

M3 - Grammar Fuzzing

Guillermo Polito

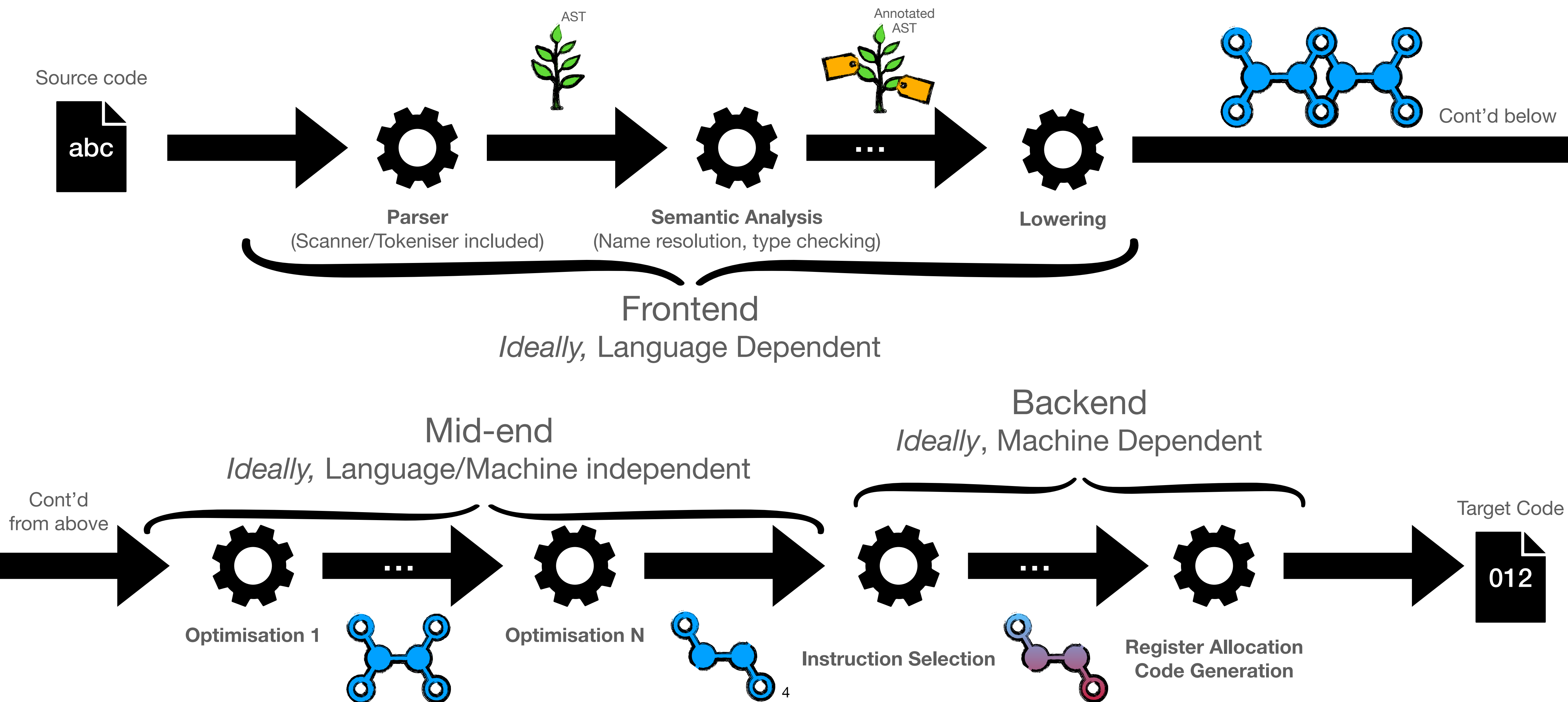
ECI'23 - Universidad de Buenos Aires

Goals

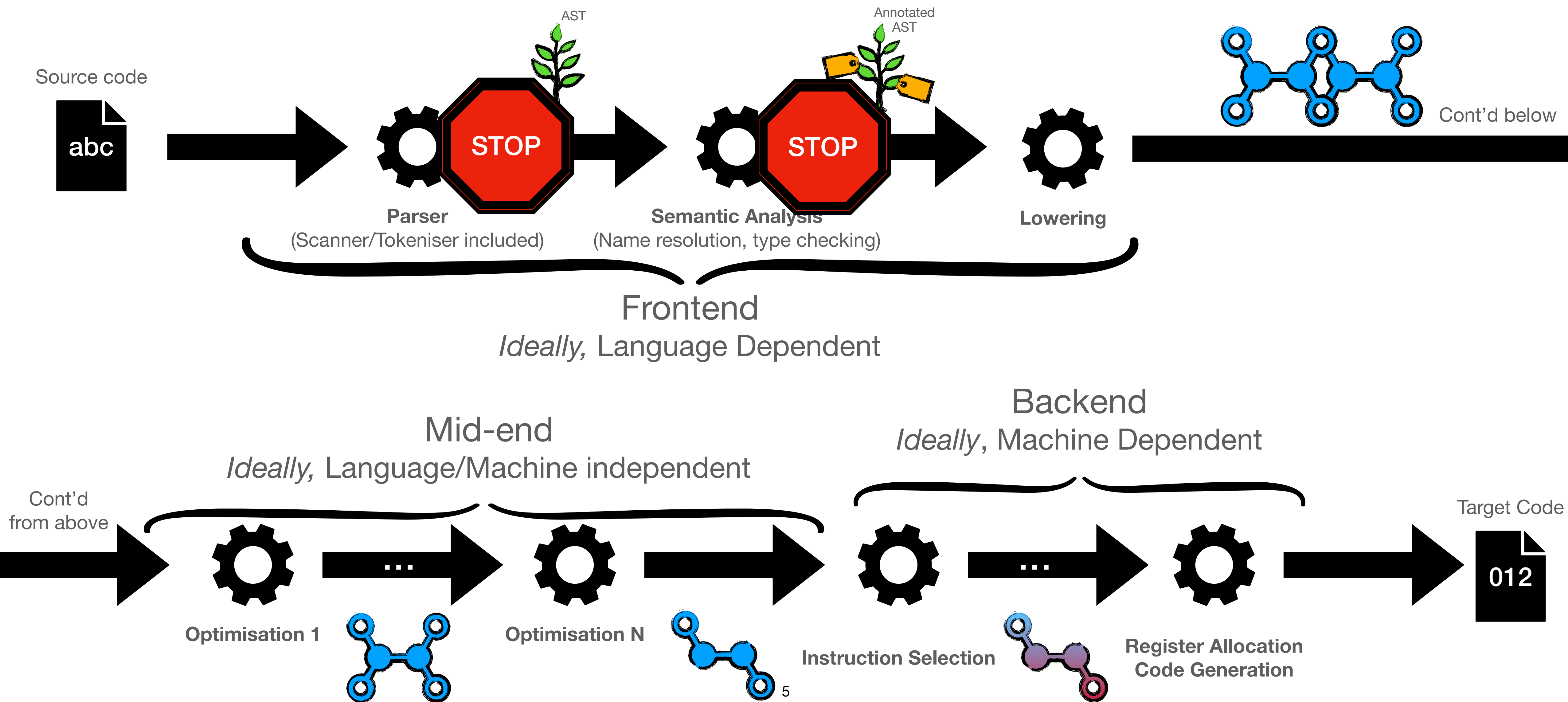
- Compilers phases act as barriers for inputs
- Syntactically valid inputs go further in the pipeline
- Grammars can be used to generate syntactically valid inputs

How can we fuzz compilers?

