Project 4 : Grammar Inference

March 16, 2017

1 Grammar Inference

Implement a grammar learner that, using the techniques presented in the lecture, is able to lean grammars from python programs that use `pyparsing` to parse inputs. For each subject your implementation must be able to learn a grammar by observing and analyzing executions with a set of sample inputs. The grammar must be saved as a file in the format specified by the `grammar` subject. For observing the executions you can use the `sys.settrace()` function as shown in the lecture. You can assume that all given sample inputs are valid and will not result in a `ParseException`.

In order to invoke the subject from your fuzzer and measure its coverage, we now recommend loading the subject module dynamically from file and calling its `main(filename)` method directly from your code:

```
import importlib.util as iu
import sys

# load the subject module
spec = iu.spec_from_file_location("arithmeticExpr.arithmetic",
                                  "arithmeticExpr/arithmetic.py")
subj = iu.module_from_spec(spec)
# initialize the subject
spec.loader.exec_module(subj)

# now you can execute the subject by calling its main method
subj.main('path/to/your/generated/input.sample')
```

Notes on Grammar Inference:

As demonstrated in the lecture the easiest way to observe the decomposition of inputs by a `pyparsing` parser is to observe the call hierarchy and return values of relevant functions. Since the parsing functionality is implemented in individual `parseImpl` functions that are provided by all subclasses of `ParseElement` we are able to derive a parse tree by fucussing dynamic analysis on these invocations. The nodes in the derived trees correspond to productions in the learned grammar and they are directly related to individual `ParseElement` instances. Since `pyparsing` implements a strategy that uses backtracking you also need to track exceptions that are thrown by the observed functions in order to be able to accurately discard a partial parse tree that corresponds to a failed parsing attempt for a input fragment.

After deriving parse trees for all sample inputs you can derive the nonterminal symbols of the grammar by identifying the unique `ParseElement` instances corresponding to nodes in the parse trees. The start symbol can be derived by identifying the unique `ParseElement` instance of all root nodes. The unique `ParseElement` instances of a node N and its children imply a production rule where the nonterminal symbol corresponding to N can be substituted by a sequence of the symbols corresponding to its children. More general and precise rules can be inferred when considering the class of the `ParseElement` instances.

Notes on nonterminal names:

In order to derive more meaningful names for nonterminal symbols you can exploit that all our subjects compose the grammar rules in a function `BNF()`. By looking at the frame when this function returns, you can check if a `ParseElement` instance is stored in a local variable and therefore the name of the variable might be useful as a descriptive nonterminal name.

2 Implementation

The subjects for the course projects are hosted as a public project on our Gitlab `https://securitytesting.cispa.saarland/kampmann/subjects`. Please make sure to pull the most recent revision of the subjects from the project (*especially since we added the sample inputs*). Each subject resides in a top level directory and you can invoke it using:

```
python <subject>/<module> <inputfile>
```

Depending on your setup you might need to substitute `python` for `python3` if your system uses Python 2.x by default. The subjects will terminate with a non-zero exit code in case of an error. Additionally to the public subjects we will also evaluate your implementation on two secret subjects and on variants of the public subjects. Your implementation is expected to be accessible as a python module `grammarinference.py` in the root directory of your project repository. The fuzzer is supposed to be invokable as:

```
python grammarinference.py -s <sample-dir>
                          -p <path-to-subject> -m <name-of-subject-module>
```

The `m` and `p` parameters are intended to be passed to the `spec_from_file_location` function. The `s` parameter is the path to the sample directory that contains the samples as individual files ending with `.sample`.

The produced grammar should be written to the working directory as `result.grammar`.

In order to evaluate your implementation we will run the grammar inference on each subject (including the secret subjects) with the provided sample set and a different representative sample set of our choice. We will use a grammar based fuzzer to generate inputs from the grammars learned by your tool and measure the grammar coverage the inputs produce on the original grammar.