**Funmilayo (Funmi) Olaiya**
Cheriton School of Computer Science
University of Waterloo

April.4.2023

## Does it Live up to the Hype?

# Extensibility of the P416 Compiler for New Targets

# INTRODUCTION

## P4 as a language

- Processing packets in a programmable data plane
- Needed something more higher level

## Why P416?

- Originally developed for network switches???
- No modularity - P414
- Embraces diversity

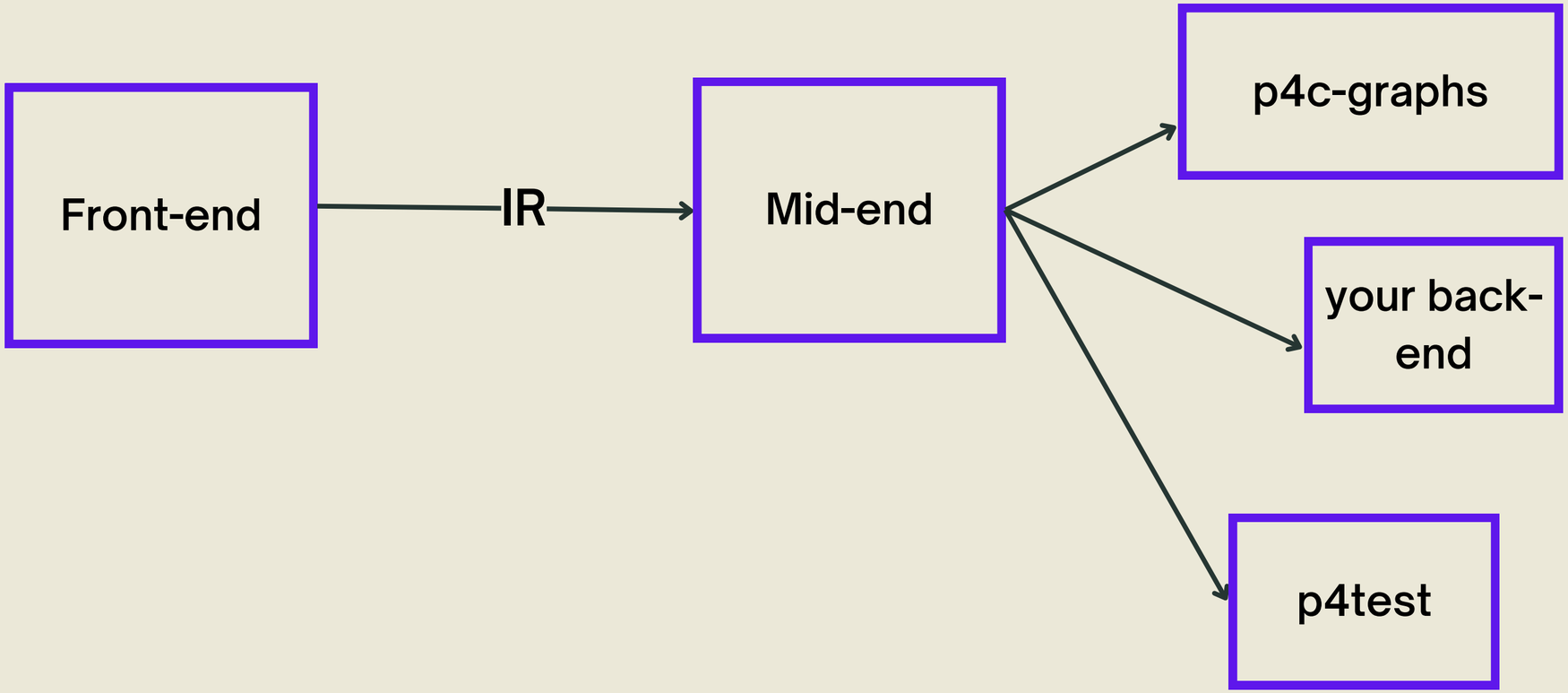
## Design goals of P416

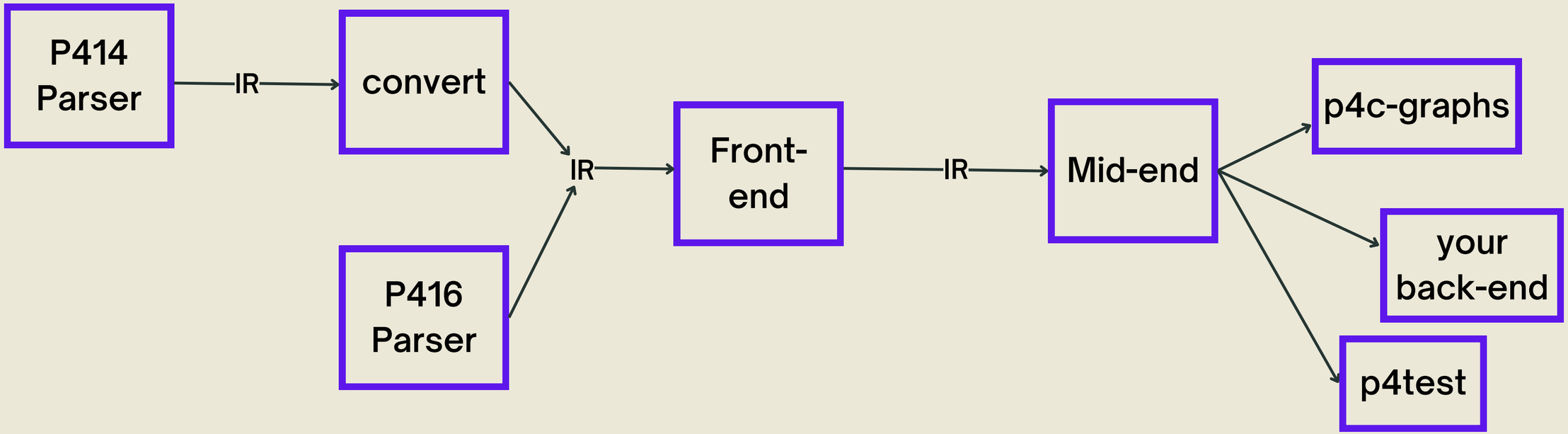
- Core design goals

# P416's Design Goals

- It should provide a solid basis to support the past, the present and future versions of P4
- It should support multiple back-ends for specific applications & targets
- Provide support for writing other software development tools e.g debuggers, P4 control planes, testing
- The compiler's front-end should be open-source - ability to bootstrap a compiler for a new architecture/backend/target easily
- Should have an extensible architecture, making it easy to add new passes/optimizations/back-ends