Modern Optimizing Compiler Phases



Bugs as Barriers



**We need to generate syntactically
and semantically valid inputs**

We need to generate **syntactically**
and semantically valid inputs

Remember Fuzzing Date Parser

```
f := PzRandomFuzzer new.  
r := PzBlockRunner on: [ :e | e asDate ].  
r expectedException: DateError.  
f run: r times: 20.
```

- Pharo 11
- String>>asDate
- DateError is an expected error
- 4/20 = 5% of errors

```
PASS "DateError: day is after month ends"  
PASS "28 April 2006"  
PASS "7 September 2029"  
PASS "9 March 1995"  
FAIL "SubscriptOutOfBounds: 73"  
PASS "DateError: day is after month ends"  
FAIL "SubscriptOutOfBounds: 0"  
PASS "DateError: day is after month ends"  
PASS "6 January 2007"  
PASS "9 January 1986"  
FAIL "SubscriptOutOfBounds: 0"  
FAIL "#isAlphaNumeric was sent to nil"  
PASS "DateError: day is after month ends"  
PASS "1 September 1989"  
PASS "DateError: day is after month ends"  
PASS "DateError: day may not be zero or negative"  
PASS "5 January 0228"  
PASS "DateError: day may not be zero or negative"  
PASS "7 September 1996"  
PASS "2 January 2008"
```


Parser was too Permissive

- Some inputs PASS but **do not respect the contract**

"Answer an instance of created from a string with format
mm.dd.yyyy or mm-dd-yyyy or mm/dd/yyyy"

'?(2/=-@=@:4?/(3\$3(8"&,!-2/&6&&' asDate.
>> 4 February 2003

- Parser is too permissive
- Our runner is too permissive too => we should detect this as an error!

Random Inputs Fail Easily

- We could expect to break something with fully random inputs
- This could be solved with **input sanitizing**
- What if we have *almost correct inputs*?
- Looks like a date, cuacks like a date, parses as a date?

Date Fuzzer

```
(1 to: 10) collect: [ :e | PzDateFuzzer new fuzz ]
```

```
23 5  
7/February-6  
7,February0  
0/february/7  
9 february 0  
7 February-9  
February 0,1  
4/February,4  
february/0 7  
1January,8
```

Grammars as Input Descriptions

- Grammars describe languages
- Usually used for parsing purposes, but...
- Key idea => structured fuzzing with grammars

Date Grammar

```
ntNumber --> ntDigit, ntNumber | ntDigit.  
ntDigit  --> ($0 - $9).
```

ntDate

```
--> ntDay, ntSeparator, ntMonth, ntSeparator, ntYear  
| ntMonth, ntSeparator, ntDay, ntSeparator, ntYear  
| ntYear, ntSeparator, ntMonth, ntSeparator, ntDay.  
ntSeparator --> ' ' | ' ' | '-' | ',' | '/'.
```

```
ntDay --> ntNumber.
```

ntMonth

```
--> ntNumber  
| 'january' | 'January'  
| 'february' | 'February'.
```

```
ntYear --> ntNumber.
```

Grammar Fuzzer

```
(1 to: 10) collect: [ :e | (PzGrammarFuzzer on: PzDateGrammar new) fuzz ]
```

```
23 5  
7/February-6  
7,February0  
0/february/7  
9 february 0  
7 February-9  
February 0,1  
4/February,4  
february/0 7  
1January,8
```

Let's test some parser

```
f := PzGrammarFuzzer on: PzDateGrammar new.  
r := PzBlockRunner on: [ :e | e asDate ].  
r expectedException: DateError.  
f run: r times: 20.
```

- Pharo 11
- String>>asDate

```
PASS 3 January 2009  
PASS 9 February 2006  
PASS 4 February 2002  
PASS-FAIL DateError: day may not be zero or negative  
FAIL #isAlphaNumeric was sent to nil  
FAIL Error: Month out of bounds: 26.  
PASS 9 January 2001  
PASS 4 January 2004  
PASS 7 February 2007  
PASS 4 February 2007  
FAIL #isAlphaNumeric was sent to nil  
PASS 4 January 2005  
PASS 8 February 2004  
PASS 8 February 2009  
PASS 8 January 2007  
PASS 3 May 2001  
PASS 7 February 2001  
FAIL #isAlphaNumeric was sent to nil  
PASS-FAIL DateError: day may not be zero or negative  
PASS 5 February 2006
```

Let's get more data

```
f := PzGrammarFuzzer on: PzDateGrammar new.  
r := PzBlockRunner on: [ :e | e asDate ].  
r expectedException: DateError.  
f run: r times: 100.
```

