# Introduction

This document aims to be a guide to the public p4lang repositories, and some other selected public sources of information about P4, related tools, and published research papers.

Caveat emptor: As of this writing, the author is not an expert on these topics, and is creating this in hopes of getting up to speed with the state of the art more quickly. Corrections and comments are very welcome.

## Quick summary

If all you want to do is compile P4_14 and/or P4_16 source code for the bmv2 behavioral model, and use the command line or thrift API it provides for adding/removing table entries, then getting copies of and following the installation instructions for these 2 repositories is all you need:

- `p4c` - P4_16 prototype compiler (also compiles P4_14 programs)
- `behavioral-model` - Rewrite of the behavioral model as a C++ project without auto-generated code (except for the PD interface)

The bash shell script `bin/install-p4dev.sh` can do this for an Ubuntu 16.04 Linux machine, with no interaction required other than entering your password for `sudo` commands on occasion.

## `p4lang` repositories by name

`p4lang` is the name of a Github 'organization'.
A Github organization is a way of grouping together multiple related repositories.

p4lang organization on Github

All p4lang repositories as of 2017-Mar-10, sorted by name (case

insensitive), with their descriptions:

- `behavioral-model` - Rewrite of the behavioral model as a C++ project without auto-generated code (except for the PD interface)
- `mininet` - Emulator for rapid prototyping of Software Defined Networks http://mininet.org (forked from mininet/mininet)
- `ntf` - Network Test Framework
- `p4-build` - Infrastructure needed to generate, build and install the PD library for a given P4 program
- `p4-hlir` - (No description)
- `p4app` - (No description)
- `p4c` - P4_16 prototype compiler (also compiles P4_14 programs)
- `p4c-behavioral` - P4 compiler for the behavioral model. Deprecated.
- `p4c-bm` - Generates the JSON configuration for the behavioral-model (bmv2), as well as the C/C++ PD code
- `p4factory` - Compile P4 and run the P4 behavioral simulator. Deprecated.
- `p4ofagent` - Openflow agent on a P4 dataplane
- `papers` - Repository for papers related to P4
- `PI` - P4 PI headers and target-independent code
- `ptf` - Packet Test Framework
- `SAI` - Switch Abstraction Interface (forked from opencomputeproject/SAI)
- `scapy-vxlan` - A scapy clone, with support for additional packet headers
- `switch` - Consolidated switch repo (API, SAI and Netlink)
- `third-party` - Third-party dependencies for p4lang software
- `thrift` - Mirror of Apache Thrift (forked from apache/thrift)
- `tutorials` - P4 language tutorials

Excerpt from 2017-Mar-07 email from Antonin Bas on p4-dev email list
link

> *IMO, there is no good reason to use p4factory today, as it is in the process of being deprecated. Either you want to run switch.p4, in which case you should refer to the instructions in the switch repo directly, or you are interested in writing your own P4 programs and running them in bmv2, in which case the best place to get started is probably the tutorials repo. To generate control plane APIs easily, the best place to look is the p4-build repository.*

## `p4lang` repositories by category

A few projects have intentionally been placed into more than one category.

Remaining to be categorized:

- `p4-build` - Infrastructure needed to generate, build and install the PD library for a given P4 program
- `p4app` - (No description)
- `p4factory` - Compile P4 and run the P4 behavioral simulator. Deprecated.
- `p4ofagent` - Openflow agent on a P4 dataplane
- `PI` - P4 PI headers and target-independent code
- `switch` - Consolidated switch repo (API, SAI and Netlink)

Documentation, research papers, and tutorials:

- `papers` - Repository for papers related to P4
- `tutorials` - P4 language tutorials

P4 compilers, some only front end, some front end plus back end for one or more P4 targets:

- `p4c` - P4_16 prototype compiler (also compiles P4_14 programs)
- `p4c-bm` - Generates the JSON configuration for the behavioral-model (bmv2), as well as the C/C++ PD code
- `p4-hlir` - P4_14 compiler, written in Python, which stops at generating an intermediate representation, from which one can start in writing a back end compiler.
- `p4c-behavioral` - P4 compiler for the behavioral model. Deprecated.

