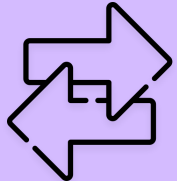


Why OpenQASM?

Convenient &
standardised format
for quantum circuits



Hardware-agnostic
representations



Closer to real
hardware



Straightforward syntax



Upgrades in OpenQASM 3.0

A complete language for quantum circuits now with salient features as compared to OpenQASM 2.0:

- Complete type system (constants, variables, operators, casting, expressions, ...)
- Control flow statements
- Support for versatile circuit and operation expression
- Dynamic circuit subroutines and external function calls
- Support for lower level operation definition
- Extended grammar for pulse operations

Physical level

`delay` statements, adding relative timings to operation

Type `stretch` to resolve concrete durations at compile time for granular calibration

Support for qubit-specific calibration instructions via `defcal` construct

Design Philosophy & execution

Arbitrary classical control flow, gate modifiers and timings

Ability to perform new kinds of circuits and experiments

Pulse level calibration and **multi level optimization**

Logical Level

Ability to use quantum-classical dependencies in quantum circuits

Native support for classical computation on measurement results

Robusts classical types with support for classical control flow

Support for `int`, `uint`, `float`, `bool`, and `bit` for classical types with functionality to specify a type with exact bit-precision for **low-level** and **bitwise** operations

Goals for the OpenQASM 3.0 Translator



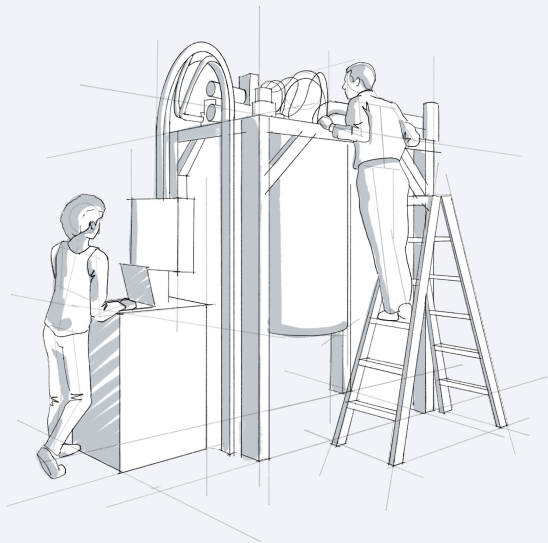
Goals

- Having a working translator
- Support most of the OpenQASM3 features
- Provide feedback on the proposed reference OpenQASM3 parser implementation (AWS team)
- Provide feedback to the OpenQASM3 TSC about the language specification

Non-Goals

- Having an **efficient** translator
- Implementation of **all** the OpenQASM3 specification

Out-of-scope OpenQASM3 features



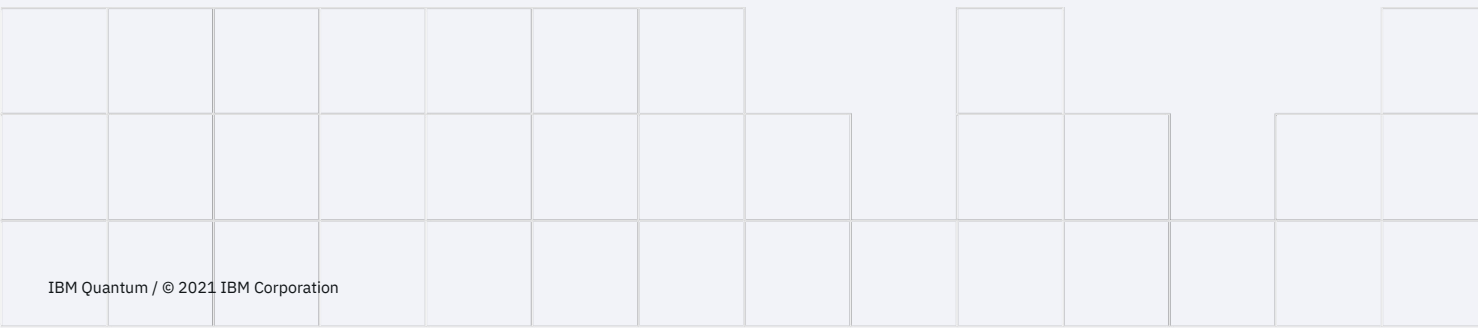
Defcal Grammars

- Completely different language embedded in the OpenQASM3 grammar
- Integration with a lot of components required at the same time (Qiskit-terra, Qiskit-Pulse, Qiskit-Runtime, etc)
- Can be implemented at a later stage when the **openpulse** grammar is mature

