ESILV

myQLM

Gaëtan Rubez, PhD
Senior expert in Quantum Computing
2022

pyAQASM

Classical bits and measurement

- Allocate classical bits

```
cbits_reg = my_program.calloc(10)
```

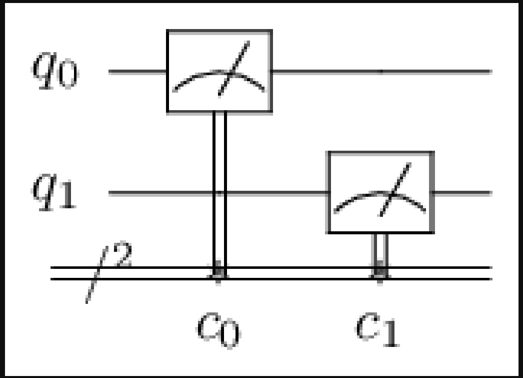
- Measure

```
my_program.measure(qbits[0], cbits[0])
```

```
my_program.measure(qbits, cbits)
```

Classical bits and measurement

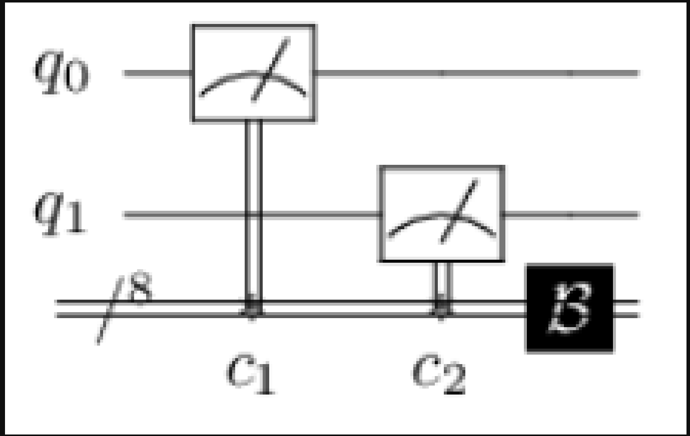
```
from qat.lang.AQASM import Program
#Create a program
prog = Program()
#Allocate qubits
qbits = prog.qalloc(2)
#Allocate bits
cbits = prog.calloc(2)
#Measure qbits in cbits
prog.measure(qbits, cbits)
#Create the circuit and print it
circ = prog.to_circ()
%qatdisplay circ
```



Atos

Logic operations on classical bits

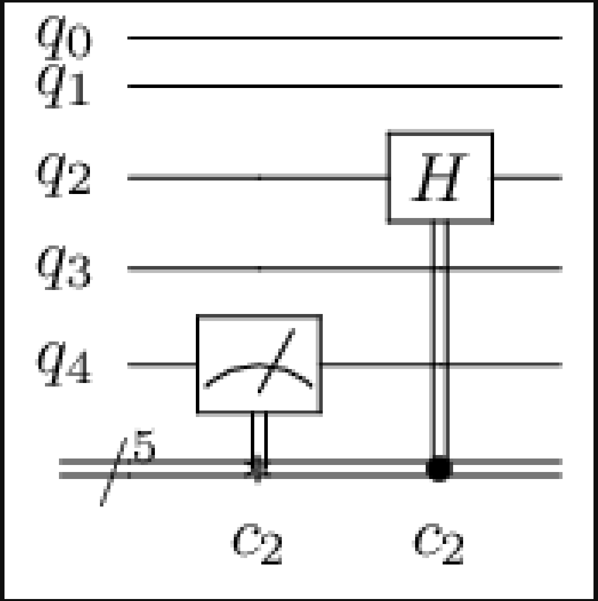
```
from qat.lang.AQASM import Program
prog = Program()
qbits = prog.qalloc(2)
cbits = prog.calloc(8)

#Measure qbits in cbits
prog.measure(qbits, cbits[1:3])
#Measure qbits in cbits
prog.logic(cbits[0], cbits[1] & cbits[2])

circ = prog.to_circ()
%qatdisplay circ
```



Boolean operators '&', '|', '^', '~'

