Quantum Computing for 3-SAT Problem

Ning Ma

Problem Description

- Boolean satisfiability problem (SAT)
  - If there exists an interpretation that satisfies a given Boolean formula
  - It asks whether the variables of a given Boolean formula can be consistently replaced by the values TRUE or FALSE
  - If the formula evaluates to TRUE, the formula is called *satisfiable*; otherwise, it is *unsatisfiable*
- Example
  - $(x1 \lor x2) \land (\neg x1 \lor x3) \land (\neg x2 \lor x3)$
  - Solutions are x1 = F, x2 = T, x3 = T; x1 = T, x2 = F, x3 = T; x1 = T, x2 = T, x3 = T.

Problem Description

- 3-SAT problem
  - Each clause has at most three literals
  - Any k-SAT problem can be transformed to a 3-SAT one
  - There is no loss of generality in focusing on the Boolean 3-SAT problem.

Problem Description

Brute Force

- Time Complexity is $O(2^N)$, where N is the amount of literals

Problem Description

Grover algorithm

- A significant quantum algorithm for unstructured search
- Initialization
  - To begin by preparing a superposition of all possible states in the search space using quantum bits (qubits) – Hadamard Gate
- Oracle Query
  - To use a quantum oracle, a black-box operation that can identify the correct solution(s)
  - When the oracle is queried with the superposition of all possible states, it flips the sign of the amplitude (the quantum state's probability amplitude) of the solution state(s), distinguishing them from the non-solution states.

Problem Description

Grover algorithm (Cont'd)

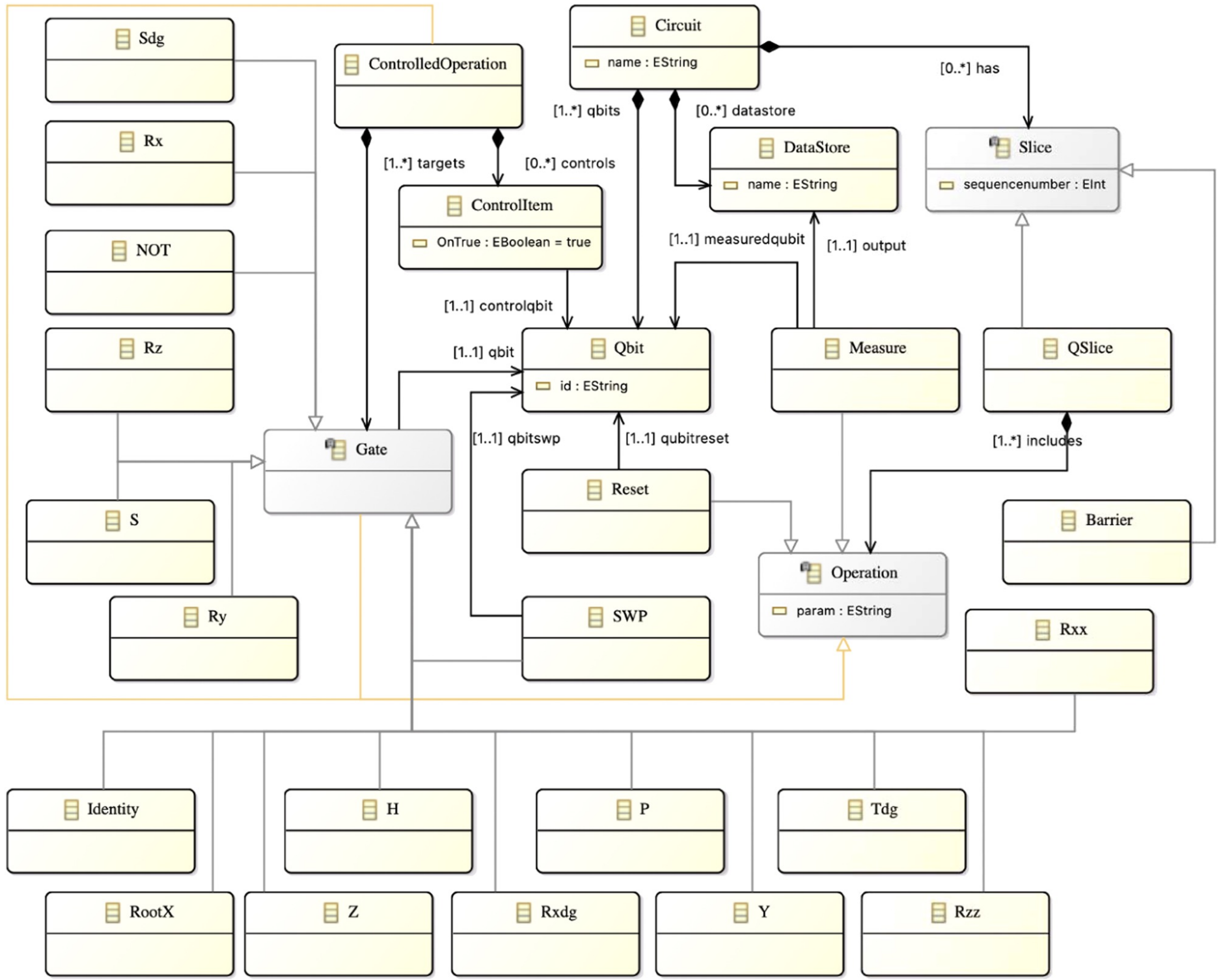
- Amplitude Amplification
  - To amplify the probability amplitude of the solution state(s) while de-amplifying the non-solution states
  - a) Inverting the amplitudes of all states.
  - b) Applying a second Hadamard transform, followed by a conditional phase shift (leaving the initial state unchanged) and another Hadamard transform.
- Iteration
  - The steps of querying the oracle and applying the diffusion transform are repeated multiple times – roughly $\sqrt{N}$, where N is the size of the search space

