TensileFuzz: Facilitating Seed Input Generation in Fuzzing via String Constraint Solving

Xuwei Liu
Purdue University

Wei You*
Renmin University of China

Zhuo Zhang
Purdue University

Xiangyu Zhang
Purdue University

ISSTA' 22

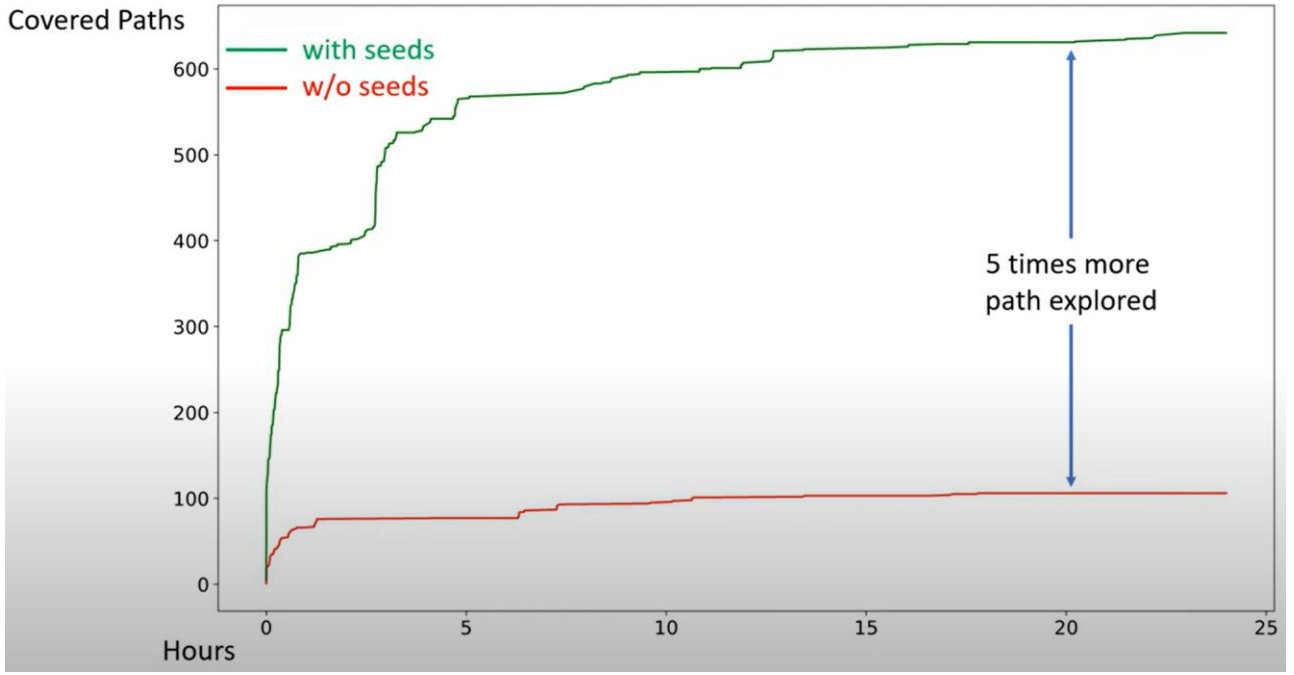
Background

基于变异的模糊测试

- 一组有效的输入实例作为初始种子
- 持续变异它们以探索各种执行路径
- 如果触发了新的覆盖，则将对应的变异输入保存到种子池
  - input growth（即，使输入越来越长），是生成种子输入的关键步骤

有效的种子至关重要：

使用AFL对libzip进行24小时fuzzing



当时的 Existing Works

Seed generation

- Traditional symbolic execution: KLEE [OSDI 08], S2E [ASPLOS 11]
- Enhanced symbolic execution: identify loop relations (Saxena et al. [ISSTA 09]) accelerate array constrains (Perry et al. [ISSTA 17])

Hybrid fuzzing

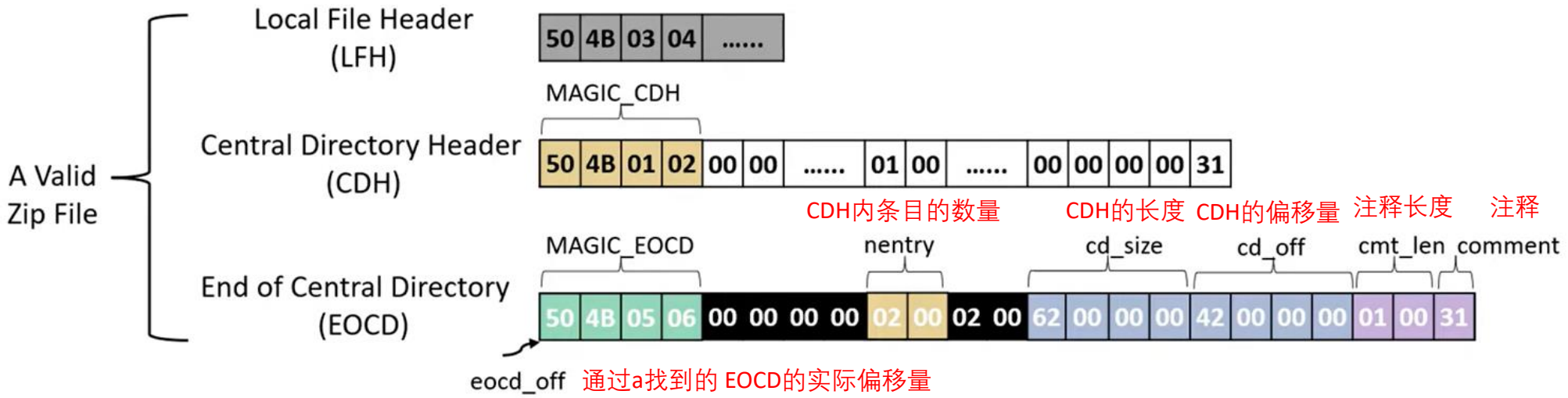
- With symbolic execution: Driller [NDSS 16], QSYM [SECURITY 18], Intriguer [CCS 19]
  - 对每一条语句和路径条件进行分析，提供最一般的约束建模
- With gradient-based search: Angora [S&P 18], SLF [ICSE 19], Matryoshka [CCS 19]
  - 只考虑数学关系