Pass	81 %
Expected-Fail	10 %
Fail	9 %

- Simple Date grammar fuzzing has a high success ratio

Looking at the bugs

- Out of 135 bugs fuzzing 1000 cases

method not understood during parsing	83 %
Out of bounds during parsing	13 %
<i>Validation</i> with generic error during parsing	4 %

Building a Grammar Fuzzer

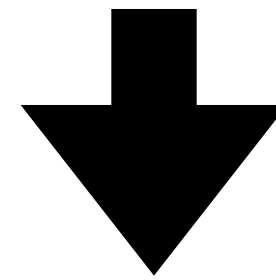
- Example, a number grammar

```
ntNumber --> ntDigit, ntNumber | ntDigit.  
ntDigit  --> ($0 - $9).
```

Desugarising into simple rules

- Example, a number grammar

```
ntNumber --> ntDigit, ntNumber | ntDigit.  
ntDigit  --> ($0 - $9).
```

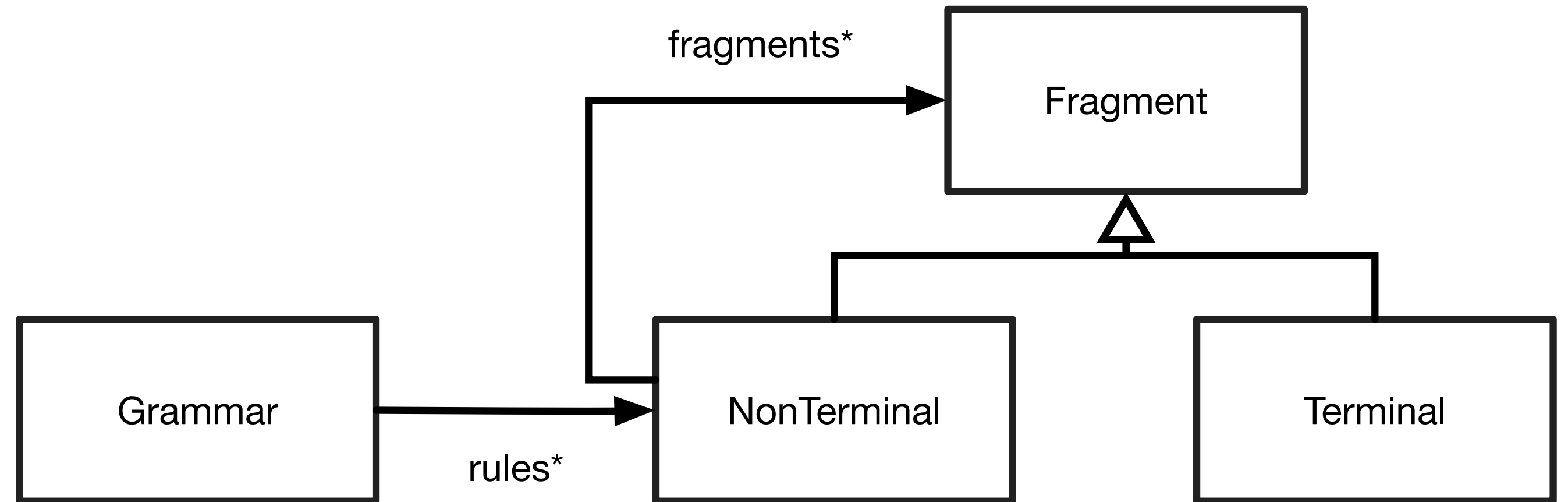


```
ntNumber --> ntDigit, ntNumber  
ntNumber --> ntDigit.  
ntDigit  --> 0.  
ntDigit  --> 1.  
...  
ntDigit  --> 8.  
ntDigit  --> 9.
```

Modelling as a Composite Pattern

- Example, a number grammar

```
ntNumber --> ntDigit, ntNumber
ntNumber --> ntDigit.
ntDigit  --> 0.
ntDigit  --> 1.
...
ntDigit  --> 8.
ntDigit  --> 9.
```



Instantiating the Model

- Example, a number grammar

ntNumber --> ntDigit, ntNumber
ntNumber --> ntDigit.