Problem Description

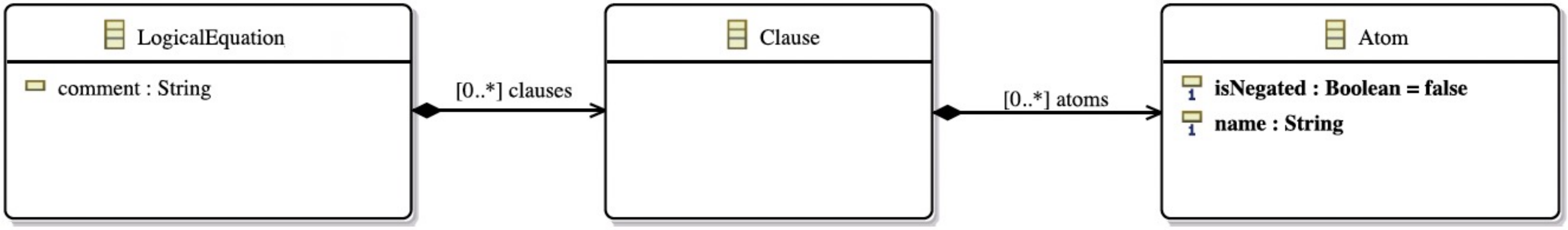
Grover algorithm (Cont'd)

- Measurement
  - After completing the iterations, the quantum state is measured
  - The probability of measuring a state corresponding to a solution is significantly higher than that of any individual non-solution state.
- Complexity and Efficiency
  - Time Complexity is $O\sqrt{N}$

Metamodel for modelling quantum circuits



Metamodel for modelling SAT problems



Epsilon Transformation Language

```
operation hadamard_input_qbits() : output!QSlice {
        var tmp : output!QSlice = new output!QSlice;
        tmp.sequencenumber = 0;
        var qbits = gen_qubits ();
        for (i  in qbits) {
                var H : output!H = new output!H;
                H.qbit = i;
                tmp.includes.add(H);
        }
        return tmp;
}
```

Epsilon Transformation Language

```
@cached
operation String gen_unique_qbit () : output!Qbit {
    var tmp : output!Qbit = new output!Qbit;
    tmp.id = self;
    return tmp;
}

@cached
operation gen_qubits () : Sequence {
        var tmp : Set;
        input!Atom.allInstances.forEach(i | tmp.add( i.name.gen_unique_qbit() ));
        var tmp2 = tmp.sortBy (i | i.id.asInteger());
        return tmp2;
}
```

Epsilon Transformation Language

```
$pre not self.isNegated
operation input!Atom gen_not() : output!NOT {
        var tmp : output!NOT = new output!NOT;
    tmp.qbit = self.name.gen_unique_qbit();
    return tmp;
}
```

```
@cached
operation input!Clause not_all_atoms_negated() : Boolean {
    return self.atoms.exists(i | not i.isNegated);
}
```

Epsilon Transformation Language

```
operation phase_flip () : output!QSlice {
    var tmp : output!QSlice = new output!QSlice;
    tmp.sequencenumber = seq_slice;
        seq_slice++;
        var or_op = new output!ControlledOperation;
        var z = new output!Z;
        z.qbit = scratch_qbits.last().controlqbit;
        or_op.targets.add(z);
        scratch_qbits.select(i | i != scratch_qbits.last()).forEach(i | or_op.controls.add(i));
        tmp.includes.add(or_op);
    return tmp;
}
```

