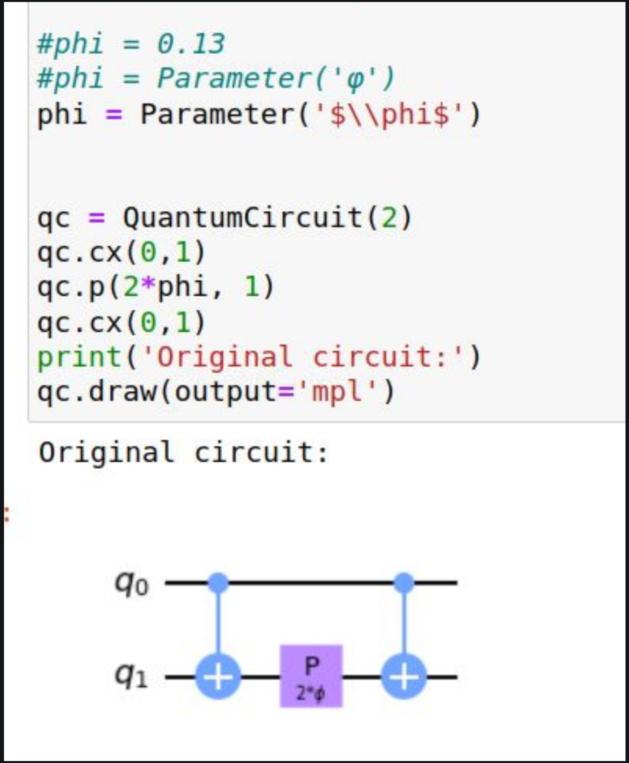
Qiskit compatibility with Parameters

Rafał Pracht, Jesús Sistos

Mentored by Nick Bronn

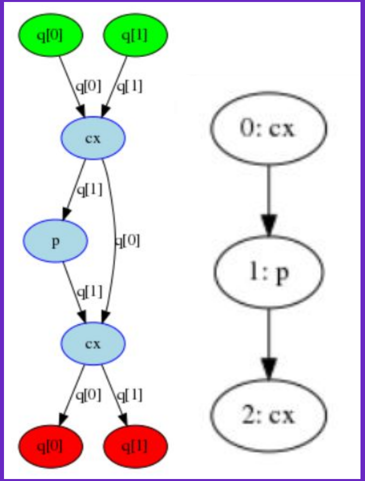
1. Passing unbound parameters through Template Optimization Pass



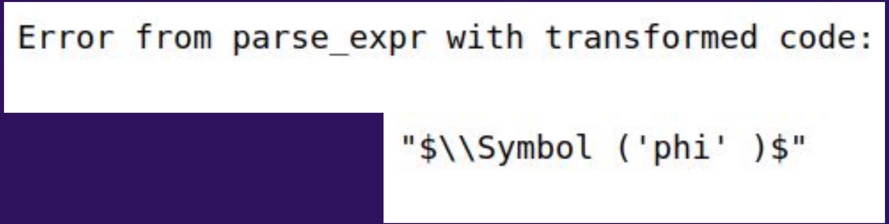
```
#phi = 0.13
#phi = Parameter('φ')
phi = Parameter('$\\phi$')


qc = QuantumCircuit(2)
qc.cx(0,1)
qc.p(2*phi, 1)
qc.cx(0,1)
print('Original circuit:')
qc.draw(output='mpl')
```

```
Original circuit:
```

Template matching makes use of the circuit's DAG (Directed Acyclic Graph)



