

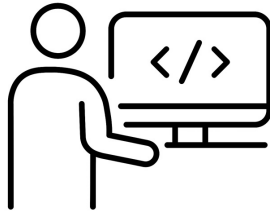
C2FSM: A Tool for Active Learning of Extended Finite State Machines from C Programs

Yoel Kim and Yunja Choi

Motivation

- A precise understanding of **software behavior** is crucial
 - Especially in **safety-critical** domains (e.g., automobiles, medical devices, avionics)
- **Formal behavioral models** provide a high-level, structured view of behaviors
 - They can help developers reason about software more systematically
 - However, constructing such models manually is **time-consuming**

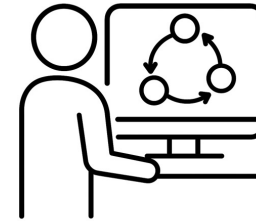
“How can I inspect
thousand lines of code?”



Source Code

VS

“Ok, it’s understandable,
but how can I construct it?”



Behavioral Model

Automated Model Inference from Source Code

Learner

Execution trace

speed	pedal	gear
Stop	T	1
Low	T	2
High	F	3
Low	F	2
Stop	...	...

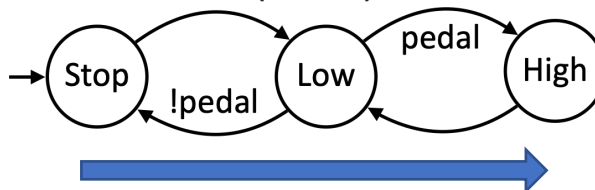
Transition predicate synthesis

```
next(Low, T, 2) = High
next(Low, F, 2) = Stop
```

Missing behavior

speed	pedal	gear
High	T	3
High	...	...

Extended Finite State Machine (EFSM)



Teacher

Source code

```
assume(Init)
while true do
  step(speed, pedal, gear)
end while
```

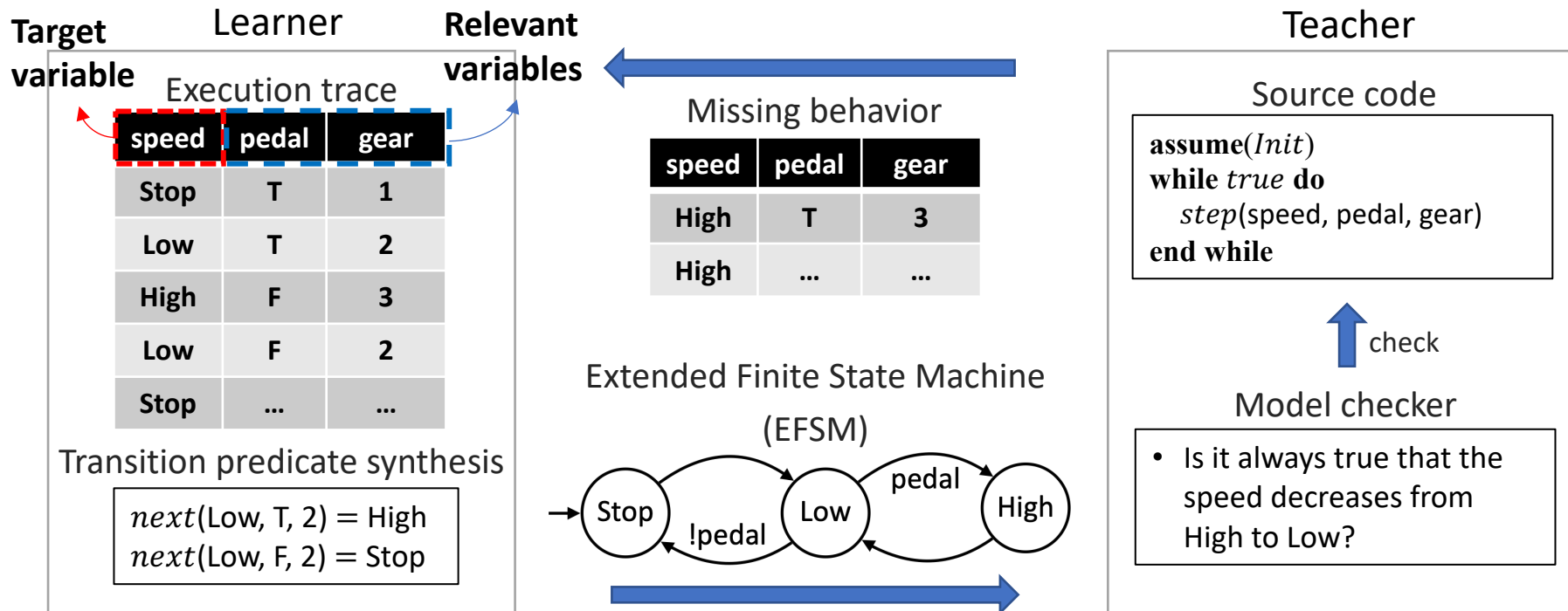


Model checker

- Is it always true that the speed decreases from High to Low?

