llvmcpy: A python interface for LLVM

Andrea Gussoni
andrea1.gussoni at polimi.it

Politecnico di Milano

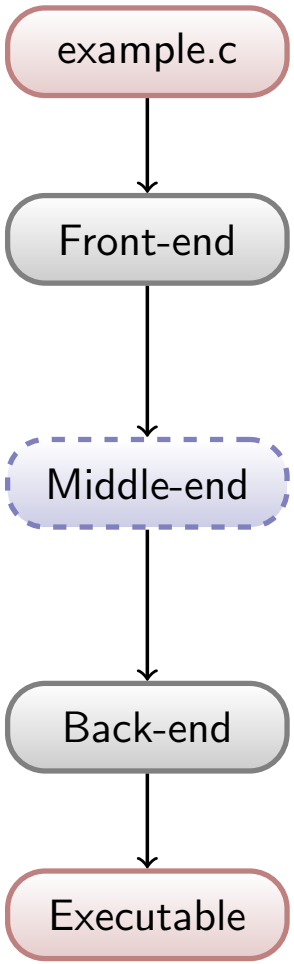
May 17, 2019

Table of Contents

LLVM

- The LLVM compiler framework
- How much do you already know?

LLVM design



LLVM design

Front-end Convert the source code into the intermediate representation

Middle-end Applies optimization to the intermediate representation

Back-end Convert the intermediate into machine code

Pass design

- KISS
- More simpler passes composed together are better than a single analysis
- The IR is the common language shared between passes

Interact with LLVM

The recommended way to interact with the LLVM framework is the C++ API:

- Advantages:
  - High power and performances
  - Completely integrated in the framework
- Disadvantages:
  - Complex
  - A lot of boilerplate code

Interact with LLVM

What if we only need to interact with LLVM to gather some info on the IR?

Table of Contents

llvmcpy

- llvmcpy[2] is a project which provides Python bindings for the LLVM project
- It uses CFFI[1] to parse the header files of the LLVM-C API[5] to generate a set of classes and methods to interact with
- It is developed and mantained as a part of the rev.ng[3] project

Instructions

- You'll need LLVM on yor system: `apt-get install llvm`
- And then you'll need `llvmcpy`: `pip install --user llvmcpy`
- If you want to hack with it, clone the repo: `git clone https://github.com/revng/llvmcpy.git`
- And the install it: `python setup.py install --user`

Support

- It has been tested on the following LLVM versions: 3.4, 3.8, 3.9, 6.0, 7.0 (PR for 8.0)
- Supporting newer versions should be very easy and effortless
- You can use the `LLVM_CONFIG` to specify which version of LLVM you want to use (in case you have multiple):
  `export LLVM_CONFIG=/usr/bin/llvm-config-7`

Example

```
import sys
from llvmcpy.llvm import *

buffer = create_memory_buffer_with_contents_of_file(sys.argv[1])
context = get_global_context()
module = context.parse_ir(buffer)
for function in module.iter_functions():
    for bb in function.iter_basic_blocks():
        for instruction in bb.iter_instructions():
            instruction.dump()
```

Generated classes and function

- The basic idea behind the project is to take the LLVM-C API function, create a class for each data type and create a method for that class for each function in the API taking an argument of that data type as first argument.

- This means that the following function:

```
LLVMModuleRef LLVMCloneModule (LLVMModuleRef M)
```

- Will become:

```
class Module(object):
    def clone(self):
        # ...
```

Generated classes and function

- To see what LLVM-C API function a method is using, we can use the `help(Module.clone)` helper
- Basically, each class in `llvmcpy` is a wrapper around a pointer to an LLVM object
- If an API function doesn't take an LLVM object as a first argument, it will be part of the LLVM module

Generated properties and generators

- llvmcpy also provides some generated properties and generators for certain well known patterns in the API

Properties

- For each function starting with LLVMGet or LLVMSet in the LLVM-C API, we generate a property:

```
void LLVMSetValueName (LLVMValueRef Val, const char *Name)
const char* LLVMGetValueName(LLVMValueRef Val)
```

- In llvmcpy the Get or Set prefixes disappear, along with Value (the name of the class) and you can use them as properties of the Value class:

```
basicblock.name = "main"
print basicblock.name
```

Generators

- The LLVM-C API has a recurrent pattern which allows you to navigate through the hierarchy of its class hierarchy, i.e. the pair of `LLVMGetSomething` and `LLVMGetNextSomething` functions. Something can be `Function`, `BasicBlock` and so on
- `llvmcpy` identifies these patterns and produces a generator method which allows you to iterate over these objects in a Pythonic way:

```
for function in module.iter_functions():
    for bb in function.iter_basic_blocks():
        for instruction in bb.iter_instructions():
            # ...
```

Documentation

- You can use the online documentation
- Or you can use *zeal*[1] + the doxygen docset for LLVM

[1] https://zealdocs.org/

Table of Contents

Demo time

DEMO TIME

References I


CFFI. URL: https://cffi.readthedocs.io/en/latest/.


The rev.ng company. *llvmcpy*. URL: https://github.com/revng/llvmcpy.


The rev.ng company. *rev.ng*. URL: https://rev.ng.

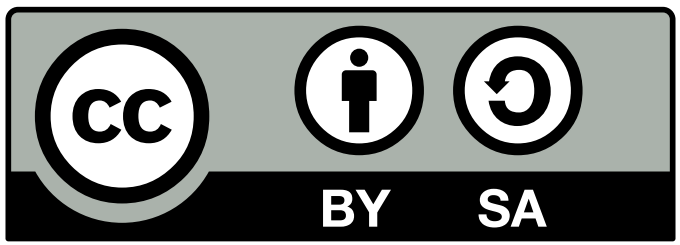

Alessandro Di Federico. *rev.ng Output Reference*. URL: https://github.com/revng/revng/blob/develop/docs/GeneratedIRReference.rst.


LLVM. *LLVM-C*. URL: https://llvm.org/doxygen/group__LLVMC.html.

Special Thanks

- Thanks to Alessandro Di Federico for developing and mantaining the `llvmcpy`[2] project

License



These slides are published under a Creative Commons Attribution-ShareAlike 4.0 license[2].

[2] https://creativecommons.org/licenses/by-sa/4.0/