在许多情况下，对input growth至关重要的约束（例如，输入长度约束）不会通过程序语句明确地体现出来。

Motivation

以zip文件格式的输入为例。

当使用 libzip 读取 zip archive时，它必须先执行一系列有效性检查。例如：




```
01 file_size = get_file_size(input);
02 while (memcmp(input + eocd_off, MAGIC_EOCD, 4) != 0)
03   eocd_off++;
04 if (eocd_off > file_size) error();
```
ⓐ check magic number

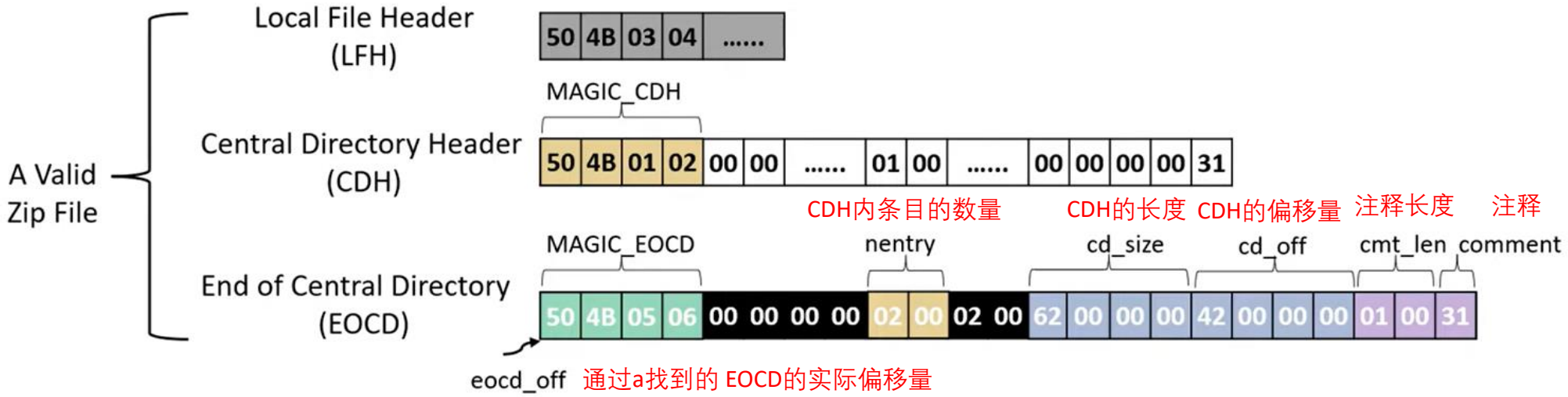
```
05 int tail_len = count_remaining(input);
06 if (cmt_len != tail_len) error();
```
check length ⓑ

```
07 if (cd_size + cd_off > eocd_off) error();
```
check offset ⓒ

```
08 while(cd_size > 0)
{
09 if(cd_size < MIN_CDH_SIZE) error();
10 if(memcmp(buf_cd_off, MAGIC_CDH, 4)) error();
11 cd_size -= entry[i].size;
12 if(++i == nentry && cd_size != 0) error();
}
```
convoluted check ⓓ

- AFL can pass check ⓐ with dictionary but fails check ⓑ
- Hybrid fuzzers can pass check ⓑ.
- Hybrid fuzzers tend to satisfy check ⓒ by decreasing lhs, which fails check ⓓ

符号执行无法意识到cd_size与CDH实际长度的匹配关系




```
01 file_size = get_file_size(input);
02 while (memcmp(input + eocd_off, MAGIC_EOCD, 4) != 0)
03   eocd_off++;
04 if (eocd_off > file_size) error();
```
ⓐ check magic number

```
05 int tail_len = count_remaining(input);
06 if (cmt_len != tail_len) error();
```
check length ⓑ

```
07 if (cd_size + cd_off > eocd_off) error();
```
check offset ⓒ

```
08 while(cd_size > 0)
{
09 if(cd_size < MIN_CDH_SIZE) error();
10 if(memcmp(buf_cd_off, MAGIC_CDH, 4)) error();
11 cd_size -= entry[i].size;
12 if(++i == nentry && cd_size != 0) error();
}
```
convoluted check ⓓ