[*] N. Y. Jeppu, T. Melham, and D. Kroening, "Enhancing active model learning with equivalence checking using simulation relations," Formal Methods in System Design, 2022

Automated Model Inference from Source Code



[*] N. Y. Jeppu, T. Melham, and D. Kroening, "Enhancing active model learning with equivalence checking using simulation relations," Formal Methods in System Design, 2022

Practical Limitations in SOTA Technique

✗ Limited Application Structure:

- Supports only programs with a single loop that invokes a *step* function
- *step* serves both as the **learning scope** and the atomic transition unit

```
assume(Init)
while true do
  step(X)
end while
```

✗ Manual Relevant Variable Instrumentation:

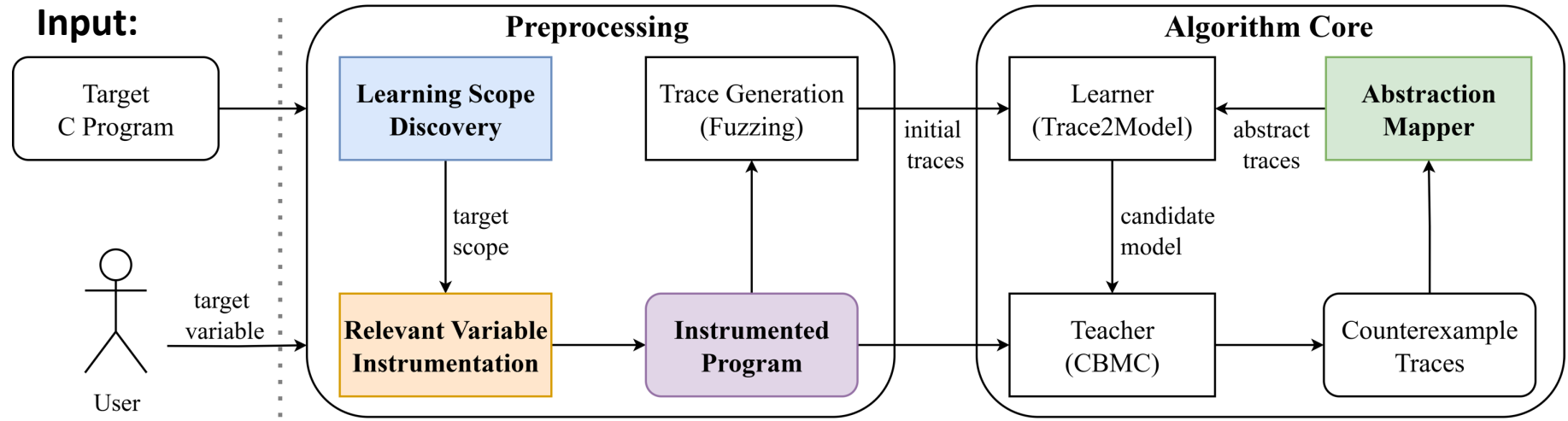
- Requires manual instrumentation of variables to capture execution traces
- **Trace conflict:** The same valuation yields different next values of the target var.

```
next(Low, 2) = High?
next(Low, 2) = Stop?
```

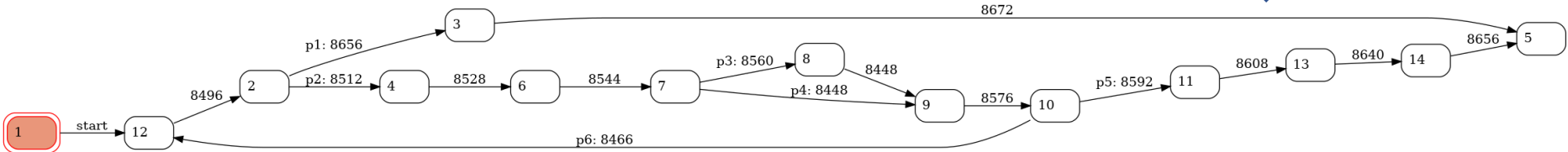
✗ Scalability Bottleneck:

