Machine-Learning-Guided Selectively Unsound Static Analysis

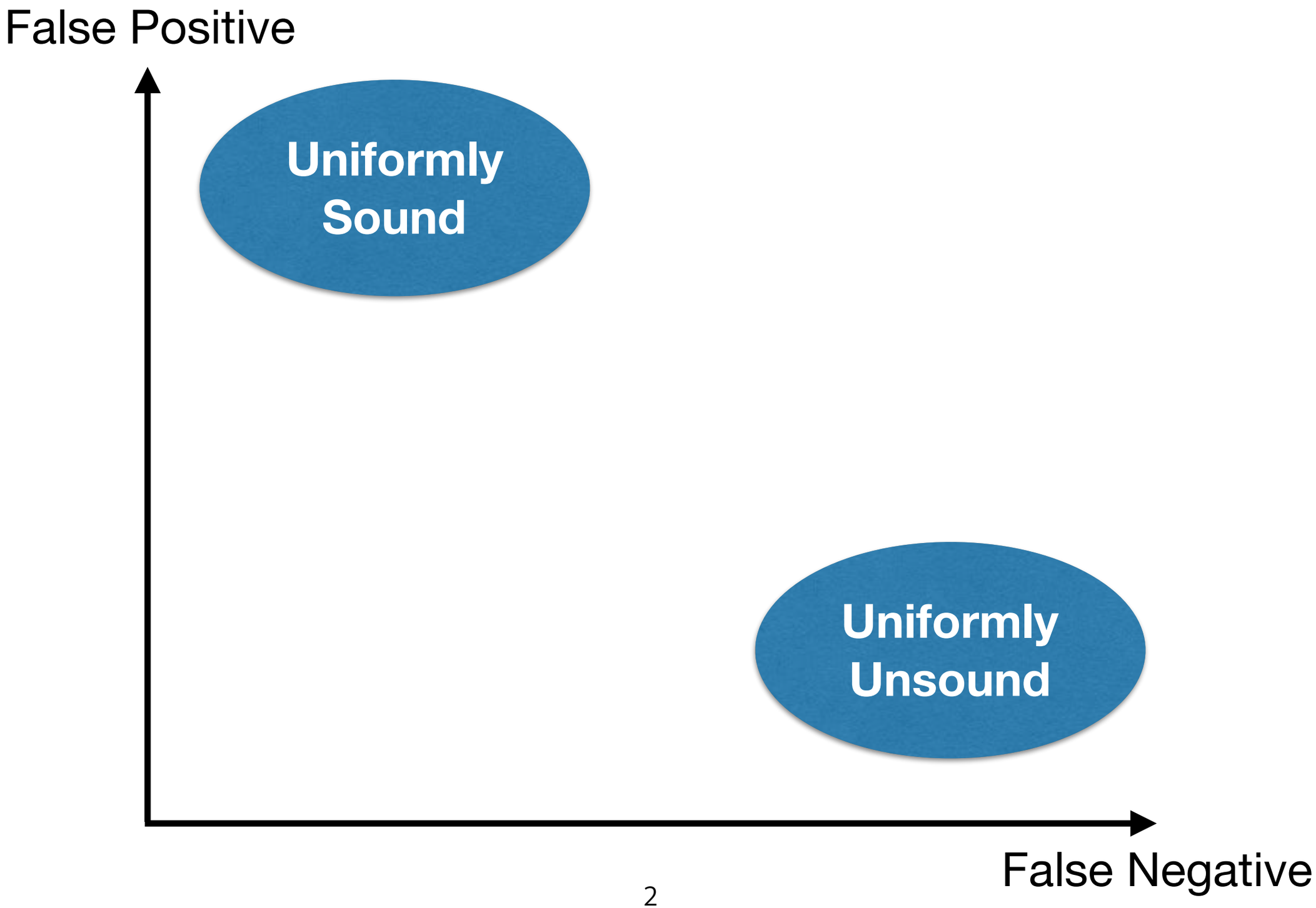
Kihong Heo
Seoul National University

Hakjoo Oh
Korea University

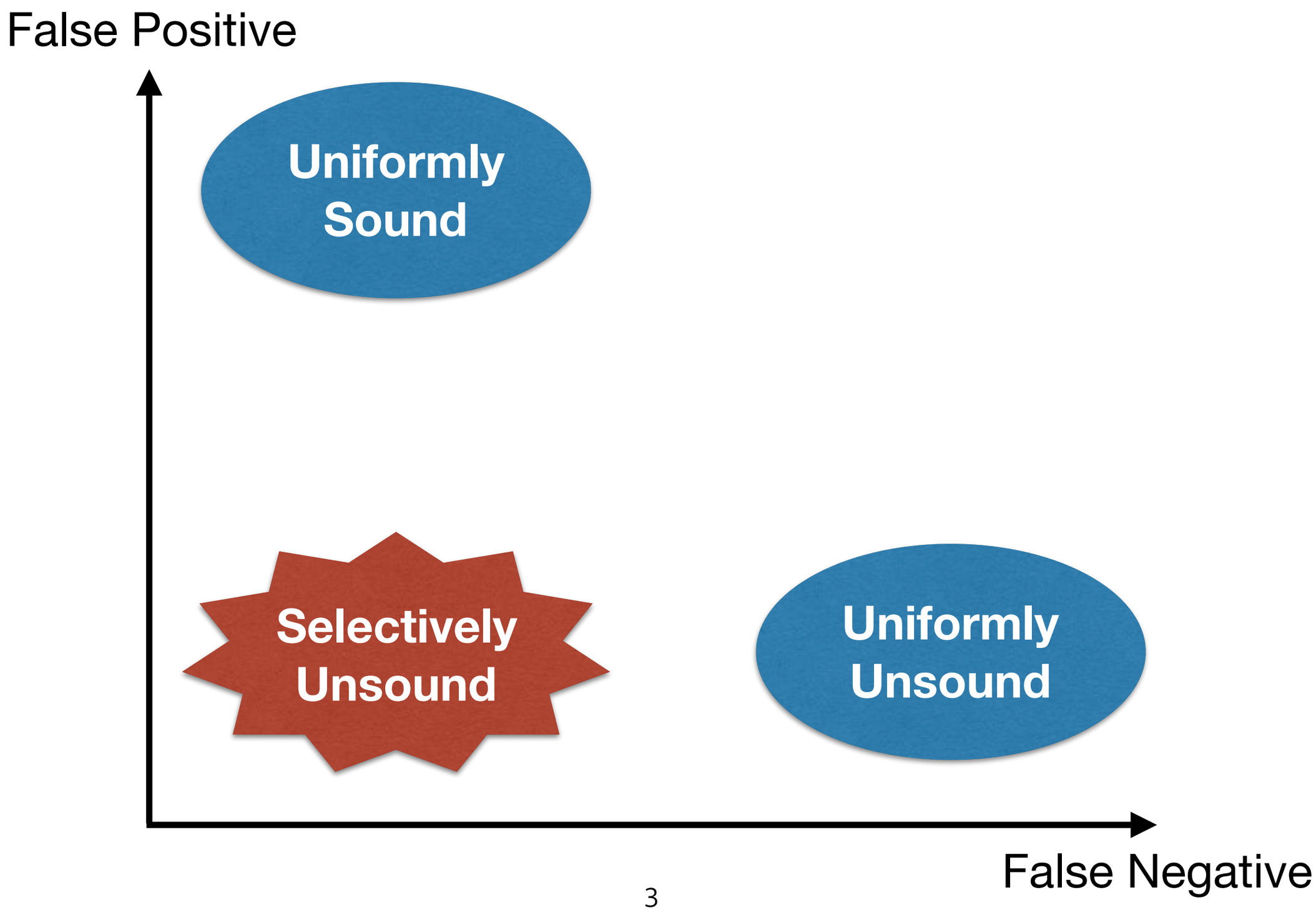
Kwangkeun Yi
Seoul National University

