Extracting a Micro State Transition Table Using KLEE

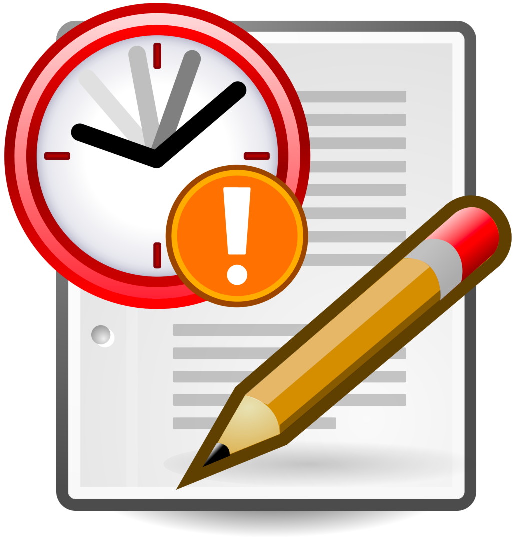
Norihiro Yoshida, Takahiro Shimizu,
Ryota Yamamoto and Hiroaki Takada

NAGOYA UNIVERSITY

Legacy code in embedded system development

Specification document is often outdated.

- ad-hoc code modification when deadline is approaching
- E.g. frequent modifications to condition branches cause the change of the specifications.



https://en.wikipedia.org/wiki/File:Ambox_outdated_serious.svg

Reuse in embedded system development

Practitioners need to address hardware variations.

They frequently reuse source code from similar hardware products.

⬇

Practitioners want to reuse code, but an outdated document is a barrier for them.

Request from industry

Practitioners need a tool for extracting a state transition table from C modules.

- Static analysis is desirable for them because it is sometimes hard to prepare a runtime environment.
- They can give a state variable.
  - Tools do not have to identify a state variable automatically.
- They use a Micro State Transition Table (MSTT).

Micro State Transition Table (MSTT)

They use a state transition table at module level.

States: a set of values of a user-specified state variable

state event	st = 1	st = 2	else

t = 1 & s < 10	s := s+1
	out := s
		s := s+1
		out := 0
		(t) st := 3
			s := s+1
			s := s+1
			(t) st := 1

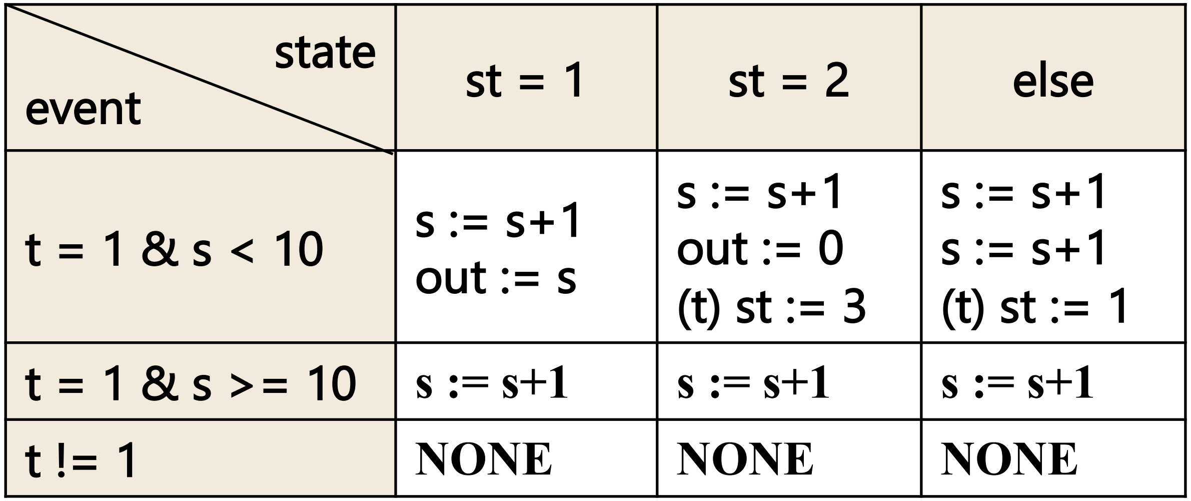
t = 1 & s >= 10	s := s+1	s := s+1	s := s+1
t != 1	NONE	NONE	NONE

Events: combinations of values of the other variables.

Why specification inference?

Extracting an MSTT manually from a module in C source code is unrealistic.

