SOFTWARE RELIABILITY GROUP

Imperial College London

Computing Summaries of String Loops in C for Better Testing and Refactoring

Timotej Kapus, Oren Ish-Shalom, Shachar Itzhaky,
Noam Rinetzky, Cristian Cadar

TEL AVIV UNIVERSITY אוניברסיטת תל אביב

הטכניון מכון טכנולוגי לישראל TECHNION Israel Institute of Technology

```c
char *p;
for (p = line; p && *p && whitespace (*p); p++)
  ;
```

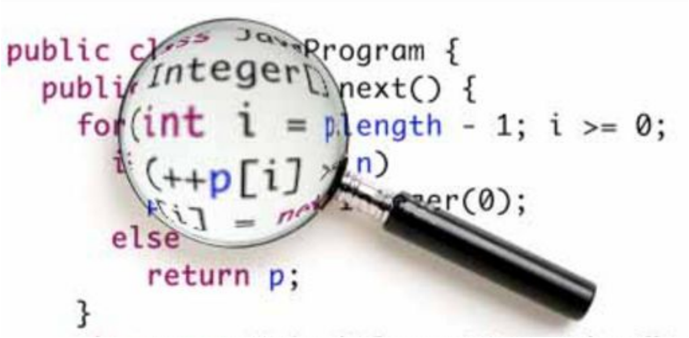
```c
char *p;
for (p = line; p && *p && whitespace (*p); p++)
  ;
```

This talk

```c
char *p = line + strspn(line, "␣\t")
```

Why?

- Give clarity to the meaning of loops



- Refactoring
- Program analysis
    - Symbolic execution
- Compiler optimisations

Motivation: Refactoring

```
char *p = path + strlen (path);
for (; *p != '/' && p != path; p--)
  ;
```

Motivation: Refactoring

```c
char *p = path + strlen (path);
for (; *p != '/' && p != path; p--)
  ;
```

summary

```c
char *p = strrchr(path, '/');
p = p == NULL ? path : p;
```

Motivation: Refactoring

- Real examples from `wget` and `bash`
- C code contains lots of loops replicating libc functions
  - Different handling of edge cases

Motivation: Program analysis

- Easier to reason about a known function than an arbitrary loop

  Example symbolic execution of `strchr`
  Two approaches:

  1. Unroll loop and gather constraints character by character
  2. Model it as `firstIndexOf` in theory of strings

Scope: Memoryless Loops

- Loops conforming to an interface:
  - Argument: single pointer to a buffer
  - Returns: pointer to an offset in the buffer
- Only reads the character under current pointer

```
char* loopSummary(char*);
```

- Need a vocabulary to express these loops

Remember?

```
char *p = line + strspn(line, "␣\t")
```

In our vocabulary

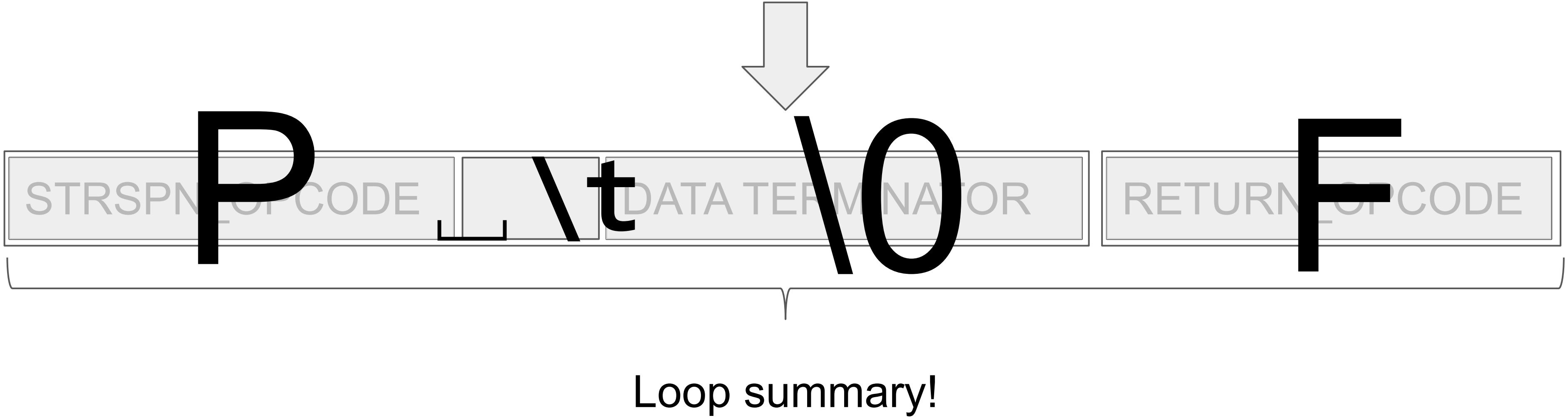
```
char *p = line + strspn(line, "␣\t")
```

STRSPN_OPCODE	␣\t	DATA TERMINATOR	RETURN_OPCODE

Loop summary!

We just used characters!

```c
char *p = line + strspn(line, "␣\t")
```



Vocabulary for expressing simple loops

- Vocabulary has meaning in an `interpreter()`
- `strspn(P)` and `return(F)`
- Adding a new vocabulary as simple as adding a new `case:`

```c
char* interpreter(char* input) {
  char *result = input;

  while(nextInstruction())
    switch(instruction)
      case 'P':
        result += strspn(result,
                   data(instruction));
      case 'F':
        return result;
}
```