26 May 2017
ICSE'17 @ Buenos Aires

Goal


False Positive
Uniformly Sound
Uniformly Unsound
False Negative

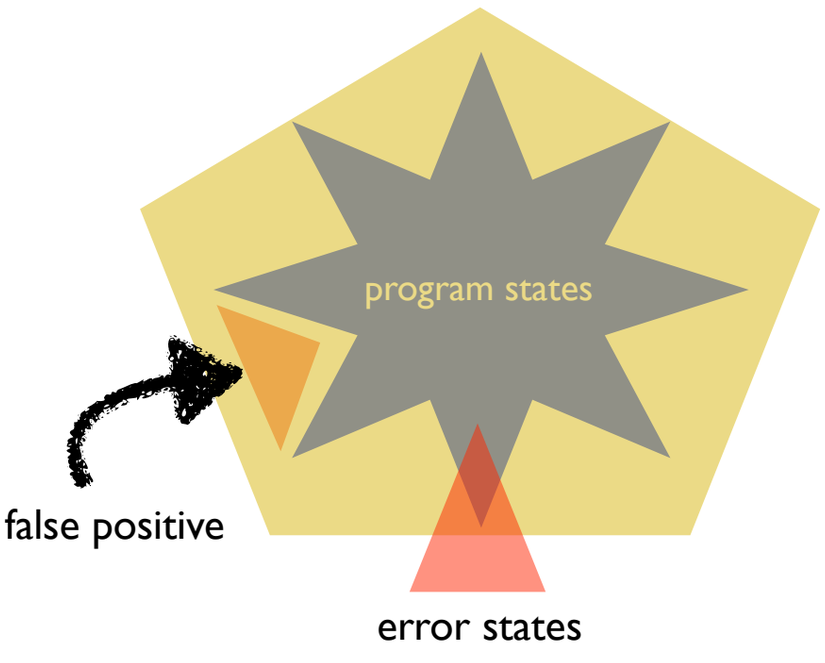
Goal


False Positive
Uniformly Sound
Selectively Unsound
Uniformly Unsound
False Negative