- Identifying missing behaviors in the learned model remains a major bottleneck
- This issue gets worse when the number of possible valuations is large

C2FSM Workflow



Output: complete Extended Finite State Machine (EFSM)



Learning Scope (*step*) Discovery

- Assumption: The target program is a **reactive system**
 - It continuously interacts with external events and updates the target variable t within loops
 - Which loop should be selected as the **learning scope**?
- Intuition 1: **Last Encountered Loop**
 - In reactive system, early loops typically handle setup
 - Later loops are more likely to implement the main control logic
- Intuition 2: **Outermost Loop**
 - The outermost loop captures the highest-level behavior
 - Nested inner loops can be interpreted as hidden states

```
Loop1:  
  update1( $t$ )  
end Loop1  
  
Loop2 (step):  
  Loop3:  
    update3( $t$ )  
  end Loop3  
  update2( $t$ )  
end Loop2
```

Relevant Variable Instrumentation

- **In-scope variables:** variables that are visible inside *step*, but are not declared within it
- **Relevant variables:** In-scope variables that affect any assignment to the target variable *t*

```
step():  
  a = x;  
  b = a + 1;  
  t = b - 2;
```

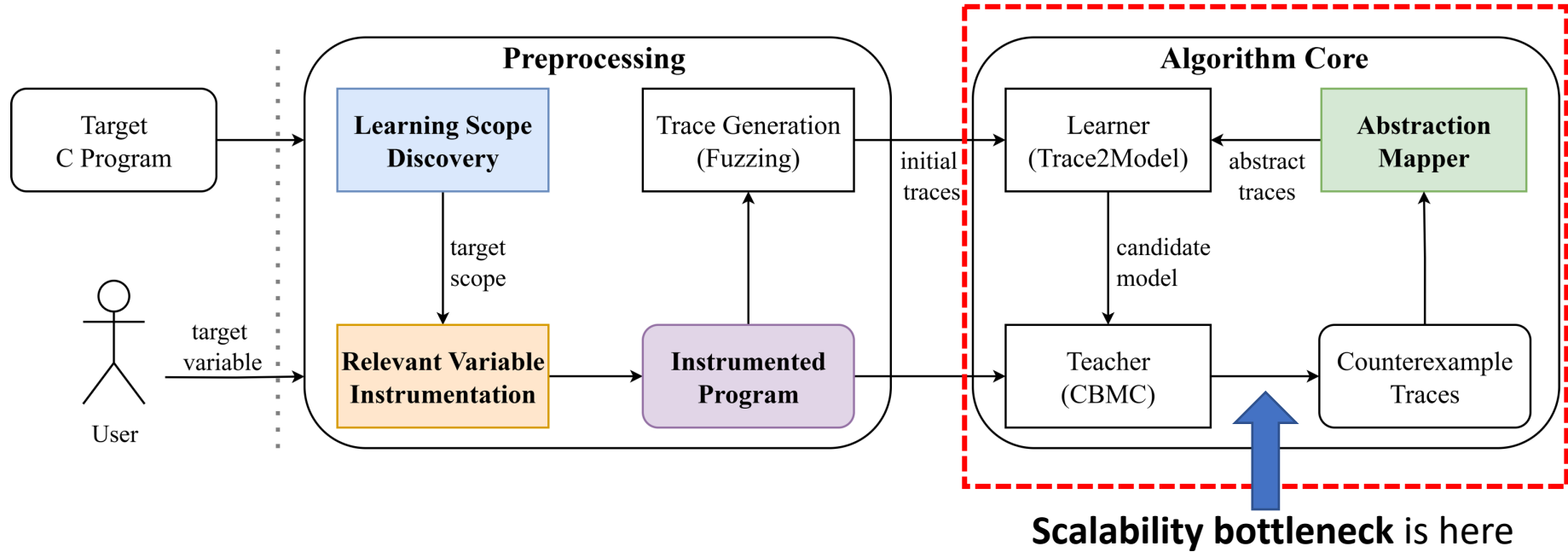
Data dependency

```
step():  
  if (a > 0)  
    if (b < 0)  
      t = t + 1;
```

Control dependency

- Data dep.: It **suffices** to instrument **only** the variables used in the assignments to *t*
- Control dep.: **All** variables used in control-dependent statements **must** be instrumented