Overview

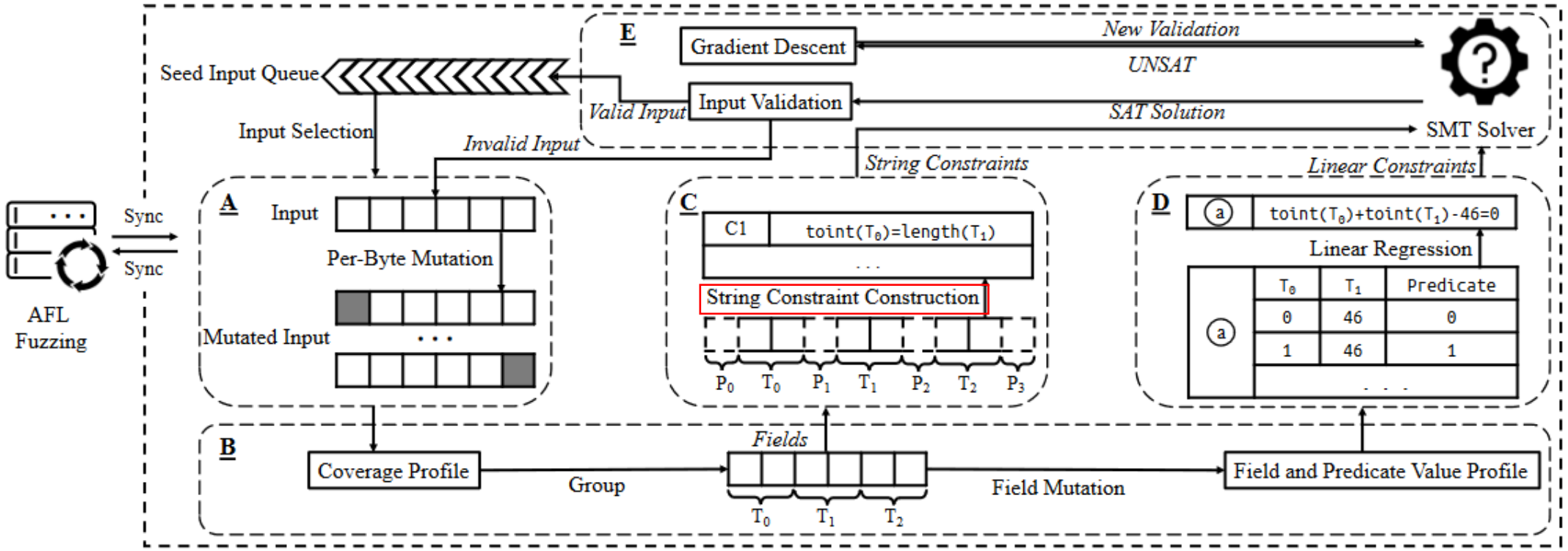
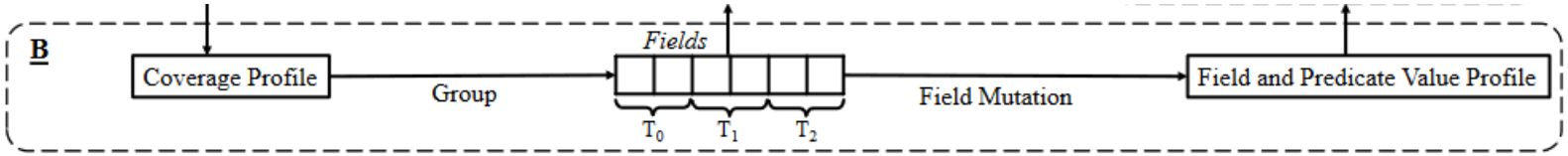


Figure 4: Overview of TensileFuzz.

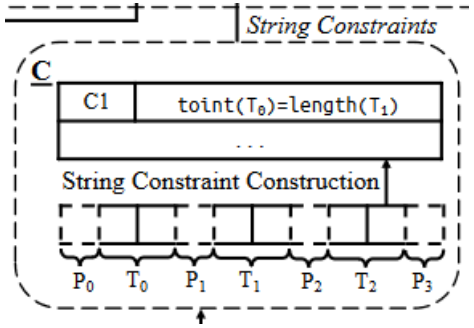
Field Probing 字段探测



利用SLF提出的技术：

- 使用初始输入执行目标程序
- 它对输入的每个字节进行变异，以生成变异输入列表，**每个输入恰好有一个变异字节**
- 投喂变异输入，基于fuzzing engine，观察每个predicate（控制执行流的条件）的左值lhs和右值rhs
- 引起**同一组predicate**发生变化的**连续字节**被分组到**一个字段**中。