- Compiler implementation should rely on modern compiler techniques e.g IR, strong type checking

# P416's Architecture


Front-end
IR
Mid-end
p4c-graphs
your back-end
p4test


P414 Parser
IR
convert
P416 Parser
IR
Front-end
IR
Mid-end
p4c-graphs
your back-end
p4test

# Setting up, Building & Exploring the Compiler

- Clone the repository, and make sure to use **--recursive**

```
- git clone --recursive https://github.com/p4lang/p4c.git
- git submodule update --init --recursive
```

- Install certain dependencies, below is for mac

```
# Have Homebrew installed

- xcode-select --install

- brew install autoconf automake libtool bdw-gc boost bison pkg-config

- brew link --force bison
export PATH="/usr/local/opt/bison/bin:$PATH"

- brew install doxygen graphviz

PB_PREFIX="$(brew --prefix --installed protobuf)"
./bootstrap.sh \
  -DProtobuf_INCLUDE_DIR="${PB_PREFIX}/include/" \
  -DProtobuf_LIBRARY="${PB_PREFIX}/lib/libprotobuf.dylib" \
  -DENABLE_PROTOBUF_STATIC=OFF
```

- Build the compiler

```
- mkdir build
- cd build
- cmake .. <optional arguments>
- make -j4
- make -j4 check
```

