Interactive Programming for LLVM TableGen

David Spickett, Staff Software Engineer, Arm

Linaro

The Problem

- TableGen powers large parts of LLVM.
- You'll need to learn it eventually.
- Existing TableGen is hard to learn from.
- Tutorials are too basic or too detailed.
  - https://llvm.org/docs/TableGen/
  - https://llvm.org/docs/TableGen/ProgRef.html

Existing Solutions

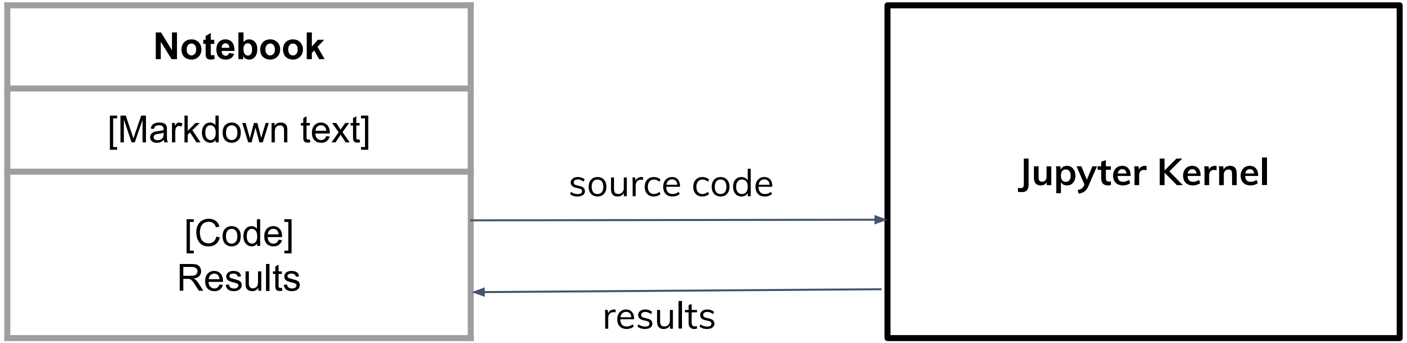
- REPLs - Python, cling, lisp, etc.
- Online compilers - https://godbolt.org/
- Explainer tools - https://explainshell.com/
- `clang -ast-dump` and similar.
- Jupyter Notebooks

Common themes:

- Learn what you want
- When you want

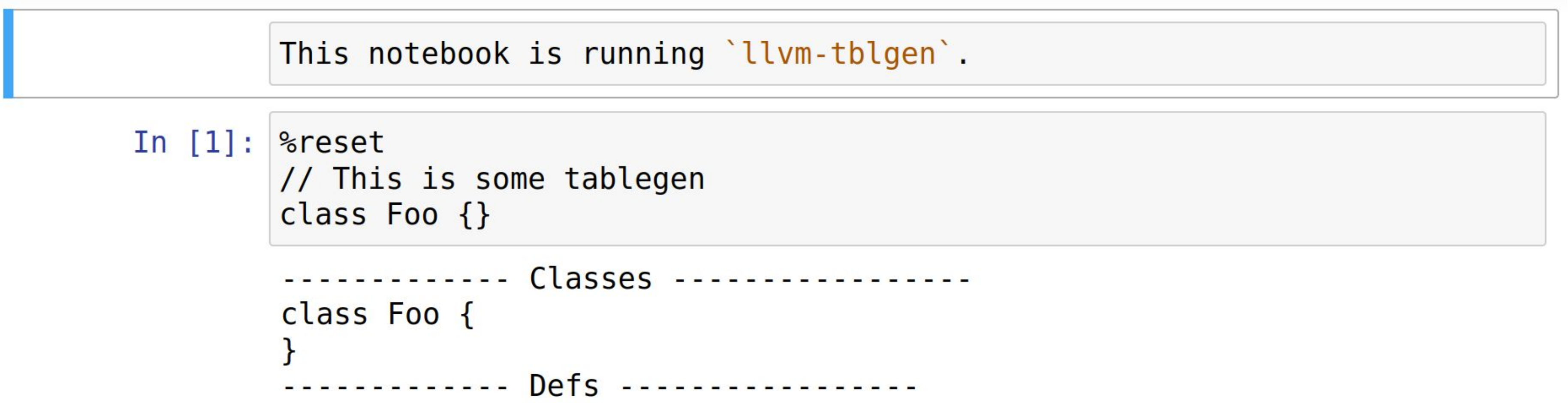
Jupyter Notebooks

- Text and code “cells”
- Edit and re-run cells
- Results shown inline



- Single .ipynb file
- Render to static formats like HTML

Jupyter Kernel for TableGen



This notebook is running `llvm-tblgen`.