- Module includes complex condition branches.
- Human resources are limited.



state event	st = 1	st = 2	else

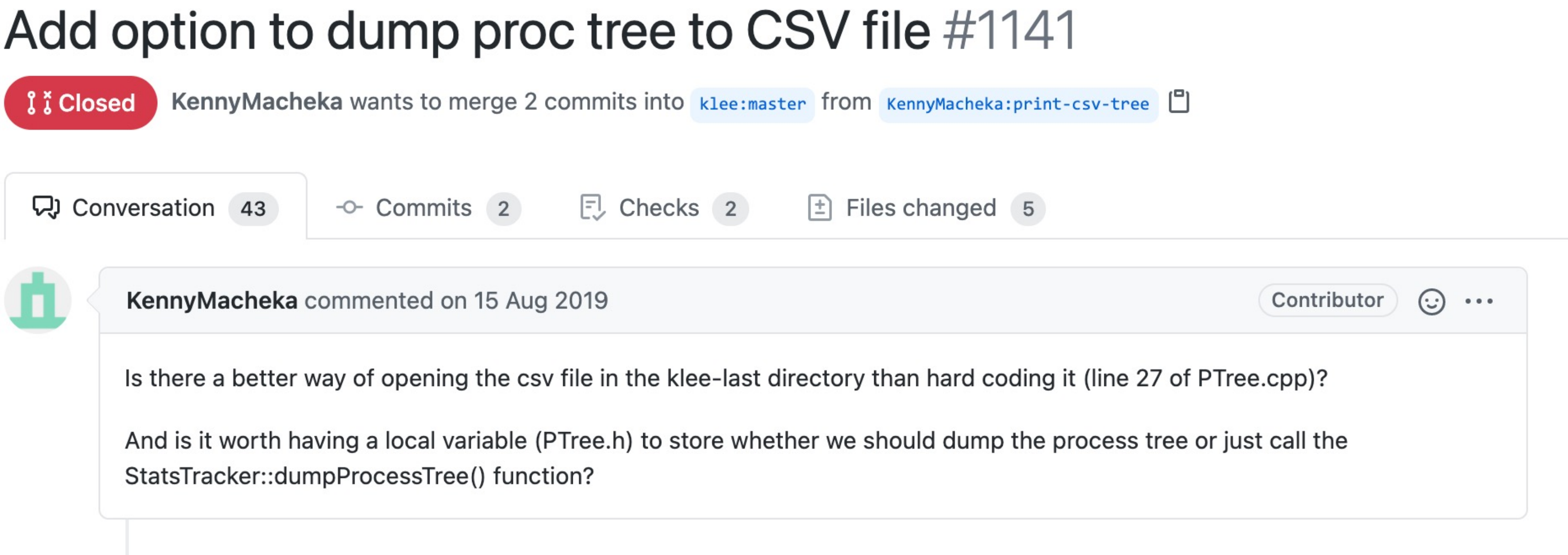
t = 1 & s < 10	s := s+1
	out := s
		s := s+1
		out := 0
		(t) st := 3
			s := s+1
			s := s+1
			(t) st := 1

t = 1 & s >= 10	s := s+1	s := s+1	s := s+1
t != 1	NONE	NONE	NONE

Extracting an MSTT using KLEE

- Generate a symbolic execution tree using KLEE
  - KLEE can analyzes directives, pointers and arrays correctly.