Vocabulary for expressing simple loops

string.h functions

- `strspn`
- `strcspn`
- `memchr`
- `strchr`
- `strrchr`
- `strpbrk`

pointer manipulation

- `increment`
- `set to start`
- `set to end`

special

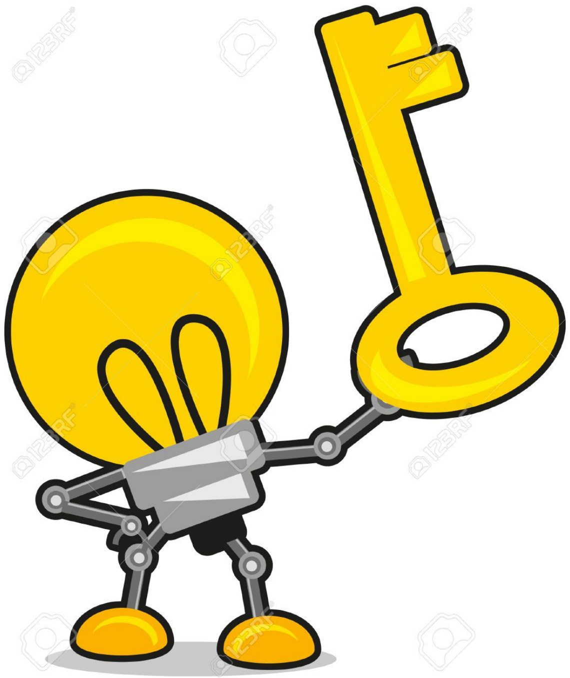
- `backward traverse`
- `return`

conditionals

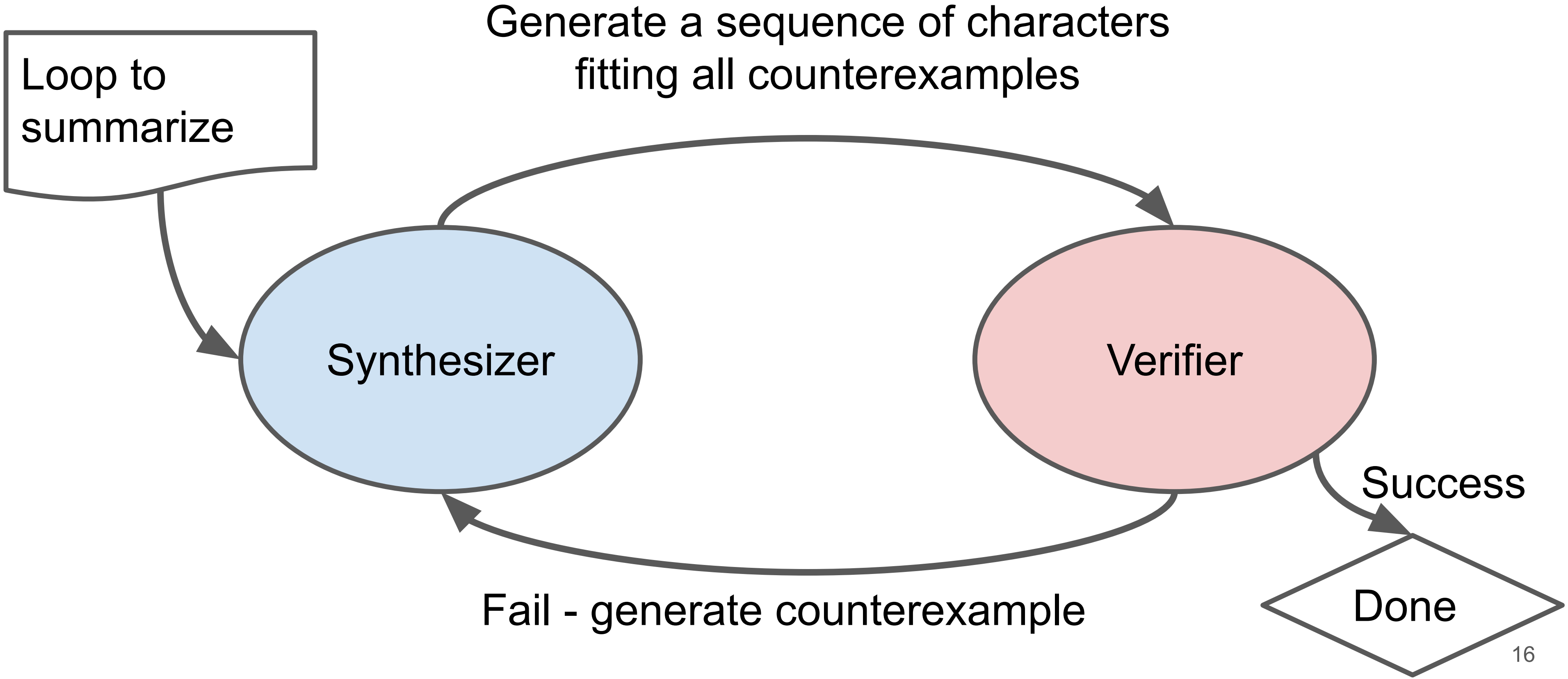
- `is null`
- `is start`

Loop Summarisation

Find sequences of characters that when executed by our interpreter have the same behaviour as the original loop



Counter-example guided synthesis



Synthesizer

- Symbolic execution
- Use a symbolic string (program)
- Constrain it to be equivalent on current counterexamples
- Ask an SMT solver for a solution

Verifier

- Symbolic execution
  - Bounded equivalence checking strings of length ≤ 3
- Loops in **our scope**
  - checking lengths ≤ 3 sufficient to show equivalence for any length (proof in the paper)

Synthesizer

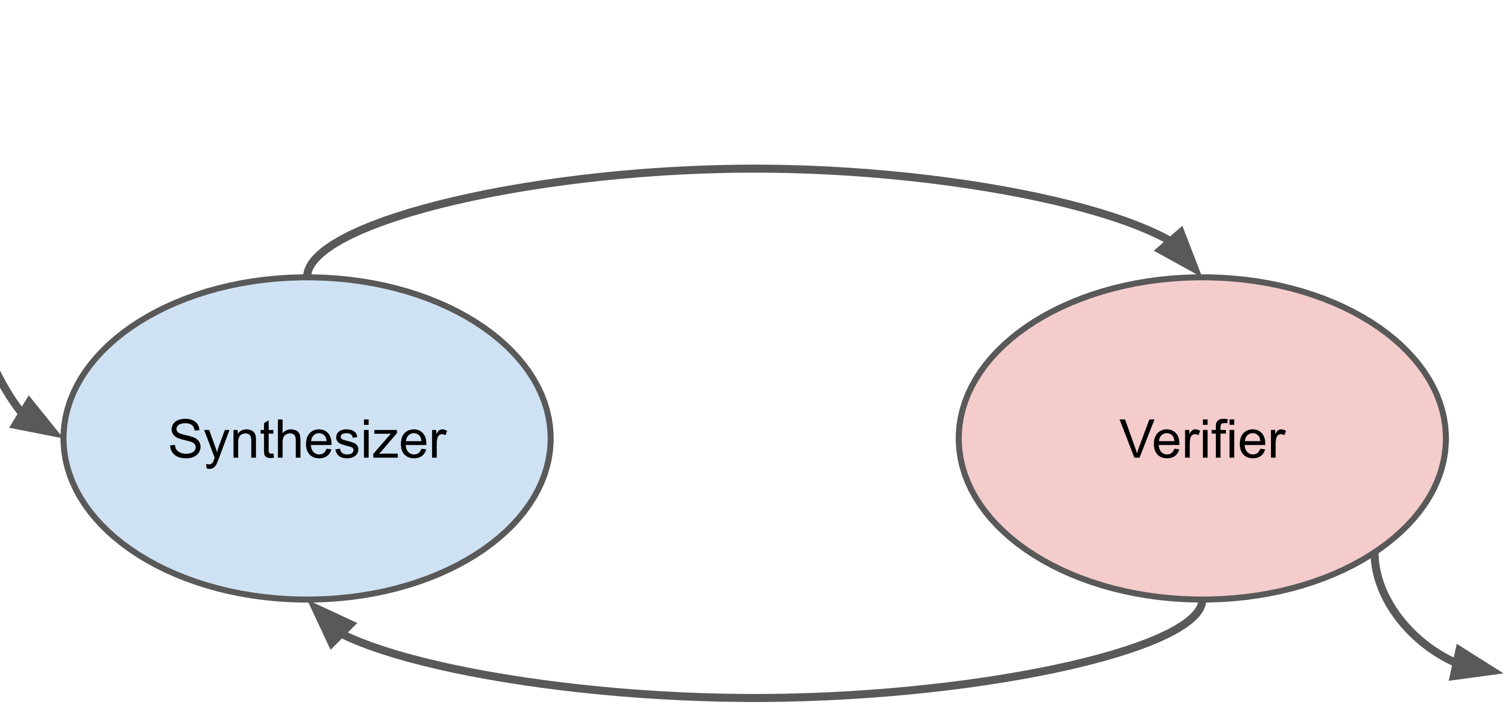
- Symbolic execution
- Use a symbolic string (program)
- Constrain it to be equivalent on current counterexamples
- Ask an SMT solver for a solution

Verifier

- Symbolic execution
  - Bounded equivalence checking strings of length ≤ 3
- Loops in **our scope**
  - checking lengths ≤ 3 sufficient to show equivalence for any length (proof in the paper)