Timing & duration

- **delay[t]**, **stretch**, **box**, and **duration**
- Computing **stretch** is hard: multi-objective linear programming problem

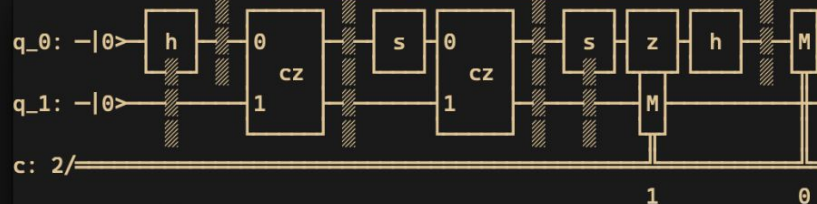
DEMO



Demo

```
*rb.qasm > *test.qasm > *qft.qasm > ... > buffers (ϕ|tests|ψ) python3 build_ast.py ../../../../examples/rb.qasm -I ../../../../examples/ -t
```

```
7 // One randomized benchmarking sequence
6 OPENQASM 3;
5 include "stdgates.inc";
4
3 qubit[2] q;
2 bit[2] c;
1
8 reset q;
1 h q[0];
2 barrier q;
3 cz q[0], q[1];
4 barrier q;
5 s q[0];
6 cz q[0], q[1];
7 barrier q;
8 s q[0];
9 z q[0];
10 h q[0];
11 barrier q;
12
13 // Original example has: measure q -> c;
14 // this will supported in the coming days :)
15 measure q[0] -> c[0];
16 measure q[1] -> c[1];
```

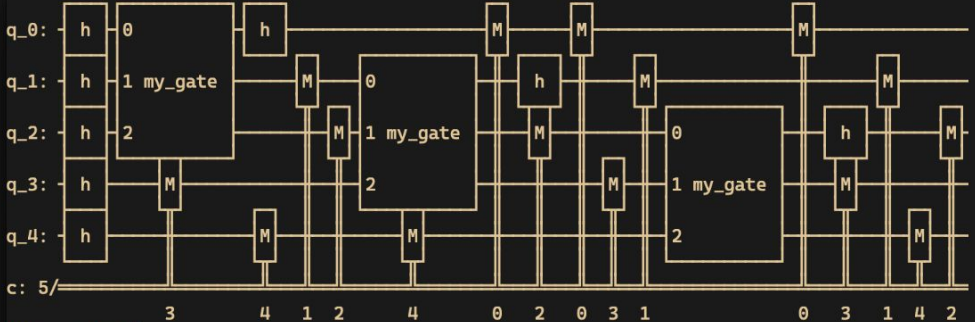


```
(ϕ|tests|ψ) |
```

Demo

```
test.qasm buffers  $\langle \phi | \text{tests} | \psi \rangle$  python3 build_ast.py ../src/openqasm/translator/tests/test.qasm -I ../../examples/ -t
```

```
1 OPENQASM 3.0;
2 include "stdgates.inc";
3
4 gate my_gate a, b, c {
5   cx a, b;
6   cx b, c;
7 }
8
9 const n = 5;
10 qubit[n] q;
11 uint[n] c;
12
13 h q;
14
15 for i in [0: n-2] {
16   my_gate q[i], q[i+1], q[i+2];
17   h q[i];
18   for j in [0: n] { c[j] = measure q[j]; }
19 }
```