- Use the implementation from the pull request #1141 by KennyMacheka



Add option to dump proc tree to CSV file #1141

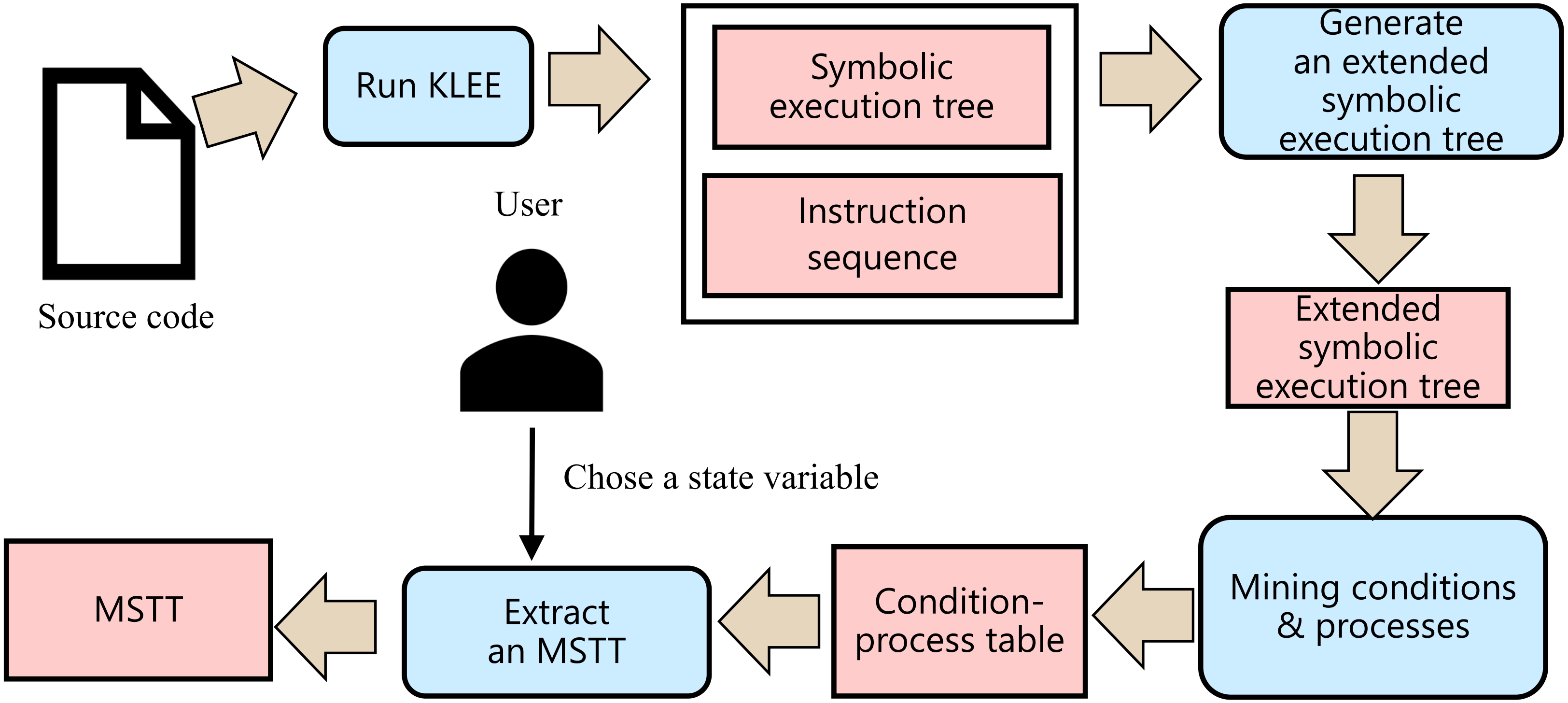
KennyMacheka wants to merge 2 commits into `klee:master` from `KennyMacheka:print-csv-tree`

KennyMacheka commented on 15 Aug 2019

Is there a better way of opening the csv file in the klee-last directory than hard coding it (line 27 of PTree.cpp)?

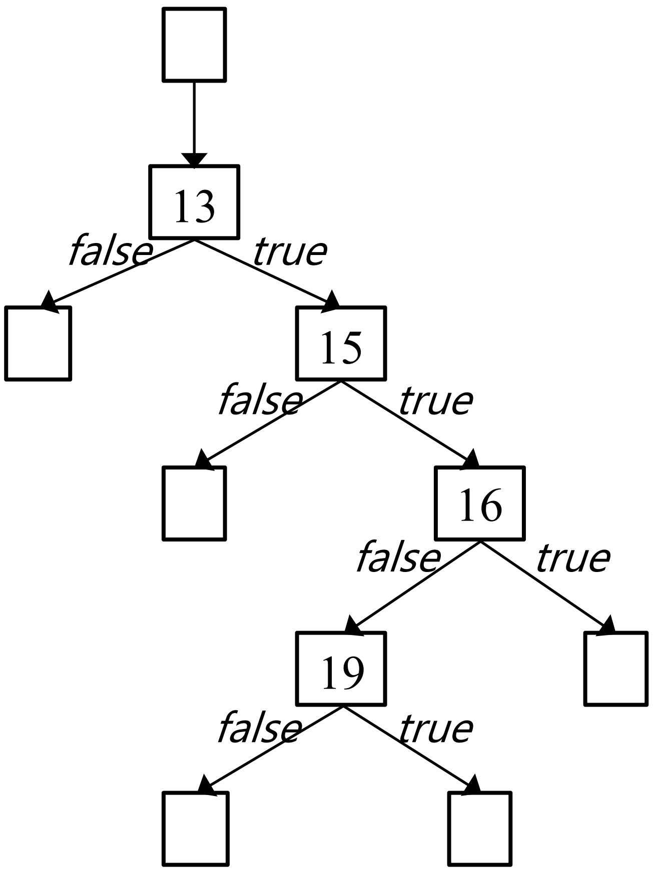
And is it worth having a local variable (PTree.h) to store whether we should dump the process tree or just call the StatsTracker::dumpProcessTree() function?

