

FockMap: A Composable Functional Framework for Symbolic Fock-Space Operator Algebra and Fermion-to-Qubit Encodings in F#

John Azariah^{*1}

¹University of Technology Sydney

February 2026

Summary

Simulating fermionic systems on quantum hardware requires a mapping from fermionic ladder operators to qubit Pauli operators. This fermion-to-qubit encoding step strongly influences Pauli weight, measurement cost, and circuit depth. Although widely used encodings (Jordan–Wigner, Bravyi–Kitaev, Parity, and tree-based variants) share substantial algebraic structure, they are often implemented in existing software as isolated transformations rather than as instances of a common formal interface.

FockMap is an open-source F# library that formalizes this shared structure through two composable abstractions: *index-set schemes*, defined by three set-valued functions (**Update**, **Parity**, **Occupation**), and *path-based tree encodings*, in which any rooted labelled tree induces a valid encoding. The index-set abstraction expresses Jordan–Wigner, Bravyi–Kitaev, and Parity in 3–5 lines each, while the path-based abstraction supports arbitrary tree topologies, including balanced binary and balanced ternary trees with optimal $O(\log n)$ asymptotic Pauli weight.

The operator-processing pipeline is implemented symbolically: Pauli strings are multiplied exactly with algebraic phase tracking, without constructing operator matrices and without introducing floating-point error in intermediate steps. Numerical coefficients are introduced only at Hamiltonian assembly. This design supports transparent correctness checks and avoids numerical artefacts common in matrix-first workflows.

In addition to the fermionic canonical anti-commutation workflow, **FockMap** now includes a bosonic canonical commutation algebra for symbolic normal ordering of ladder-operator expressions. This extends the same typed expression pipeline to bosonic sectors and supports mixed-statistics model assembly in a single representation.

The library is implemented with algebraic data types and pure functions, includes a persistent Fenwick tree ADT, and is validated by an extensive xUnit + FsCheck test suite (427 passing tests), including both property-based algebraic checks and targeted edge-case regressions. Current coverage is 96.51% line and 86.94% branch on the core library. A complete H₂/STO-3G example is provided and reproduced across all five built-in encodings.

A companion walkthrough (*Library Cookbook*) is available both as hosted documentation on the repository website and as a standalone typeset document [Azariah \[2026a\]](#), providing 13 progressive chapters that cover every public type, function, and workflow in the library.

Statement of Need

Quantum simulation of molecular electronic structure is widely regarded as one of the most promising near-term applications of quantum computing [[Feynman, 1982](#), [Aspuru-Guzik et al., 2005](#)]. Between the molecular Hamiltonian in second quantization and the measurements performed on quantum hardware lies a critical middleware step: the fermion-to-qubit encoding. The choice of encoding determines

^{*}ORCID: [0009-0007-9870-1970](#)

the Pauli weight of each operator (and hence circuit depth), the number of measurement terms, and ultimately whether a simulation is feasible on a given device.

Current tools for this step, OpenFermion [McClean et al., 2020], Qiskit Nature [Qiskit contributors, 2023], and PennyLane [Bergholm et al., 2022], implement each encoding as a monolithic function mapping `FermionOperator` \rightarrow `QubitOperator`. Adding a new encoding requires writing hundreds of lines of bespoke code. The mathematical structure shared across encodings (Majorana decomposition, parity tracking, update sets) is duplicated rather than abstracted. There is no mechanism for users to define, compose, or compare custom encodings programmatically.

`FockMap` addresses this gap by representing an encoding as a *value* (a record of three functions) rather than as an opaque class hierarchy. This design enables:

- **Exploration:** researchers can define and test novel encodings in 3–5 lines of code.
- **Comparison:** all encodings share the same verification pipeline (anti-commutation tests, eigen-spectrum comparison).
- **Pedagogy:** the implementation remains close to the formal definitions in the literature; an `EncodingScheme` directly encodes the mathematical specification.

Many contemporary models also require bosonic modes (for example, vibrational and photonic degrees of freedom), where canonical commutation relations govern symbolic rewriting. A practical software stack therefore needs both fermionic and bosonic ladder-operator normal-ordering support before any downstream qubit mapping or truncation strategy is applied.