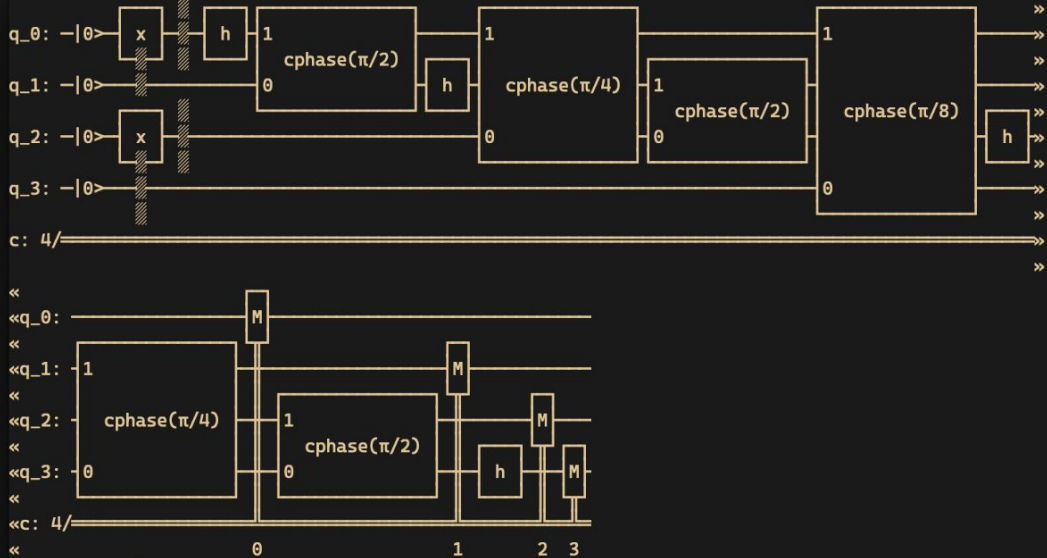


$\langle \phi | \text{tests} | \psi \rangle$

Demo

```
...> ^1pong.qasm ^2qec.qasm ^3qft.qasm ^4qpt.qasm ^5... buffers <| tests |> python3 build_ast.py ../../examples/qft.qasm -I ../../examples/ -t
```

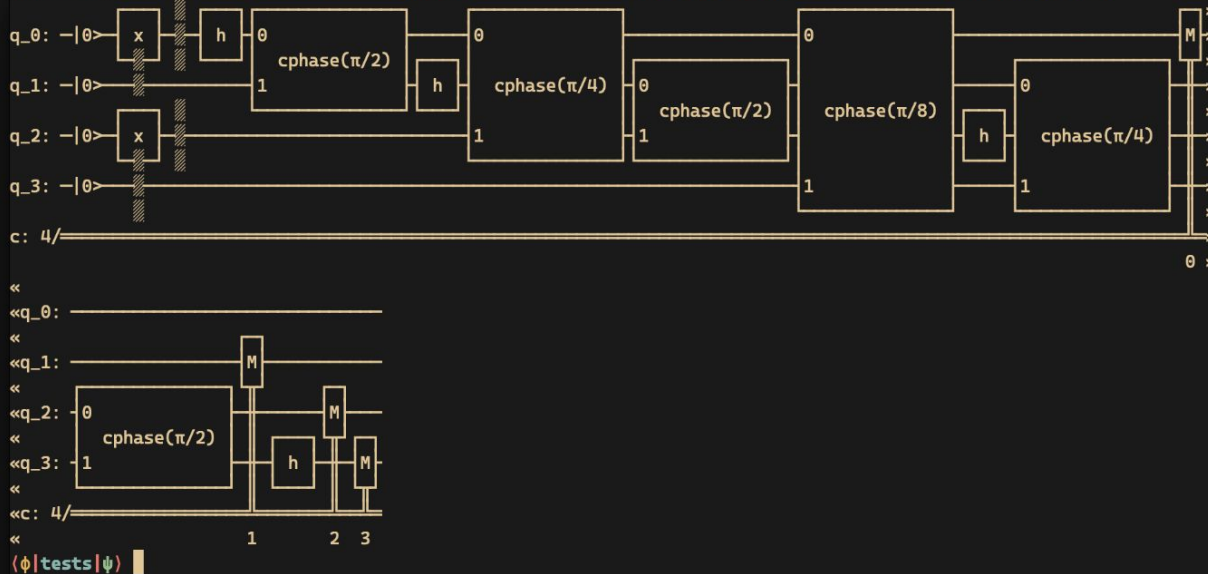
```
20 // quantum Fourier transform
19 OPENQASM 3;
18 include "stdgates.inc";
17 qubit[4] q;
16 bit[4] c;
15 reset q;
14 x q[0];
13 x q[2];
12 barrier q;
11 h q[0];
10 cphase(pi / 2) q[1], q[0];
9 h q[1];
8 cphase(pi / 4) q[2], q[0];
7 cphase(pi / 2) q[2], q[1];
6 h q[2];
5 cphase(pi / 8) q[3], q[0];
4 cphase(pi / 4) q[3], q[1];
3 cphase(pi / 2) q[3], q[2];
2 h q[3];
1 c[0] = measure q[0];
21 c[1] = measure q[1];
1 c[2] = measure q[2];
2 c[3] = measure q[3];
```