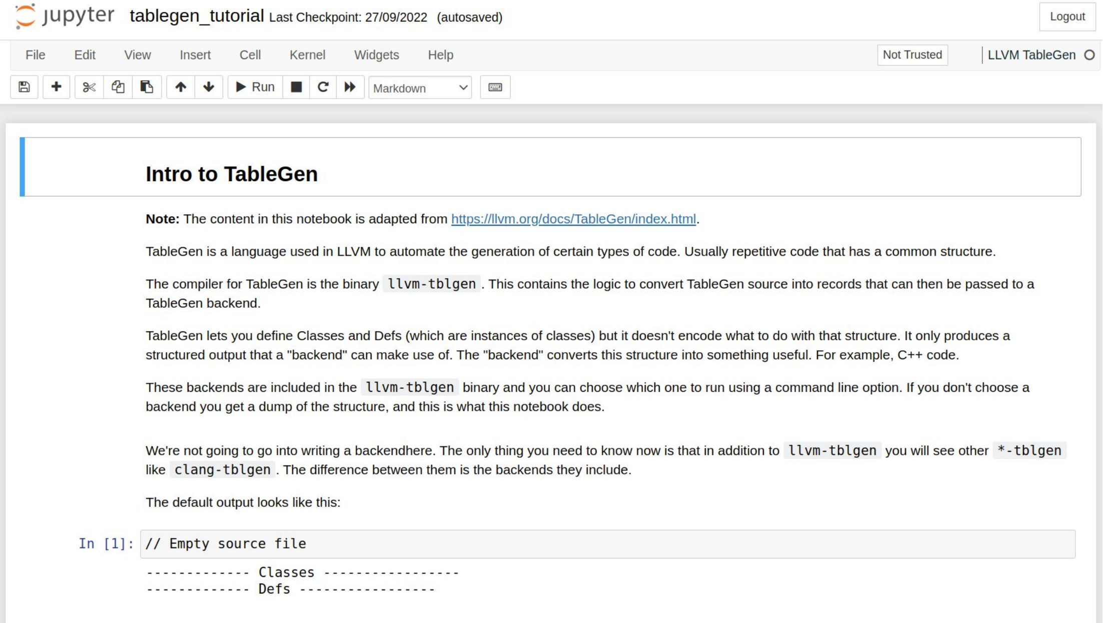
```
In [1]: %reset
        // This is some tablegen
        class Foo {}
```

```
------------- Classes -----------------
class Foo {
}
------------- Defs -----------------
```

- Based on the existing MLIR kernel
- Compiles with `llvm-tblgen`
- Cells linked by default (reset cache with `%reset`)
- Set compiler arguments with `%args`

The Goal

- An interactive, editable, TableGen tutorial.
- Read in Jupyter or as a static document.


jupyter
tablegen_tutorial Last Checkpoint: 27/09/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
LLVM TableGen
Run
Markdown

Intro to TableGen

Note: The content in this notebook is adapted from https://llvm.org/docs/TableGen/index.html.

TableGen is a language used in LLVM to automate the generation of certain types of code. Usually repetitive code that has a common structure.

The compiler for TableGen is the binary `llvm-tblgen`. This contains the logic to convert TableGen source into records that can then be passed to a TableGen backend.

TableGen lets you define Classes and Defs (which are instances of classes) but it doesn't encode what to do with that structure. It only produces a structured output that a "backend" can make use of. The "backend" converts this structure into something useful. For example, C++ code.

These backends are included in the `llvm-tblgen` binary and you can choose which one to run using a command line option. If you don't choose a backend you get a dump of the structure, and this is what this notebook does.

We're not going to go into writing a backendhere. The only thing you need to know now is that in addition to `llvm-tblgen` you will see other `*-tblgen` like `clang-tblgen`. The difference between them is the backends they include.

The default output looks like this:

In [1]:
```
// Empty source file
```

```
------------- Classes -----------------
------------- Defs -----------------
```

Status

- RFC: https://discourse.llvm.org/t/rfc-a-jupyter-kernel-for-tablegen/65003
- Patch series - https://reviews.llvm.org/D132378
- First tutorial notebook - https://reviews.llvm.org/D137085

Future work:

- Domain specific tutorials
- Visualise class structure
- Output filtering

Thank you

(thanks to Jacques Pienaar for the MLIR kernel)