Classical control

```
from qat.lang.AQASM import Program, H
prog = Program()
cbits = prog.calloc(5)
qbits = prog.qalloc(5)

# Initializing cbits[2]
prog.measure(qbits[4], cbits[2])
# Apply Hadamard only if cbits[2] is set
prog.cc_apply(cbits[2], H, qbits[2])

circ = prog.to_circ()
%qatdisplay circ
```



Qubit reset & Cbit reset

```python
from qat.lang.AQASM import Program

prog = Program()
qbits = prog.qalloc(8)
cbits = prog.calloc(8)

# Reseting qubits 1, 3 and 4, and cbit 0.
prog.reset([qbits[1], qbits[3:5]], [cbits[0]])
# Reseting only cbit 1
prog.reset([], [cbits[1]])
```

Abstract gates

- pyAQASM natively supports 4 parametrized gates (e.g RX, RY, RZ, PH)
- New gates can be defined using AbstractGate
- *There is two main interests :*
  - *Create boxes to implement latter during the development phase*
  - *If you know that the QPU you are target has a specific gate*

Abstract gates

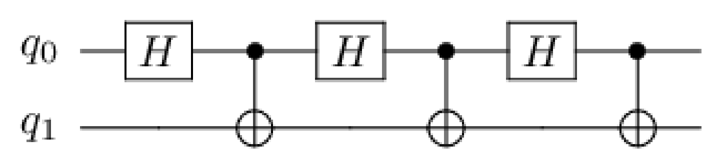
```python
from qat.lang.AQASM import Program, AbstractGate
prog = Program()
q = prog.qalloc(2)

# Abstract gates do not require a particular matrix description:
 they are boxes with a name and a signature:
my_gate = AbstractGate("mygate", # The name of the gate
                       [float], # Its signature: (here a single float)
                       arity=2) # Its arity
prog.apply(my_gate(0.3), q[0], q[1])

circ = prog.to_circ()
%qatdisplay circ
```

Abstract gates

```
from qat.lang.AQASM import Program, AbstractGate
prog = Program()
q = prog.qalloc(2)

My_CNOT = AbstractGate("MY\_CNOT", [], arity=2,
                       matrix_generator=lambda : np.array([[1,0,0,0],
                                                            [0,1,0,0],
                                                            [0,0,0,1],
                                                            [0,0,1,0]]))


prog.apply(My_CNOT(), q[0], q[1])
circ = prog.to_circ()
%qatdisplay circ
```

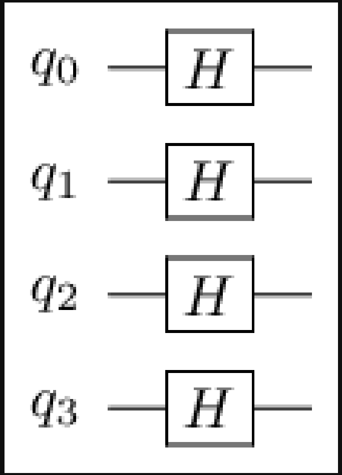
pyAQASM – QRoutine

```
from qat.lang.AQASM import H, CNOT, QRoutine
#Define your routine
my_routine = QRoutine()
my_routine.apply(H, 0)
my_routine.apply(CNOT, 0, 1)

#Use this routine
my_program.apply(my_routine, qubits_reg[0], qubits_reg[1])
my_program.apply(my_routine, qubits_reg[0:2])
my_program.apply(my_routine, qubits_reg)
```



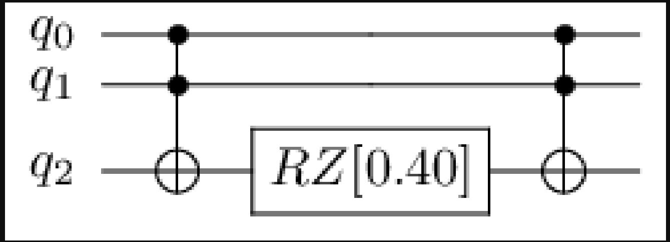
pyAQASM – QRoutine: Walsh Hadamard

```
from qat.lang.AQASM import QRoutine, H
#Define your walsh Hadamard
def WALSH_HADAMARD(n):
    walsh_Hadamard_routine =  QRoutine()
    for i in range(n):
        walsh_Hadamard_routine.apply(H, i)
    return walsh_Hadamard_routine

#Use this routine
my_program.apply(WALSH_HADAMARD(4), qubits_reg)
```