Demo

```
...> *qft.qasm *qft_forloop.qasm *adder.qasm > buffers (φ|tests|ψ) python3 build_ast.py ../src/openqasm/translator/tests/qft_forloop.qasm -I ../../../../examples/ -t
```

```
21 // quantum Fourier transform
20 OPENQASM 3;
19 include "stdgates.inc";
18
17 const n = 4;
16
15 qubit[n] q;
14 bit[n] c;
13
12 reset q;
11 x q[0];
10 x q[2];
9 barrier q;
8
7 for i in [0: n-1] {
6   h q[i];
5   for j in [0: i+1] {
4     cphase(pi / (2**(i-j+1))) q[j], q[i+1];
3   }
2 }
1 h q[n-1];
22 []
1 for i in [0: n] {
2   c[i] = measure q[i];
3 }
```

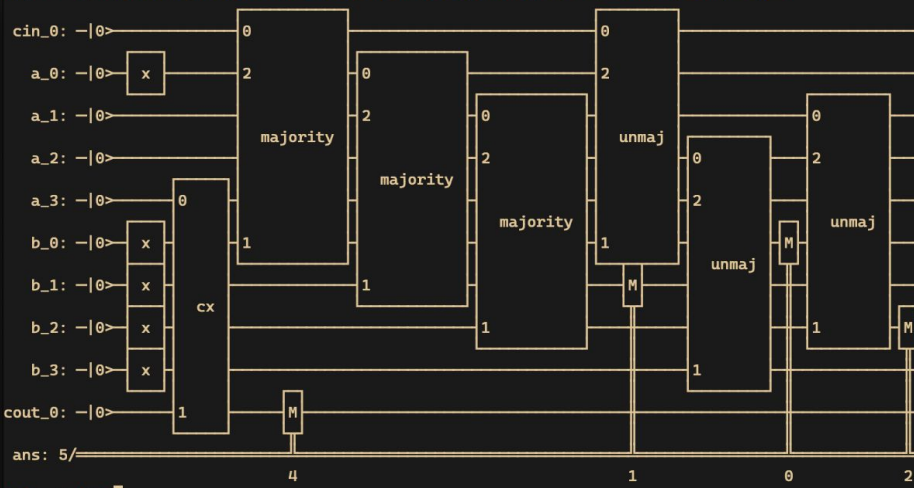


```
«q_0: _____
«
«q_1: _____
«
«q_2: 0 _____
«      1 _____
«q_3: 1 _____
«
«c: 4/ _____
«
(φ|tests|ψ) █
```

