Google Summer of Code 2019

Final Work Product

Organization:	**Bazel**
Project:	**Starlark Language**
Mentor:	**Laurent Le Brun**
Author:	**Marwan Tammam**

Project Summary

Starlark language was designed for Bazel. The language has now a specification, 3 implementations (in Java, in Go, in Rust), and is used outside Bazel.

The goal of this project was to ensure the different implementations are in sync. This includes creating a common test suite, identifying and resolving corner-cases, suggesting changes to the language specification, comparing the performance of the implementations (to find performance issues) and improving the interpreter in Java.

Initial Proposal

In my initial proposal, my plan was as follows:

- Create a standard test runner
- Merge existing tests of the 3 implementations and add more tests to increase coverage
- Resolve inconsistencies between implementations and specs
- Add benchmark suite
- Improve the Starlark Interpreter

Contributions

Creating a Common test suite

The common test suite consisted of two main components; a test runner that is responsible for running the tests against the binaries of each implementation and report the results, and the second part is the tests themselves.

Creating the test runner was straight forward since I adapted the same idea of the test runner from the Java implementation and modified it a bit in order to execute the other binaries, starlark-go and starlark-Rust, as well against the test cases.

For the actual tests, I merged all the existing tests of the three implementations. This took some time because there were some variations between each implementation's tests. For example:

- Error messages were not the same, so I had to make sure that the tests accept the different error messages of each implementation.
- Test framework was not the same. Things like signatures of assertion functions and handling tests that expect to fail were not implemented the same way. This required fixing a single test framework and changing all tests to use it.
- The feature set was not the same; some implementations implemented features that others didn't or did in a different way. Such tests were commented out so that the final set of tests can contain passing tests only, but such differences in features or inconsistencies between implementations were noted, to be resolved later. This step was crucial since it helped identified many bugs and inconsistencies between the implementations and between the language specifications.

Relevant PRs:

- Add skeleton code for starlark common test suite
- Implement test runner and add Java's test data
- add bazelci presubmit and shell script to setup workspace
- Remove expected error msgs from code before executing it
- Add go and rust test data

Resolving Inconsistencies and Bugs

In addition to some already existing Starlark issues on GitHub, I worked on creating new issues for the inconsistencies that were identified using the common test suite, and fixing them by making necessary changes to the implementations and the specification of Starlark. In this part I mainly focused on the Java implementation of Starlark considering it the main implementation and the one used in Bazel.

The following are some of the main changes implemented and their corresponding PRs:

- Restrict unknown string escape sequences (\z, \c, ...)
- Restrict *in* operator with dict to hashable types only
- Implement *list.clear()*
- Add default parameters to *bool(), list(), tuple()*

- Add optional parameters to *enumerate* and *list.index*
- Reject empty separator in *string.split*
- Reject unhashable types in *dict.get*
- Add key and reverse parameters to builtin *sorted* function
- Add unary plus operator, +int
- Disabled *string.partition* default parameter
- Support bitwise operations
- Support two-step method calls and allow retrieving built-in methods using *getattr*
- Set local scope for LHS identifiers of assignments
- Allow trailing commas in argument lists after **args* and ***kwargs*
- Reject hashing of frozen mutable types
- Allow *enumerate, join, extend* functions to accept dicts
- Enable unpacking of *dicts* into positional args
- Stop lexer from scanning characters in ints if first token in not a zero

Benchmarking

The original plan was to create a benchmark suite similar to the common test suite, add some benchmark tests, then identify performance issues from the results, the same as we identified inconsistencies in language features, and finally work on improving the Starlark interpreter to resolve those performance issues.

This wasn't fully completed though; I managed to create a simple prototype of a benchmark suite which helped me identify performance issues in a few functions and fix them:

- Add benchmark test suite
- Improve performance of *string.replace*
- Improve performance of string methods *count* and *split*

The first PR is still open and most probably won't be merged since as I mentioned the benchmark runner is a simple one, and other ideas to implement a better, more accurate one, have been suggested (see the comments on the PR for more details). However, the tests included in this PR could be useful when creating another benchmark suite.

My last contribution in this part of the project was creating a benchmark tool for Java using OpenJDK's JMH (Java Microbenchmark Harness). As was suggested, this is expected to give more accurate results compared to measuring the time of executing a Starlark Java binary as a process, which in this case would include the JVM startup time in the measurement and can lead to misleading results. The implementation of this approach is working but may still need some modifications or design changes, so the PR had not been merged at the time of writing this.

- Add JMH benchmarking tool for Starlark

Conclusion

In this project, I created a common test suite which provides a portable way of running tests against the different implementations of Starlark. This test suite helped identify many inconsistencies and issues between the implementations. Many of these issues were fixed and many still exist, and I've created a list of the inconsistencies that were discovered but have not been resolved yet, which you can find here: Inconsistencies between Starlark implementations

Concerning the problem of Starlark benchmarking, the work I've done although not completed, I believe represents a good starting point towards creating a complete benchmarking test suite for Starlark.

Potential Future Work

- Adding more tests to the common test suite to increase coverage
- Resolving remaining inconsistencies between implementations
- Continue working on the benchmarking test suite
- Use the benchmark suite to identify performance issues and improve the Starlark interpreter

Final Words

I would like to thank Laurent Le Brun, my mentor, for guiding me throughout the project and being extremely responsive. I would also like to thank every member of the Bazel community for helping with code reviews and getting around problems that I encountered.

I learned many new things by participating in GSoC and working on this project. I've been wanting to try working on open source projects for a while and GSoC and Bazel have given me this great opportunity and I think I will continue to make contributions to open source in the future and hope to keep in touch with its community.

Links

- GSoC Proposal
- Contributions to buildbazel/bazel
- Contributions to buildbazel/starlark