String Constraint Construction 字符串约束构造



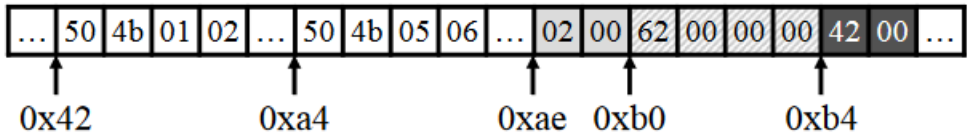
目的：推导出一组对**input growth**很重要的约束

- 包括字段之间的**长度**、**偏移量**和**计数**关系

基于string solver，可以改变字符串的长度，而不是固定长度

Ti: 由Field Probing得到的原始字段；**Pi**: 填充字符串；**S**: 整个字符串

相关操作：length（长度），offset（偏移），cnt（计数），toint（字符串对应的数值），和concat（连接）



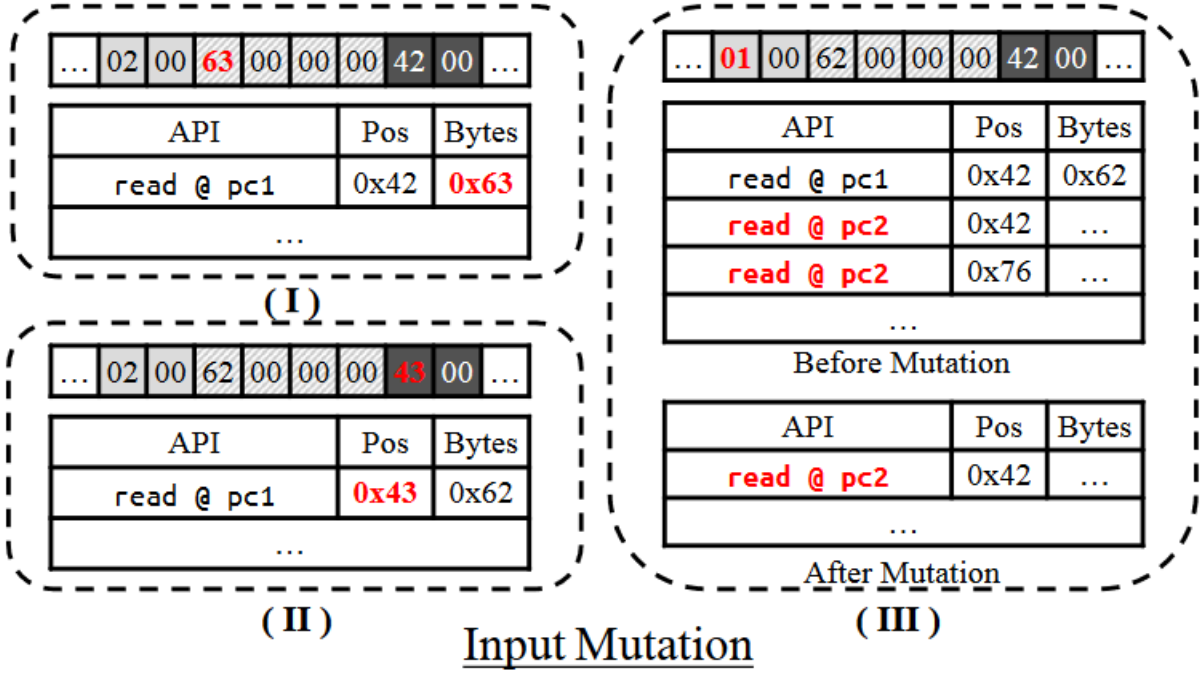
Input

(I) $\text{toint}(T_{0xb0\text{-}0xb3}) = \text{length}(P_{0x42\text{-}0x45}+T_{0x42\text{-}0x45}+\ldots+P_{0xa0\text{-}0xa3}+T_{0xa0\text{-}0xa3}+P_{0xa4\text{-}0xa7})$

(II) $\text{toint}(T_{0xb4\text{-}0xb7}) = \text{offset}(S, P_{0x42\text{-}0x45}+T_{0x42\text{-}0x45}+\ldots+P_{0xa0\text{-}0xa3}+T_{0xa0\text{-}0xa3}+P_{0xa4\text{-}0xa7})$

(III) $\text{toint}(T_{0xae\text{-}0xaf}) = \text{cnt}(P_{0x42\text{-}0x45}+T_{0x42\text{-}0x45}+\ldots+P_{0xa0\text{-}0xa3}+T_{0xa0\text{-}0xa3}+P_{0xa4\text{-}0xa7}, P_{0x42\text{-}0x45}+T_{0x42\text{-}0x45}+\ldots+P_{0x72\text{-}0x75}+T_{0x72\text{-}0x75})$

Constraints

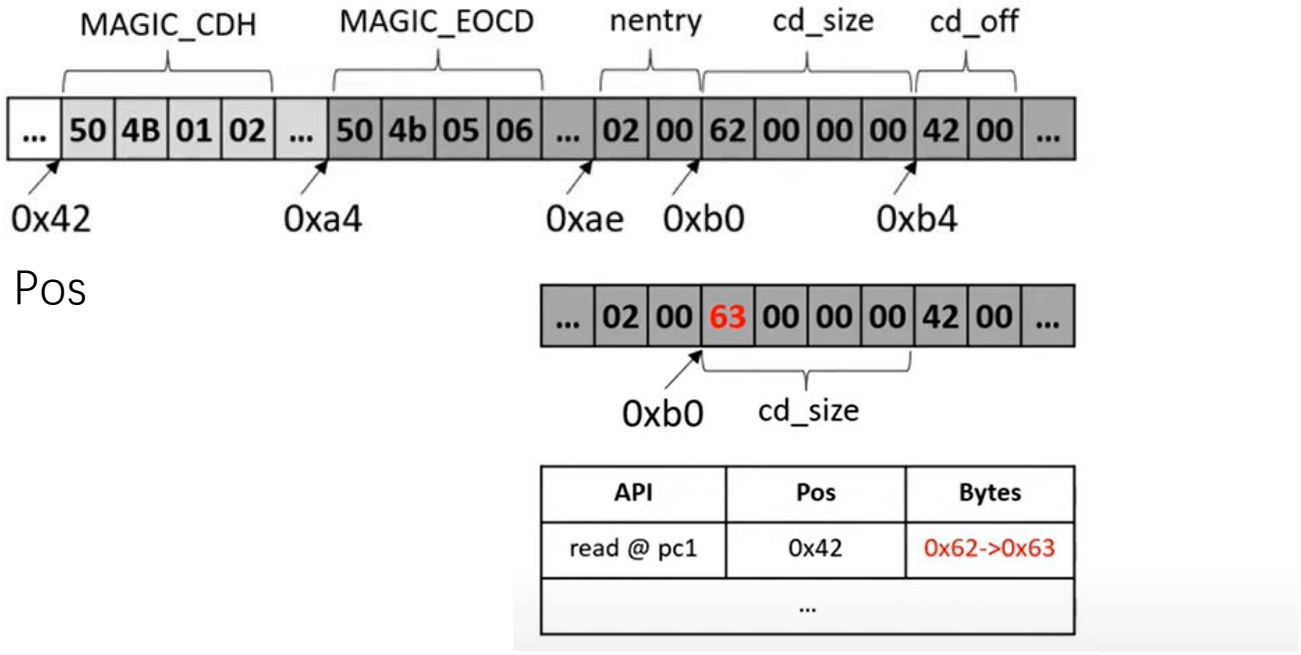


Input Mutation

String Constraint Construction 字符串约束构造

(I) Length Constraints Across Fields 格式: $toint(T_0) = length(T_1)$

- 截取file read API
  - 假设所有这些操作都采用一个表示读取起始位置的**参数Pos**和一个要读取的**字节数参数Bytes**
- 将$toint\ (Ti\ )$分别增加1和2，观察read的参数Bytes的变化



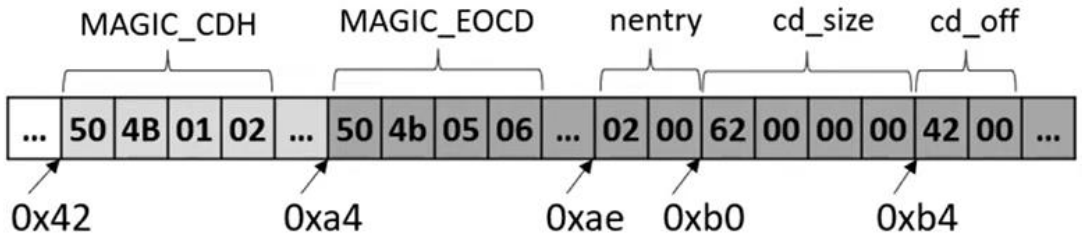
(I) $toint(T_{0xb0\text{-}0xb3}) = length(P_{0x42\text{-}0x45}+T_{0x42\text{-}0x45}+\ldots+P_{0xa0\text{-}0xa3}+T_{0xa0\text{-}0xa3}+P_{0xa4\text{-}0xa7})$