Single run of symbolic execution

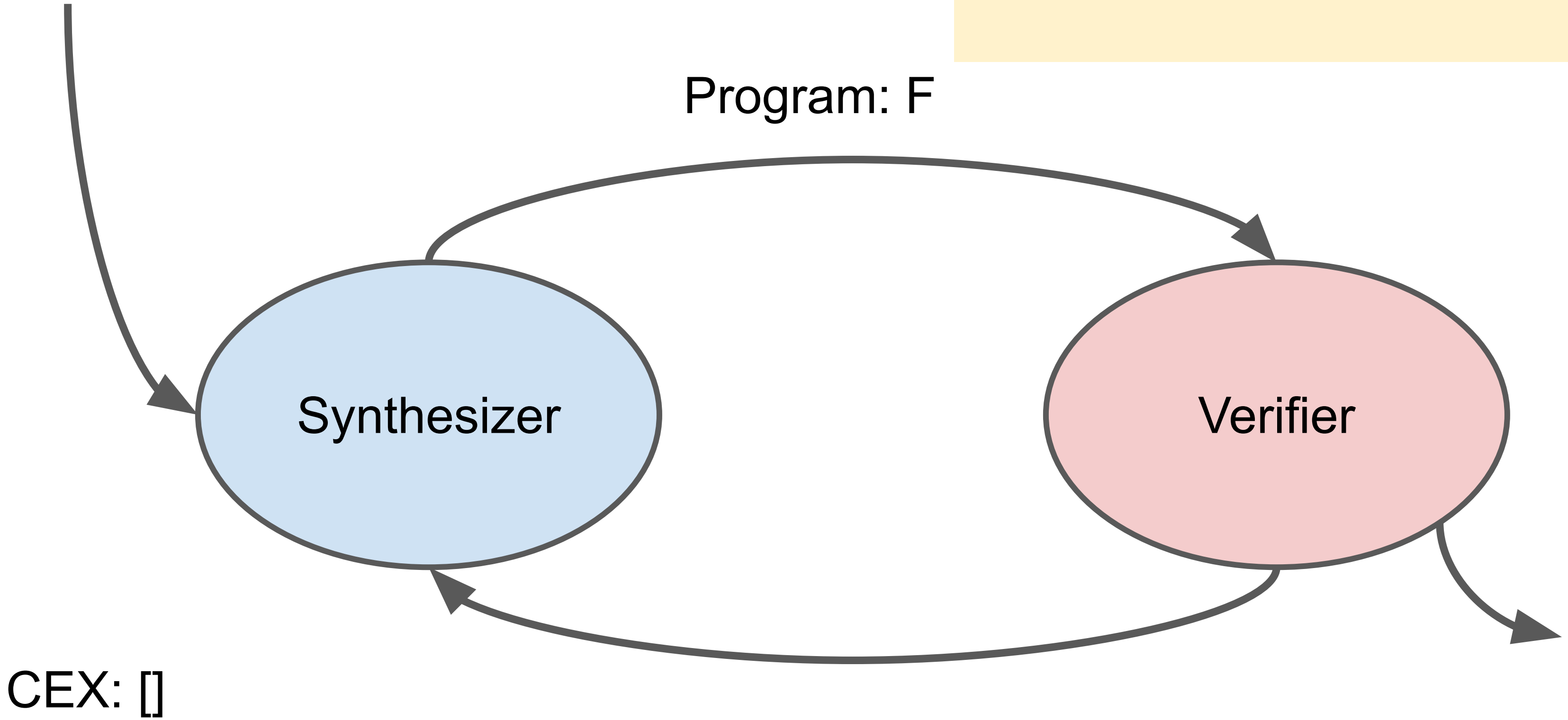
```
char *p;
for (p = line;
     p && *p && whitespace (*p);
     p++)
  ;
```



CEX: []

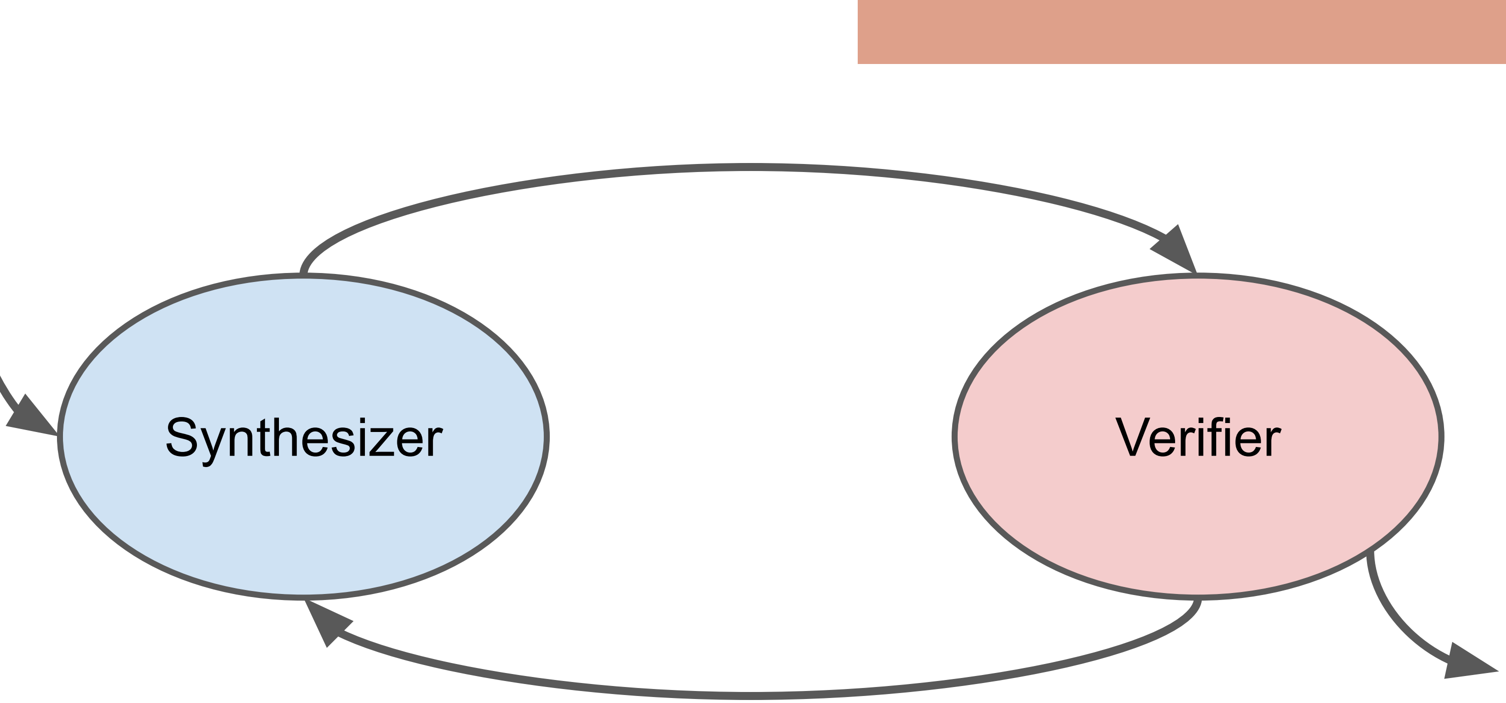
```c
char *p;
for (p = line;
     p && *p && whitespace (*p);
     p++)
  ;
```

```c
return result;
```



```c
char *p;
for (p = line;
     p && *p && whitespace (*p);
     p++)
  ;
```

```c
return result;
```