Selectively Unsound Analysis

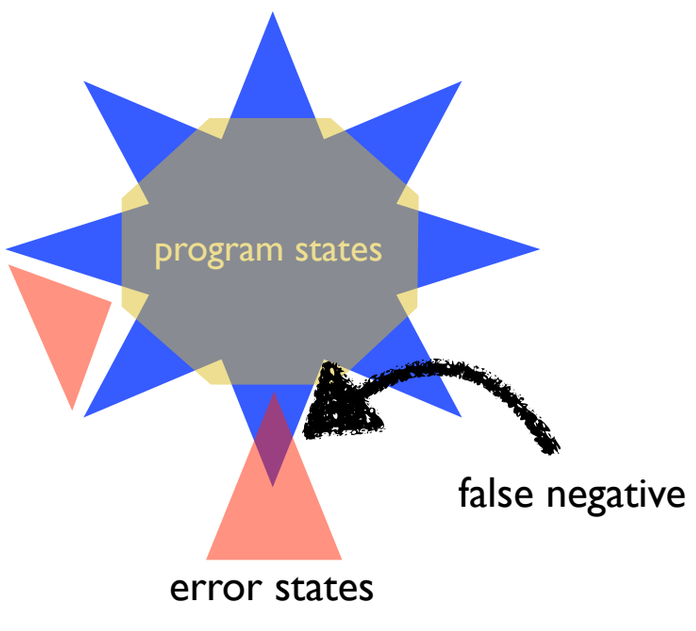
- Selectively apply unsound strategies
  - e.g.) unrolling loops, skipping lib calls

`while(e){ C }` ▶ `if(e){ C }` `A;lib();B;` ▶ `A;B;`



Uniformly Sound

Selectively Unsound



Uniformly Unsound

Example

- Sound buffer-overrun analyzer with interval domain
  - soundly analyze all the loops

```
str = "hello world";
for(i=0;  str[i]; i++)// buffer access 1
  skip;

size = positive_input();
for(i=0; i<size; i++)
  skip;

... = str[i];              // buffer access 2
```

Example

- Sound buffer-overrun analyzer with interval domain
  - soundly analyze all the loops

```
str = "hello world";
for(i=0;  str[i]; i++)// buffer access 1
  skip;

size = positive_input();
for(i=0; i<size; i++)
  skip;

... = str[i];                  // buffer access 2
```


str.size: [12, 12]

i: [0, +oo]

size: [0, +oo]

i: [0, +oo]

Example

- Uniformly unsound buffer-overrun analyzer
  - unsoundly unroll all the loops

```
str = "hello world";
i = 0;
if ( str[i])              // buffer access 1
  skip;

size = positive_input();
i = 0;
if (i < size)
  skip;

... = str[i];             // buffer access 2
```

Example

- Uniformly unsound buffer-overrun analyzer
  - unsoundly unroll all the loops

```
str = "hello world";
i = 0;
if ( str[i])                    // buffer access 1
  skip;

size = positive_input();
i = 0;
if (i < size)
  skip;

... = str[i];                   // buffer access 2
```

i: [0, 0]

i: [0, 0]

Example

- Selectively unsound buffer-overrun analyzer
  - unsoundly unroll only harmless loops

```
str = "hello world";
i = 0;
if( str[i])          // buffer access 1
  skip;

size = positive_input();
for(i = 0; i < size; i++)
  skip;

... = str[i];        // buffer access 2
```

Example

- Selectively unsound buffer-overrun analyzer
  - unsoundly unroll only harmless loops

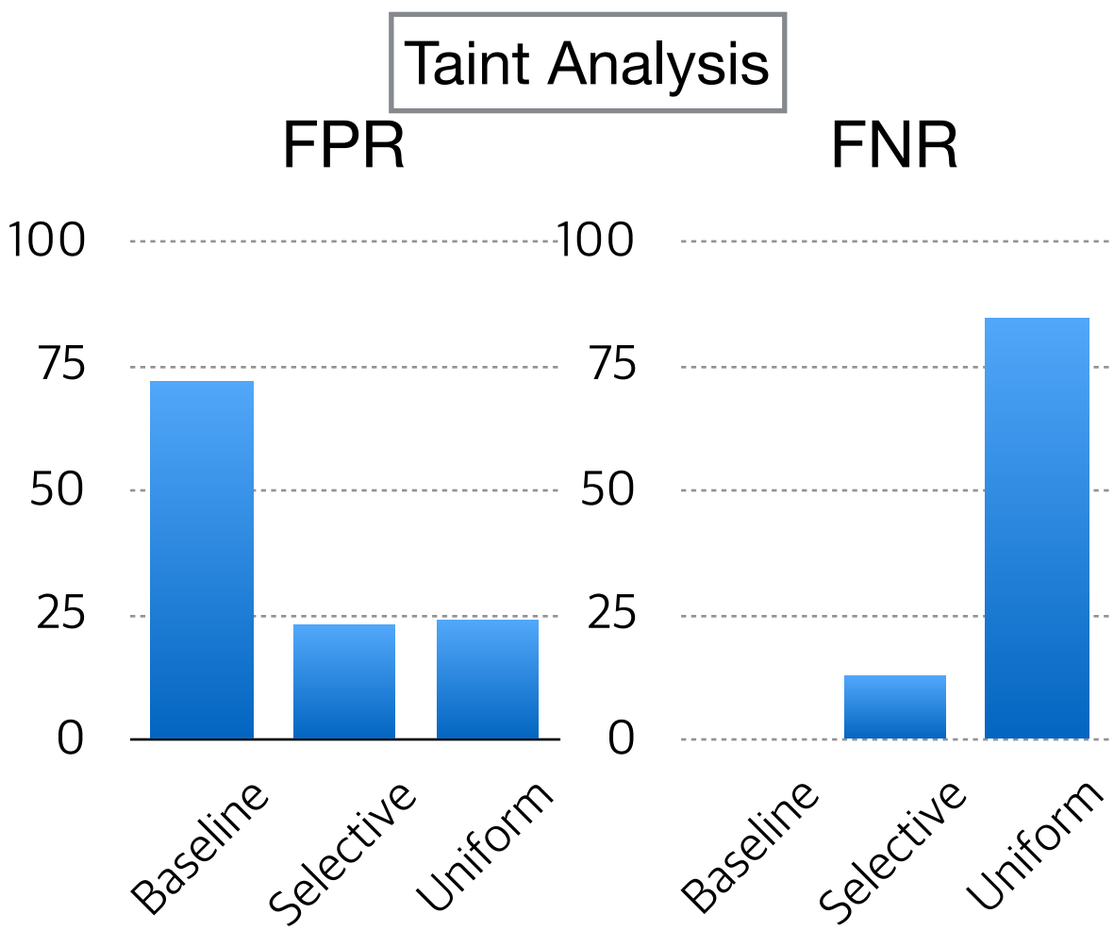
```
str = "hello world";
i = 0;
if( str[i])           // buffer access 1
  skip;

size = positive_input();
for(i = 0; i < size; i++)
  skip;

... = str[i];         // buffer access 2
```

