



Better Starlark analysis unit tests

Please read Bazel [Code of Conduct](#) before commenting.

- **Authors:** hvd@google.com
- **Status:** **Draft** | In review | Approved | Rejected | In progress | Implemented
- **Reviewers:**
- **Created:** 2021-06-16
- **Updated:** 2021-06-18
- **Discussion thread:** <link>
-

Abstract

As we move more code out of native Blaze into Starlark, it becomes increasingly important to provide rule owners with a better experience writing the accompanying tests. This proposal attempts to reduce boilerplate, improve readability and prevent duplicated code.

Current situation

The current documentation provides something like the following as a minimal example:

```
def _foo_test_assert(ctx):  
    env = analysistest.begin(ctx)  
  
    # asserts ...  
  
    return analysistest.end(env)  
  
foo_test = analysistest.make(_foo_test_assert)  
  
def _test_foo_arrange():  
    myrule(name = "target1")  
  
    foo_test(name = "foo_test", target_under_test = "target1")
```



```

def myrules_test_suite(name):
    _test_foo_arrange()

    # ...

    native.test_suite(
        name = name,
        tests = [
            ":foo_test",
            # ...
        ],
    )

```

The problems with this are:

- **Boilerplate:**
 - The test name is repeated several times and structure will often be copy/pasted with renaming and hence error prone
 - The implementation function (assert section) always starts and ends with `analysistest.{begin,end}`
- **Poor readability:** The arrange, act, assert phases do not follow linearly. Some rearrangement is possible, but is not ideal.
- **Missing utilities:** Almost all analysis test suites would need to find artifacts, actions, transitive closures, etc and the lack of these utility functions in the standard test library leads to unnecessary duplication

Proposal

Without significant change, the same example can look as follows:

```

foo_test_suite = analysistest.new_test_suite()

def _foo_arrange():
    myrule(name = "target1")

    foo_test(name = "foo_test", target_under_test = "target1")

```

```
def _foo_assert(env):  
    # asserts ...  
  
foo_test_suite.make_test("test_name", _foo_arrange, _foo_assert,  
    "target1")  
  
myrules_test_suite = foo_test_suite.generate
```

This is currently impossible since Starlark would require `rule1` to be exported before it is invoked and export occurs only post-assignment during a bzl file load. While this can be worked around by returning `rule1` from `_make_test()` and assigning it to an arbitrary, unused variable, it would be nicer if this was not necessary. Therefore, we need a mechanism to export rules without assignment (possibly restricted only to analysis tests).

The corresponding new code in `analysistest` is available in the [appendix](#) of this document.

Compatibility

No incompatibility is expected with these changes.

Document History

Date	Description
2021-06-14	First proposal

Appendix

```
def create_test_suite():

    _arrange_list = []

    _test_list = []

    return struct(

        make_test = lambda *args: _make_test(_arrange_list,
        _test_list, *args),

        generate = lambda name: _generate(name, _arrange_list,
        _test_list),

    )

def _make_test(_arrange_list, _test_list, name, arrange_impl,
assert_impl, target):

    _test_list.append(name)

    _arrange_list.append(arrange_impl)

    rule1 = analysistest.make(lambda ctx: _wrap(assert_impl, ctx))

    _arrange_list.append(lambda: rule1(name = name, target_under_test
= target))

    return rule1

def _wrap(impl, ctx):

    env = analysistest.begin(ctx)

    impl(env)

    return analysistest.end(env)

def _generate(name, _arrange_list, _test_list):

    for arrange in _arrange_list:
```

```
    arrange()  
  
    native.test_suite(name = name, tests = _test_list)
```