String Constraint Construction 字符串约束构造

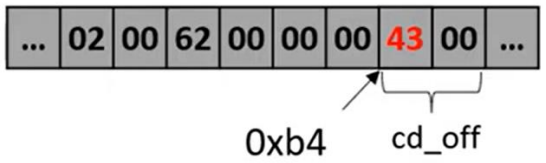
(II) Offset Constraints Across Fields　格式：$toint(T_0) = indexof(T_1, T_2)$

- 一个**偏移字段**的值可能不等同于read API中的**起始位置参数**s，但s是偏移字段T0的线性函数：

$$s = a \times toint(T_0) + b$$



将$toint\ (Ti\ )$分别增加1和2，观察read的参数Pos的变化



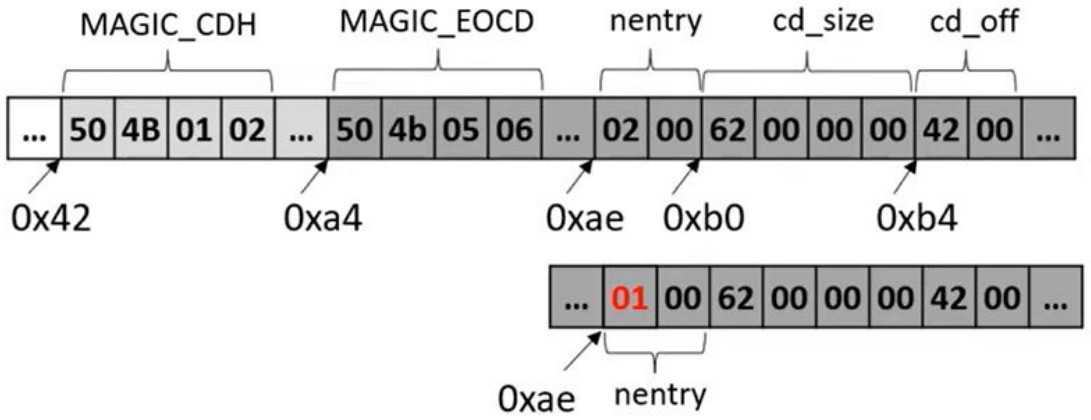
API	Pos	Bytes
read @ pc1	0x42->0x43	0x63
...		

(II) $toint(T_{0xb4\text{-}0xb7}) =$
$offset(S, P_{0x42\text{-}0x45}+T_{0x42\text{-}0x45}+\ldots+P_{0xa0\text{-}0xa3}+T_{0xa0\text{-}0xa3}+P_{0xa4\text{-}0xa7})$

String Constraint Construction 字符串约束构造

(III) Count Constraints Across Fields 格式： $toint(T_0) = cnt(T_1, T_2)$

- 改变一个**计数字段**，会**影响**与相应数据字段相关的文件**读取次数**
- 将$toint\ (Ti\ )$分别变为1和0，观察read次数的变化



API	Pos	Bytes
read @ pc2	0x42	...
read @ pc2	0x76	...
...		

API	Pos	Bytes
read @ pc2	0x42	...
...		

(III) $toint(T_{0xae\text{-}0xaf}) =$

$cnt(P_{0x42\text{-}0x45}+T_{0x42\text{-}0x45}+\ldots+P_{0xa0\text{-}0xa3}+T_{0xa0\text{-}0xa3}+P_{0xa4\text{-}0xa7},$

$P_{0x42\text{-}0x45}+T_{0x42\text{-}0x45}+\ldots+P_{0x72\text{-}0x75}+T_{0x72\text{-}0x75})$

Constraints from Program Paths 路径约束

仅字符串约束不足以促进程序路径探索。

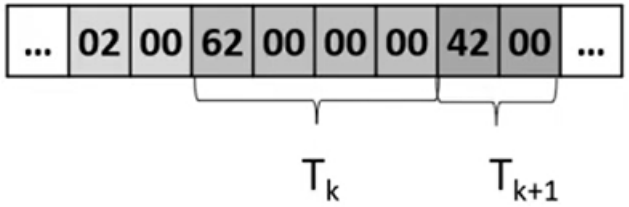
引入路径约束：处理程序执行路径的变化。

提出不需要进行重量级的指令级跟踪或建模（如污点分析）的方法：

Predicate: p

假设$lhs(p)$与一系列字段线性相关：

对于与 p 相关的每个字段 Ti，我们对其进行几次变异，并观察 $lhs(p)$ 的变化



Predicate	Sample Values			
	T_k	T_{k+1}	lhs	rhs
ⓓ ≤	0x62	0x42	0xa4	0xa4
	0x63	0x42	0xa5	0xa4
	0x63	0x43	0xa6	0xa4
...	...			