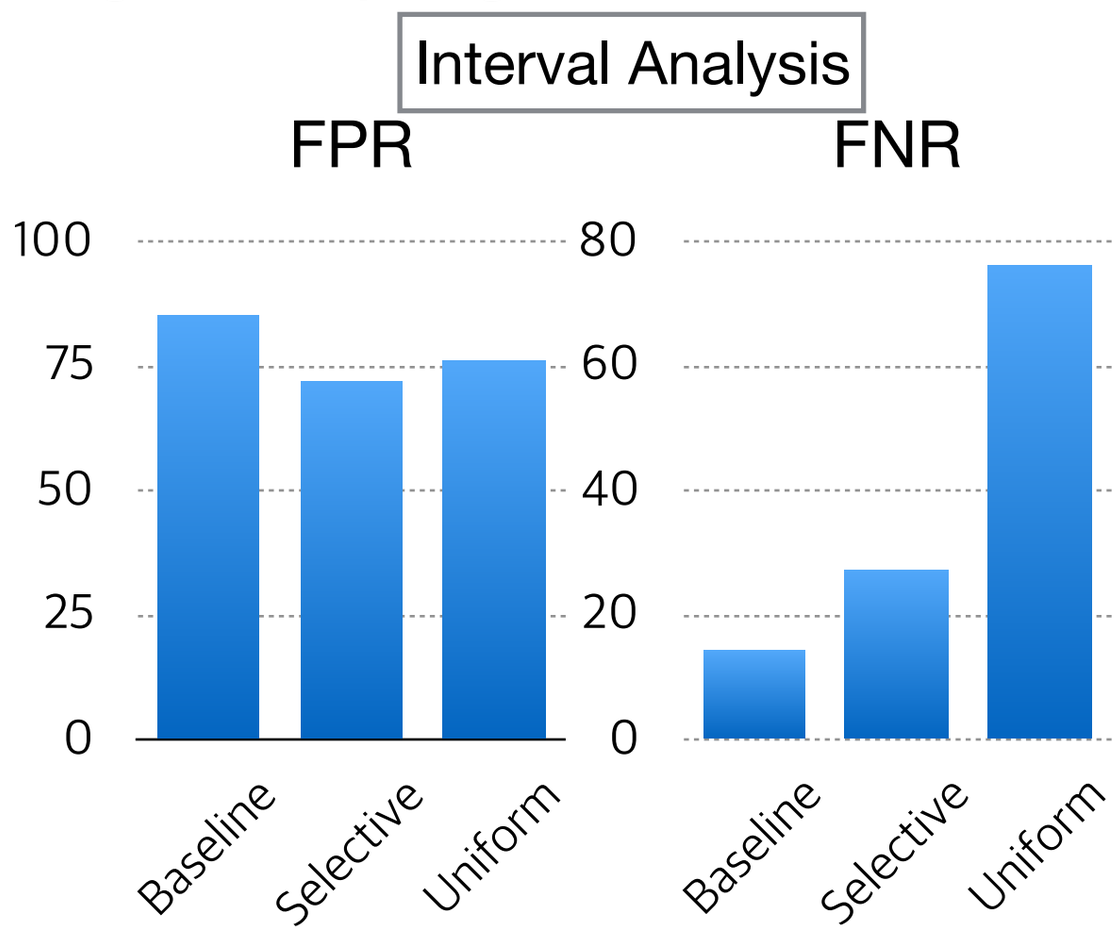
i: [0, 0]

i: [0, +oo]