Sympy fails when parsing LaTeX names

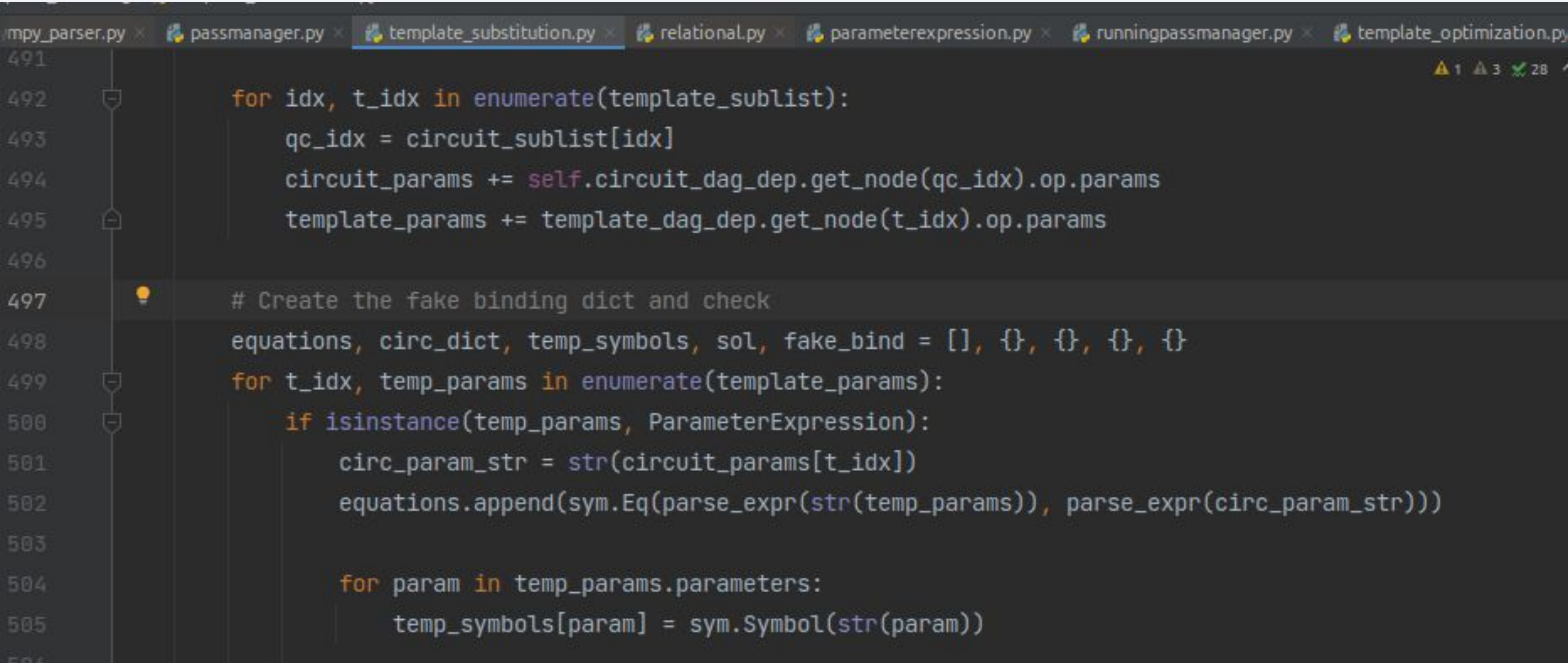


```
Error from parse_expr with transformed code:

                    "$\\Symbol ('phi' )$"
```

Qiskit

Finding the bug

Parsing occurs in
qiskit/transpiler/passes/optimization/template_matching/template_substitution.py



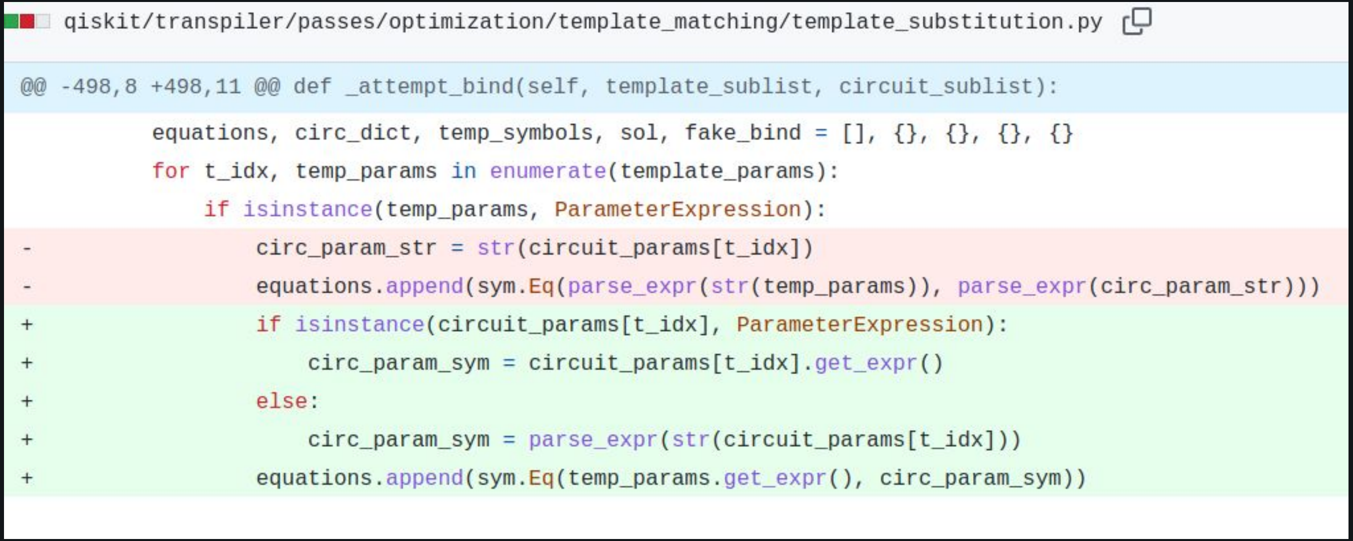
```
mpy_parser.py ×   passmanager.py ×   template_substitution.py ×   relational.py ×   parameterexpression.py ×   runningpassmanager.py ×   template_optimization.py
```

```
        for idx, t_idx in enumerate(template_sublist):
            qc_idx = circuit_sublist[idx]
            circuit_params += self.circuit_dag_dep.get_node(qc_idx).op.params
            template_params += template_dag_dep.get_node(t_idx).op.params

        # Create the fake binding dict and check
        equations, circ_dict, temp_symbols, sol, fake_bind = [], {}, {}, {}, {}
        for t_idx, temp_params in enumerate(template_params):
            if isinstance(temp_params, ParameterExpression):
                circ_param_str = str(circuit_params[t_idx])
                equations.append(sym.Eq(parse_expr(str(temp_params)), parse_expr(circ_param_str)))

                for param in temp_params.parameters:
                    temp_symbols[param] = sym.Symbol(str(param))
```