Demo

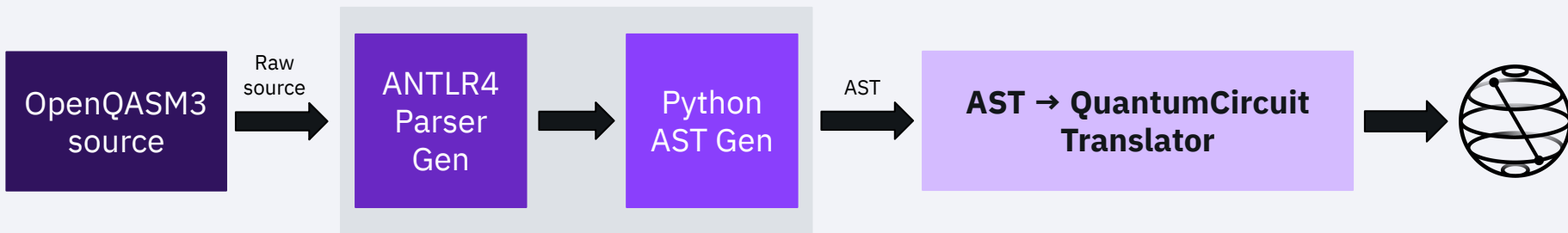
```
adder.qasm *alignment.qasm > *cphase.qasm > *dd.qasm >... buffers (phi|tests|psi) python3 build_ast.py ../../examples/adder.qasm -I ../../examples/ -t
```

```
18 */
17 OPENQASM 3;
16 include "stdgates.inc";
15
14 gate majority a, b, c {
13   cx c, b;
12   cx c, a;
11   ccx a, b, c;
10 }
9
8 gate unmaj a, b, c {
7   ccx a, b, c;
6   cx c, a;
5   cx a, b;
4 }
3
2 qubit[1] cin;
1 qubit[4] a;
22 qubit[4] b;
1 qubit[1] cout;
2 bit[5] ans;
3 uint[4] a_in = 1; // a = 0001
4 uint[4] b_in = 15; // b = 1111
5 // initialize qubits
6 reset cin;
7 reset a;
8 reset b;
9 reset cout;
10
11 // set input states
12 for i in [0: 4] {
13   if(bool(a_in[i])) x a[i];
14   if(bool(b_in[i])) x b[i];
15 }
16 // add a to b, storing result in b
17 majority cin[0], b[0], a[0];
18 for i in [0: 2] { majority a[i], b[i + 1], a[i + 1]; }
19 cx a[3], cout[0];
20 for i in [2: -1: 0] { unmaj a[i], b[i+1], a[i+1]; }
21 unmaj cin[0], b[0], a[0];
22 measure b[0] -> ans[0];
23 measure b[1] -> ans[1];
24 measure b[2] -> ans[2];
25 measure cout[0] -> ans[4];
```



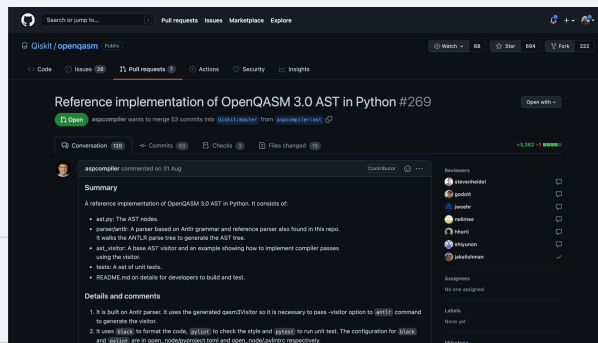
```
(phi|tests|psi) █
```

Architecture



```

19 OPENQASM 3;
18 include "stdgates.inc";
17 qubit[4] q;
16 bit[4] c;
15 reset q;
14 x q[0];
13 x q[2];
12 barrier q;
11 h q[0];
10 cphase(pi / 2) q[1], q[0];
9 h q[1];
8 cphase(pi / 4) q[2], q[0];
7 cphase(pi / 2) q[2], q[1];
6 h q[2];
5 cphase(pi / 8) q[3], q[0];
4 cphase(pi / 4) q[3], q[1];
3 cphase(pi / 2) q[3], q[2];
2 h q[3];
1 c[0] = measure q[0];
21 c[1] = measure q[1];
1 c[2] = measure q[2];
2 c[3] = measure q[3];
  