Performance

- Experiments with 2 analyzers & open source SW
  - Taint: 106 format string bugs / 13 programs
  - Interval: 138 buffer overrun bugs / 23 programs



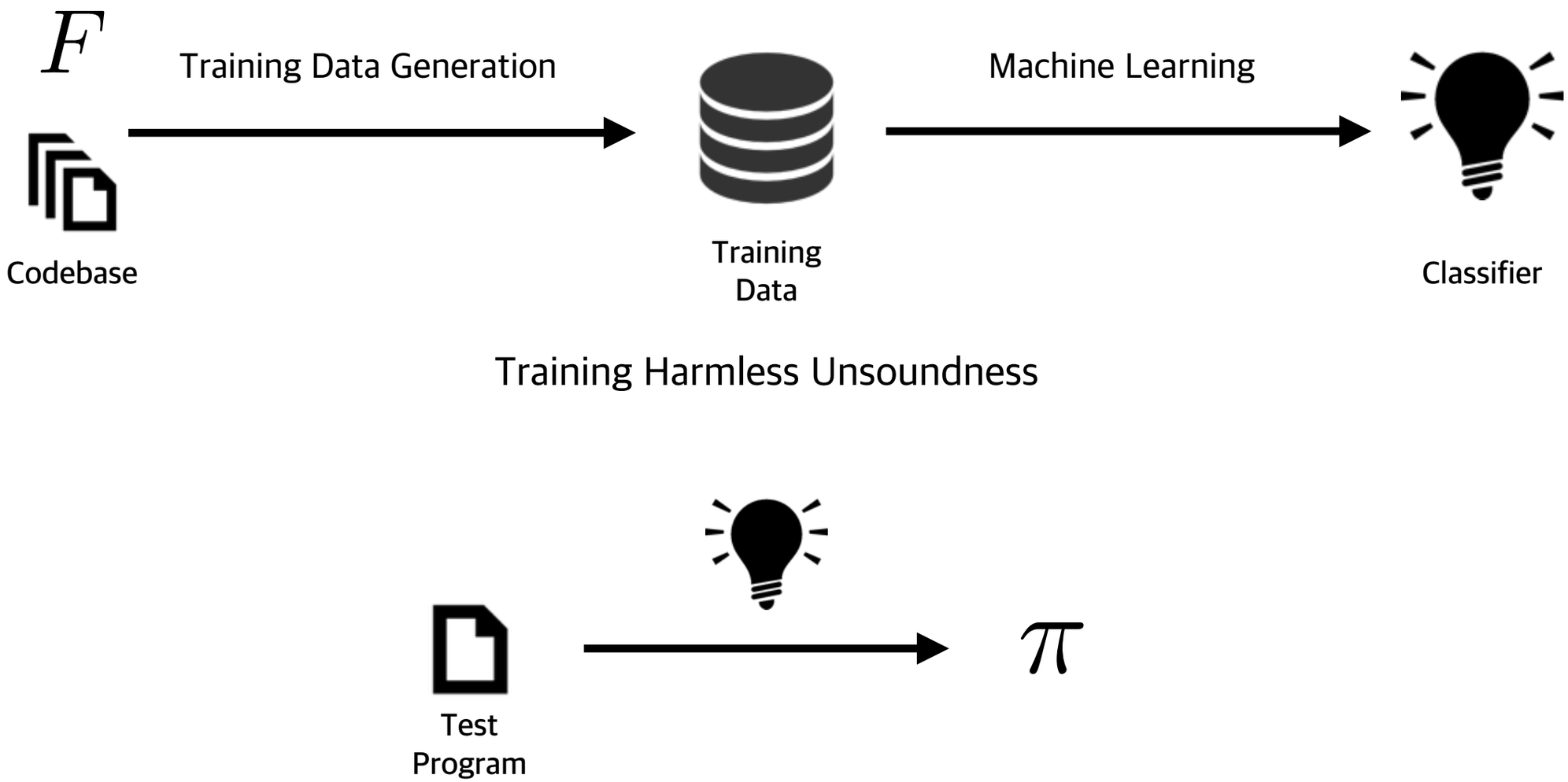


Setting

$$F \in Pgm \times \Pi \rightarrow \mathcal{A}$$

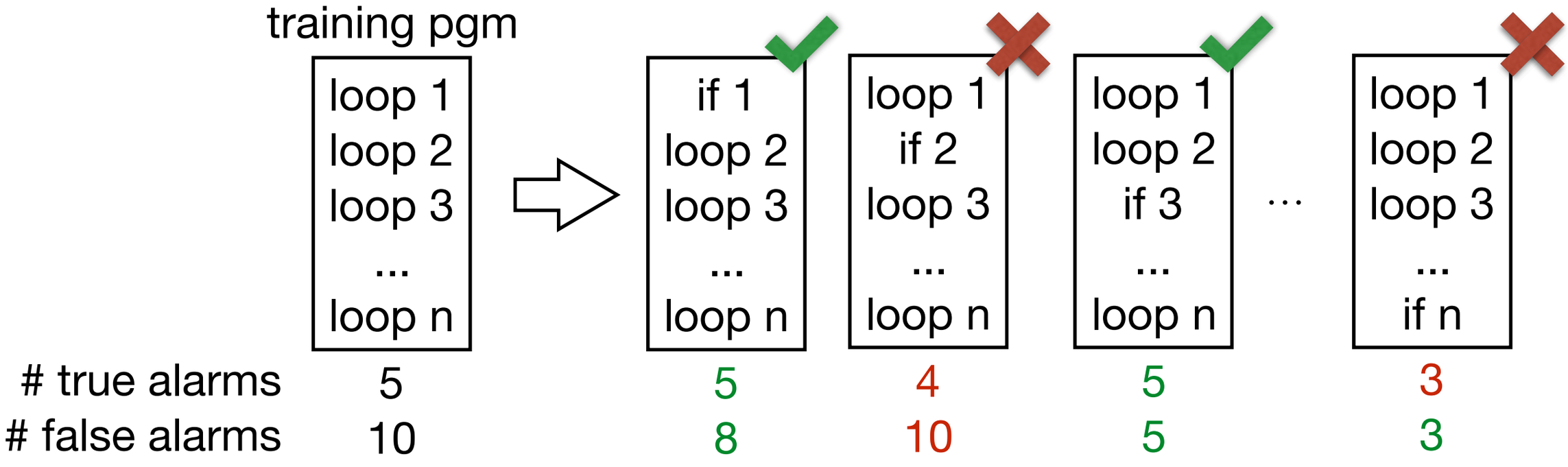
- Find a set of targets $\pi \in \Pi$ for unsound strategies
  - loops to analyze unsoundly ($\Pi = 2^{Loop}$)
  - library calls to analyze unsoundly ($\Pi = 2^{Lib}$)
- Selectively apply unsound strategies to $p \in \pi$

System Overview



Training Data Generation

- Given a codebase w/ known bugs + a sound static analyzer
- Collect precision-decreasing yet harmless pgm components
  - e.g.) unrolling a loop reduces only FP but retains all TP



Features & Learning

- Encode each program component as a feature vector

$$f(x) = \langle f_1(x), f_2(x), \ldots, f_n(x) \rangle$$

$$f(loop_1) = \langle 1, 0, \ldots, 1 \rangle$$

$$f(loop_2) = \langle 0, 1, \ldots, 1 \rangle$$

$$f(lib_1) = \langle 0, 1, \ldots, 0 \rangle$$

$$f(lib_2) = \langle 1, 1, \ldots, 1 \rangle$$

- Derive a classifier using an off-the-shelf algorithm
  - e.g.) SVM

Features

- 22 features for loops

Feature	Property	Type	Description
Null	Syntactic	Binary	Whether the loop condition contains nulls or not
Const	Syntactic	Binary	Whether the loop condition contains constants or not
Array	Syntactic	Binary	Whether the loop condition contains array accesses or not
Conjunction	Syntactic	Binary	Whether the loop condition contains `&&` or not
IdxSingle	Syntactic	Binary	Whether the loop condition contains an index for a single array in the loop
IdxMulti	Syntactic	Binary	Whether the loop condition contains an index for multiple arrays in the loop
IdxOutside	Syntactic	Binary	Whether the loop condition contains an index for an array outside of the loop
InitIdx	Syntactic	Binary	Whether an index is initialized before the loop
Exit	Syntactic	Numeric	The (normalized) number of exits in the loop
Size	Syntactic	Numeric	The (normalized) size of the loop
ArrayAccess	Syntactic	Numeric	The (normalized) number of array accesses in the loop