# How does p4test work?

```
- cd backends/p4test
- run the command: p4test

error:
  [--Werror=expected] error: No input files specified
  p4test: Compile a P4 program

  ...
```

```
- cd backends/p4test
- run the command: p4test

error:
  [--Werror=expected] error: No input files specified
  p4test: Compile a P4 program

  ...
```

```
- cd backends/p4test
- run the command: p4test prog.p4

output:
  nothing, which shows it passes
```

```
#include <core.p4>
#include <v1model.p4>

typedef bit<48> EthernetAddress;
typedef bit<32> IPv4Address;

header ethernet_t {
   ....
}

......

control my_egress(inout headers_t hdr,
                  inout metadata_t meta,
                  inout standard_metadata_t standard_metadata)
{
    apply { }
}

V1Switch(my_parser(),
         my_verify_checksum(),
         my_ingress(),
         my_egress(),
         my_compute_checksum(),
         my_deparser()) main;
```

```
- cd backends/p4test
- run the command: p4test prog.p4
```

```
#include <core.p4>
#include <v1model.p4>

typedef bit<48> EthernetAddress;
typedef bit<32> IPv4Address;

header ethernet_t {
    ....
}

......

control my_egress(inout headers_t hdr,
                  inout metadata_t meta,
                  inout standard_metadata_t standard_metadata)
{
    apply { }
}

V1Switch(my_parser(),
         my_verify_checksum(),
         my_ingress(),
         my_egress(),
         my_compute_checksum(),
         my_deparser()) main;
```

```
- cd backends/p4test
- run the command: p4test prog.p4
```

```
prog.p4(7):syntax error, unexpected {, expecting (
head ethernet_t {
                ^
[--Werror=overlimit] error: 1 errors encountered, aborting compilation
```

```
#include <core.p4>
#include <v1model.p4>

typedef bit<48> EthernetAddress;
typedef bit<32> IPv4Address;

header ethernet_t {
    ....
}

......

control my_egress(inout headers_t hdr,
                  inout metadata_t meta,
                  inout standard_metadata_t standard_metadata)
{
    apply { }
}

V1Switch(my_parser(),
         my_verify_checksum(),
         my_ingress(),
         my_egress(),
         my_compute_checksum(),
         my_deparser()) main;
```

# Deeper into p4test's code structure


p4test
CMakeLists.txt
midend.cpp
midend.h
p4test.cpp
prog.p4
U
README.md
run-p4-sample.py
version.h.cmake

# p4test.cpp

```
#include <fstream>
#include <iostream>

#include "backends/p4test/version.h"
#include "control-plane/p4RuntimeSerializer.h"
#include "frontends/common/applyOptionsPragmas.h"
#include "frontends/common/parseInput.h"
#include "frontends/p4/evaluator/evaluator.h"
#include "frontends/p4/frontend.h"
#include "frontends/p4/toP4/toP4.h"
#include "ir/ir.h"
#include "ir/json_loader.h"
#include "lib/crash.h"
#include "lib/error.h"
#include "lib/exceptions.h"
#include "lib/gc.h"
#include "lib/log.h"
#include "lib/nullstream.h"
#include "midend.h"
```

# p4test.cpp

```
int main(int argc, char *const argv[]) {
    setup_gc_logging();
    setup_signals();

    AutoCompileContext autoP4TestContext(new P4TestContext);
    auto &options = P4TestContext::get().options();
    options.langVersion = CompilerOptions::FrontendVersion::P4_16;
    options.compilerVersion = P4TEST_VERSION_STRING;

    if (options.process(argc, argv) != nullptr) {
        if (options.loadIRFromJson == false) options.setInputFile();
    }
    if (::errorCount() > 0) return 1;
    const IR::P4Program *program = nullptr;
    auto hook = options.getDebugHook();
    if (options.loadIRFromJson) {
        std::ifstream json(options.file);
        if (json) {
            JSONLoader loader(json);
            const IR::Node *node = nullptr;
            loader >> node;
            if (!(program = node->to<IR::P4Program>()))
                error(ErrorType::ERR_INVALID, "%s is not a P4Program in json format",
options.file);
        } else {
            error(ErrorType::ERR_IO, "Can't open %s", options.file);
        }
    } else {
        program = P4::parseP4File(options);

        if (program != nullptr && ::errorCount() == 0) {
            P4::P4COptionPragmaParser optionsPragmaParser;
            program->apply(P4::ApplyOptionsPragmas(optionsPragmaParser));

            if (!options.parseOnly) {
                try {
                    P4::FrontEnd fe;
                    fe.addDebugHook(hook);
                    program = fe.run(options, program);
                } catch (const std::exception &bug) {
                    std::cerr << bug.what() << std::endl;
                    return 1;
                }
            }
        }
    }
```

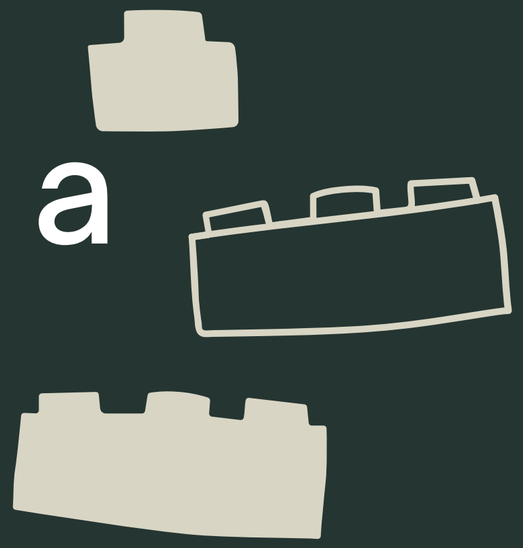
## frontend/parser_options.cpp

```cpp
void ParserOptions::setInputFile() {
    if (remainingOptions.size() > 1) {
        ::error(ErrorType::ERR_OVERLIMIT, "Only one input file must be specified: %s",
                cstring::join(remainingOptions.begin(), remainingOptions.end(), ","));
        usage();
        exit(1);
    } else if (remainingOptions.size() == 0) {
        ::error(ErrorType::ERR_EXPECTED, "No input files specified");
        usage();
        exit(1);
    } else {
        file = remainingOptions.at(0);
    }
}
```