The library serves quantum computing researchers exploring encoding-aware circuit synthesis, students learning the algebraic structure of encodings, and developers building simulation pipelines who need a correct and composable symbolic operator layer.

Functionality

Encoding Schemes (Index-Set Framework)

The `EncodingScheme` record type captures the three index-set functions that define a fermion-to-qubit encoding. Three concrete schemes are provided: `jordanWignerScheme`, `bravyiKitaevScheme`, and `parityScheme`. User-defined schemes are first-class values of the same type:

```
let myScheme : EncodingScheme =
  { Update = fun j n -> set [ j + 1 .. n - 1 ]
    Parity = fun j -> if j > 0 then Set.singleton (j - 1)
                      else Set.empty
    Occupation = fun j -> if j > 0 then set [j-1; j]
                        else Set.singleton j }
```

The framework automatically constructs Majorana operators c_j and d_j from these three functions, then derives ladder operators a_j^\dagger and a_j by linear combination.

Tree Encodings (Path-Based Framework)

Any rooted labelled tree defines a fermion-to-qubit encoding. The library provides `balancedBinaryTree` and `balancedTernaryTree` constructors; users can build arbitrary trees from `TreeNode` values. The path-based encoding function traverses the tree to construct Majorana operators without requiring the index-set monotonicity constraint, making it strictly more general than the index-set framework.

Symbolic Algebra Engine

A distinguishing feature of `FockMap` is that operator multiplication is entirely symbolic. The `PauliRegister` type represents a Pauli string (e.g. $XZIY$) together with an exact `Phase` drawn from $\{+1, -1, +i, -i\}$. Multiplying two Pauli registers applies the single-qubit multiplication table ($X \cdot Y = iZ$, etc.) position-wise and accumulates the phase algebraically; no $2^n \times 2^n$ matrices are ever constructed. The resulting `PauliRegisterSequence` (a weighted sum of Pauli strings) is the symbolic representation of an encoded operator.

This design means that encoding a ladder operator on 100 modes produces a compact list of Pauli strings, not a $2^{100} \times 2^{100}$ sparse matrix. Correctness can be verified symbolically: the anti-commutation tests in the verification suite check $\{a_i, a_j^\dagger\} = \delta_{ij}$ by Pauli string cancellation, not by matrix eigenvalue comparison.

The library also includes a persistent `FenwickTree<'a>` (parameterised over any monoid) and a `Hamiltonian` module for constructing molecular Hamiltonians from one-body and two-body integrals.

Normal ordering is parameterized by a combining-algebra interface. `FermionicAlgebra` implements canonical anti-commutation relations, while `BosonicAlgebra` implements canonical commutation relations. This allows both statistics to share the same typed expression representation, while preserving distinct rewrite rules ($a_i a_i^\dagger = 1 - a_i^\dagger a_i$ versus $b_i b_i^\dagger = 1 + b_i^\dagger b_i$).

Verification Suite

The verification strategy combines unit tests, property-based tests, and cross-encoding consistency checks. At submission time, 427 tests pass for the software package, with 96.51% line coverage and 86.94% branch coverage on the core `Encodings` library.

Tests cover five categories:

- **Anti-commutation:** $\{a_i, a_j^\dagger\} = \delta_{ij}$ verified symbolically for all mode pairs.
- **Commutation:** $[b_i, b_j^\dagger] = \delta_{ij}$ verified symbolically for bosonic rewrite cases, including same-index identity generation and different-index swap behavior.
- **Number conservation:** $a_j^\dagger a_j$ produces diagonal Pauli operators.
- **Cross-encoding agreement:** all five encodings produce isospectral Hamiltonians for H_2 (eigenvalue agreement to 5×10^{-16}).
- **Parser and ordering robustness:** malformed input handling, normal/index-order canonicalization, and swap-tracking edge branches for deterministic symbolic normalization.

These checks provide evidence that symbolic rewriting is robust in both fermionic and bosonic workflows, including malformed parser inputs, non-canonical operator orderings, and same-index rewrite edge cases.

Design Principles

Encodings as data. An `EncodingScheme` is a value, not a class hierarchy. Jordan–Wigner, Bravyi–Kitaev, and Parity are different values of the same type. This enables algebraic reasoning: one can ask whether two schemes agree on a given mode without running a full encoding.

