Tutorial – ABC

ALCom Lab

Outline

- ABC Introduction
- Logic Circuit Descriptions
- Data Structure in ABC
- Useful Commands in ABC
- Programming with ABC
- GitHub Tips

ABC INTRODUCTION

Introduction

- ABC [2](ABC: System for Sequential Logic Synthesis and Formal Verification) is an academic open source front-end EDA tool
  - Logic synthesis
  - Optimization
  - Verification
  - Technology mapping

Installation

- Prerequisites: linux or mac OS environment
  - for windows, try wsl (recommended) or virtual machine
  - or just work on your work station
  - For newly installed linux, you may need to run
    - sudo apt-get update
    - sudo apt-get upgrade
    - sudo apt-get install build-essential libreadline-dev
- Simply type "*make*" at the root directory
  - the compilation could take a few minutes
  - "make –j8"

LOGIC CIRCUIT DESCRIPTIONS

BLIF

- https://people.eecs.berkeley.edu/~alanmi/publications/other/blif.pdf

```
.model adder
.inputs a b cin
.outputs s cout
.names a b cin s
001 1
010 1
100 1
111 1
.names a b cin cout
11- 1
1-1 1
-11 1
.end
```

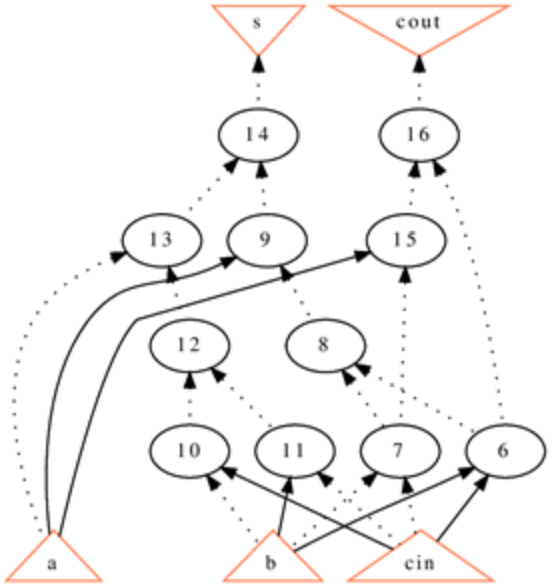
Data Structure in ABC

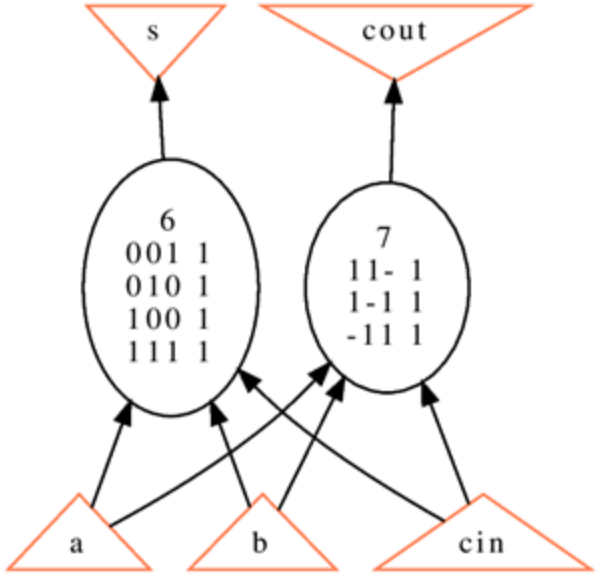
Abc_Frame_t

- Manage the whole ABC shell
  - command/package registration
  - manage all the data including designs, network status
- Don’t really have to look into it for now
  - We will show how to add your own package and command later

Abc_Ntk_t

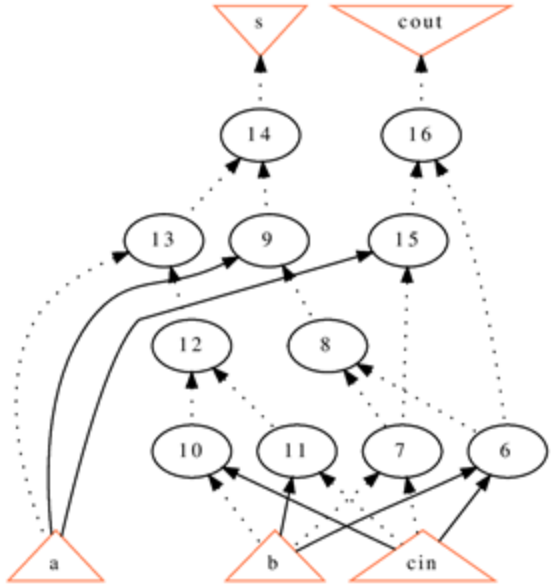
- The top level network
- A combinational network is represented as a directed cyclic graph
- There can be complemented edges (e.g. in AIG)