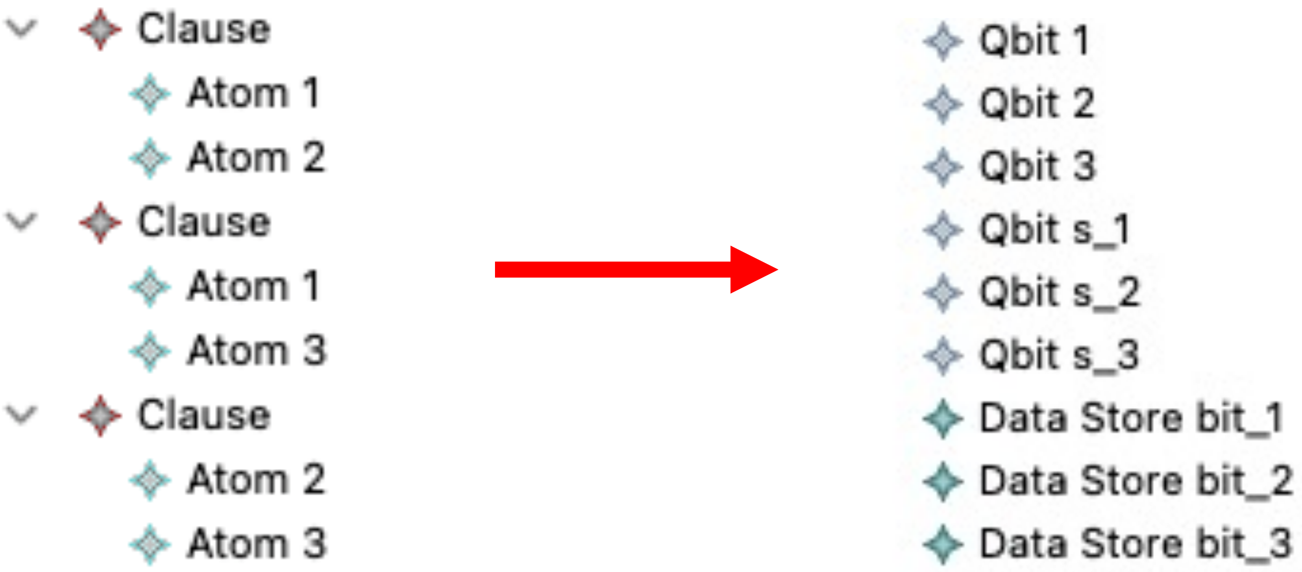
Epsilon Transformation Language

```
rule Root
    transform i : input!LogicalEquation
    to o : output!Circuit
       , s : output!QSlice
{
        o.name =  i.comment;
        for (i in seq_qbits) {
                o.qbits.add(i);
                var ds = new output!DataStore;
                ds.name = "bit_"+i.id;
                o.datastore.add(ds);
                var m = new output!Measure;
                m.output = ds;
                m.measuredqubit=i;
                s.includes.add(m);
        }
        measure_qslice = s;
        //seq_qbits.forEach(i | o.qbits.add(i) );
}
```