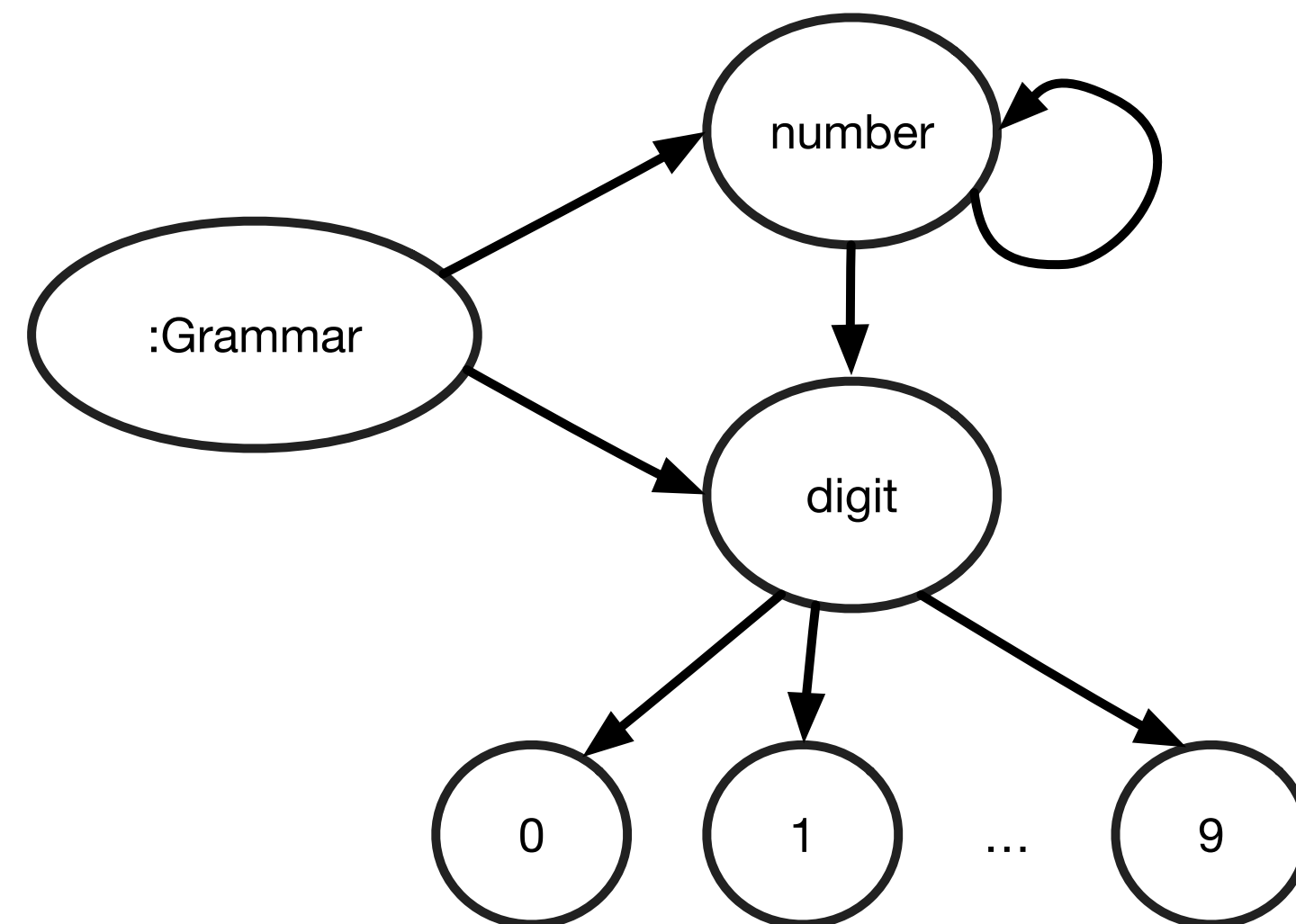
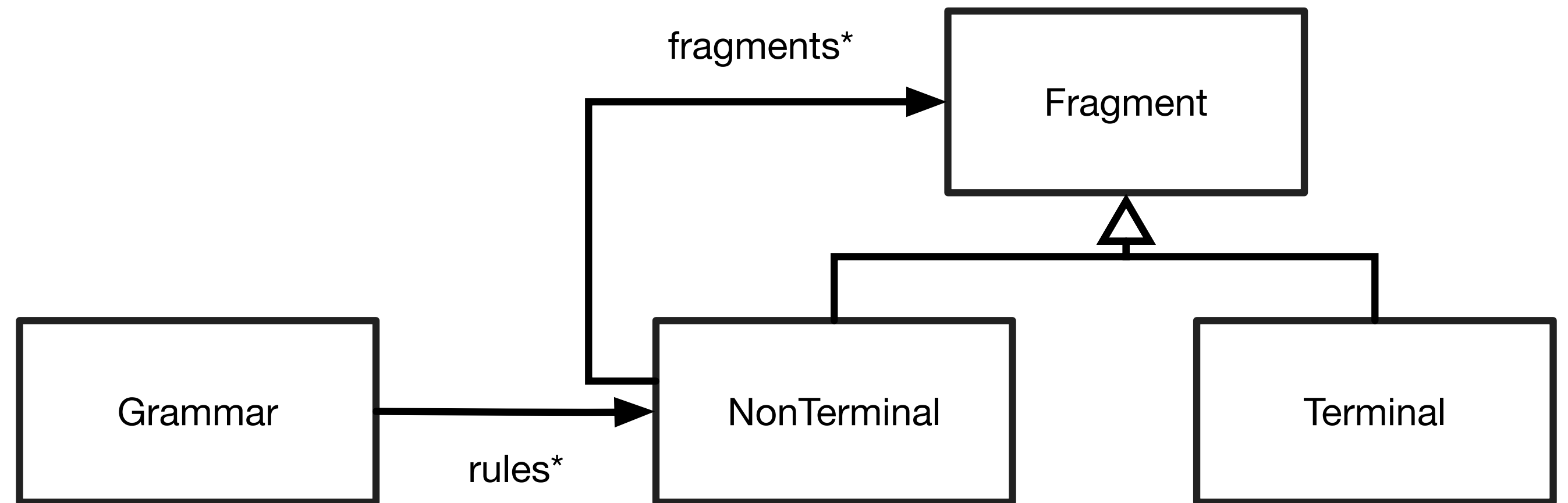
ntDigit --> 0.