```



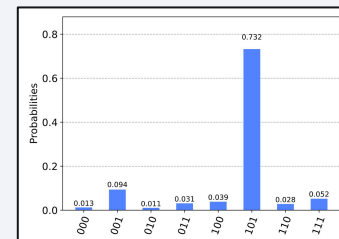
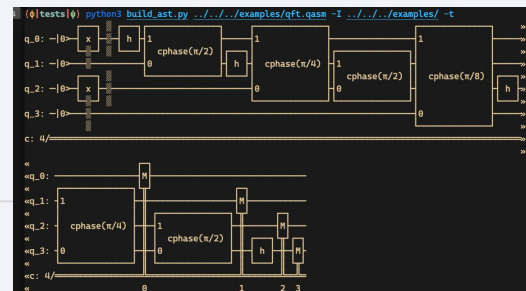
Reference implementation of OpenQASM 3.0 AST in Python #269

Summary

- A reference implementation of OpenQASM 3.0 AST in Python. It consists of:
 - `ast.py`: The AST nodes.
 - `parser/antlr4`: A parser based on Antlr grammar and reference parser also found in this repo.
 - `generator`: Builds the ANTLR source files to generate the AST files.
 - `ast_visitor`: A base AST visitor and an example showing how to implement compiler passes using the visitor.
 - `tests`: A set of unit tests.
- README.md on details for developers to build and test.

Details and comments

- It is built on Antlr parser. It uses the generated `qasm3Visitor` so it is necessary to pass `-visitor` option to `antlr4` command to generate the visitor.
- It uses `pytest` to format the code, `pylint` to check the style and `pytest` to run unit test. The configuration for `pylint` and `pytest` are in `open_nodebyproject.toml` and `open_nodebyproject.pyproject` respectively.

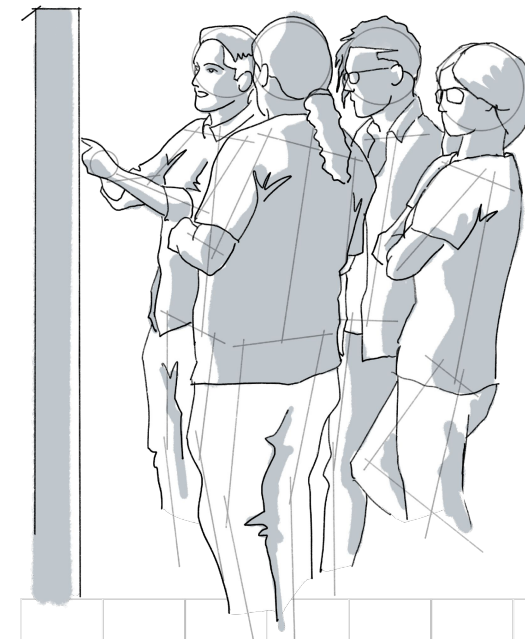


How are we bringing about changes?

Impact of “QAMP” #23 on the shaping of OpenQASM3 compiler Development

Under the guidance of Technical Steering Committee (TSC)

- Providing an **end-user’s perspective** to the Technical Steering Committee (TSC)
- **Reporting inconsistencies** in the Language Specification and Examples
- **Tightening** the AST implementation of OpenQASM3
- Building a platform for **translating** and **running** OpenQASM3 code with the existing Qiskit specifications
- **Implementing features** from the current grammar specification for OpenQASM3
 - ***for/while***: only quantum vs. mix of quantum-classical operations
 - ***if-else***: Implementing conditions based on measurement result



How are we bringing about changes?

- Tight integration with TSC-Approved OpenQASM3 AST

- Lots of bug fixes while working on the Translator

Open Casting angles to floats - slow and precise or quick and lossy? #280
 mbhealy opened this issue 13 days ago · 4 comments

would really love to see the ability to choose to preserve or promote a specific operator or identifier to ensure better precision in C, I have seen where people will explicitly encode the floating point number they want to ensure speed&accuracy. Something like that might be a useful addition as well.

Hope this helps some :)

jwoehr commented 2 days ago Contributor

Some folks concerned with this issue might be interested in QAMP Fall 2021 Team No. 23's ways of handling this which will be presented in our Oct. 7 session for QAMP checkpoint.

