DynaPyt: A Dynamic Analysis Framework for Python

By Aryaz Eghbali and Michael Pradel

Nat Bourassa
Shahzeb Faheem Khan
Abhijeet Shah

Problem

- Python is very dynamic in nature
- Lack of universal dynamic analysis tool
- Existing frameworks are mostly ad-hoc and require manual effort and are time-consuming to use

Motivation

- Python is widely used and the second most popular languages in code hosted at GitHub
- Frameworks that exist for other languages do not exist for Python
- Event hierarchy enables the user to specify run-time events of interest and limit overhead

Proposed Solution

- General-purpose framework for dynamic analysis - DynaPyt
- Rich set of 97 hooks - allows developers to inject their analysis logic
  - Function calls
  - Writes of object attributes
  - Control flow decisions
- Enables focused analysis - reduces overhead and runtime impact
- Specific needs met by alteration during analysis phase
- Modular and concise analysis implementation using hierarchically organized hooks
  - Higher level hooks provide broader monitoring of program execution
  - Lower level hooks offer specific points of observation

Evaluation

- How efficient is DynaPyt when instrumenting source code? -> runtime
- Does an instrumented program remain faithful to the semantics of the original program?
- How complex is the implementation of analyses built on top of the framework?
- What runtime overhead does DynaPyt impose when performing analysis?

Q1: How efficient is DynaPyt when instrumenting source code?

- How often new code can be analyzed
- How many analyses can be performed in a given time budget
- Measured how long the instrumenter takes to transform all Python files for the TraceAll analysis
- TraceAll
  - Implementing all hooks

#	Repository	Application domain	Instrument time (mm:ss)	Python files	Lines of Code
1	ansible/ansible	Automation framework	06:59	2,188	176,173
2	django/django	Web framework	14:07	3,603	318,602
3	keras-team/keras	Deep learning framework	05:41	678	155,407
4	pandas-dev/pandas	Data analysis library	12:32	2,727	358,195
5	psf/requests	HTTP library	00:16	54	6,370
6	Textualize/rich	Text tool	00:57	178	24,362
7	scikit-learn/scikit-learn	Machine learning framework	06:52	1,419	180,185
8	scrapy/scrapy	Web crawler	01:49	505	37,181
9	nvbn/thefuck	Commandline tool	01:21	620	12,070

Q2: Does an instrumented program remain faithful to the semantics of the original program?

- Must ensure transformations and hooks do not interfere with semantics
- Instrument and run TraceAll analysis to see how many tests still pass

Failures

Two Reasons:

- Original program makes assertions based on execution stack
- Edge cases which are not handled by DynaPyt

	Passing tests	
#	# without instrumentation	% after instrumentation
1	2,862	99.4%
2	191	98.4%
3	363	99.7%
4	136,898	99.8%
5	374	100.0%
6	545	99.6%
7	9,400	97.8%
8	1,841	99.6%
9	1,798	100.0%

Q3: How complex is the implementation of analyses built on top of the framework?

Name	Description	Analysis hooks	LoC
BranchCoverage	Measures how often each branch gets covered (Listing 1)	1	6
CallGraph	Computes a dynamic call graph	1	19
KeyInList	Warns about performance anti-pattern of linearly search through a list	2	10
MLMemory	Warns about memory leak issues in deep learning code	4	29
SimpleTaint	Taint analysis useful to, e.g., detect SQL injections	7	53
AllEvents	Implements the `runtime_event` analysis hook to trace all events	1	4

Q4: What runtime overhead does DynaPyt impose when performing analysis?

- Run TraceAll which is most expensive
- Run BranchCoverage which only instruments the entry points of control flow statements
- Trace all additions performed by program for a low-overhead analysis
- Result:
  - TraceAll overhead of 1.2x - 16x where Janlangi for JavaScript is about 26x
  - 5.6% -88.6% faster than sys.settrce depending on analysis hooks selected

Novelty

- Offers a wide range of analysis hooks arranged in a hierarchical structure allowing developers to concisely implement analyses
- Features selective instrumentation and execution modification
- Use of an event hierarchy - select events at an appropriate level of abstraction
- Use of pay-per-use - tracks events the analysis is interested in

Assumptions

- The program is deterministic
- The program is expected to terminate within a reasonable timeframe
- The program has a well-defined control flow
- The program allows modification during analysis without causing unexpected behavior
- The program has standard python language features

Limitations

- Limited scope due to python opcodes not being covered by available hooks
- Limited control flow modification, restricts certain analysis capabilities
- Assumed program characteristics
  - Might not work perfectly with programs violating those assumptions
- Limited to programs using standard libraries and libraries with introspection

Practical Significance

- Enhanced Debugging capabilities
- Improved performance analysis
- Security vulnerability assessment
- Research and development

Discussion 1

Support for Concurrent and Parallel Programming

- Mechanisms for tracking and analyzing events across multiple threads or processes.