Overview of the proposed tool



STEP1: Dump a symbolic execution tree

- Dump a symbolic execution tree and the corresponding instruction sequence



Symbolic execution tree

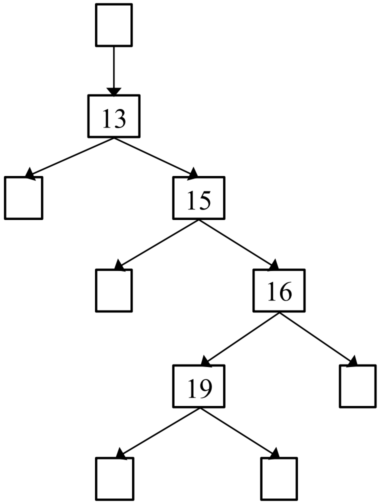
```
33  int main(){
34  task();
06  void task(){
07  int t,s;
08  klee_make_symbolic(&state,
    sizeof(state), "state");
09  klee_make_symbolic(&t,
    sizeof(t), "t");
10  klee_make_symbolic(&s,
    sizeof(s), "s");
13  if(t == ON){
31  }
35  return 0;
14  s++;
15  if(s < 10){
31  }
35  return 0;
16  if(state == 1){
    ……
```

depth-first sequence of symbolically executed instructions

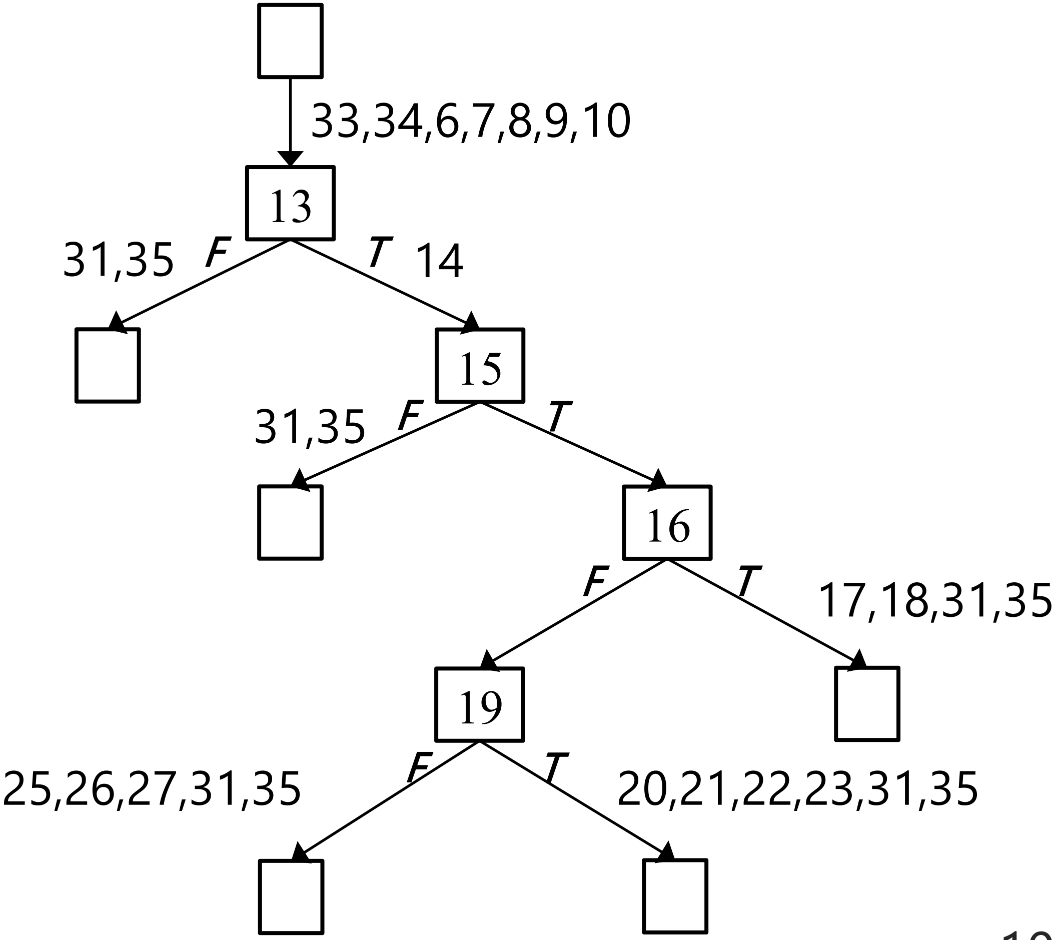
Instruction sequence

STEP2: Extend the symbolic execution tree

- Add the corresponding line number to each edge of the symbolic execution tree



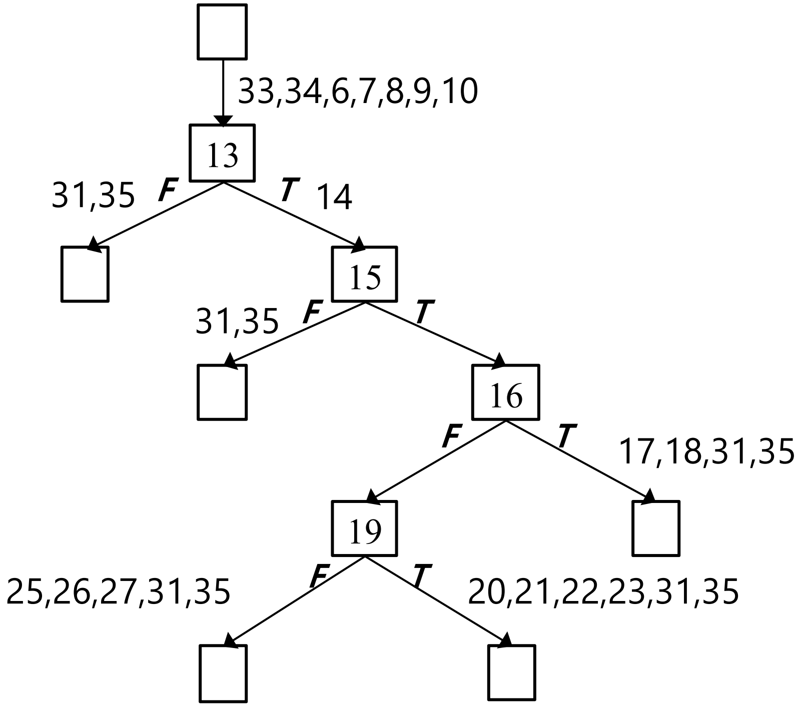
```
33  int main(){
34  task();
06  void task(){
07  int t,s;
08  klee_make_symbolic(&state,
    sizeof(state), "state");
09  klee_make_symbolic(&t,
    sizeof(t), "t");
10  klee_make_symbolic(&s,
    sizeof(s), "s");
13  if(t == ON){
31  }
35  return 0;
14  s++;
15  if(s < 10){
31  }
35  return 0;
16  if(state == 1){
    ……
```



Extended symbolic execution tree

STEP3: Generate a condition-process table

