JOE ROZNER / @JROZNER

RE-TARGETABLE GRAMMAR BASED TEST CASE GENERATION

TESTING PARSERS IS HARD

HOW WE GOT HERE

- Mostly black box (ish) implementation of complex languages (context-free-ish)
- ~35k lines of grammar in total (ANTLR)
- Implemented from incomplete, inaccurate, and contradictory documentation
- Radically different parsing algorithm(s) from original implementations
- Lack of public test cases for most dialects

PROBLEM AREAS FOR PARSING

OVERFIT/UNDERFIT GRAMMAR DEFINITION

- Does the parser accurately recognize the language?
- Lack of access to real grammars
- Poor documentation
- Differences in parsing algorithms means differences in grammar definition
- Ambiguity, recursion, precedence differences
- Universally proving CFG equivalence is undecidable

TREE GENERATION FLAWS

- Does the parse tree accurately represent the sentence?
- Shows the syntactic relationship between tokens
- Most parser generators require manual tree construction
- Typically this stage translates string data to it’s real type representation

```
lexer_body_part: ast::Operation = {
    <r:UpperId> <q:Quantifier*> => ast::unroll_quantifier(q, ast::Operation::Token(r)),
    <l:StringLiteral> ".." <r:StringLiteral> => ast::Operation::Range((l, r)),
    <r:StringLiteral> <q:Quantifier*> => ast::unroll_quantifier(q, ast::Operation::StringLiteral(r)),
    "(" <r:LexerBody> ")" <q:Quantifier*> => ast::unroll_quantifier(q, ast::Operation::Group(r)),
    "." <q:Quantifier*> => ast::unroll_quantifier(q, ast::Operation::Any),
    CharacterClass <q:Quantifier*> => ast::unroll_quantifier(q, ast::Operation::CharacterClass),
};
```

UNSAFE/INCORRECT VALIDATED INPUT

- Have we validated that the input is safe and correct?
- Correct parsing proves validity but doesn’t ensure future proper handling
- Syntactic/Semantic correctness doesn’t ensure safety
- Opaque handling of tokens is fairly common due to language complexity
- Once you’re past the parser it’s back to smashing the stack/logic flaws/etc

HOW DO WE TEST THIS?

GETTING MORE TEST CASES

- Write by hand
- Crawl the web/open source project pulling out examples
- Automatically generate test cases with a fuzzer

STYLES OF FUZZING

INSTRUMENTATION + RANDOM MUTATION

- Focus on path exploration and code coverage
- No concept of syntax/semantics
- Wont necessarily provide lot’s of coverage for variations of a specific parse tree
- Might spend a lot of time on uninteresting/non-relevant code paths
- Not immediately clear how to build a proper test harness
- Example of this strategy is AFL (American Fuzzy Lop)
- https://lcamtuf.blogspot.com/2014/11/pulling-jpegs-out-of-thin-air.html

“THE FIRST IMAGE, HIT AFTER ABOUT SIX HOURS ON AN 8-CORE SYSTEM. . . ”

“. . . CERTAIN TYPES OF ATOMICALLY EXECUTED CHECKS WITH A LARGE SEARCH SPACE MAY POSE AN INSURMOUNTABLE OBSTACLE TO THE FUZZER. . . ”

```
if (strcmp(header.magic_password, "h4ck3d by p1gZ”))
   goto terminate_now;
```

“IN PRACTICAL TERMS, THIS MEANS THAT AFL-FUZZ WON'T HAVE AS MUCH LUCK ‘INVENTING’ PNG FILES OR NON-TRIVIAL HTML DOCUMENTS FROM SCRATCH…”

INSTRUMENTATION + SOLVING

- Focus on path exploration and code coverage
- Instrument the code and solve for new paths
- Still doesn’t care about syntax/semantics
- Still not clear how to build a more customer test harness
- Not necessarily easy to gate off specific paths that are uninteresting
- Example of this is KLEE

GRAMMAR BASED

- Uses a grammar to generate syntactically correct sentences
- Typically provide their own grammar language
- Mostly targeted at regular/context-free text based languages
- Example of this is Mozilla Dharma
- https://github.com/MozillaSecurity/dharma