Discussion 2

External Library Use

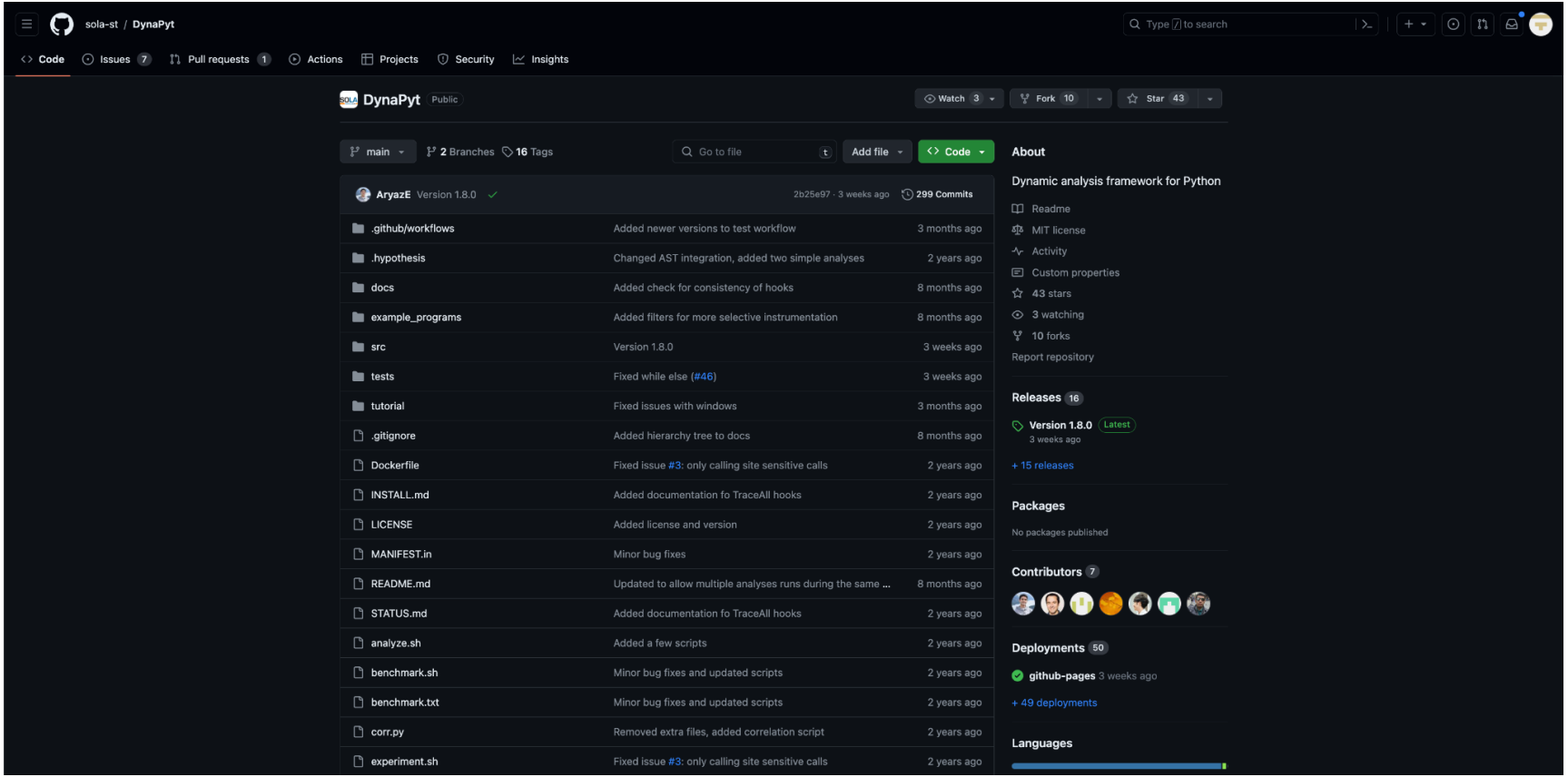
- Difficulties:
  - It can be difficult to predict how external libraries will affect code functionality
  - They may have security vulnerabilities that DynaPyt may miss
  - They may use caching mechanisms which DynaPyt could affect
- Analyze how DynaPyt perform with a wide variety of external libraries
- Increasing monitoring of libraries could increase overhead

Discussion 3

Beyond Debugging and Performance Analysis

- Program Verification (Symbolic Execution)
- Software Testing (Mutation Testing and Dynamic Test Generation)
- Memory Usage Profiling (Leak Detection and Usage Patterns)
- Program Slicing

DynaPyt Github Repository Link: https://github.com/sola-st/DynaPyt



Questions?