C2FSM Workflow



Solution: Abstraction Mapper

- C2FSM abstracts concrete valuations using **Boolean predicate abstraction**

Trace group for $t = 0$

Concrete state	Abstract state	Next value
<1, 2, 3, 4>	<T, F>	0
<2, 1, 5, 6>	<T, T>	1
<3, 1, 7, 8>	<F, T>	1
<4, 3, 9, 10>	<F, F>	1

Relevant variables: <a, b, c, d>

Boolean variables (BVs): <(a ≤ 2), (b == 1)>

- These BVs can dramatically reduce the search space during teaching (10^4 vs 2^2 valuations)

Boolean Predicate Synthesis

- Initially, atomic propositions in the transition predicates are treated as BVs
- BVs are refined by resolving **abstract trace conflicts**:

Abstract trace conflict group for $\langle T, T \rangle$

Concrete state	Abstract state	Next value	
$\langle 2, 1, 5, 6 \rangle$	$\langle T, T \rangle$	1	
$\langle 1, 1, 1, 1 \rangle$	$\langle T, T \rangle$	0	← new missing behavior

- A distinguishing predicate ($a == b$) can be synthesized and treated as a new BV

$\langle 2, 1, 5, 6 \rangle$	$\langle T, T, F \rangle$	1
$\langle 1, 1, 1, 1 \rangle$	$\langle T, T, T \rangle$	0

[*] S. Padhi, R. Sharma, and T. Millstein, “Data-driven precondition inference with learned features,” PLDI 2016.

Experiments on Simulink Benchmarks

Metrics	C2FSM	C2FSM without mapper
Total iterations	145	130
Learning time (s)	287	230
Teaching time (s)	14,292	20,440
Total time (L + T)	14,579	20,670
# of complete EFSMs	43 / 45	41 / 45

- With abstraction mapper:
 1. Successfully learned 2 additional complete EFSMs (within 1-hour per benchmark)
 2. Reduced the average teaching time per iteration from 157.2s to 98.5s

Experiments on SV-COMP Benchmarks

Category	$ P $	LoC	# Vars	# Inst	# Succ	# Iter	Learn (s)	Teach (s)	Total (s)	# States	# Trans	# BVs
ntdrivers-simplified	4	1,240	199.8	134.0	1	30.0	72	2,706	2,779	5.8	14.0	83.0
product-lines	3	2,340	42.0	12.0	3	2.7	26	196	223	6.0	14.3	11.7
systemc	2	669	26.5	12.5	2	3.0	3	63	66	4.0	5.5	1.5
seq-mthreaded	6	468	34.8	6.7	6	5.8	6	28	34	4.2	5.5	6.0
eca-rers2012	4	1,145	20.5	18.0	4	7.8	10	1,742	1,751	3.5	6.8	19.5
psyco	4	417	33.0	26.3	3	13.5	366	944	1,310	8.0	25.3	46.3
openssl-simplified	2	602	71.0	28.5	1	164.0	636	1,317	1,952	15.0	18.0	21.0
Total / Avg.	25	943.0	61.7	34.8	20	582	3,182	25,088	28,270	6.0	12.3	28.4

- C2FSM successfully learned **20 complete EFSMs** out of 25 benchmarks, each within a 1-hour time limit
- Without the abstraction mapper, only **9 complete EFSMs** were learned

Conclusion

- We presented C2FSM, the first fully automated tool for active learning of complete EFSMs from reactive C programs
- **Limitations:**
 1. Relevant variable instrumentation in C2FSM may still cause trace conflicts
 2. Limited support for target/relevant variables of pointer or floating-point types
 3. Scalability remains a challenge, even with abstraction mapping
- **Practical Value:**
 1. Enables understanding of program behavior without manual code inspection
 2. Supports formal verification through specification generation and model checking
 3. Provides a benchmark for evaluating new active learning algorithms
- **Demonstration:** <https://youtu.be/W10qblxkHNk>

Toward More Complicated Structures

- Since C2FSM traces relevant variables **outside** *step*, it may miss some internal values **inside** *step*

```
step():  
  a = input(); // def  
  t = a + 1; // use  
  a = x; // def
```

Def-Use-Def pattern

```
step():  
  for (i = 0; i < n; i++)  
    if (input()) t++;  
  t = t + 1;
```

Nested reactive loop

- Candidate solution: Introduce a **temporary variable** to preserve internal values

```
step():  
  a = input(); // def  
  a_0 = a;  
  t = a_0 + 1; // use  
  a = x; // def
```

```
step():  
  for (i = 0; i < n; i++)  
    if (input()) t++;  
    t_0 = t;  
  t = t_0 + 1;
```