ArithInc	Syntactic	Numeric	The (normalized) number of arithmetic increments in the loop
PointerInc	Syntactic	Numeric	The (normalized) number of pointer increments in the loop
Prune	Semantic	Binary	Whether the loop condition prunes the abstract state or not
Input	Semantic	Binary	Whether the loop condition is determined by external inputs
GVar	Semantic	Binary	Whether global variables are accessed in the loop condition
FinInterval	Semantic	Binary	Whether a variable has a finite interval value in the loop condition
FinArray	Semantic	Binary	Whether a variable has a finite size of array in the loop condition
FinString	Semantic	Binary	Whether a variable has a finite string in the loop condition
LCSize	Semantic	Binary	Whether a variable has an array of which the size is a left-closed interval
LCOffset	Semantic	Binary	Whether a variable has an array of which the offset is a left-closed interval
#AbsLoc	Semantic	Numeric	The (normalized) number of abstract locations accessed in the loop

Features

- 15 features for library calls

Feature	Property	Type	Description
Const	Syntactic	Binary	Whether the parameters contain constants or not
Void	Syntactic	Binary	Whether the return type is void or not
Int	Syntactic	Binary	Whether the return type is int or not
CString	Syntactic	Binary	Whether the function is declared in `string.h` or not
InsideLoop	Syntactic	Binary	Whether the function is called in a loop or not
#Args	Syntactic	Numeric	The (normalized) number of arguments
DefParam	Semantic	Binary	Whether a parameter are defined in a loop or not
UseRet	Semantic	Binary	Whether the return value is used in a loop or not
UptParam	Semantic	Binary	Whether a parameter is update via the library call
Escape	Semantic	Binary	Whether the return value escapes the caller
GVar	Semantic	Binary	Whether a parameters points to a global variable
Input	Semantic	Binary	Whether a parameters are determined by external inputs
FinInterval	Semantic	Binary	Whether a parameter have a finite interval value
#AbsLoc	Semantic	Numeric	The (normalized) number of abstract locations accessed in the arguments
#ArgString	Semantic	Numeric	The (normalized) number of string arguments