Making the fix

Current code uses a custom parser.

We can use Sympy's tools to do it in a compatible way.



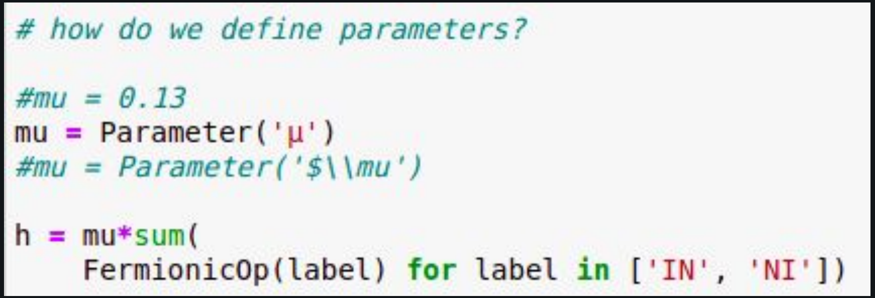
```
qiskit/transpiler/passes/optimization/template_matching/template_substitution.py

@@ -498,8 +498,11 @@ def _attempt_bind(self, template_sublist, circuit_sublist):
          equations, circ_dict, temp_symbols, sol, fake_bind = [], {}, {}, {}, {}
          for t_idx, temp_params in enumerate(template_params):
              if isinstance(temp_params, ParameterExpression):
-                 circ_param_str = str(circuit_params[t_idx])
-                 equations.append(sym.Eq(parse_expr(str(temp_params)), parse_expr(circ_param_str)))
+                 if isinstance(circuit_params[t_idx], ParameterExpression):
+                     circ_param_sym = circuit_params[t_idx].get_expr()
+                 else:
+                     circ_param_sym = parse_expr(str(circuit_params[t_idx]))
+                 equations.append(sym.Eq(temp_params.get_expr(), circ_param_sym))
```

These changes have already been added to the main pull request in Terra's repository.

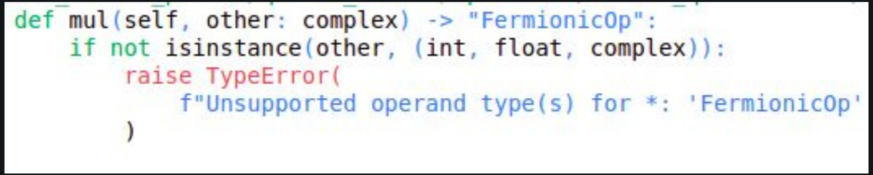
2. Parameters in Qiskit Nature: Second Quantization Operators

Operator classes directly block the usage of Parameters

$$H = \mu \sum_{i=1}^{2} a_i^{\dagger} a_i$$

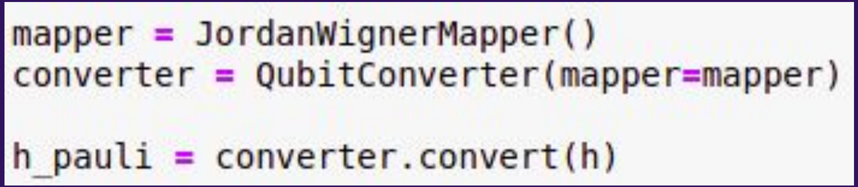


```
# how do we define parameters?

#mu = 0.13
mu = Parameter('μ')
#mu = Parameter('$\\mu')

h = mu*sum(
    FermionicOp(label) for label in ['IN', 'NI'])
```



```
def mul(self, other: complex) -> "FermionicOp":
    if not isinstance(other, (int, float, complex)):
        raise TypeError(
            f"Unsupported operand type(s) for *: 'FermionicOp'
        )
```

What if we ‘bypass’ this restriction?