Open Reference implementation of OpenQASM 3.0 AST in Python #269
 aspcompiler wants to merge 53 commits into @qiskit/master from aspcompiler/ast

AbeerVaishnav13 commented 7 days ago

Hi @aspcompiler, we were in the process of using your QASM3 to AST generator for the Qiskit Advocate Mentorship Program project on OpenQASM3 and noticed a few bugs/potential changes...

1. No restriction on re-assignment to const types

Presently there are no restrictions on re-assignment to const types. For example, if I write the following code:

```
const my_const = 10; // parses as a 'ConstantDeclaration' statement
my_const = 5; // parses as a 'ClassicalAssignment' statement - WRONG
```

Shouldn't the AST gen step throw an error for statement-2 i.e. re-assignment to an immutable identifier?

2. 'bool' instead of NoDesignatorTypeName.bool

Open Reference implementation of OpenQASM 3.0 AST in Python #269
 aspcompiler wants to merge 53 commits into @qiskit/master from aspcompiler/ast

nelimee commented 19 days ago

Hi @aspcompiler,

We started to use your AST generator for the Qiskit Advocate Mentorship Program project on OpenQASM3 and found some things that might be considered as issues.

Here is a quick description of each, sadly I did not have the time to investigate the source of these issues yet:

```
# Code to get the AST from one of the example files in https://github.com/aspcompiler/openqasm/tree/ast/examples
from openqasm.parserantlr.qasm_parser import parse

file = "adder.qasm"
with open(file, "r") as f:
    qasm_str = f.read()

ast = parse(qasm_str)
```

Open Reference implementation of OpenQASM 3.0 AST in Python #269
 aspcompiler wants to merge 53 commits into @qiskit/master from aspcompiler/ast

jakeshman commented 7 days ago · edited · Contributor

@AbeerVaishnav13, some quick answers, since I'm looking at this (but am not on the main author team).

- reassignment to const: that probably isn't the role of the AST to throw an error on that, because AST generation is more about syntax than programme correctness. The first pass of AST generation (which the reference implementation here is) doesn't generally track which identifiers are being assigned, and what their types are - that would most likely come in a later, compilation stage.
- cast operator doesn't preserve enum: that looks like a bug to me.

Fixed NoDesignatorType parsing ✓ @93ddd

aspcompiler commented 7 days ago Contributor Author

@AbeerVaishnav13: @jakeshman just answered 1. I just pushed a fix for 2.