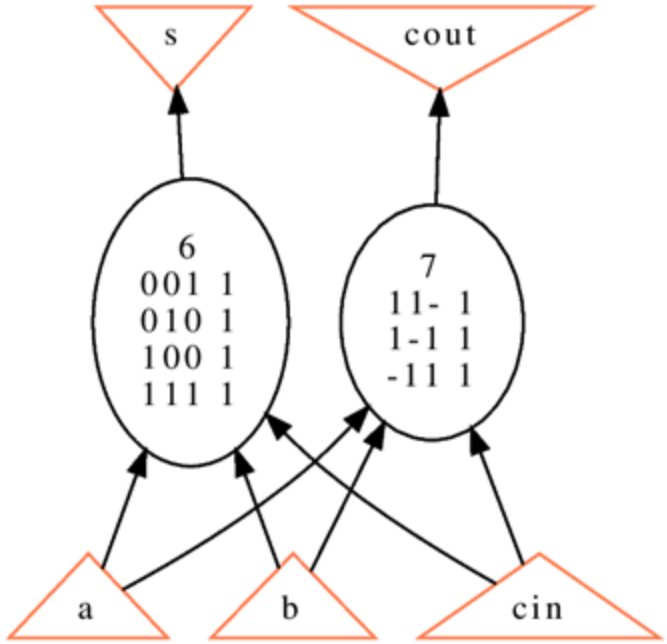




Abc_Obj_t

- Each node (PI, PO, gate) is an Abc_Obj_t
- Each node has an ID
- The functionality of nodes can be represented in many different ways
  - SOP
  - BDD
  - AIG





USEFUL COMMANDS IN ABC

Commands Usage

- *help* lists all the commands
- *help –d* list all the commands with details
- Adding option *-h* shows the usage and description of a command

```
abc 01> read_blif -h
usage: read_blif [-nmach] <file>
                 reads the network in binary BLIF format
                 (if this command does not work, try "read")
        -n     : toggle using old BLIF parser without hierarchy support [default = no]
        -m     : toggle saving original circuit names into a file [default = no]
        -a     : toggle creating AIG while reading the file [default = no]
        -c     : toggle network check after reading [default = yes]
        -h     : prints the command summary
        file   : the name of a file to read
```

Read / Write

- *read, read_blif, read_verilog, read_aiger*
- *write_blif, write_verilog, write_aiger*

Printing

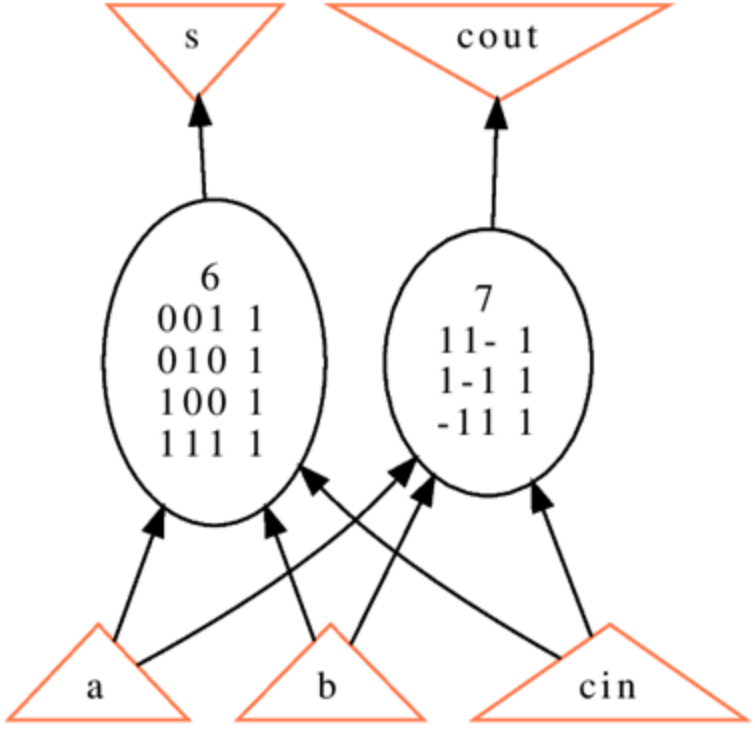
- *print_stats, print_io*

```
UC Berkeley, ABC 1.01 (compiled Aug 11 2022 15:09:19)
abc 01> read_blif adder.blif
abc 02> print_stats
adder                          : i/o =    3/    2  lat =    0  nd =     2
 edge =      6  cube =     7  lev = 1
abc 02> print_io
Primary inputs (3):  0=a 1=b 2=cin
Primary outputs (2): 0=s 1=cout
Latches (0):
abc 02>
```

```
12 .model adder
11 .inputs a b cin
10 .outputs s cout
 9 .names a b cin s
 8 001 1
 7 010 1
 6 100 1
 5 111 1
 4 .names a b cin cout
 3 11- 1
 2 1-1 1
 1 -11 1
13 .end
```