$$lhs\ (p) = a_1 toint\ (T_k) + a_2 toint\ (T_{k+1}) + b$$

$$rhs\ (p) = \text{const}$$

$$\text{toint}(T_k) + \text{toint}(T_{k+1}) \leq 0xa4$$

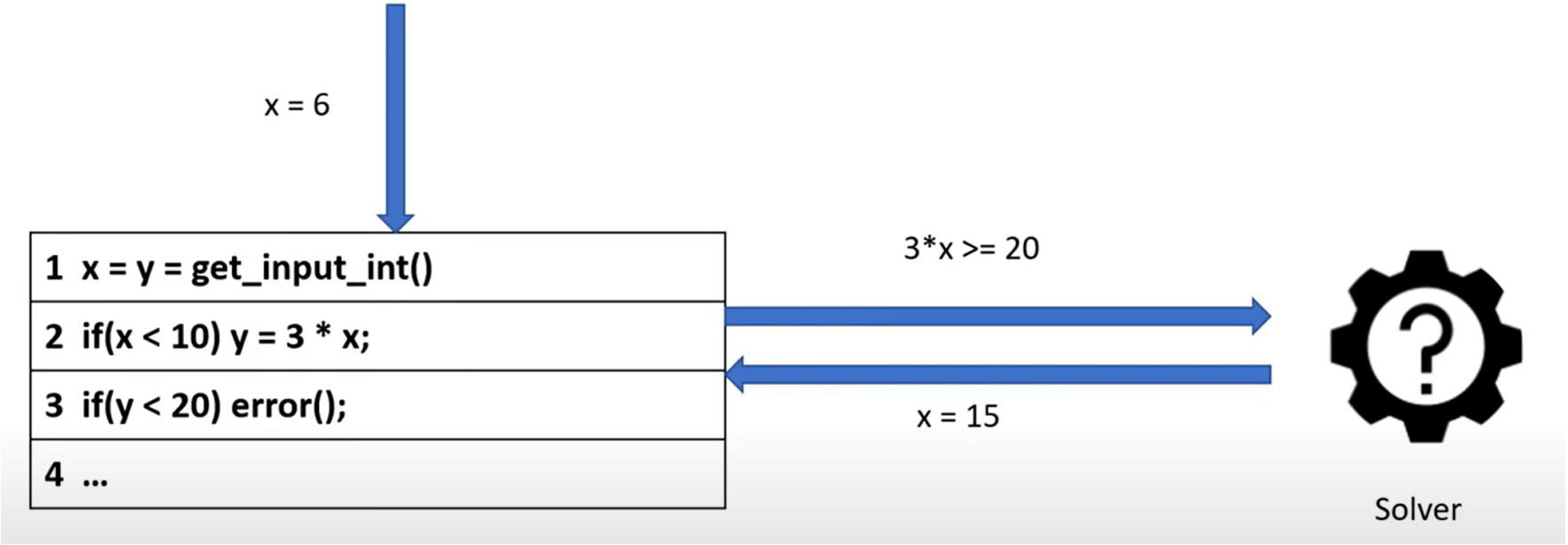
b 是来自具有非线性相关性字段contribution的汇总，可以通过线性回归确定

Constraints from Program Paths 路径约束

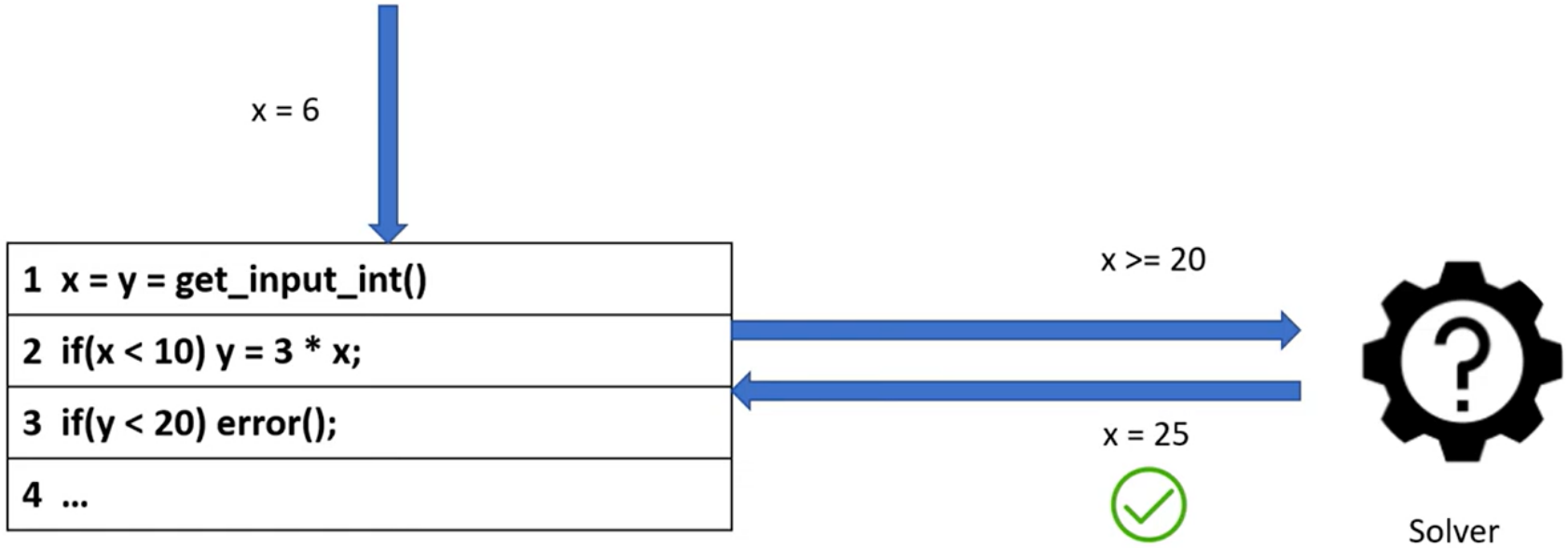
Incompleteness：没有显式地建模某些程序行为（例如符号数组索引和高阶函数）

Unsoundness：例

- 假设 x 的值为 6
- 通过一些额外的变异，例如 x = 7 和 x = 8，
- 如果预期是前往第四行，路径是第二行True，第三行False，则求出一个线性约束 3 * x >= 20
  - 而y是关于x的分段函数

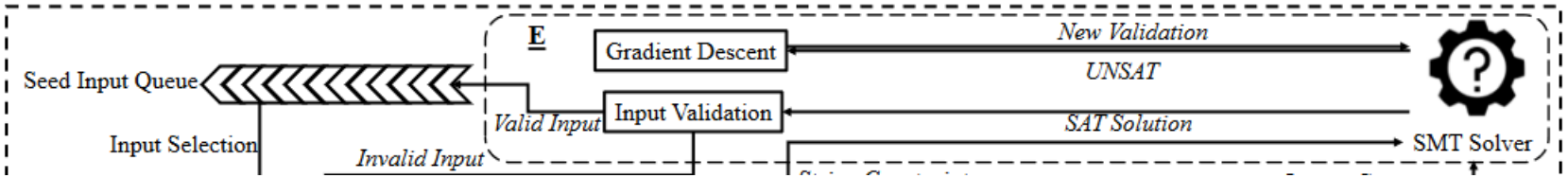


Iterative Constraint Solving 迭代约束求解



使用新输入，重新进行：字段探测（field probing） 和 约束推导（ constraint derivation ），直到产生有效解;