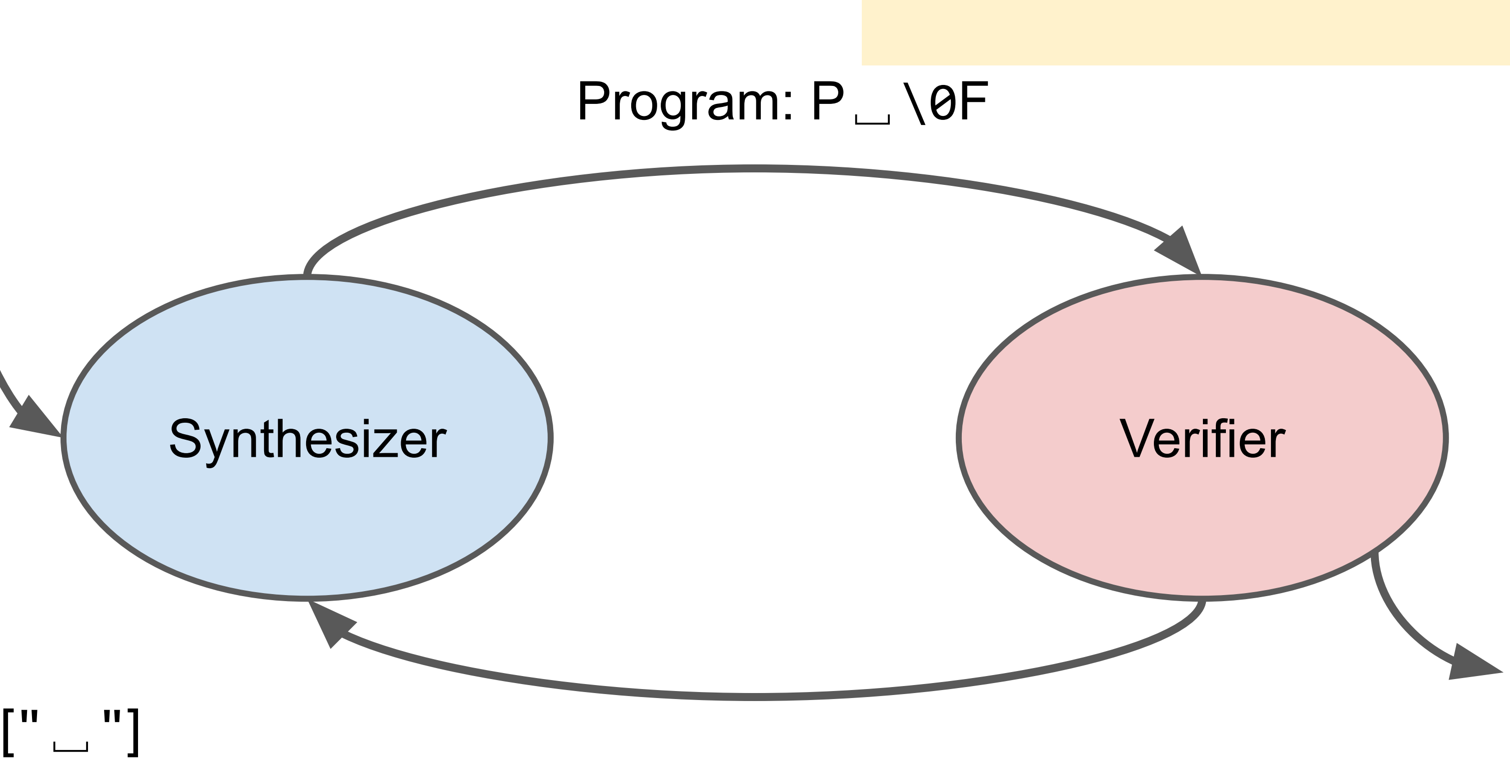


CEX: []

Counterexample: "␣"

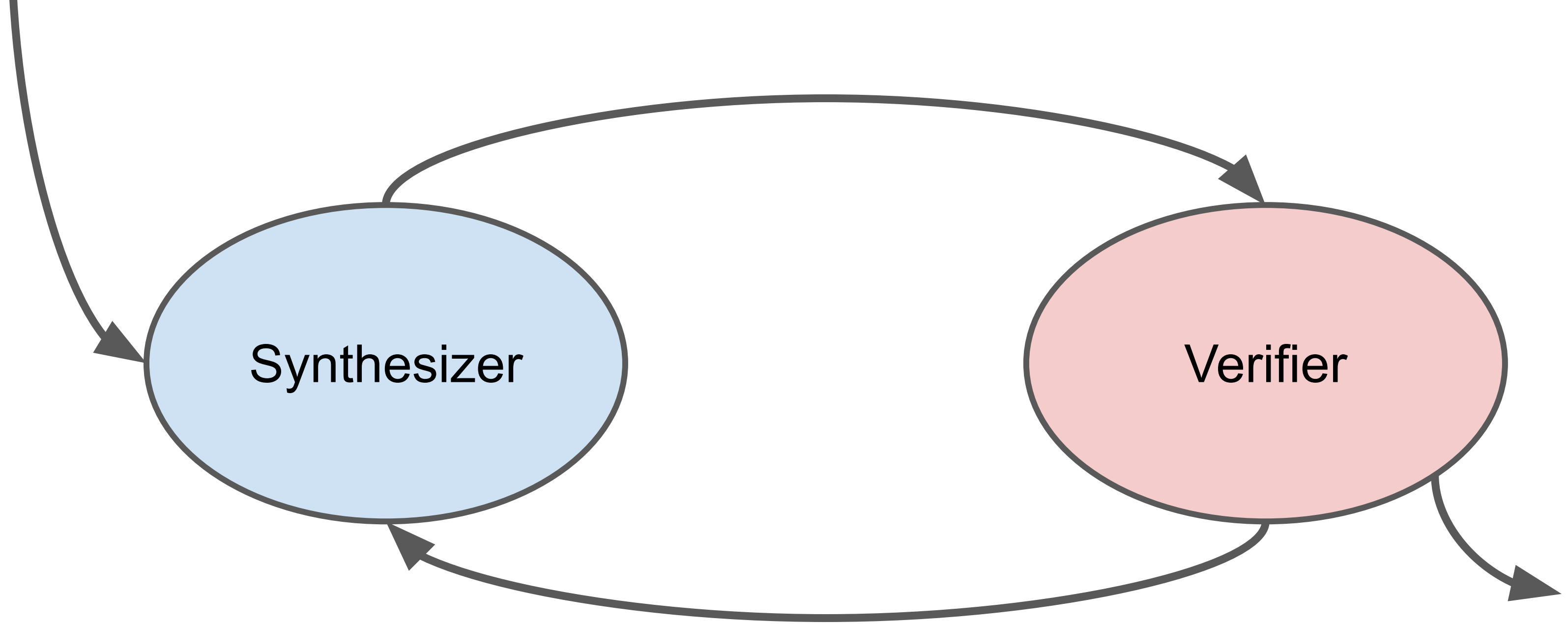
```c
char *p;
for (p = line;
     p && *p && whitespace (*p);
     p++)
  ;
```

```c
result += strspn(result, "␣");
return result;
```



```c
char *p;
for (p = line;
     p && *p && whitespace (*p);
     p++)
  ;
```

```c
result += strspn(result, "␣");
return result;
```