PROBLEMS WITH TRADITIONAL TEST CASE GENERATION

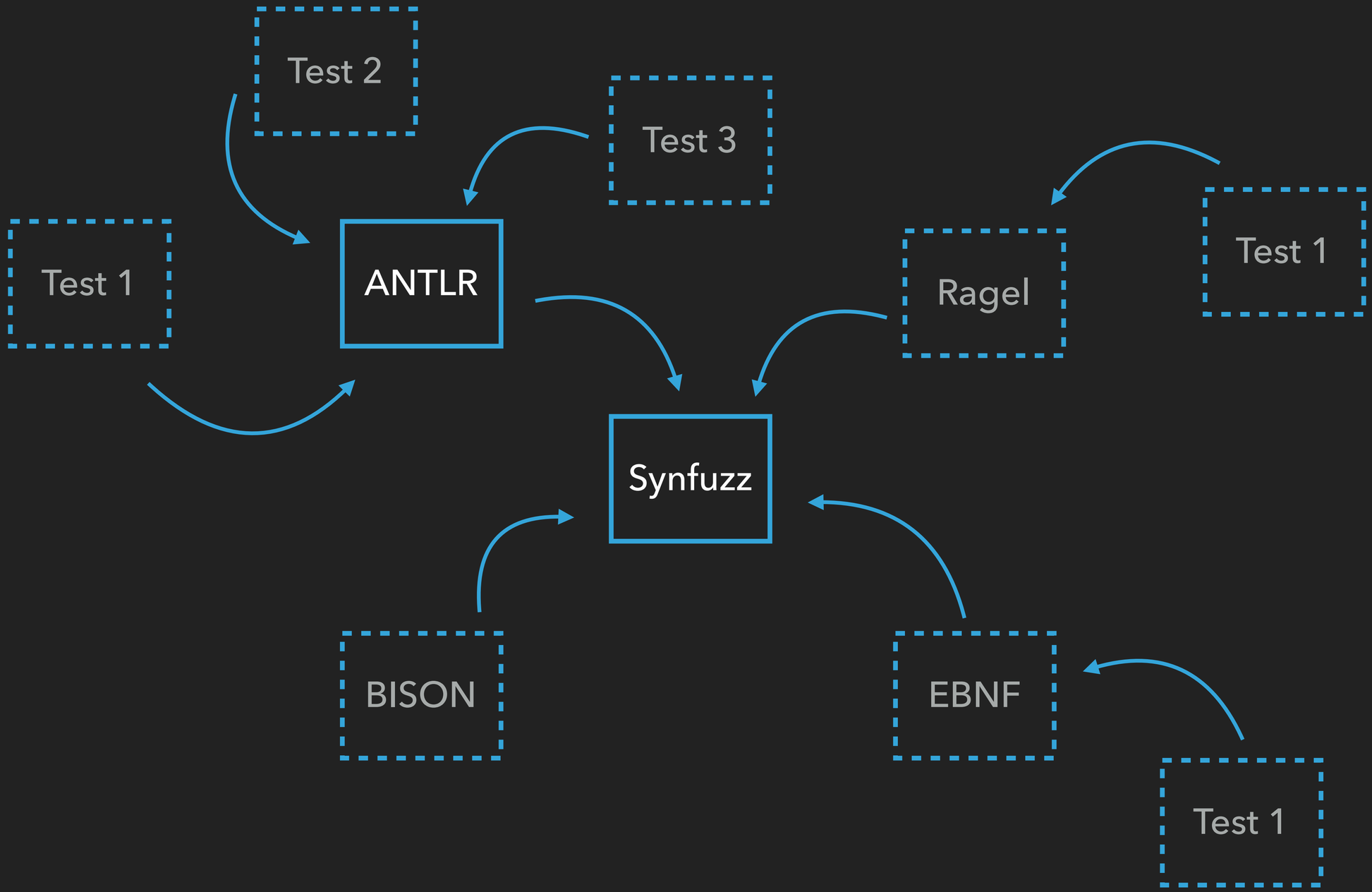
- Inflexibility with using test cases
- Inflexibility with providing feedback
- Existing tools solve many cases but as you deviate they become less useful

HOW CAN WE DO BETTER?

- Easy to build flexible test harnesses
- Directly use grammar definition without manual translation
- Expressive enough for regular, context-free, and context-sensitive languages both text and binary
- Embeddable into and usable from any language
- As much code re-use as possible to avoid duplication

A GRAMMAR BASED TEST CASE GENERATION PLATFORM

SYNFUZZ


Test 2
Test 3
Test 1
ANTLR
Ragel
Test 1
Synfuzz
BISON
EBNF
Test 1

Values	Quantification	Logical	Grouping
CharLiteral	RepeatN	Choice	Sequence
Byte	Many	Not	JoinWith
String	Many1		SepBy
CharRange	Range		SepBy1
	Optional		

```
let mut f = File::open("bnf.g4").unwrap();
let mut buf = String::new();
f.read_to_string(&mut buf).unwrap();

let rules = antlr4::generate_rules(&buf).unwrap();

let r = rules.read().unwrap();
let root = r.get("rulelist").unwrap();
let generated = root.generate();
let s = String::from_utf8_lossy(&generated);
println!("{}", s);
```

DEMO

DESIGNING TEST HARNESSES

DOES IT CRASH?

1. Start process
2. Generate input and feed it
3. Listen for SIGSEGV/SIGABRT

OVERFIT

1. Generate test case
2. Find oracle that specifies whether a syntax or runtime error
3. Feed test case
4. Categorization
    1. If failure and syntax error parser is overfit
    2. If failure and runtime error may or may not be overfit

```
mysql> select * from a where id ^^^^^ 3;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
 corresponds to your MySQL server version for the right syntax to use near '^^^^ 3' at line 1
```

```
mysql> select * from a where fakecolumn = 3;
ERROR 1146 (42S02): Table 'mysql.a' doesn't exist
```

UNDERFIT

1. Generate test case from reference implementation grammar
2. Parse with re-implementation
3. Categorization
    1. If fails re-implementation is underfit
    2. If succeeds re-implementation is correct

WHAT'S READY TODAY

- Combinator library for regular and context-free grammars
- ANTLR4 frontend
- https://www.github.com/jrozner/synfuzz
- https://www.github.com/jrozner/rust-antlr4

WHAT'S NEXT?

- Cycle detection + forced progression
- Expose a C-compatible API + language bindings
- Better negation logic
- Context-Sensitive/Introspective generators
- Bit level values
- Additional frontends
- Grammar coverage information

JOE@DEADBYTES.NET / @JROZNER

HTTPS://WWW.GITHUB.COM/JROZNER/SYNFUZZ

QUESTIONS?