或没有新关系被发现：通常是因为没有建模**非线性关系**，则利用多目标梯度下降法来生成新的输入值



Evaluation

Benchmark:

- 12 real world programs
- Google Fuzzer Test Suite (Google FTS)

Baselines:

- Vanilla Fuzzer: AFL
- Hybrid Fuzzer: Qsym
- Gradient Guided Fuzzers: Angora and SLF

fuzzing without source code or any valid seeds

所有工具都从4字节的空种子输入开始

Experiments:

- Path coverage and seed generation
- Time to reach the buggy point

Evaluation: Real World Applications

Table 1: Evaluation for real world application

Program	AFL		Qsym		Angora		SLF		TensileFuzz	
	Path	Seed	Path	Seed	Path	Seed	Path	Seed	Path	Seed
libzip	172	0	316	0	337	0	381	6	642	19
exiv2	1230	843	1886	1316	1468	826	1413	1101	2105	1958
giflib	167	0	180	17	228	16	276	26	328	53
libpng	375	0	777	0	527	0	674	0	1046	1
libtiff	1011	62	977	69	1060	78	1173	112	1613	114
openjpeg	853	113	860	663	783	537	858	577	988	652
lame	1269	1073	1614	1512	2114	1976	1320	1172	1372	1260
odt2txt	243	0	420	0	279	0	342	0	637	0
7z	1035	1	1123	2	/	/	1307	4	1519	8
readelf	4119	3662	5138	4600	5857	1056	5131	4521	6758	5826
objdump	650	193	1372	371	1451	369	981	324	1627	897
size	1060	323	1159	432	1460	514	1215	479	1587	659

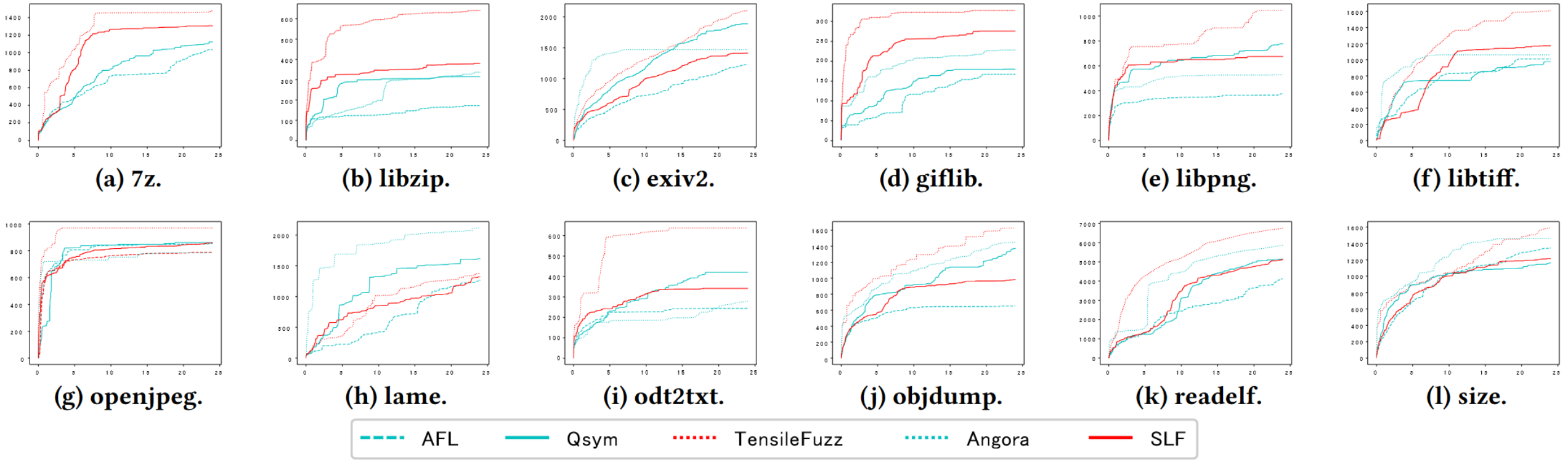


Figure 9: Path coverage. X-axis: time (bounded at 24 hours), Y-axis: the number of unique paths.

- AFL很快到达了极限
- Angora和SLF这样使用基于梯度方法的工具，在解决CRC等算术约束方面表现出色
- TensileFuzz在几乎所有测试应用中实现了最高的路径覆盖率
  - 在需要专门解析的程序中表现不佳:
    - 如“odt2txt”需要XML解析，或像“lame”程序那样有bit level growth related constraints

Evaluation: Growth related constraint identification

Table 2: Growth related constraint identification. TP denotes True Positive, FP denotes False Positive. A., S., and T. denotes AFL, SLF, and TensileFuzz, respectively.

Program	Total	TP			FP			Program	Total	TP			FP		
		A.	S.	T.	A.	S.	T.			A.	S.	T.	A.	S.	T.
libzip	10	0	10	7	0	23	1	lame	2	0	0	0	0	0	0
exiv2	2	1	0	1	9	0	0	odt2txt	10	1	0	6	0	0	0
giflib	1	0	0	1	2	0	0	7z	6	1	0	4	0	0	2
libpng	2	2	0	2	0	0	0	readelf	11	1	0	4	8	2	0
libtiff	2	1	0	1	6	0	0	objdump	11	1	0	4	16	0	0
openjpeg	3	2	0	3	2	0	0	size	11	1	0	4	16	0	0

Total： the number of **growth related relations** in a valid seed.

TP and FP show how many **fields** are correctly or mistakenly identified, respectively.