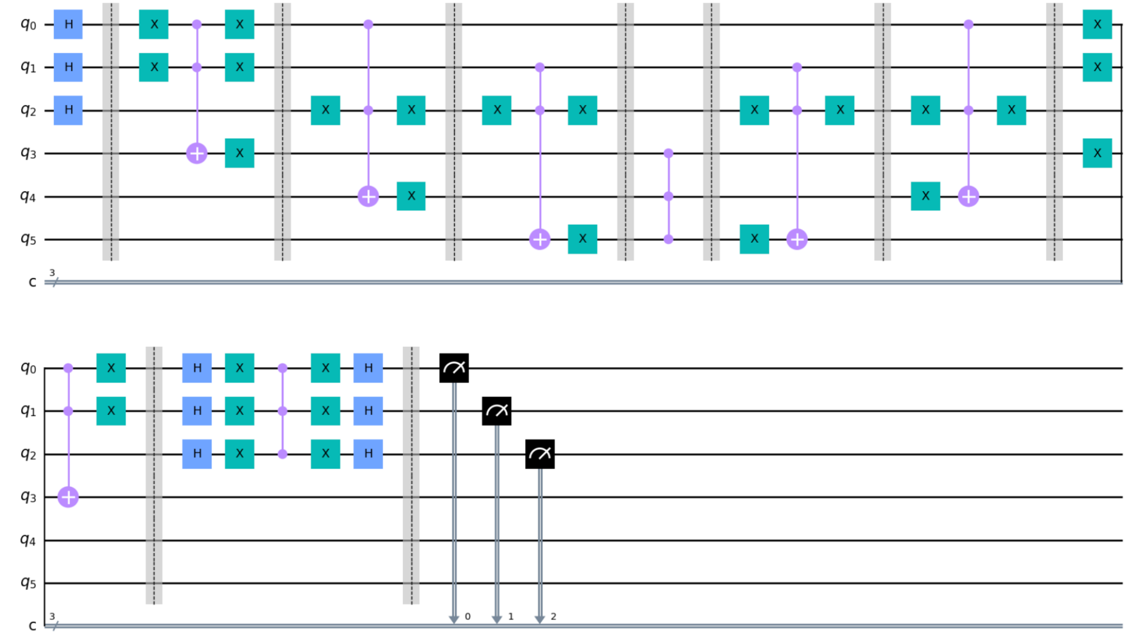
Epsilon Transformation Language

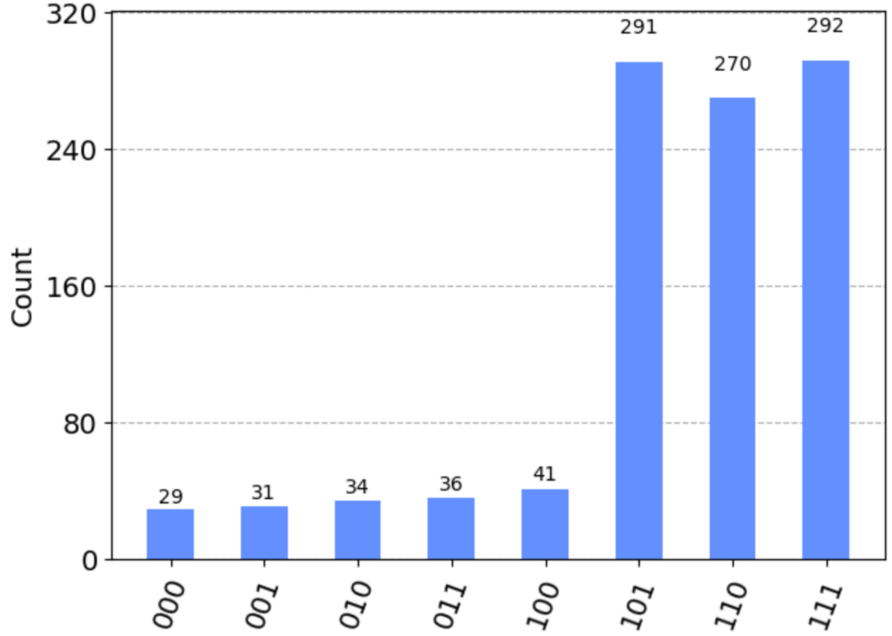
```
rule Clause
        transform i : input!Clause
        to q : output!Qbit
           , s_or : output!QSlice
           , or_op : output!ControlledOperation
           , c_n : output!NOT
           , s_undo : output!QSlice
           , n : output!NOT
           , c_item : output!ControlItem
           , barrier : output!Barrier
{

        q.id = "s_"+scratch_int;
        scratch_int++;
        the_circuit().qbits.add(q);
        c_item.controlqbit = q;
        scratch_qbits.add(c_item);

        // do
        if (i.not_all_atoms_negated()) {
                var s_do : output!QSlice = new output!QSlice;
                s_do.sequencenumber = seq_slice;
                seq_slice++;
                i.atoms.select(i | not i.isNegated).
                  forEach(i | s_do.includes.add(i.gen_not()) );
                sat_circuit.add(s_do);
        }

        // control-not
        s_or.sequencenumber = seq_slice;
        seq_slice++;
        s_or.includes.add(or_op);
        or_op.targets.add(c_n);
        c_n.qbit=q;
        i.atoms.forEach(i | or_op.controls.add(i.gen_controlIt()) );
        sat_circuit.add(s_or);

        // undo
        s_undo.sequencenumber = seq_slice;
        seq_slice++;
        i.atoms.select(i | not i.isNegated).
                    forEach(i | s_undo.includes.add(i.gen_not()) );
        n.qbit=q;
        s_undo.includes.add(n);
        sat_circuit.add(s_undo);
        sat_circuit.add(barrier);
}
```

XMI


Clause
Atom 1
Atom 2
Clause
Atom 1
Atom 3
Clause
Atom 2
Atom 3
Qbit 1
Qbit 2
Qbit 3
Qbit s_1
Qbit s_2
Qbit s_3
Data Store bit_1
Data Store bit_2
Data Store bit_3

Demo





- The corresponding solutions are: 1) x3 = T, x2 = F, x1 = T; 2) x3 = T, x2 = T, x1 = F; 3) x3 = T, x2 = T, x1 = T.
- Link: https://github.com/CGZJDX/LOG8505E/blob/main/SAT_Solution.ipynb

Thank You

Ning Ma