ntDigit --> 1.

...

ntDigit --> 8.

ntDigit --> 9.



Generation: Visitor Pattern

```
GrammarFuzzer >> visitRule: rule
```

```
  ^ rule fragments inject: ‘’ into: [ :accum :each |  
    accum , (each visit: self) ].
```

```
GrammarFuzzer >> visitTerminal: terminal
```

```
  ^ terminal
```

```
GrammarFuzzer >> visitNonTerminal: nonTerminal
```

```
  | rule |  
  rule := self selectRule: nonTerminal rules.  
  ^ rule visit: self withSubLevel
```

Generation: Visitor Pattern

```
GrammarFuzzer >> visitRule: rule
```

```
^ rule fragments inject: `` into: [ :accum :each |  
  accum , (each visit: self) ].
```

```
GrammarFuzzer >> visitTerminal: terminal
```

```
^ terminal
```

```
GrammarFuzzer >> visitNonTerminal: nonTerminal
```

```
| rule |  
rule := self selectRule: nonTerminal rules.  
^ rule visit: self withSubLevel
```

Random selection

```
GrammarFuzzer >> selectRule: nonTerminal
```

```
^ nonTerminal rules atRandom
```

Possible Extensions

- *Limiting* the output, by size, by tree depth
- *Guide* grammar generation by
 - adding *weights* to the derivations
 - grammar *coverage*
 - *genetic algorithms*?

Takeaways

- Structured inputs can help
 - *penetrate* complex programs e.g., compilers
 - bypass validations e.g., syntax validations in parsers
- Grammars describe languages
 - not only parsing!
 - but also generation (goes back to '67, '70, '72!)

Material

- The Fuzzing Book. Grammars Chapter. A. Zeller et al
<https://www.fuzzingbook.org/html/Grammars.html>
- Gnocco
<https://github.com/Alamvic/gnocco/>