Printing

- *show*




```
 12 .model adder
 11 .inputs a b cin
 10 .outputs s cout
  9 .names a b cin s
  8 001 1
  7 010 1
  6 100 1
  5 111 1
  4 .names a b cin cout
  3 11- 1
  2 1-1 1
  1 -11 1
13  .end
```

```
UC Berkeley, ABC 1.01 (compiled Aug 11 2022 15:09:19)
abc 01> read_blif adder.blif
abc 02> print_stats
adder                         : i/o =    3/    2  lat =    0  nd =     2
 edge =      6  cube =     7  lev = 1
abc 02> print_io
Primary inputs (3):  0=a 1=b 2=cin
Primary outputs (2): 0=s 1=cout
Latches (0):
abc 02>
```

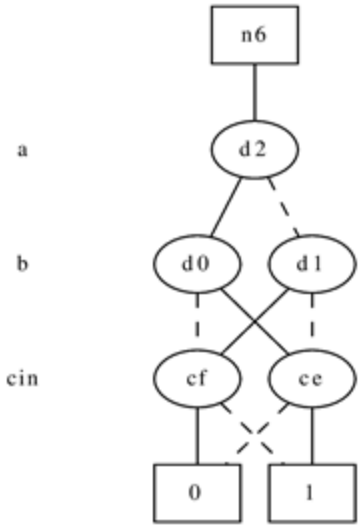
Local Function Representation

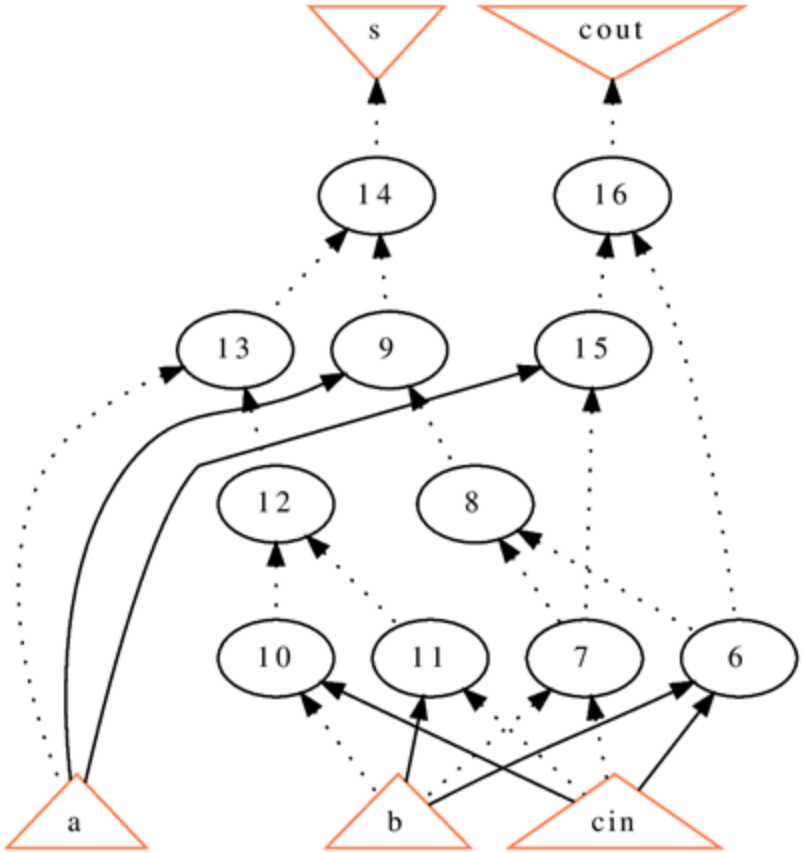
- *sop, aig, bdd*

```
abc 01> read_blif adder.blif
abc 02> print_stats
adder                          : i/o =    3/    2  lat =    0  nd
=     2  edge =      6  cube =     7  lev = 1
abc 02> aig
abc 02> print_stats
adder                          : i/o =    3/    2  lat =    0  nd
=     2  edge =      6  aig  =    13  lev = 1
abc 02> bdd
abc 02> print_stats
adder                          : i/o =    3/    2  lat =    0  nd
=     2  edge =      6  bdd  =     7  lev = 1
```