- Extracting pairs of a condition and the corresponding process



Extended symbolic execution tree

Condition-Process Table

condition	process
!(t == ON)	int t,s;

t == ON
& !(s < 10)
	int t,s;
	s++;

t == ON
& s < 10
& !(state == 1)
& !(state == 2)
	int t,s;
	s++;
	s++;
	state = 1;
	printf(“state changed¥n”);

STEP4: Extract an MSTT

- Extract an MSTT based on a user-specified state variable

Condition-Process Table

condition	process
!(t == ON)	int t,s;

t == ON
& !(s < 10)
	int t,s;
	s++;

t == ON
& s < 10
& !(state == 1)
& !(state == 2)
	int t,s;
	s++;
	s++;
	state = 1;
	printf(“stat

Extract the processes and the transitions

MSTT

	!(state == 1) & !(state == 2)	!(state == 1) & state == 2	state == 1
!(t == ON)	int t,s;	int t,s;	…

t == ON
& !(s < 10)
	int t,s;
	s++;
		int t,s;
		s++;
			…

t == ON
& s < 10
	int t,s;
	s++;
	s++;
	(t)state = 1;
	printf(“state changed¥n”);
		int t,s;
		s++;
		out=0;
		(t)state = 3;
		printf(“state changed¥n”);
			…

Summary

- We proposed a tool for extracting an MSTT from source code using KLEE.
  1. Dump a symbolic execution tree
  2. Extend the symbolic execution tree
  3. Generate a condition-process table
  4. Extract an MSTT

- Future Works
  - Lager-scale case study
  - Extraction of MSTTs with floating point numbers