SymboliQ: A python framework for Symbolic Quantum computation

Victory Omole[1] and Bob Sutor[2]

- [1] Super.tech, a division of Infleqtion
- [2] ColdQuanta, a division of Infleqtion

A 2022 QIP talk titled “Software of QIP, by QIP, and for QIP” by Dave Bacon asked “What would a symbolic library for quantum computing look like?”. Even though it’s natural for most scientists to perform calculations with symbols, popular quantum programming languages like Cirq and Qiskit don’t emphasize symbolic derivations of expressions such as manipulating quantum state amplitudes symbolically instead of assuming finite precision complex numbers. SymPy: A Python framework for Symbolic Computing contains the capabilities for symbolically manipulating quantum mechanical equations but it doesn't provide an easy way to perform symbolic manipulations in Dirac notation. SymboliQ builds off of SymPy to support quantum computing calculations in Dirac Notation.

SymPy

- Performs operations on a statevector via matrix multiplication
  - Operations are performed on states via a method called *_apply_operator_Qubit*
  - An inspection of *_apply_operator_Qubit* shows that operations are represented as matrices before they’re multiplied by the input state

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \times 1 + 1 \times 0 \\ 1 \times 1 + 0 \times 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

- Doesn’t provide a method for retrieving the steps it went through to simplify quantum mechanical expressions

```
state = Qubit(0)
```

```
XGate(0) * state
```

$X_0|0\rangle$

```
qapply(XGate(0) * state)
```

$|1\rangle$

```
XGate(0)._apply_operator_Qubit(state)
```

$|1\rangle$

```
def apply_operator_Qubit(operator, qubits):
    targets = operator.targets

    # returns Matrix([[0, 1], [1, 0]]) for an X gate
    target_matrix = operator.get_target_matrix()

    # Find which column of the target matrix this applies to.
    column_index = 0
    n = 1
    for target in targets:
        column_index += n * qubits[target]
        n = n << 1
        column = target_matrix[:, int(column_index)]

    # Now apply each column element to the qubit.
    result = 0
    for index in range(column.rows):
        new_qubit = qubits.__class__(*qubits.args)
        # Flip the bits that need to be flipped.
        for bit, target in enumerate(targets):
            if new_qubit[target] != (index >> bit) & 1:
                new_qubit = new_qubit.flip(target)
        # The value in that row and column times the
        # flipped-bit qubit is the result for that part.
        result += column[index] * new_qubit
    return result
```

```
apply_operator_Qubit(XGate(0), state)
```

$|1\rangle$

SymboliQ with SymPy

- Based on 2 rewrite rules that were specified in [1] called BaseReduce and MulReduce
- Performs operations symbolically by using MulReduce to break expressions down into inner products via associative and distributive laws. The resulting inner products can then be cancelled with BaseReduce
- BaseReduce and MulReduce have been implemented in Coq: https://github.com/Vickyswj/DiracRepr, but Python is a more popular and accessible language

```
symboliq.qapply(XGate(0) * state)
```

$|1\rangle$

```
print(symboliq.get_simp_steps(XGate(0) * state))
```

```
(0) X(0)*|0>
(1) |0><1|*|0> + |1><0|*|0>
(2) |1>
```

$$BaseReduce : \langle 0|0\rangle = \langle 1|1\rangle = 1 \text{ and } \langle 0|1\rangle = \langle 1|0\rangle = 0$$

$$MulReduce : x * (y + z) = (x * y) + (x * z) \text{ and } (x * y) * z = z * (y * z)$$

For example, when an X gate is applied to a state $|0\rangle$, the X gate is translated to how it's represented in Dirac notation; which is $|0\rangle\langle 1| + |1\rangle\langle 0|$. MulReduce is used to distribute the state $|0\rangle$ to $|0\rangle\langle 1| + |1\rangle\langle 0|$ to get $|0\rangle\langle 1|0\rangle + |1\rangle\langle 0|0\rangle$. By applying BaseReduce to this expression we get $\langle 0|0\rangle\langle 1| + |1\rangle\langle 0|0\rangle$ which simplifies to $0|0\rangle + 1|1\rangle$, which is equal to $|1\rangle$

https://github.com/vtomole/SymboliQ

SymboliQ is an extension of Sympy's Quantum Mechanics subpackage. It's the only Python package we know of that allows for simulation of quantum circuits via Dirac Notation. See the introductory notebook for an in-depth explanation.