Network Structure

- *strash* (structural hashed aig)
- *collapse* (one BDD for each PO)
  - *show_bdd* shows the first PO
  - *show_bdd -g* shows all POs





Optimization

- Commands
  - *strash* (structural hashed aig)
  - *fraig* (functionally reduced aig)
  - *dc2, rewrite, balance, resub, refactor, …*
- Scripts (sequence of commands, defined in “abc.rc”)
  - *resyn, resyn2, compress, share, …*

Verification

- *cec* (combinational equivalence check)
- *miter* (construct miter)
- *sim* (simulate to see if the output can be 1)
- *sat* (SAT solving)

```
abc 03> cec ./adder.blif
Networks are equivalent after structural hashing.  Time =     0.00 sec
abc 03> cec ./notAnAdder.blif
Networks are NOT EQUIVALENT.  Time =     0.01 sec
Verification failed for at least 1 outputs:  s
Output s: Value in Network1 = 1. Value in Network2 = 0.
Input pattern:  a=1 b=0 cin=0
abc 03>
```

PROGRAMMING WITH ABC

Adding Your Own Commands

- Creating an external package
  - It works without the need to change other parts of the ABC
  - You can then use the data structures and functions defined in ABC and other packages

Create External Package in ABC

- First, create a directory named “ext…”, e.g. “ext-lsv” under “./src”
  - The makefile would automatically look for any directory under ".src/" whose name starts with “ext”
- Your code should only be inside this folder

Create External Package in ABC (cont.)

- Under the created directory, create a file named "module.make" and the c/cpp files you need.
- In "module.make", type:

```
SRC += src/ext-lsv/file1.cpp \
       src/ext-lsv/file2.cpp \
       …
       src/ext-lsv/file3.cpp
```

Create External Package in ABC (cont.)

- In one of your .cpp files, you have to register your package and commands

Create External Package in ABC (cont.)

- Each command function should take exactly these three arguments
- Commands are registered in this init() function

```
1   #include "base/abc/abc.h"
2   #include "base/main/main.h"
3   #include "base/main/mainInt.h"
4
5   static int Lsv_CommandPrintGates(Abc_Frame_t* pAbc, int argc, char** argv);
6
7   void init(Abc_Frame_t* pAbc) {
8     Cmd_CommandAdd(pAbc, "LSV", "lsv_print_gates", Lsv_CommandPrintGates, 0);
9   }
```

Create External Package in ABC (cont.)

```
15    struct PackageRegistrationManager {
16      PackageRegistrationManager() { Abc_FrameAddInitializer(&frame_initializer); }
17    } lsvPackageRegistrationManager;
```

Just a variable name, can be anything

Create External Package in ABC (cont.)

- The package is already created and registered in our PA
- You just have to register your own command

Create External Command in ABC (cont.)

Will this function change the current network?