pyAQASM – QRoutine: Ancilla qubits management

```
from qat.lang.AQASM import *
#Create a QRoutine
routine = QRoutine()
#Allocate Qubits using wires
input_wires = routine.new_wires(2)
temp_wire = routine.new_wires(1)
#Apply your gates
routine.apply(CCNOT, input_wires, temp_wire)
routine.apply(RZ(0.4), temp_wire)
routine.apply(CCNOT, input_wires, temp_wire)

%qatdisplay routine
```



pyAQASM – QRoutine: Ancilla qubits management

```
#Defining temporary qubits
routine.set_ancillae(temp_wire)
#Printing arity
print("Now the routine has arity", routine.arity)
#Create a program calling our QRouting
prog = Program()
qbits = prog.qalloc(2)
prog.apply(routine, qbits) # No exceptions!
circ = prog.to_circ(include_locks=True)
print("But the circuit has arity", circ.nbqbits)

%qatdisplay circ
```

pyAQASM – QRoutine: Ancilla qubits management

```
#Defining temporary qubits
routine.set_ancillae(temp_wire)
#Printing arity
print("Now the routine has arity", routine.arity)
#Create a program calling our QRouting
prog = Program()
qbits = prog.qalloc(2)
prog.apply(routine, qbits) # No exceptions!
circ = prog.to_circ(include_locks=True)
print("But the circuit has arity", circ.nbqbits)

%qatdisplay circ
```

Now the routine has arity 2

pyAQASM – QRoutine: Ancilla qubits management

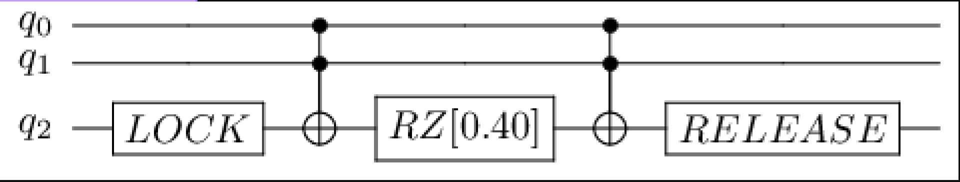
```
#Defining temporary qubits
routine.set_ancillae(temp_wire)
#Printing arity
print("Now the routine has arity", routine.arity)
#Create a program calling our QRouting
prog = Program()
qbits = prog.qalloc(2)
prog.apply(routine, qbits) # No exceptions!
circ = prog.to_circ(include_locks=True)
print("But the circuit has arity", circ.nbqbits)

%qatdisplay circ
```

Now the routine has arity 2
But the circuit has arity 3

pyAQASM – QRoutine: Ancilla qubits management

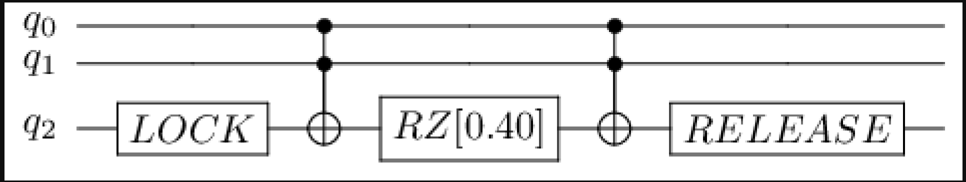
```
#Defining temporary qubits
routine.set_ancillae(temp_wire)
#Printing arity
print("Now the routine has arity", routine.arity)
#Create a program calling our QRouting
prog = Program()
qbits = prog.qalloc(2)
prog.apply(routine, qbits) # No exceptions!
circ = prog.to_circ(include_locks=True)
print("But the circuit has arity", circ.nbqbits)

%qatdisplay circ
```

