Refactoring AwkwardForth Generation in Uproot

Seth Bendigo[1]
Mentors: Ioana Ifrim[2] Jim Pivarski[2]

2023

[1]University of Minnesota - Twin Cities

[2]Princeton University

Intro to Uproot

Intro to Uproot

Uproot is an I/O library for reading and writing ROOT files for use in Python [1]. To keep it lightweight and portable, it is kept strictly as an I/O library, and does not depend on ROOT.

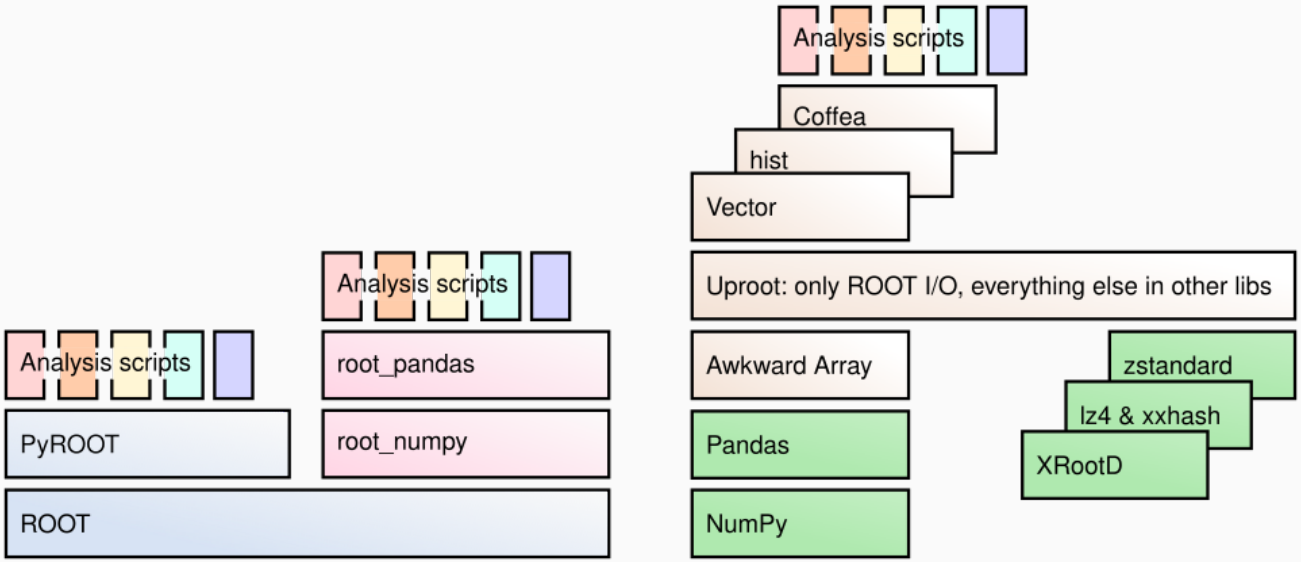

Figure 1: Abstraction layers of various methods to use ROOT files in Python.

Intro to Uproot

When reading in non-columnar data types, iteration is required, and Python loops are slow compared to compiled languages.

- A compiled language cannot be used since, at compile time, the byte-for-byte layout in ROOT files with complex data types is unknown.
- Just-in-time compilation could be used, but this affects portability.

AwkwardForth

AwkwardForth

Uproot instead implements the use of AwkwardForth, an internal domain specific language [2].

By generating AwkwardForth code to read in the incoming complex data types, Uproot is significantly optimized. In the case of std::vector<std::vector<float», AwkwardForth is faster than Python by a factor of about 400.

AwkwardForth

Unfortunately, the current implementation of AwkwardForth generation in Uproot has some problems.

Issues

- Excessively mutable: Objects that change their attributes in arbitrary ways as information needed to generate AwkwardForth accumulates.
- Readability: Dead code, nondescript attribute names.

Refactor

Refactor

To fix these issues, the refactor is focusing on rewriting the generation to avoid as much mutability as possible, while utilizing test-driven development.

There are $\sim$140 tests relating to AwkwardForth in the current implementation.

Refactor

I started by removing all the code that relied on AwkwardForth. My next step was to understand in depth how AwkwardForth *currently* generates for the case std::vector<std::vector<float».

Then, I refactored AwkwardForth for just that case, using the already-written tests to ensure I had done it correctly.

Refactor

Note: What AwkwardForth code gets generated is not changing. The outer "generation-machinery" is.

As the program reads through the data, it decides what AwkwardForth code to generate, if any.

The program stores collected AwkwardForth code-snippets in nodes. These nodes are then worked into a tree.

At the end, it recurses through the tree to generate the complete AwkwardForth code that will read in the data.

Refactor

The code-snippet tree is highly mutable. The main focus of the refactor has been making it append only.

Each case

- Get result in current implementation.
- Understand which parts are logic to generate AwkwardForth.
- Deconstruct, reorganize, and rework to get same result, but with append only.

Refactor

Example

```
# AwkwardForth testing: test_0637's 01,02,08,09,11,12,13,15,16,29,38,45,46,49,50
read_members.extend(
    [
        "        if forth_stash is not None:",
        "            temp_node, temp_node_top, temp_form, temp_form_top, temp_prev_form = forth_obj.replace_form_and_model(None, {'name': 'TOP', 'content': {}})",
        f"        self._bases.append(c({self.name!r}, {self.base_version!r}).read(chunk, cursor, context, file, self._file, self._parent, concrete=self.concrete))",
        "        if forth_stash is not None and not context['cancel_forth']:",
        "            temp_prev_form1 = forth_obj.prev_form",
        "            temp_form1 = forth_obj.top_form",
        "            temp_model1 = forth_obj.top_node",
        "            temp_model_ref = forth_obj.awkward_model",
        "            forth_obj.awkward_model = temp_node",
        "            forth_obj.top_node = temp_node_top",
        "            forth_obj.aform = temp_form",
        "            forth_obj.prev_form = temp_prev_form",
        "            forth_obj.top_form = temp_form_top",
        "            temp_model1 = temp_model1['content']",
        "            forth_obj.add_node_whole(temp_model1, temp_model_ref)",
        "            content.update(temp_form1['contents'])",
        "            forth_obj.enable_adding()",
    ]
)
```

Figure 2: Python code that generates Python code that generates Forth code in current implementation.

Bibliography

Jim Pivarski, Henry Schreiner, Angus Hollands, Pratyush Das, Kush Kothari, Aryan Roy, Jerry Ling, Nicholas Smith, Chris Burr, and Giordon Stark.
Uproot, June 2023.

Jim Pivarski, Ianna Osborne, Pratyush Das, David Lange, and Peter Elmer.
AwkwardForth: accelerating uproot with an internal DSL.
EPJ Web of Conferences, 251:03002, 2021.