The string to call this command

```
void init(Abc_Frame_t* pAbc) {
  Cmd_CommandAdd(pAbc, "LSV", "lsv_print_gates", Lsv_CommandPrintGates, 0);
}
```

Group of your command (shown in *help*)

The function that implements the command

Create External Command in ABC

Parse options

Do anything you want

Print usage

```
33  int Lsv_CommandPrintGates(Abc_Frame_t* pAbc, int argc, char** argv) {
34    Abc_Ntk_t* pNtk = Abc_FrameReadNtk(pAbc);
35    int c;
36    Extra_UtilGetoptReset();
37    while ((c = Extra_UtilGetopt(argc, argv, "h")) != EOF) {
38      switch (c) {
39        case 'h':
40          goto usage;
41        default:
42          goto usage;
43      }
44    }
45    if (!pNtk) {
46      Abc_Print(-1, "Empty network.\n");
47      return 1;
48    }
49    Lsv_NtkPrintGates(pNtk);
50    return 0;
51
52  usage:
53    Abc_Print(-2, "usage: lsv_print_gates [-h]\n");
54    Abc_Print(-2, "\t        prints the gates in the network\n");
55    Abc_Print(-2, "\t-h    : print the command usage\n");
56    return 1;
57  }
```

Example Code in LSV PA

- This example can be found here

```
19  void Lsv_NtkPrintGates(Abc_Ntk_t* pNtk) {
20    Abc_Obj_t* pObj;
21    int i;
22    Abc_NtkForEachObj(pNtk, pObj, i) {
23      printf("Object Id = %d, name = %s\n", Abc_ObjId(pObj), Abc_ObjName(pObj));
24      Abc_Obj_t* pFanin;
25      int j;
26      Abc_ObjForEachFanin(pObj, pFanin, j) {
27        printf("  Fanin-%d: Id = %d, name = %s\n", j, Abc_ObjId(pFanin),
28               Abc_ObjName(pFanin));
29      }
30    }
31  }
```

Programming in ABC

- Many inline functions are provided for the basic operations, for example:
  - Abc_NtkForEachAnd(): iterate through all and gates
  - Abc_NtkPoNum(): get the number of primary outputs
  - Abc_ObjIsPi(): check if an object is PI
  - Abc_ObjFanin0(): get the first fanin object of the object
  - Can be found in *"src/base/abc/abc.h"*
- You can refer to other commands to see how they use these functions
  - most commands can be traced from *"./src/base/abci/abc.c"*

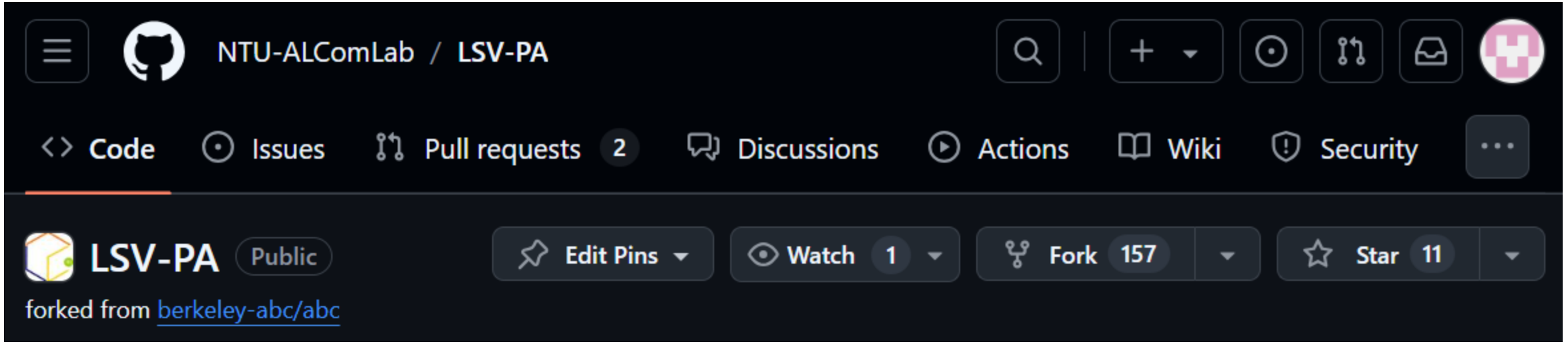
Tips

- Built-in vectors
  - requires some time to understand
  - but works well with the built-in functions
  - see src/misc/vec
- Try not to access the data inside Abc_Obj_t or Abc_Ntk_t directly. Use the defined interface (api functions)