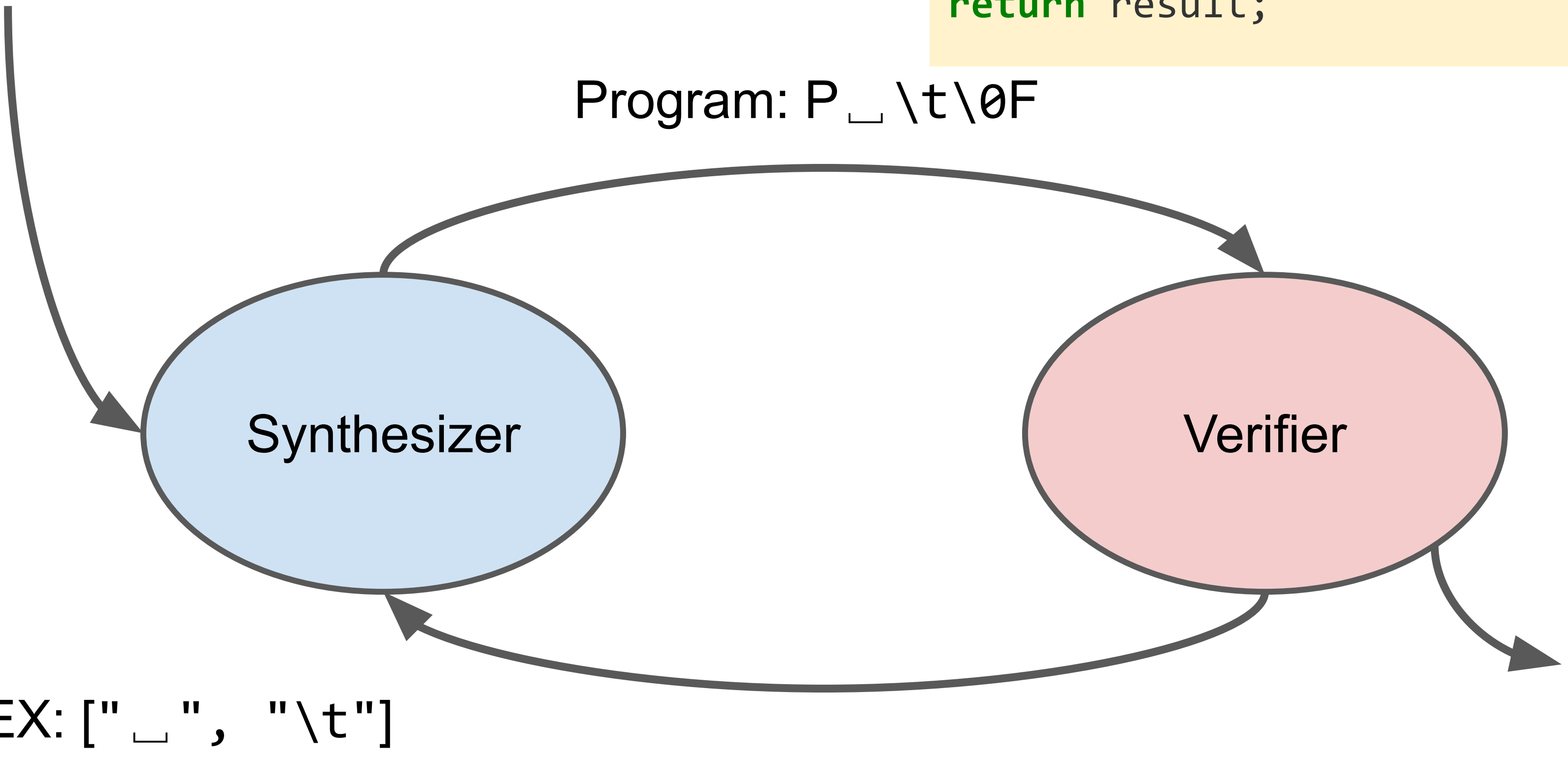


CEX: ["␣"]

Counterexample: "\t"

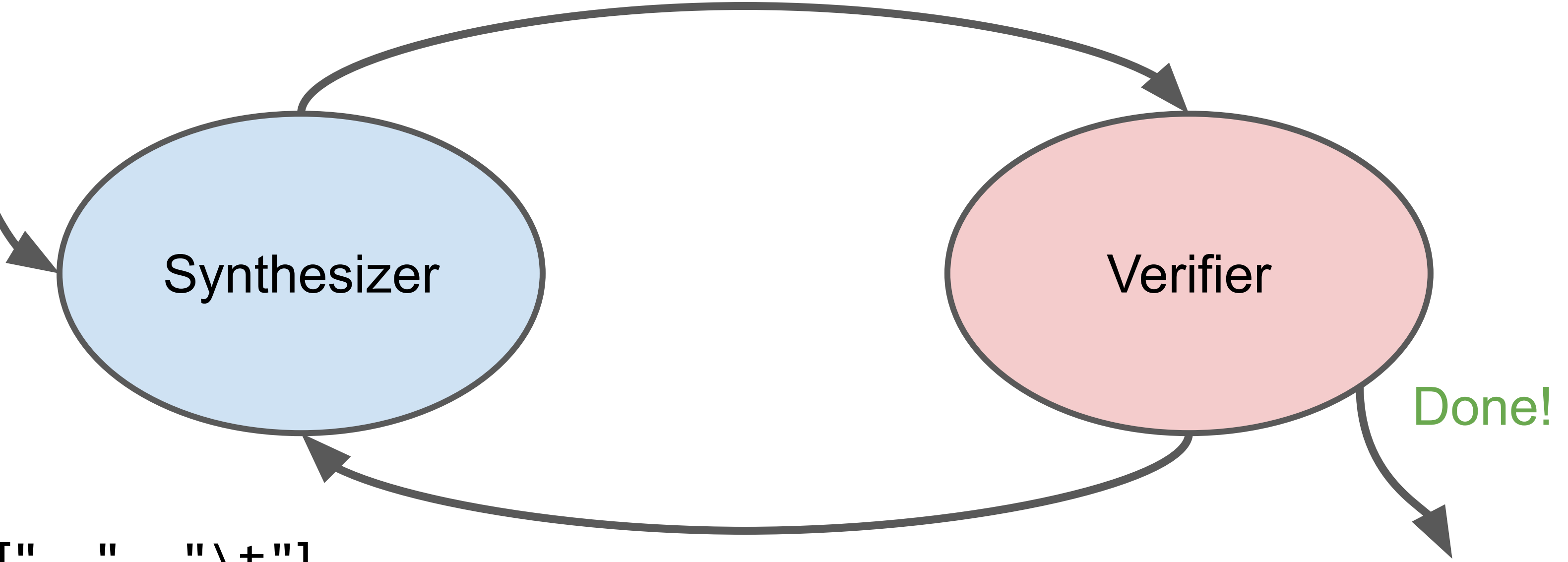
```c
char *p;
for (p = line;
     p && *p && whitespace (*p);
     p++)
  ;
```

```c
result += strspn(result,
"␣\t");
return result;
```



```c
char *p;
for (p = line;
     p && *p && whitespace (*p);
     p++)
  ;
```

```c
result += strspn(result,
"␣\t");
return result;
```



CEX: ["␣", "\t"]

P␣\t\0F

Synthesis Evaluation

- 13 open source programs
- Semi-automatic process
- Extracted 115 loops fitting our scope
- In total 88/115 synthesised