## p4test.cpp

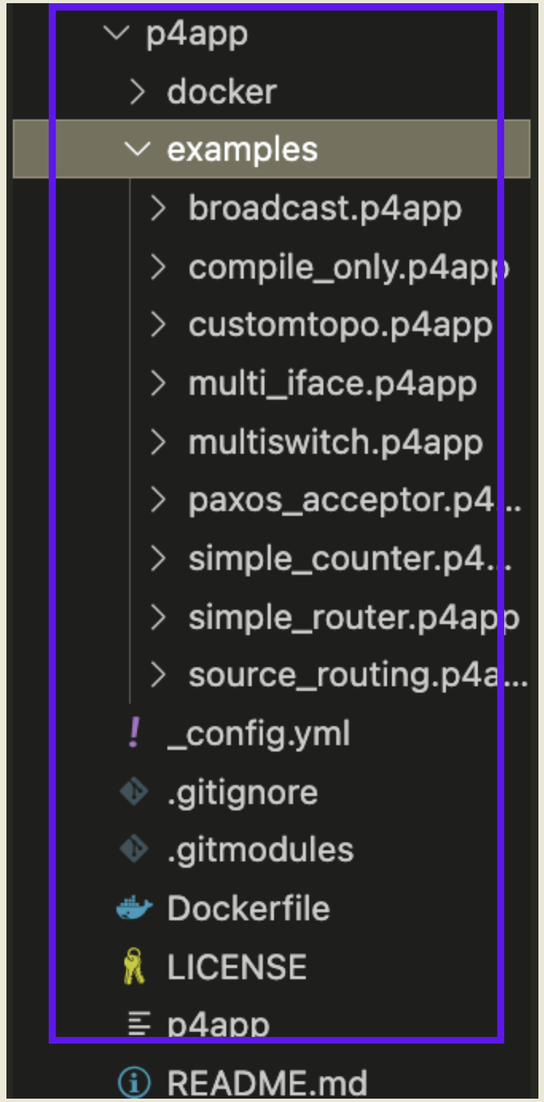
```cpp
    log_dump(program, "Initial program");
    if (program != nullptr && ::errorCount() == 0) {
        P4::serializeP4RuntimeIfRequired(program, options);

        if (!options.parseOnly && !options.validateOnly) {
            P4Test::MidEnd midEnd(options);
            midEnd.addDebugHook(hook);
#if 0
            /* doing this breaks the output until we get dump/undump of srcInfo */
            if (options.debugJson) {
                std::stringstream tmp;
                JSONGenerator gen(tmp);
                gen << program;
                JSONLoader loader(tmp);
                loader >> program;
            }
#endif
            const IR::ToplevelBlock *top = nullptr;
            try {
                top = midEnd.process(program);
                // This can modify program!
                log_dump(program, "After midend");
                log_dump(top, "Top level block");
            } catch (const std::exception &bug) {
                std::cerr << bug.what() << std::endl;
                return 1;
            }
        }
```