P4 behavioral models, for running P4 programs on general purpose computers:

- `behavioral-model` - Rewrite of the behavioral model as a C++ project without auto-generated code (except for the PD interface). Also known as `bmv2`.
- `p4c-behavioral` - P4 compiler for the behavioral model. Deprecated.

Open source tools created by organizations other than p4.org, used by one or more `p4lang` repositories:

- `mininet` - Emulator for rapid prototyping of Software Defined Networks http://mininet.org (forked from mininet/mininet)
- `SAI` - Switch Abstraction Interface (forked from opencomputeproject/SAI)
- `scapy-vxlan` - A scapy clone, with support for additional packet headers
- `third-party` - Third-party dependencies for p4lang software
- `thrift` - Mirror of Apache Thrift (forked from apache/thrift)

For creating and running automated tests:

- `mininet` - Emulator for rapid prototyping of Software Defined Networks http://mininet.org (forked from mininet/mininet)
- `ntf` - Network Test Framework
- `ptf` - Packet Test Framework
- `scapy-vxlan` - A scapy clone, with support for additional packet headers

## `p4lang` repository descriptions

Glossary:

- `API` - Application Programming Interface
- `bmv2` - Behavioral Model Version 2. Contained in the `behavioral-model` repository.
- `bmv2 JSON configuration file` - a data file produced by some of the compilers below that is read by the bmv2 behavioral model during initialization. Contains all data about a particular source P4 program that is needed by the behavioral model code to forward packets as that P4 program specifies.
- `HLIR` - High Level Intermediate Representation. See IR.
- `IR` - Intermediate Representation - data structures created as a result of parsing P4 source code, representing all relevant details about the source code needed for the back end portion of a compiler to generate configuration specific to a particular P4 target.
- `PD API` - Protocol Dependent API ? TBD where to find out more about this.
- `PI API` - Protocol Independent API. See `PI` repository docs directory for some more about this, although I do not know if that particular document is up to date with the code.
- `v1.0.x` - As of 2017-Mar-12, v1.0.3 is the latest version of the P4 specification in the v1.0.x series, although there is a v1.0.4 planned by the P4 language design committee to clarify a few things, e.g. eliminating the portion of the specification that says that primitive actions within a compound action are to be performed in parallel -- v1.0.4 will specify sequential behavior within a compound action.
- `v1.1.x` - As of 2016-Dec-14 when a draft version of the P4_16 language specification was released, the v1.1.x series of specifications was no longer publicized and effectively deprecated.

### p4c

`p4c` is a front end compiler for both P4_14 and P4_16 programs. The repository also contains a back end for `behavioral-model` (aka `bmv2`), and a couple of other sample back ends. It is intended to be able to easily add new back ends to it.

`p4c` is written in C++, not Python as `p4-hlir` is.

## behavioral-model

`behavioral-model` contains the code for what is often called `bmv2` (an abbreviation for "Behavioral Model Version 2").

The 1st version of the behavioral model, produced as output from the code in the `p4c-behavioral` repository, is like a 'P4 to C compiler', i.e. source to source translation. Changing the P4 program requires recompiling to a new C program, then recompiling that C code.

bmv2 is more like an interpreter for all possible P4 programs, which bmv2 configures itself to behave as, using the contents of the bmv2 JSON configuration file produced by a couple of the compilers above. Changing the P4 source code requires recompiling it to produce a new bmv2 JSON configuration file, but does not require recompiling bmv2.

See
here
for more discussion of why bmv2 was created.

## p4-hlir

Written in Python. Parses source code for P4_14 (v1.0.x) and P4 v1.1.x versions of P4. Creates Python objects in memory representing P4 source code objects such as headers, tables, field lists, actions, etc.