patch

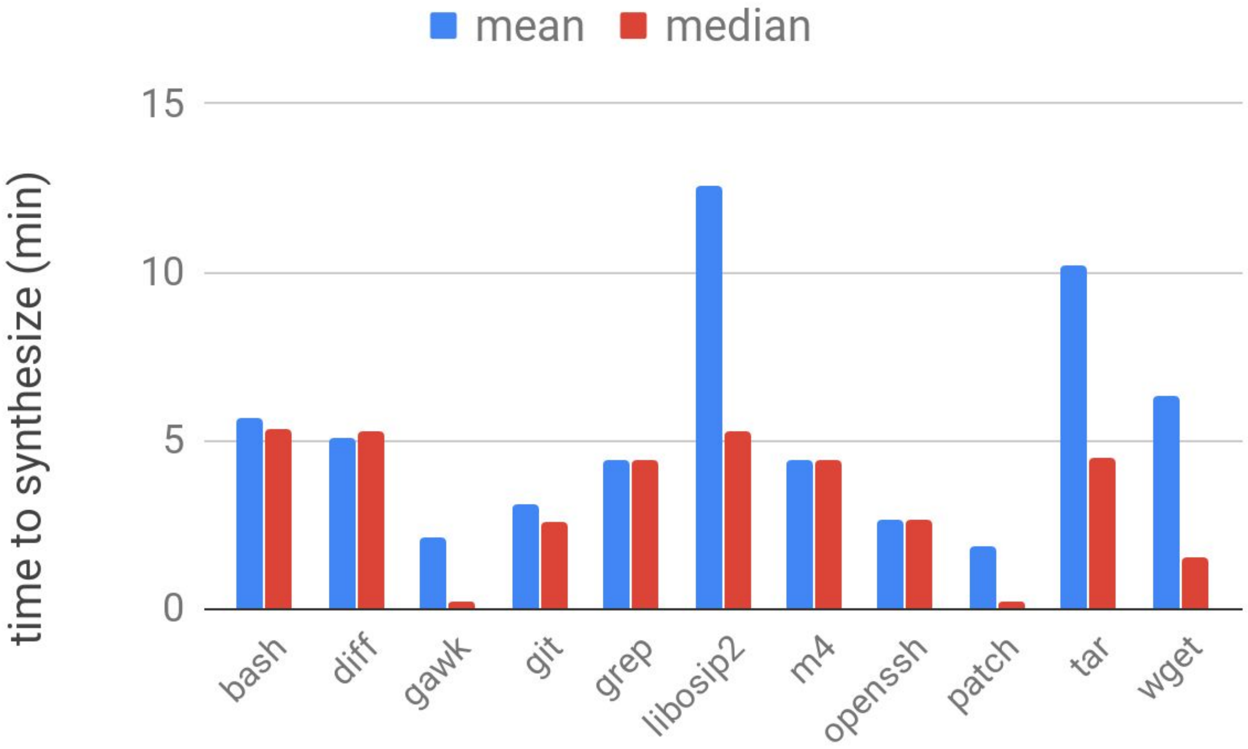
libosip

diff

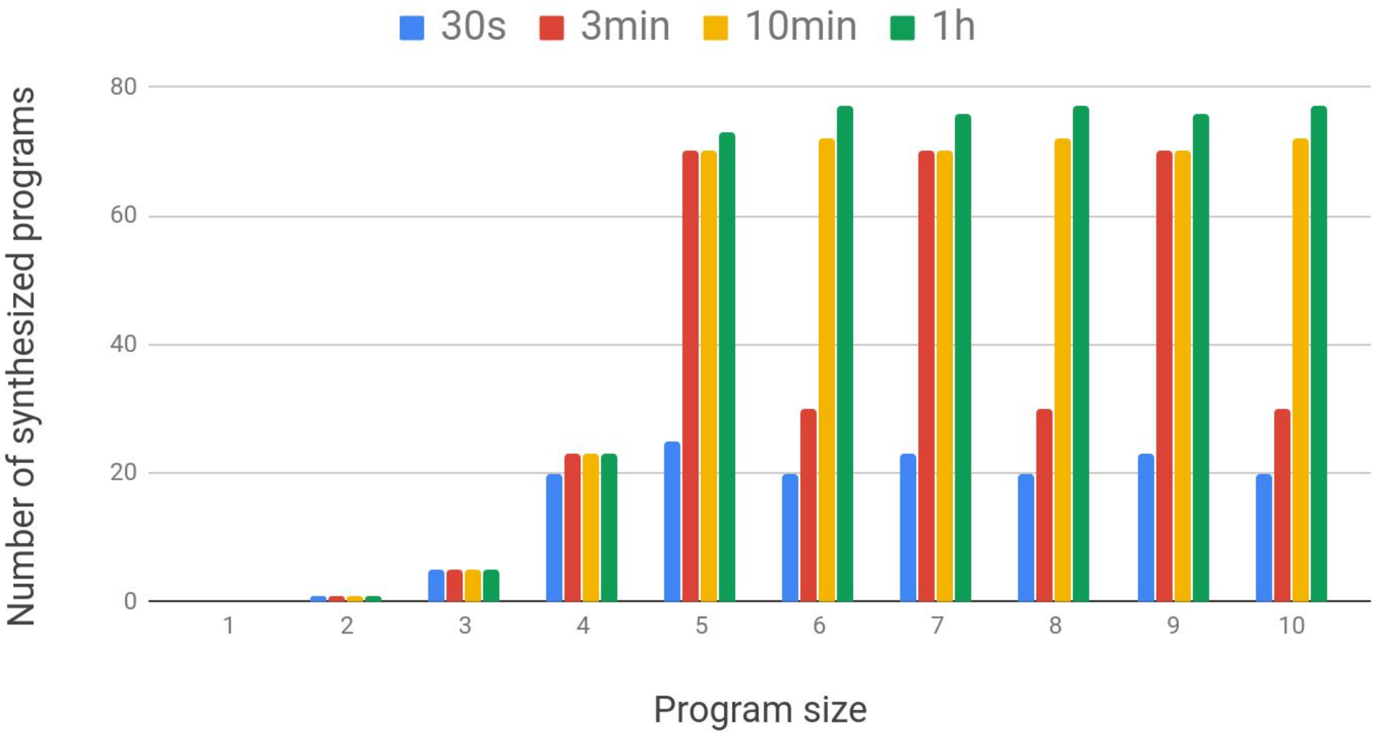
m4

make

2h/loop synthesis timeout: 77/115 loops



Impact of timeout and program size



Vocabulary optimisation

- Find a subset of vocabulary that synthesises more loops
- Gaussian process optimization
- 5 minute timeout
- 81/115 loops with 5min timeout
- 7 additional loops found with full vocabulary and 2h timeout