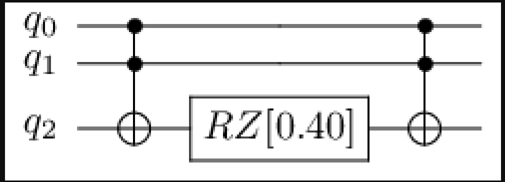
Now the routine has arity 2
But the circuit has arity 3



pyAQASM – QRoutine: Ancilla qubits management

```
#Removing locks
circ = circ.remove_locks()
%qatdisplay circ
```

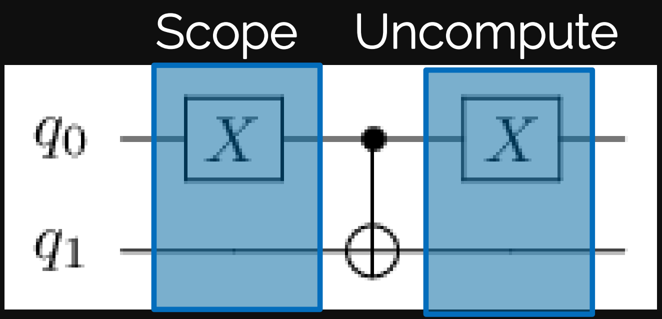




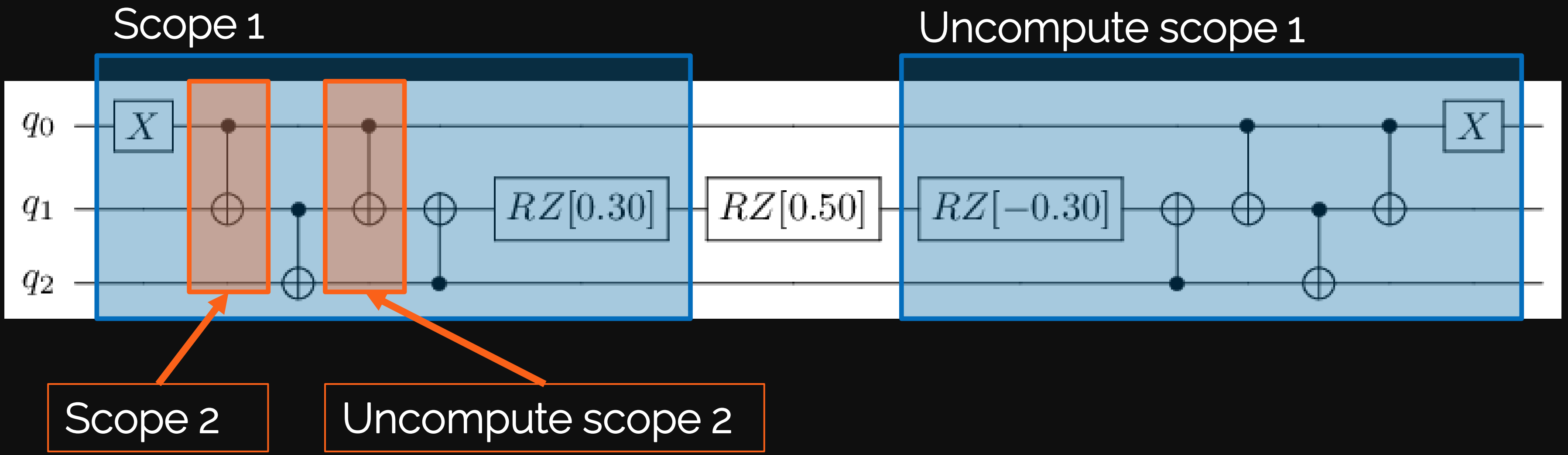
Atos

pyAQASM – QRoutine: Computation scopes

```
from qat.lang.AQASM import *

routine = QRoutine()

#Open a fresh "computation scope"
with routine.compute():
    # This gate will be stored in the scope
    routine.apply(X, 0)

#Out of the scope and apply another gate
routine.apply(CNOT, 0, 1)

#Uncompute the last scope
routine.uncompute()
```



pyAQASM – QRoutine: Nested scopes manipulation



Thank you!

For more information please contact:
gaetan.rubez@atos.net