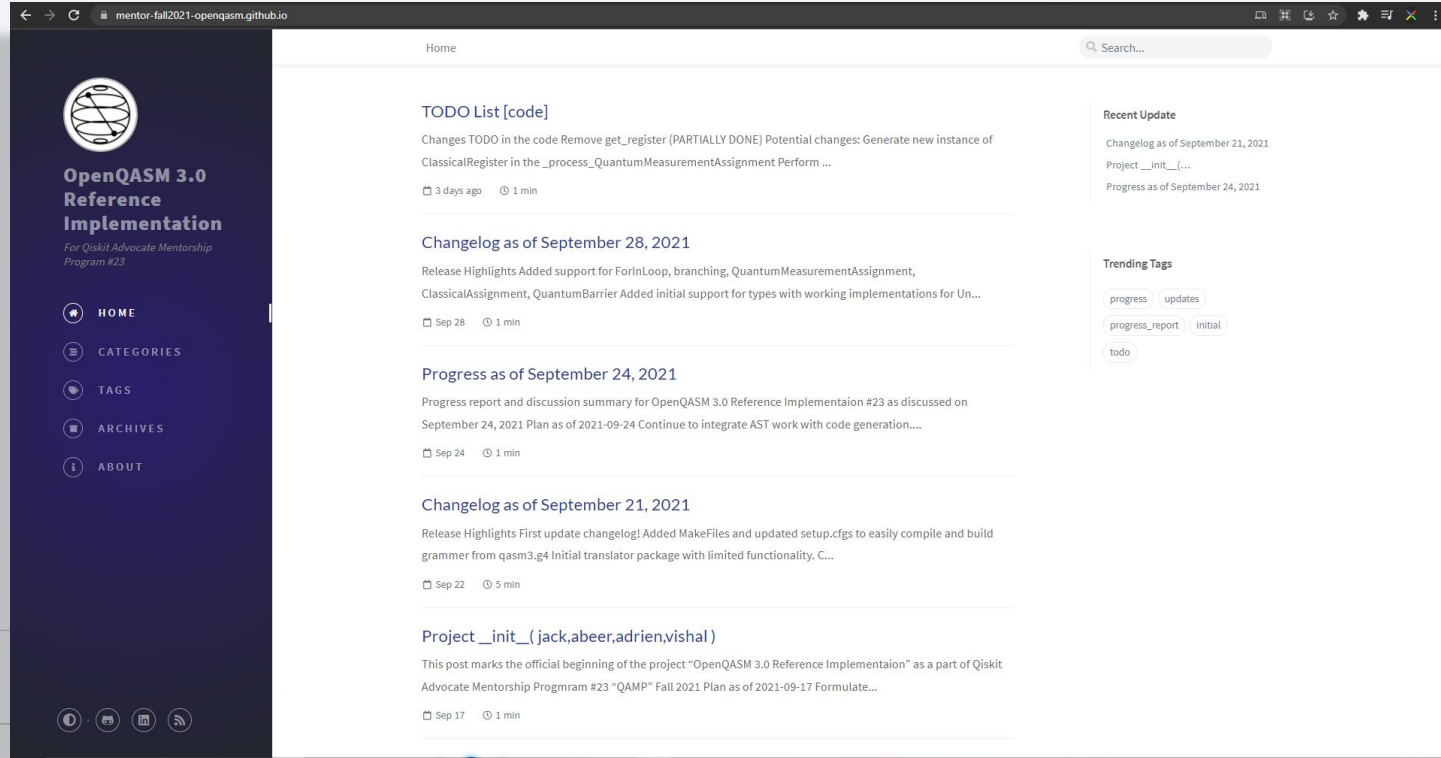
shiyunon and others added 5 commits 18 days ago

- fixed some issues raised in #269 ✗ ad0a9ce
- Small subtle adjustment to the identifier list and constant declaration ✗ 347841c
- adding a code snippet from adder.qasm as a new test case. ✗ 5696e17
- Merge branch 'master' into ast 2ccb56a
- Removed fixed type to sync up with grammar ✓ f380a95

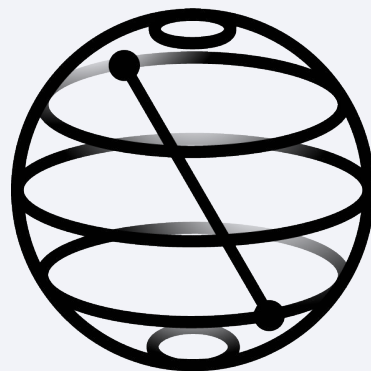
Dedicated blog website

Check it out here:

<https://mentor-fall2021-openqasm.github.io/>



The screenshot shows a web browser displaying the OpenQASM 3.0 Reference Implementation blog. The page has a dark blue sidebar on the left with the Qiskit logo and navigation links: HOME, CATEGORIES, TAGS, ARCHIVES, and ABOUT. The main content area is white and features a search bar at the top right. Below the search bar, there are several article entries, each with a title, a brief description, and a timestamp. The entries are: 'TODO List [code]' (3 days ago, 1 min), 'Changelog as of September 28, 2021' (Sep 28, 1 min), 'Progress as of September 24, 2021' (Sep 24, 1 min), 'Changelog as of September 21, 2021' (Sep 22, 5 min), and 'Project __init__(jack,abeer,adrien,vishal)' (Sep 17, 1 min). On the right side of the page, there are two sections: 'Recent Update' showing a changelog for September 21, 2021, and 'Trending Tags' with buttons for 'progress', 'updates', 'progress_report', 'initial', and 'todo'.



Thank You!