# Trying to Build/Add a Backend/Target

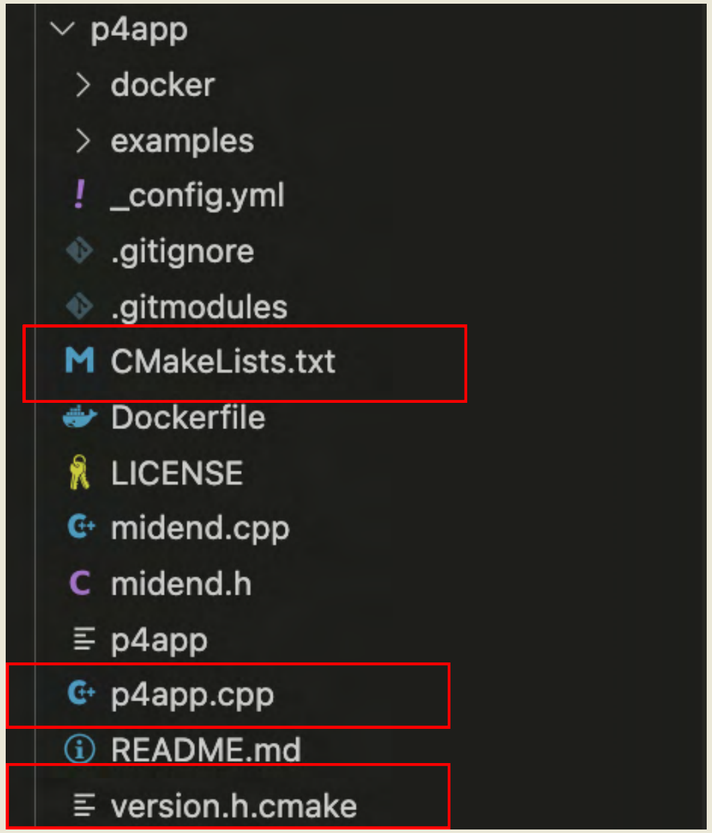


- p4app is a tool that can build, run, debug, and test P4 programs
- Designed to make small, simple P4 programs easy to write and easy to share with others

```
# clone p4app
# create a path in your .zshrc file like below

- export PATH="/users/funmilayoolaiya/documents/github/p4app:$PATH"

# try running one of the provided examples

- p4app run examples/simple_router.p4app
```




**in the compiler**

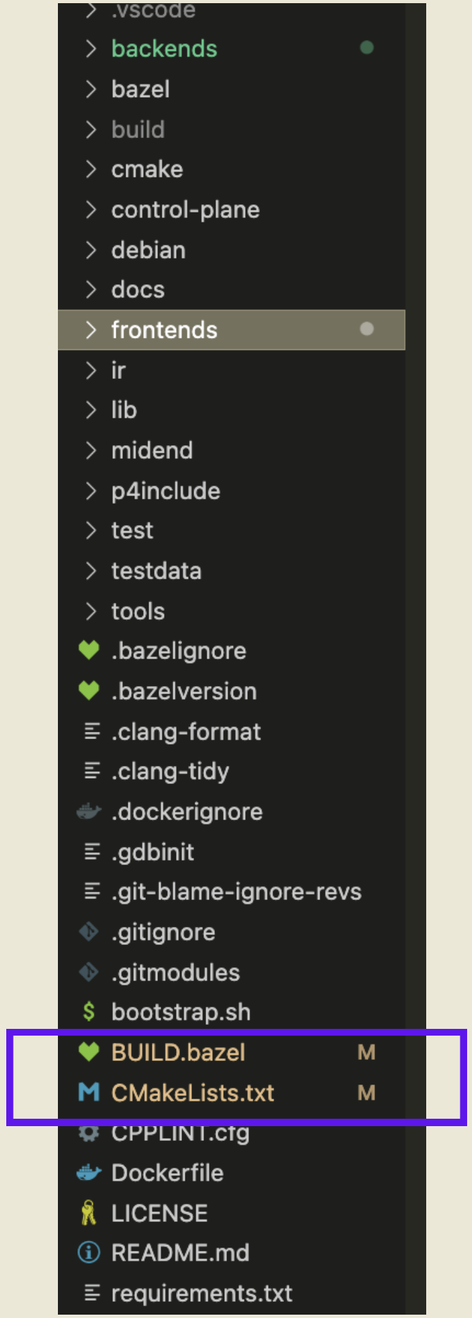


## CMakeLists.txt within the p4app folder

```cmake
configure_file("${CMAKE_CURRENT_SOURCE_DIR}/version.h.cmake"
  "${CMAKE_CURRENT_BINARY_DIR}/version.h" @ONLY)

set (P4APP_SRCS
  p4app.cpp
  midend.cpp
  )
set (P4APP_HDRS
  midend.h
  )

add_executable(p4app ${P4APP_SRCS} ${EXTENSION_P4_14_CONV_SOURCES})
target_link_libraries (p4app ${P4C_LIBRARIES} ${P4C_LIB_DEPS})
add_dependencies(p4app genIR frontend)

install (TARGETS p4app
  RUNTIME DESTINATION ${P4C_RUNTIME_OUTPUT_DIRECTORY})

file(RELATIVE_PATH
  CURRENT_BINARY_DIR_PATH_REL
  ${P4C_BINARY_DIR}
  ${CMAKE_CURRENT_BINARY_DIR}
)

file(RELATIVE_PATH
  P4C_BINARY_DIR_PATH_REL
  ${CMAKE_CURRENT_BINARY_DIR}
  ${P4C_BINARY_DIR}
)

add_custom_target(linkp4app
  COMMAND ${CMAKE_COMMAND} -E create_symlink ${CURRENT_BINARY_DIR_PATH_REL}/p4app
${P4C_BINARY_DIR}/p4app
  )
add_dependencies(p4c_driver linkp4app)
```

