** - reduce the query complexity gap between FRACTAL and LearnLib
- **Discovery** - apply FRACTAL to more real-world systems to discover state machine vulnerabilities

Table 2. Membership query comparisons between FRACTAL and LearnLib (LL) on the LearnLib integration test suite with query caching

Algorithm	FRACTAL	LL	Overhead
L*	6760	6018	12%
KV	2681	2335	15%
TTT	3805	3494	9%
NL*	268	258	4%

Table 3. Comparison of FSMs in OCaml (OC) and CompCert-compiled C on strings of length 10^6 averaged over 100 runs. Memory usage is the maximum Resident Set Size throughout all runs. Improvement (Imp.) is given by $\frac{\text{OCaml}}{\text{C}}$.

	Runtime (ms)			Mem. Usage (MB)		
Test	OC	C	Imp.	OC	C	Imp.
dfa.alternating	42.9	2.8	15.2×	30.7	9.5	3.23×
dfa.mod3	42.7	2.8	15.3×	30.8	9.5	3.24×
dfa.div7	134.7	3.1	44.0×	31.1	9.3	3.34×
nfa.suffix	191.0	2.8	68.4×	43.8	9.3	4.71×
moore.traffic	41.2	2.5	16.4×	30.6	9.4	3.26×
mealy.vending	301.6	3.0	100.6×	109.4	17.2	6.36×