Installation

The SymboliQ package is available via `pip` and can be installed in your current Python environment with the command:

```
pip install symboliq
```

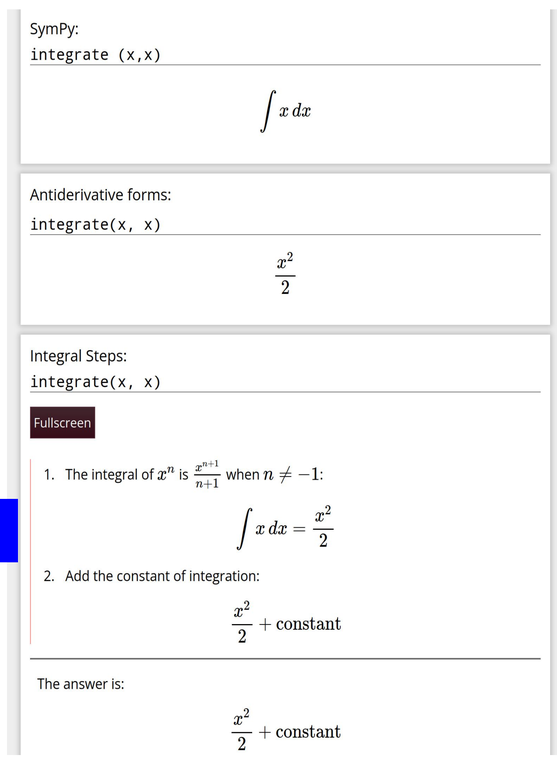
Getting started

```
from sympy.physics.quantum import TensorProduct
from symboliq.dirac_notation import DiracNotation, cx, h, i, x, ket_0,  ket_1

print(DiracNotation(ket_0))
# prints
#|0>

print(DiracNotation(x * ket_1))
# prints
# (|0><1| + |1><0|)*|0>

bell_state = DiracNotation(
        cx * TensorProduct(h, i) * TensorProduct(ket_0, ket_0)
    )
print(bell_state.operate_reduce())
# prints
# sqrt(2)*|0>x|0>/2 + sqrt(2)*|1>x|1>/2
```

Future Work

- Solicit user feedback to improve the package
- Offer step by step descriptions, similar to SymPy Gamma
- Handle more complicated expressions
- Parameterized circuits
- Jupyter notebooks for quantum education
- Implement the Density matrix formalism

https://gamma.sympy.org/



Acknowledgments

Victory thanks Pranav Gokhale for fostering an environment where this work was possible

Conclusions

- Industrial strength software has accelerated the progress of QIP by orders of magnitude
- Most QIP researchers work in Dirac notation
- Most QIP Software is in Python
- There is no library that allows researchers to work with Dirac notation in Python
- SymboliQ builds off the popular symbolic computing platform SymPy to allow for the manipulation of symbols in Dirac notation

References

1. Shi, WJ., Cao, QX., Deng, YX. *et al.* Symbolic Reasoning About Quantum Circuits in Coq. *J. Comput. Sci. Technol.* **36**, 1291–1306 (2021). https://doi.org/10.1007/s11390-021-1637-9
2. Meurer A, Smith CP, Paprocki M, Čertík O, Kirpichev SB, Rocklin M, Kumar A, Ivanov S, Moore JK, Singh S, Rathnayake T, Vig S, Granger BE, Muller RP, Bonazzi F, Gupta H, Vats S, Johansson F, Pedregosa F, Curry MJ, Terrel AR, Roučka Š, Saboo A, Fernando I, Kulal S, Cimrman R, Scopatz A. (2017) SymPy: symbolic computing in Python. *PeerJ Computer Science* 3:e103 https://doi.org/10.7717/peerj-cs.103
3. Granger , Brian E. “Sympy/quantum_notebooks: Jupyter Notebooks That Demonstrate SymPy's Symbolic Quantum Mechanics Package.” *GitHub*, 8 Oct. 2017, https://github.com/sympy/quantum_notebooks.