Winning Features

- Interval analysis
  - loops iterating on finite strings
  - library calls that return integers or manipulate strings

```
str = “hello world”;
for (p = str; *p; p++)
 ...
```

```
int r = lib1();
lib2(str1, str2);
```

Winning Features

- Interval analysis
  - loops iterating on finite strings
  - library calls that return integers or manipulate strings

```
str = “hello world”;
for (p = str; *p; p++)
 ...
```

finite string

array access

ptr increment

return integer

```
int r = lib1();
lib2(str1, str2);
```

str manipulation

Winning Features

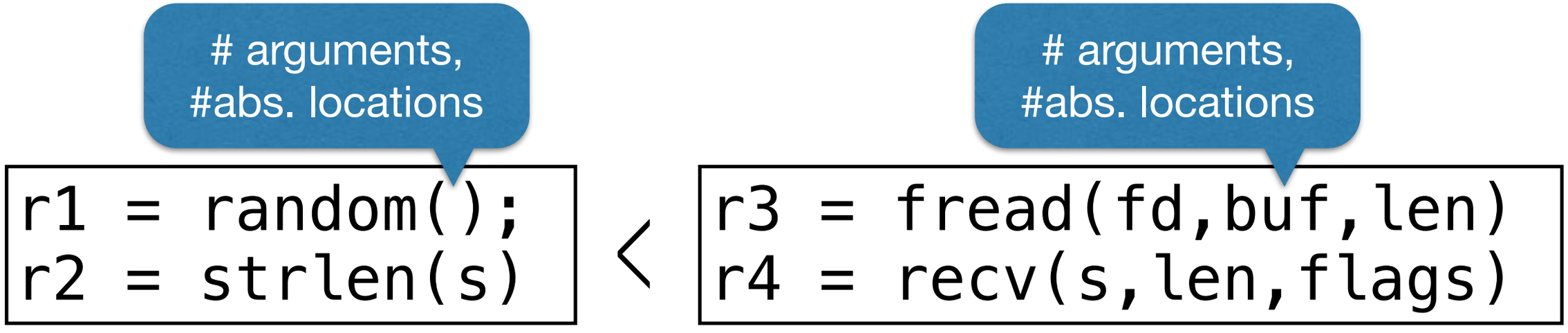
- Taint analysis
  - library calls not propagating user inputs

```
r1 = random();
r2 = strlen(s)
```

```
r3 = fread(fd,buf,len)
r4 = recv(s,len,flags)
```