GitHub Tips

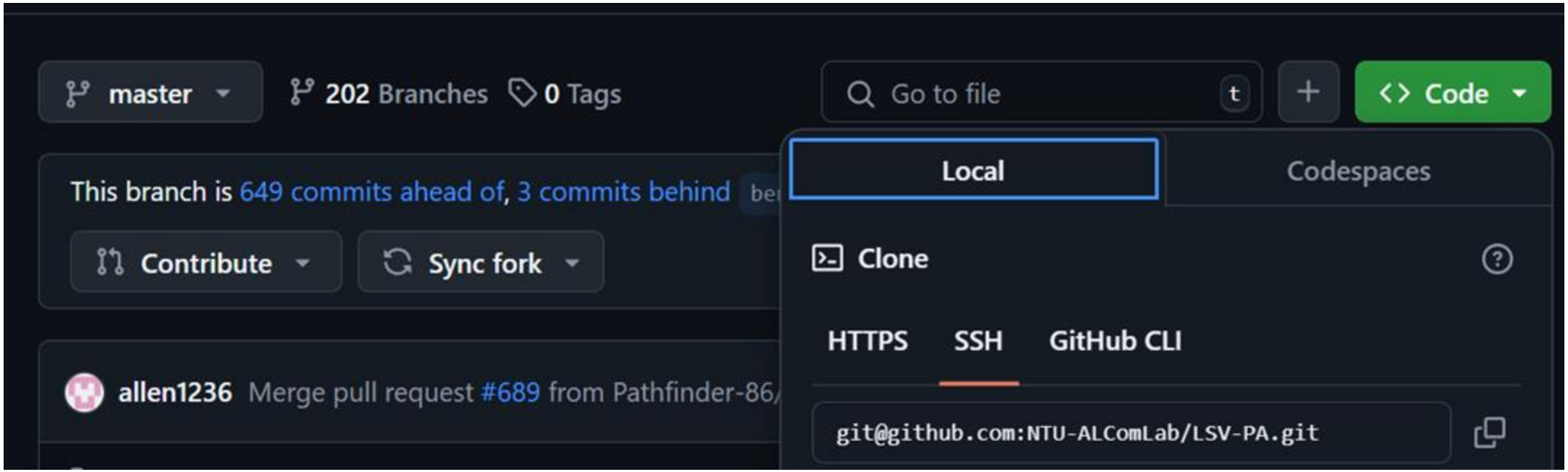
Forking the Project

- Go to the LSV-PA repo on GitHub, click “Fork”
- This will create your own repository



Start Working on Your Repo locally

- Go to your forked repository, click the green button “code”
- Copy the url under SSH tab
- In your terminal, type “git clone <url>”

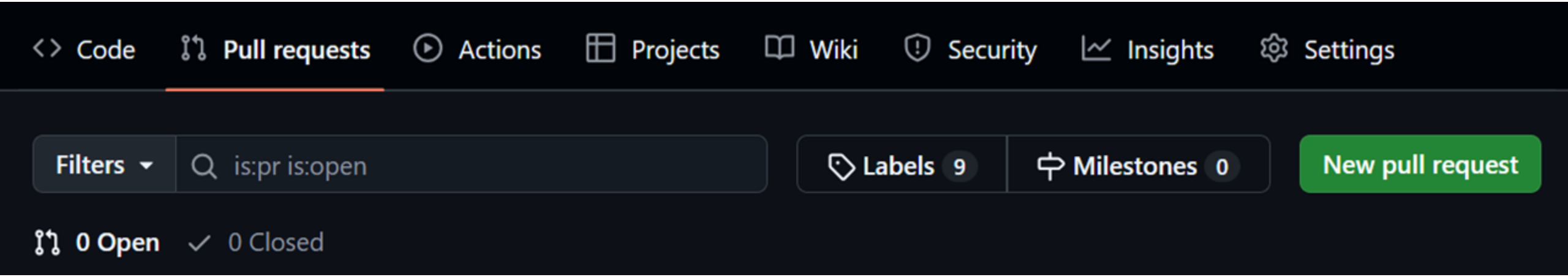


Developing on Your Forked Repo

- Use “git status” to view your modification status
- Use “git diff” to view your detail changes
- After you are done, use “git add <file>” to add your changes
- Then, use “git commit -m <comment>” to commit them
- Finally, use “git push” to push the change to your GitHub repository

Creating Pull Request

- In your repository on GitHub, go to the “Pull requests” tab
- click “New pull request”



Creating Pull Request (cont.)

- select the source and target repo and branches
  - For the first part of the assignment, send the pull request to LSV-PA/master
  - For the other part, send the pull request to LSV-PA/<your_student_id>