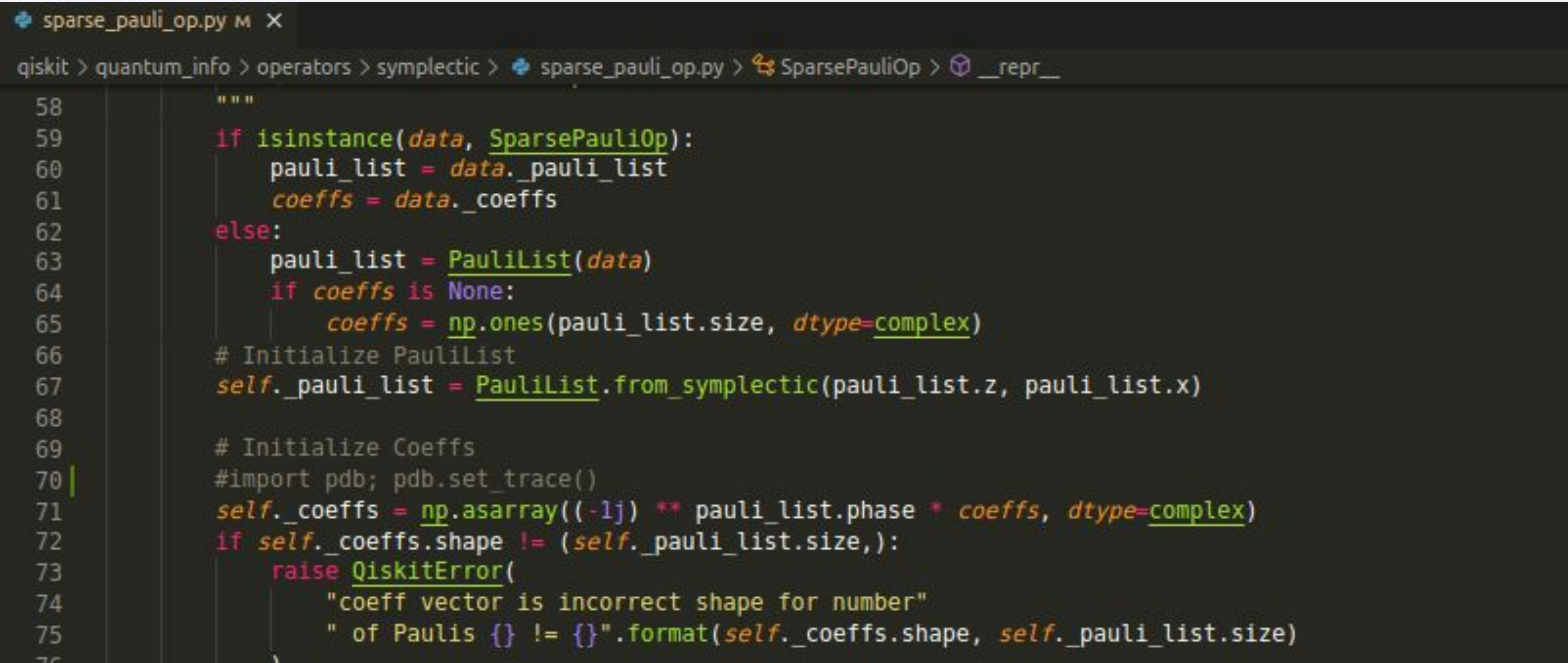


```
mapper = JordanWignerMapper()
converter = QubitConverter(mapper=mapper)

h_pauli = converter.convert(h)
```

Qiskit

State of the problem

Usage of Numpy limits versatility of parameters


sparse_pauli_op.py M ×
qiskit > quantum_info > operators > symplectic > sparse_pauli_op.py > SparsePauliOp > __repr__

```
        """
        if isinstance(data, SparsePauliOp):
            pauli_list = data._pauli_list
            coeffs = data._coeffs
        else:
            pauli_list = PauliList(data)
            if coeffs is None:
                coeffs = np.ones(pauli_list.size, dtype=complex)
        # Initialize PauliList
        self._pauli_list = PauliList.from_symplectic(pauli_list.z, pauli_list.x)

        # Initialize Coeffs
        #import pdb; pdb.set_trace()
        self._coeffs = np.asarray((-1j) ** pauli_list.phase * coeffs, dtype=complex)
        if self._coeffs.shape != (self._pauli_list.size,):
            raise QiskitError(
                "coeff vector is incorrect shape for number"
                " of Paulis {} != {}".format(self._coeffs.shape, self._pauli_list.size)
            )
```

Future challenges

Template Optimization and inverses of parametrized gates.

Qiskit Opflow parameter management.

Hardware transpiling of custom VQE Ansatz circuits.