88/115 total

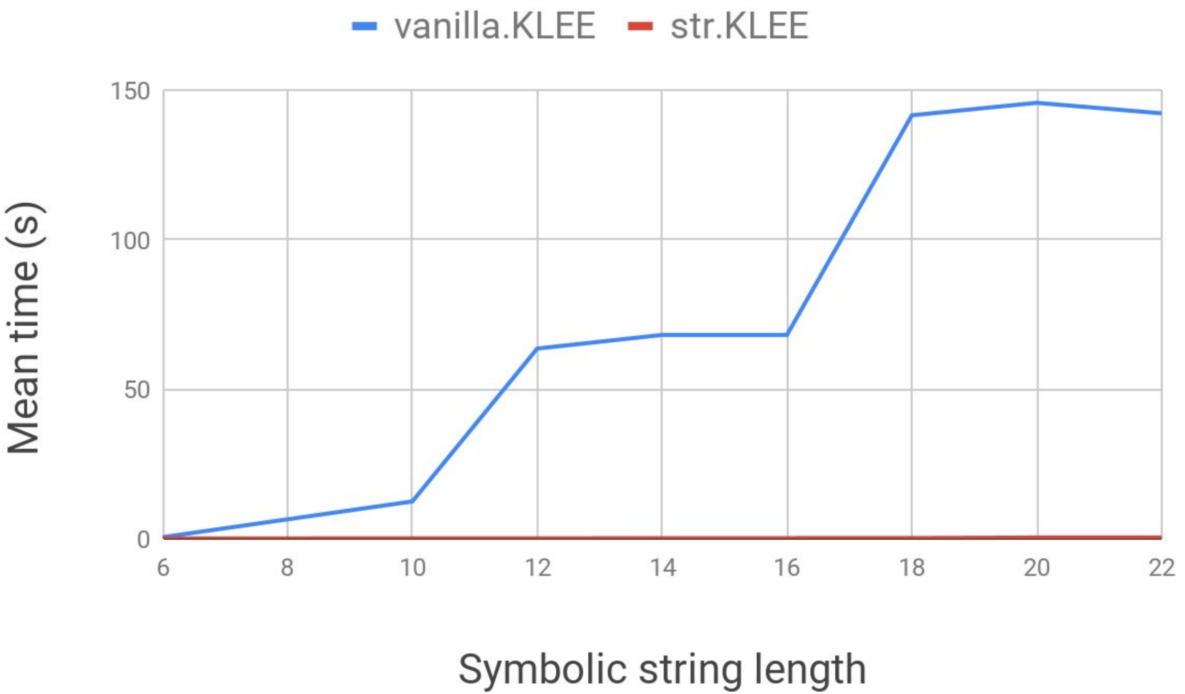
Best performing vocabulary

- `strspn`
- `strcspn`
- `memchr`
- `increment`
- `backward traverse`
- `return`

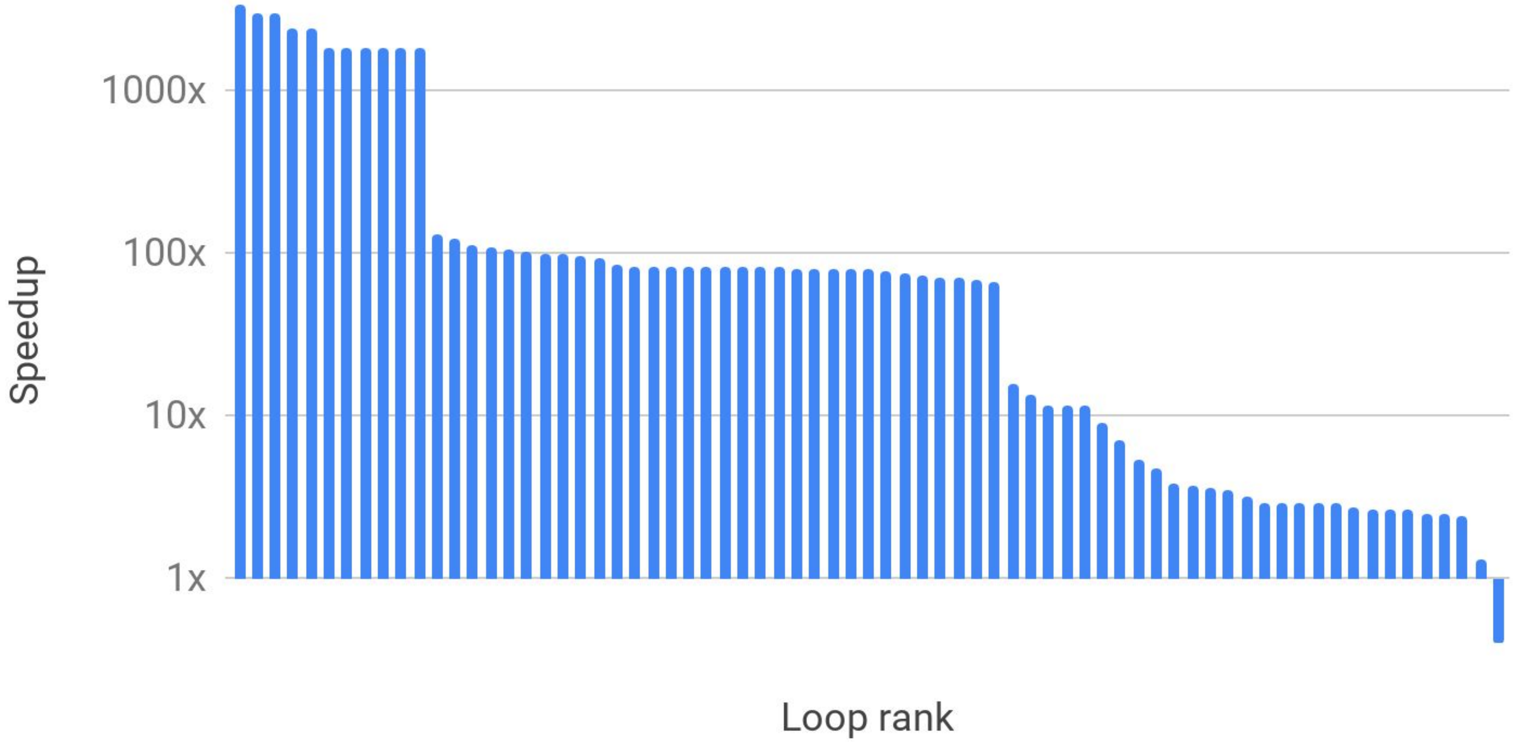
Improving symbolic execution

- Use loop summaries to gather more efficient constraints
- Intercept calls to `str` functions and encode them in theory of strings
- Compare with character by character constraints
  - Theory of strings should have an advantage for longer strings
- Implemented in KLEE
- Compared (only) on the loops we extracted

Improving symbolic execution

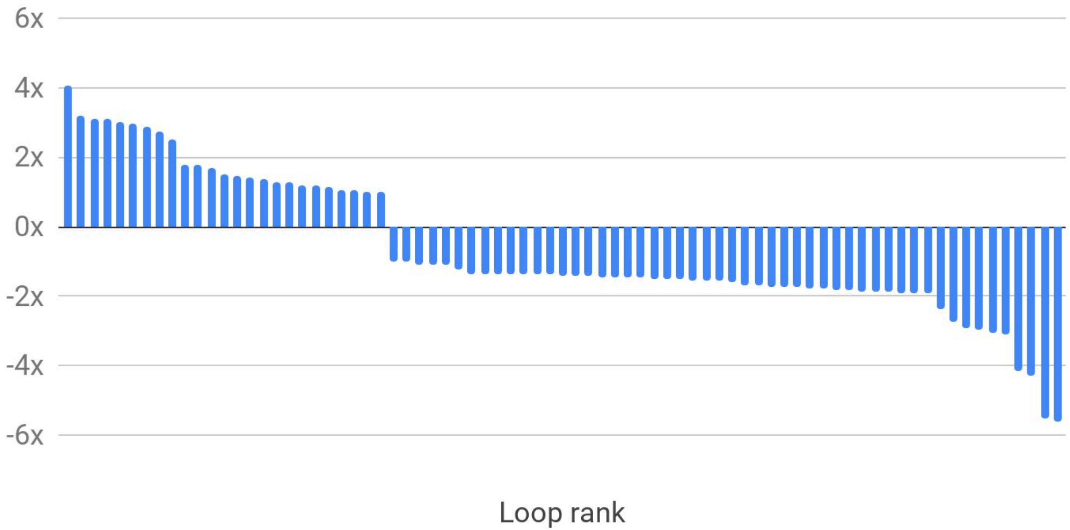


Improving symbolic execution



Compiler optimisation potential?

- Compare the loop summaries (libc library functions) with `-O3 -march` compiled loops



Refactoring

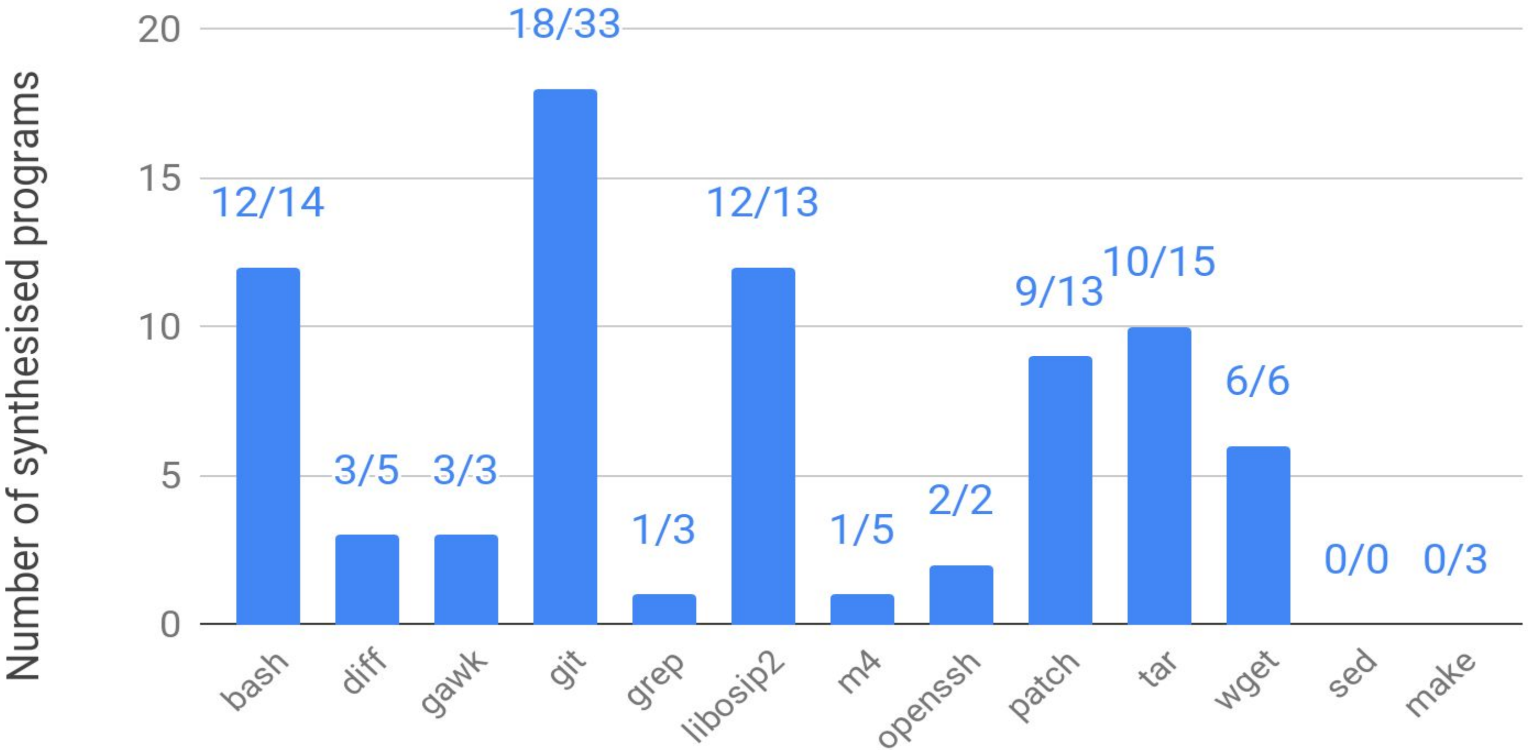
- Use summaries to create patches and send them to developers
- Developers of `libosip`, `patch` and `wget` accepted the patches

```
- for(; *tmp == ' ' || *tmp == '\t'; tmp++){
- }
- for(; *tmp == '\n' || *tmp == '\r'; tmp++){
- }                              /* skip LWS */
+ tmp += strspn(tmp, " \t");
+ tmp += strspn(tmp, "\n\r");
```