Also performs target-independent semantic checks, such as references to undefined tables or fields, and dead code elimination, e.g. eliminating tables that are never applied, or actions that are not an action of any live table.

`p4-hlir` does not parse P4_16 programs. The `p4c` repository contains the new compiler for both P4_14 and P4_16.

## p4c-behavioral

Deprecated, and may be deleted by 2018. Use `behavioral-model` instead.

`p4c-behavioral` uses `p4-hlir` as its front end to parse source code and produce an IR. From that IR it generates a C/C++ behavioral model (version 1, not for use with bmv2). This repository has seen very few changes since Aug 2016. This is because bmv2 in `behavioral-model` is recommended over this v1 kind of behavioral model (see `behavioral-model` below for more info).

There may be a conflict between installing this software, and that in the `p4c-bm` repo, as they both seem to install a Python module called p4c-bm.

### p4c-bm

`p4c-bm` uses `p4-hlir` as its front end to parse source code and produce an IR. From that IR it can generate a bmv2 JSON configuration file, used as input to the `behavioral-model`. It can also generate C++ PD code.

## Executable files created during installation

The executables are shown with the path where they are created/installed using the latest README instructions as of March 2017.

| Repository | Executable | Notes |
|---|---|---|
| p4-hlir | /usr/local/bin/p4-shell | |
| p4-hlir | /usr/local/bin/p4-validate | |
| p4-hlir | /usr/local/bin/p4-graphs | |
| p4c-bm | /usr/local/bin/p4c-bmv2 | Compile P4_14 or v1.1 to bmv2 JSON configuration file, PD API files, and optionally a few other things. |
| p4c | /build/p4c-bm2-ss | Compile P4_14 or P4_16 to bmv2 JSON configuration file |
| p4c | /build/p4c-ebpf | |
| p4c | /build/p4test | |
| behavioral-model | /usr/local/bin/bm_CLI | wrapper script for runtime_CLI.py |

| | | |
|---|---|---|
| behavioral-model | /usr/local/bin/bm_nanomsg_events | wrapper script for nanomsg_client.py |
| behavioral-model | /usr/local/bin/bm_p4dbg | wrapper script for p4dbg.py |
| behavioral-model | /usr/local/bin/simple_switch | executable compiled from C/C++ code |
| behavioral-model | /usr/local/bin/simple_switch_CLI | wrapper script for sswitch_CLI.py |

Sample command lines to compile P4 source file foo.p4 to bmv2 JSON configuration file:

```
# foo.p4 is P4_14 source code
p4c-bmv2 --json foo.json foo.p4
p4c-bm2-ss --p4v 14 -o foo.json foo.p4

# foo.p4 is P4_16 source code
p4c-bm2-ss -o foo.json foo.p4
```

Sample command line for converting P4_14 source code to P4_16 source code:

```
p4test --p4v 14 --pp foo-translated-to-p4-16.p4 foo-in-p4-14.p4
```

## Python modules created during installation

Python modules currently installed can be shown using 'pip list' command. You can see which directory the files are in using 'pip show ' command. This could be a system-wide directory requiring root privileges, or if you have created a Python virtual environment, it may be a directory anywhere you wish in the file system, without requiring root privileges to add modules.

| **Repository** | **Python module** |
|---|---|
| p4-hlir | p4-hlir |
| p4c-bm | wheel (not P4-specific module) |
| p4c-bm | Tenjin (not P4-specific module) |

| | |
|---|---|
| p4c-bm | p4-hlir |
| p4c-bm | p4c-bm |
| behavioral-model | Does not install anything that shows up in output of 'pip list', but does install many Python files in /dist-packages directory, e.g. bmpy_utils.py, bm_runtime/ and sswitch_runtime/ directories, and .py files that have shell wrapper scripts for them installed in /usr/local/bin, listed above |