## version.h.cmake within the p4app folder

```
/*
Copyright 2018-present Barefoot Networks, Inc.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

# ifndef _BACKENDS_P4APP_VERSION_H
# define _BACKENDS_P4APP_VERSION_H

/**
  Set the compiler version at build time.
  The build system defines P4C_VERSION as a full string as well as the
  following components: P4C_VERSION_MAJOR, P4C_VERSION_MINOR,
  P4C_VERSION_PATCH, P4C_VERSION_RC, and P4C_GIT_SHA.

  They can be used to construct a version string as follows:
  #define VERSION_STRING "@P4C_VERSION@"
  or
  #define VERSION_STRING
"@P4C_VERSION_MAJOR@.@P4C_VERSION_MINOR@.@P4C_VERSION_PATCH@@P4C_VERSION_RC@"

  Or, since this is backend specific, feel free to define other numbering
  scheme.

  */

# define P4APP_VERSION_STRING "@P4C_VERSION@"

# endif  // _BACKENDS_P4APP_VERSION_H
```




## BUILD.bazel

```
# This builds the p4app backend.
cc_binary(
    name = "p4c_backend_p4app",
    srcs = [
        "backends/p4app/p4app.cpp",
    ],
    deps = [
        ":ir_frontend_midend_control_plane",
        ":lib",
        ":p4c_backend_p4app_lib",
    ],
)

genrule(
    name = "p4c_p4app_version",
    srcs = ["backends/p4app/version.h.cmake"],
    outs = ["backends/p4app/version.h"],
    cmd = "sed 's|@P4C_VERSION@|0.0.0.0|g' $(SRCS) > $(OUTS)",
    visibility = ["//visibility:private"],
)

cc_library(
    name = "p4c_backend_p4app_lib",
    srcs = [
        "backends/p4app/midend.cpp",
    ],
    hdrs = [
        "backends/p4app/midend.h",
        "backends/p4app/version.h",
    ],
    deps = [
        ":ir_frontend_midend_control_plane",
        ":lib",
    ],
)
```

## CMakeLists.txt

```
if (ENABLE_P4APP)
    add_subdirectory (backends/p4app)
endif ()

OPTION (ENABLE_P4APP "Build the P4APP backend" ON)
```

## p4app.cpp

```
#include <fstream>
#include <iostream>
using namespace std;

int main(int argc, char *const argv[]) {}
```

- run the necessary commands

build — -zsh — 116×26

```
(base) funmilayoolaiya@funmilayos-MacBook-Pro p4c % cd build
(base) funmilayoolaiya@funmilayos-MacBook-Pro build % make -j4
```

p4app — com.docker.cli ◂ p4app run examples/simple_router.p4app — 116×26

```
(base) funmilayoolaiya@funmilayos-MacBook-Pro p4app % p4app run examples/simple_router.p4app
Entering build directory.
Extracting package.
> touch /tmp/p4app_logs/p4s.s1.log
> ln -s /tmp/p4app_logs/p4s.s1.log /tmp/p4s.s1.log
Reading package manifest.
> p4c-bm2-ss --p4v 16 "simple_router.p4" -o "simple_router.json"
> python2 "/scripts/mininet/single_switch_mininet.py" --log-file "/var/log/simple_router.p4.log" --cli-message "mini
net_message.txt" --num-hosts 2 --switch-config "simple_router.config" --behavioral-exe "simple_switch" --json "simpl
e_router.json"
Adding host h1
Adding host h2
```

# Positives & Negatives

## Positives:

- A very strong ***IR*** that doesn't just parse any file
- When trying to build with ***make -j4***, it gets errors immediately
- Strong modularity!
- Good ReadMe files
- Surprising acceptance of foreign targets

## Negatives:

- Takes a longer time to build when trying to make significant changes
- A very steep learning curve

## Limitations:

- Time constraints / Knowledge of C++

## Concluding Remarks:

- So does it live up?


THANKS