IBM ProjectQ 프로그래밍 관련 정보

작성자: 한성대학교 장경배

#1 양자 프로그래밍 기본

양자 프로그래밍에서 사용하려는 양자 게이트를 import 한다.

```
from projectq.ops import H, CNOT, Measure, Toffoli, X, All, Swap
```

아래와 같이 하나의 큐빗을 선언할 수 있고, 4개의 큐빗을 배열 형태로 선언할 수도 있다. 여기서 선언한 큐빗의 상태는 모두 0이다.

```
a = eng.allocate_qubit()
b = eng.allocate_qureg(4)
```

선언한 큐빗에 import한 양자 게이트 연산을 수행할 수 있다. 수행 시 매개변수 순서에 주의해야 한다.

```
H | a  # Apply H gate
All(H) | b

X | a
CNOT | (b[0], b[1]) #b[1] = b[1] XOR b[0]
Toffoli | (b[0], b[1], b[2]) #b[2] = b[2] XOR (b[0] AND b[1])
```

양자 게이트에 대한 간단한 설명은 아래와 같다.

- ✓ H 게이트는 큐빗을 중첩 상태로 만든다.
- ✓ All (Gate)는 배열 단위로 양자 게이트를 수행할 수 있다.
- ✓ X 게이트는 큐빗의 값을 반전시킨다. (0 → 1, 1 → 0)
- ✓ CNOT 게이트는 2 큐빗 간의 XOR 연산을 두번째 큐빗(b[1]에 수행한다.
- ✓ Toffoli 게이트는 2 큐빗 간의 AND 연산을 세번째 큐빗(b[2])에 XOR 한다.

#2 양자 프로그래밍 응용

양자 프로그래밍에서는 앞서 수행한 연산을 거꾸로 다시 수행해주는 Reverse연산이 필요할 때가 있다. 이는 Compute 그리고 Uncompute 문을 활용하여 아래와 같이 간단히 수행할 수 있다.

```
with Compute(eng):
    X | a
    CNOT | (b[0], b[1]) #b[1] = b[1] XOR b[0]
    Toffoli | (b[0], b[1], b[2]) #b[2] = b[2] XOR (b[0] AND b[1])

Uncompute(eng)
```

Compute에서 연산들을 수행하면(X, CNOT, Toffoli), Uncompute는 Compute의 연산들을 거꾸로 수행해 준다. (Toffoli, CNOT, X)

큐빗의 if 문 적용은 아래와 같이 Control 문으로 대체할 수 있다.

```
with Control(eng, b[0]):
    Toffoli | (b[1], b[2], b[3])
```

b[0]값이 1인 경우에만 아래 Toffoli 게이트를 수행한다. 이는 CCCNOT 연산을 의미한다. CCNOT은 Toffoli을 의미한다.

Uncompute, Compute, Control는 양자 게이트 사용과 동일하게 import하여 사용한다.

```
from projectq.meta import Compute, Uncompute, Control
```

#3 구현한 양자 프로그램 실행

구현한 양자 프로그램은 MainEngine과 함께 다양한 컴파일러를 사용하여 실행시킬 수 있다.

```
from projectq import MainEngine
from projectq.backends import ResourceCounter, ClassicalSimulator
```

ProjcetQ의 실제 양자 컴퓨터 시뮬레이터를 사용한다면 실행 시 30 큐빗 정도의 제한이 생긴다. 하지만 ClassicalSimulator를 사용한다면 중첩을 제공하지 않는 10000 큐빗 이상의 양자 프로그램을 실행시킬 수 있다. 암호 구현 시에는 큐빗 중첩 상태를 배제해도 되기 때문에 ClassicalSimulator의 출력 값을 통해 제대로 구현하였는지 확인해볼 수 있다. 양자컴퓨팅 결과인 큐빗 값 출력을 위해서는 Measure 연산이 필수이다.

```
# Result value check
sim = ClassicalSimulator()
eng = MainEngine(sim)
print(test(eng))
```

ResourceCounter는 구현한 양자 프로그램에 사용된 양자 자원들을 분석한다.

```
# Quantum resource check
Resource = ResourceCounter()
eng = MainEngine(Resource)
test(eng)
print(Resource)
```

```
/Users/kb/PycharmProjects/pro
(0, 0, 0, 0)
Gate class counts:
    AllocateQubitGate : 4
    CCCXGate : 1
    DeallocateQubitGate : 4
    MeasureGate : 4

Gate counts:
    Allocate : 4
    CCCX : 1
    Deallocate : 4
    Measure : 4

Depth : 2.
```

결과값(ClassicalSimulator)

양자자원(ResourceCounter)

* 첨부 코드

```
from projectq import MainEngine
from projectq.ops import H, CNOT, Measure, Toffoli, X, All, Swap
from projectq.backends import ResourceCounter, ClassicalSimulator
from projectq.meta import Compute, Uncompute, Control

def test(eng):

    a = eng.allocate_qubit()
    b = eng.allocate_qureg(4)

    #H | a   # Apply H gate
    #All(H) | b

    with Compute(eng):
        X | a
        CNOT | (b[0], b[1]) #b[1] = b[1] XOR b[0]
        Toffoli | (b[0], b[1], b[2]) #b[2] = b[2] XOR (b[0] AND b[1])

    Uncompute(eng)

    with Control(eng, b[0]):
        Toffoli | (b[1], b[2], b[3])



    All(Measure) | b

    return int(b[0]), int(b[1]), int(b[2]), int(b[3])

# Result value check
sim = ClassicalSimulator()
eng = MainEngine(sim)
print(test(eng))

# Quantum resource check
Resource = ResourceCounter()
eng = MainEngine(Resource)
test(eng)
print(Resource)

eng.flush()
```