它错过了一些情况，包括探测中固有的不确定性（文件读取 API 的更改不是由于字段扰动引起的）

以及缺乏位字段的支持

Evaluation: Contribution Breakdown

each query consisting of a set of constraints denoting the intention of exploring some selected branch while respecting the growth related conditions

Table 3: Contribution breakdown. SLS, GDS, and ICS denote String & Linear Solving, Gradient Descent Solving, and Iterative Constraint Solving, respectively.

Program	SLS	GDS	ICS	Program	SLS	GDS	ICS
libzip	88.98%	4.72%	6.30%	lame	21.31%	67.21%	11.48%
exiv2	84.55%	10.91%	4.55%	odt2txt	90.85%	5.99%	3.17%
giflib	54.76%	35.71%	9.52%	7z	63.87%	27.99%	8.14%
libpng	93.48%	6.52%	0.00%	readelf	75.28%	22.47%	2.25%
libtiff	63.76%	29.53%	6.71%	objdump	76.67%	23.09%	0.23%
openjpeg	26.09%	53.62%	20.29%	size	79.11%	20.67%	0.22%
Overall	SLS: 68.23%			GDS: 25.70%		ICS: 6.06%	

对于某些应用程序（例如 openjpeg、lame），迭代约束求解起着重要作用。

原因是这些应用程序包含合理数量的相互依赖的predicate，这可能会导致字符串和线性求解中的场探测失败，以及梯度下降求解中的梯度消失。

迭代约束求解在一定程度上弥补了其他两个求解组件的unsoundness and incompleteness。

Evaluation: Case Study

```
1  (offset,width,height,bitcount) = get_input_ints();
2  length = width * height * bitcount / 8;
3  if (read( length, offset ) != length) error();
4  if (width<MIN_WIDTH || height<MIN_HEIGHT) error();
5  if (bitcount != 24) error();
```

Figure 10: Checks for openjpeg.

对BMP文件进行验证检查的部分内容。

1: 从输入中读取offset和length相关字段

3: 读取offset指定位置处的一些字节

- 由于从四字节种子开始，因此种子的长度通常很小。许多fuzzers无法读取足够的字节。
- 如果length不够大，通过第3行的唯一方法就是使offset很小，这对于符号执行来说优先级较低。
  - 因此， hybrid fuzzers依靠随机突变来增加种子的长度，然后通过此检查。

4: length相关字段的检查

5: 检查位数是否为 24

Evaluation: Google Fuzzer Test Suite

Google fuzzer-test-suite 包含了24个不同类别的程序，

其中有8个使用table-based parsers，不在考虑范围内；(如gcc, 代码覆盖几乎没有透露有关输入约束的信息)

wpantund没有被标记任何buggy位置，因此被排除在外。

Table 4: Evaluation for Google Fuzzing Test Suite. T/O denotes tool cannot reach locations in 24 hours. / denotes tool cannot work on the program.

Program	Location	Tool				
		AFL	Qsym	Angora	SLF	TensileFuzz
libjpeg	jdmarker.c:659	1245.26	12.07	/	378.8	403.67
woff2	woff2_dec.cc:500	T/O	T/O	T/O	T/O	900.79
	woff2_dec.cc:1274	45.26	14.80	10.53	37.08	5.35
llvm-libcxxabi	first crash	47.25	46.37	42.00	49.53	48.06
vorbis	codebook.c:479	T/O	T/O	/	T/O	1206.00
	codebook.c:407	T/O	T/O	/	T/O	T/O
	res0.c:690	T/O	T/O	/	T/O	T/O
libpng	png.c:1035	187.23	812.76	0.87	273.00	1.72
	pngread.c:757	T/O	T/O	T/O	37.61	36.27
	pngrutil.c:1393	T/O	836.63	T/O	T/O	473.20
	pngread.c:738	190.65	822.40	19.22	29.40	15.43
	pngrutil.c:3182	T/O	T/O	T/O	T/O	55.47
	pngrutil.c:139	0.25	0.45	0.45	0.33	0.30
libarchive	archive...warc.c:537	T/O	T/O	T/O	T/O	73.96

没有任何工具可以到达 剩余15 个程序中的 9 个程序的任何 buggy位置：

- 都不适用于网络协议程序（如 libssh、openssl、boringssl 和 c-ares）
  - 难以符号执行的built-in arrays
  - 多个相互依赖的非线性predicates，阻止了梯度搜索。
  - TensileFuzz 降级为 Angora，因为这些程序没有字符串约束，并且有许多相互依赖的非线性约束。
- 对于其他程序（例如 lcms、harfbuzz 和 freetype），它们接受许多不同类型的输入。如果没有diverse seeds的引导，想要到达特定位置是相当困难的。

Limitation

1. **字段识别限制**：TensileFuzz通过将连续字节分组来识别输入字段，因此它只适用于基于块的二进制程序，而不适用于基于文本或语法的程序。
2. **约束建模**：TensileFuzz构建的约束通常是欠约束的，因为它只建模字符串约束和线性路径约束。
3. **非线性约束处理**：缺失的非线性约束由外部的梯度下降引擎处理，因此在程序包含许多复杂的位级或非线性检查时，TensileFuzz的性能会下降。
4. **文件I/O API限制**：一些程序可能不会多次调用文件I/O API，而是将整个文件读取到缓冲区并处理缓冲区。在这种情况下，TensileFuzz不能依赖file read API来构建字符串约束，而是应用启发式规则来推断字段之间的约束，这可能导致字符串约束的准确性降低。

Conclusion

- 他们开发了一种新型的变异模糊测试种子输入生成技术，该技术可以在尊重输入字段间内在约束的同时自动增长输入。
- 他们提出了一种新颖的观点，将二进制字段视为字符串变量，并明确地将重要的跨字段约束建模为字符串约束，以及将字段间的线性关系作为路径约束。
- 他们开发了一个原型TensileFuzz，并在12个真实世界应用程序和Google fuzzer-test-suite基准测试上对其进行了评估。他们的结果显示，与其他工具相比，他们的覆盖率提高了39%-98%。

- https://github.com/TensileFuzz/TensileFuzz
  - 有仓库，但没开源