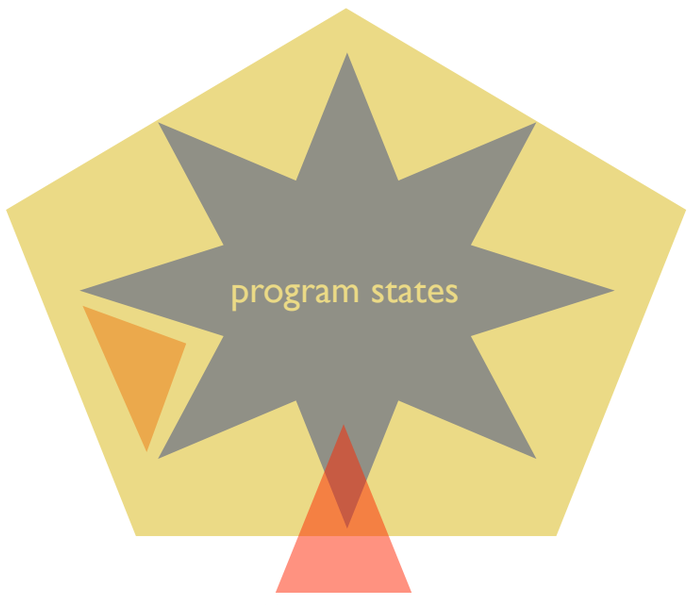
Winning Features

- Taint analysis
  - library calls not propagating user inputs

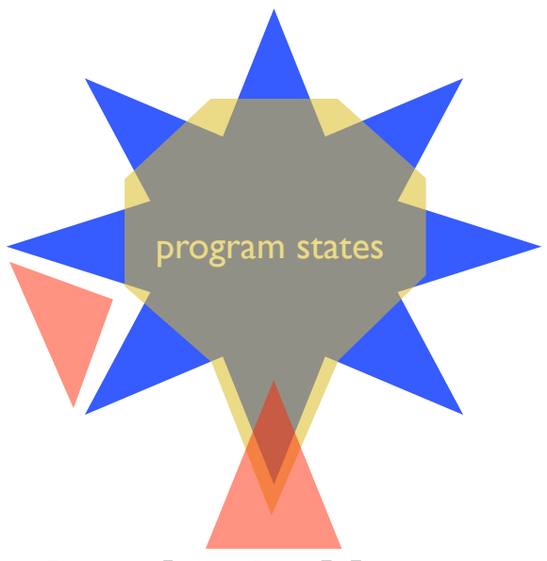


Summary

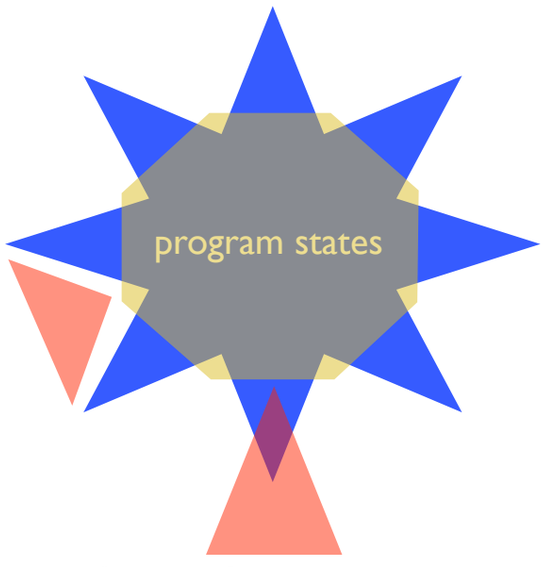
- First **selectively unsound** static analysis
  - more effective than uniformly sound / unsound ones
  - systematic way to tune unsoundness by ML



Sound



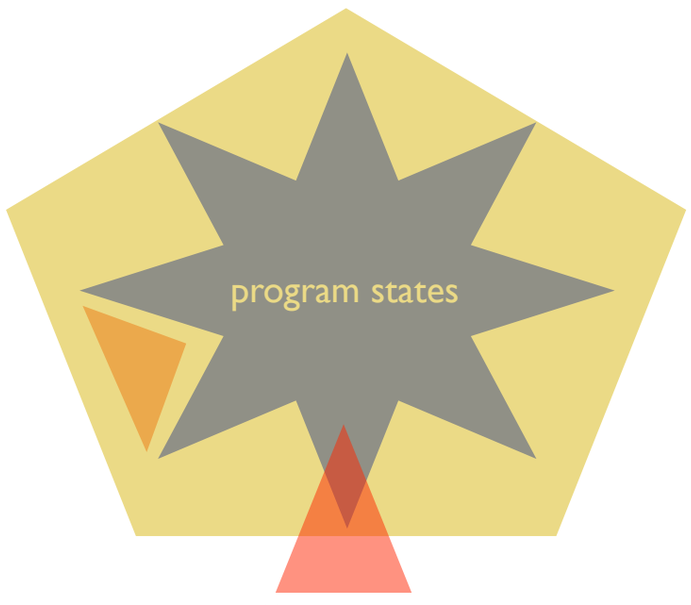
Selectively Unsound



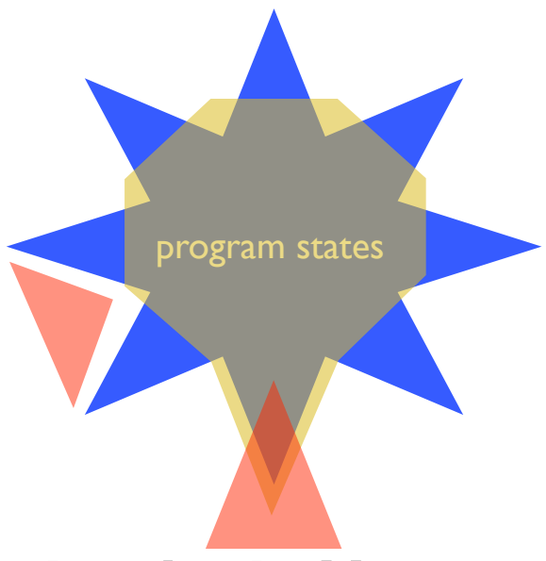
Uniformly Unsound

Summary

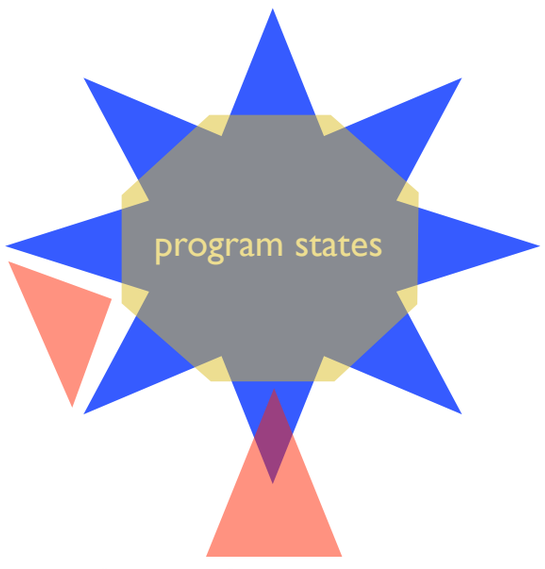
- First **selectively unsound** static analysis
  - more effective than uniformly sound / unsound ones
  - systematic way to tune unsoundness by ML



Sound



Selectively Unsound



Uniformly Unsound

Thank you