Conclusion

- Counterexample guided synthesis based technique for summarisation of simple loops in C
- 88/115 loops synthesized
- Applications:
  - Program analysis (symbolic execution)
  - Compiler optimisations
  - Refactoring

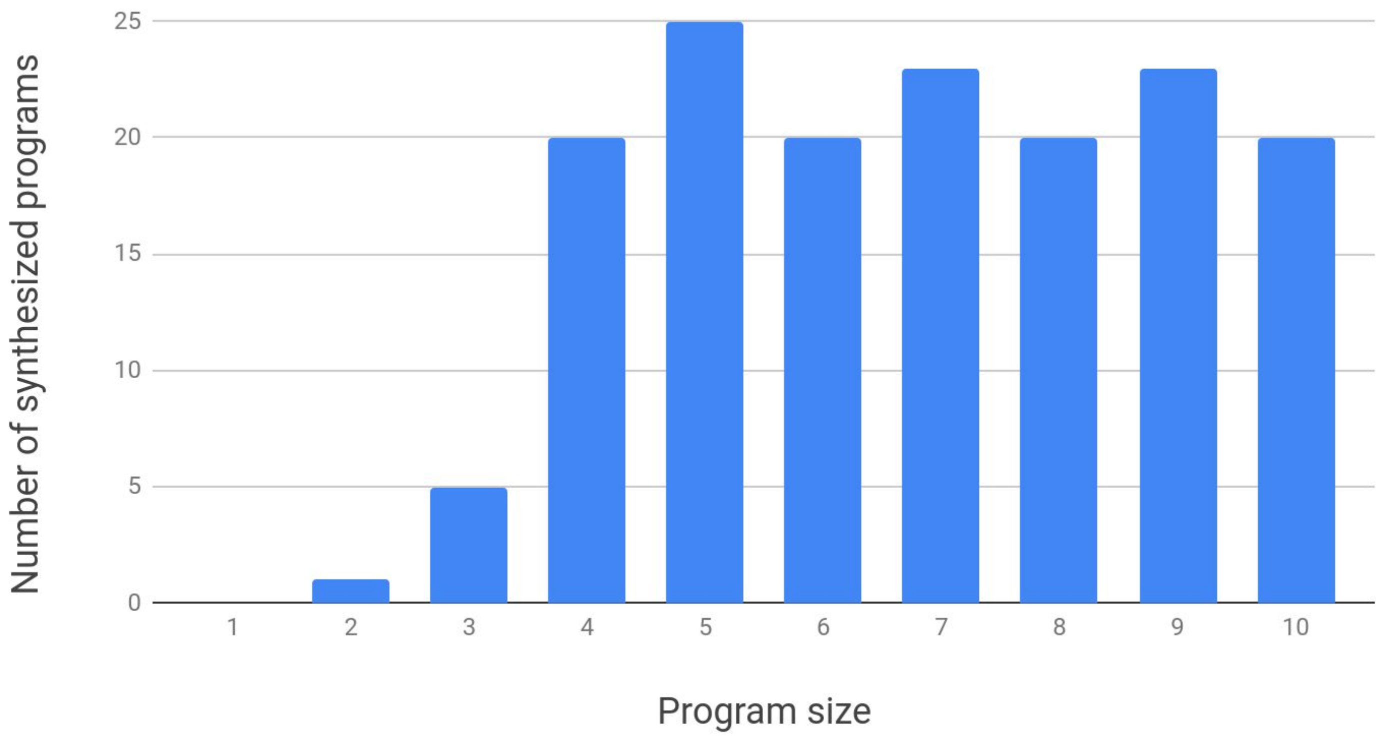
2h/loop synthesis timeout: 77/115 loops



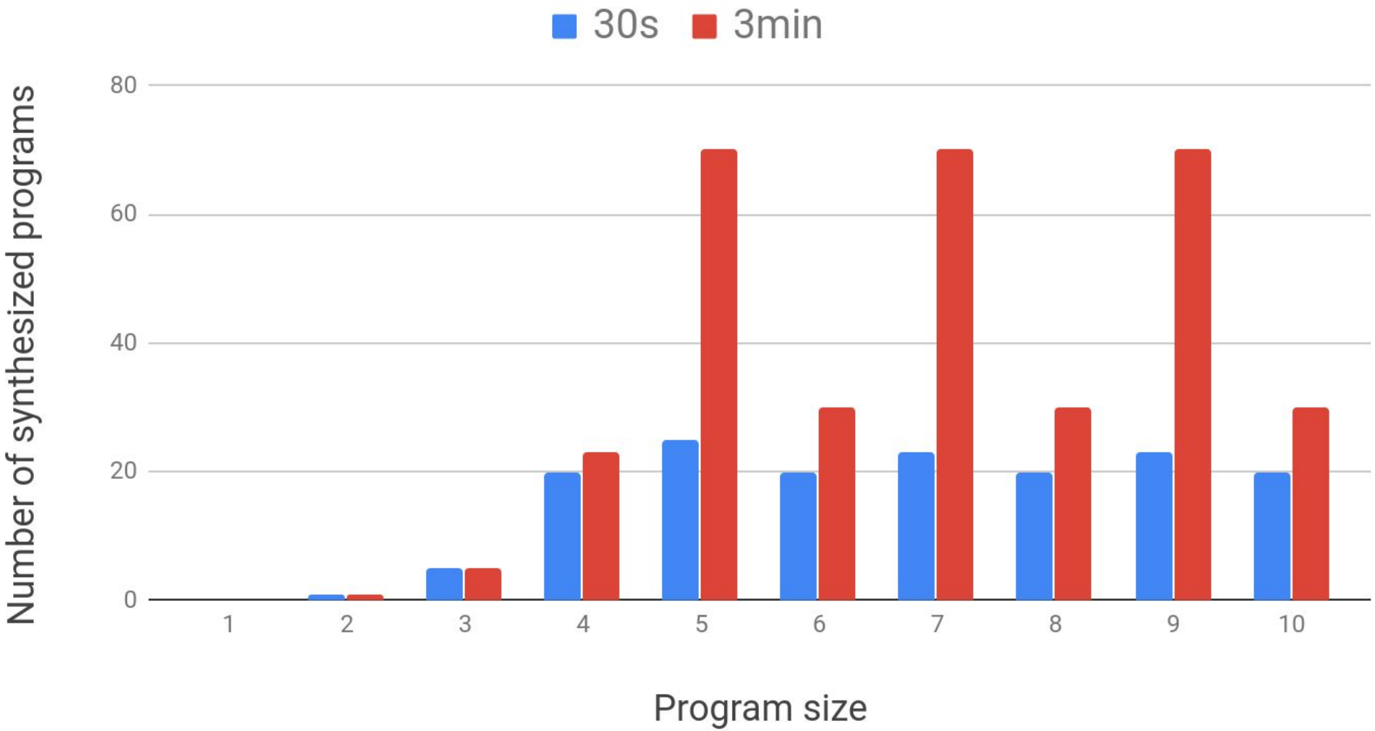
utility	Total loops	Inner loops	Loops without pointer call	Read only loops	Loops with a read from single pointer
bash	1085	944	438	264	45
diff	186	140	60	40	14
gawk	608	502	210	105	17
git	2904	2598	725	495	108
grep	222	172	72	42	9
m4	328	286	126	78	12
make	334	262	129	102	13
patch	207	172	88	67	20
sed	125	104	35	19	1
ssh	604	544	227	84	12
tar	492	432	155	106	33
torture_test	100	95	39	30	25
wget	228	197	115	83	14
SUM	7423	6448	2419	1515	323

Has Goto	2
IOsideeffects	3
Non Pointer Return	74
Return In Loop	70
Too Many Arguments	28
Too Many Return Values	31
SUM	208

Impact of timeout and program size - 30s timeout



Impact of timeout and program size



Impact of timeout and program size