Two complementary frameworks. The index-set framework (`MajoranaEncoding.fs`) is fast and algebraically transparent but requires a monotonicity condition on ancestor indices. The path-based framework (`TreeEncoding.fs`) works for *any* tree topology. Both produce the same output type (`PauliRegisterSequence`), so downstream code is encoding-agnostic.

Symbolic over numerical. Existing libraries represent operators as sparse matrices or coefficient dictionaries indexed by opaque integer keys. `FockMap` represents them as typed Pauli strings with exact algebraic phases. This makes the intermediate representation human-readable, composable, and free of floating-point error accumulation. Numerical coefficients enter only at the Hamiltonian assembly stage, where they multiply symbolic Pauli terms.

Pure functions, no mutation. All data structures are immutable: persistent Fenwick trees, recursive tree ADTs, and Pauli register sequences. The library has zero mutation and no side effects in its core modules.

Discovered constraints. Implementation and testing revealed that the index-set framework’s monotonicity requirement (ancestor indices must exceed descendant indices) is satisfied only by star-shaped trees, a structural constraint not previously documented in the literature. This discovery motivated the path-based framework as a universal alternative and is explored further in a companion paper.

Documentation

The repository includes four tiers of documentation:

1. **Library Cookbook** — a 13-chapter progressive tutorial covering every public type and function, from Pauli operators through custom encodings to mixed bosonic–fermionic Hamiltonians. Available as hosted Markdown on the documentation site and as a companion preprint [Azariah \[2026a\]](#).
2. **From Molecules to Qubits** — a pedagogical walkthrough of the complete H_2 pipeline, available as both web documentation and a separate preprint [Azariah \[2026b\]](#).
3. **Theory pages** — seven background articles on second quantization, Pauli algebra, and encoding theory.
4. **Interactive labs** — six runnable F# scripts with guided exercises.

Comparison with Related Software

Feature	OpenFermion	Qiskit Nature	PennyLane	FockMap
JW / BK / Parity	✓/✓/✓	✓/✓/✓	✓/✓/—	✓/✓/✓
Tree encodings	Steiner ext.	—	—	Binary, Ternary
User-defined encodings	—	—	—	✓
User-defined trees	—	—	—	✓
Generic encoding abstraction	—	—	—	✓
Symbolic Pauli algebra	—	—	—	✓
Typed / functional	—	—	—	✓
Persistent Fenwick tree	—	—	—	✓

Table 1: Feature comparison with existing fermion-to-qubit libraries. FockMap additionally provides symbolic bosonic normal ordering.

OpenFermion [[McClean et al., 2020](#)] is the most comprehensive existing tool, offering extensive support for operator manipulation and circuit synthesis. Qiskit Nature [[Qiskit contributors, 2023](#)] integrates tightly with IBM quantum hardware. PennyLane [[Bergholm et al., 2022](#)] excels at differentiable quantum computing. FockMap does not compete on scope; it provides the *framework* abstraction that these libraries lack, enabling systematic exploration and comparison of encodings.

Acknowledgements

This work is dedicated to Dr. Guang Hao Low, whose early encouragement to study Bravyi–Kitaev encodings motivated the development of this symbolic algebra framework. The author also acknowledges the F# Software Foundation and the .NET open-source community for the language and runtime ecosystem supporting this work.

References

- John Azariah. FockMap library cookbook: A progressive tutorial for symbolic fock-space operator algebra, 2026a. URL <https://github.com/johnazariah/encodings/tree/main/docs/guides/cookbook>. Companion documentation. Available at <https://github.com/johnazariah/encodings>.
- Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21:467–488, 1982. doi: 10.1007/BF02650179.
- Alán Aspuru-Guzik, Anthony D. Dutoi, Peter J. Love, and Martin Head-Gordon. Simulated quantum computation of molecular energies. *Science*, 309:1704–1707, 2005. doi: 10.1126/science.1113479.

- Jarrold R. McClean, Nicholas C. Rubin, Kevin J. Sung, et al. OpenFermion: the electronic structure package for quantum computers. *Quantum Science and Technology*, 5(3):034014, 2020. doi: 10.1088/2058-9565/ab8ebc.
- Qiskit contributors. Qiskit: An open-source framework for quantum computing. 2023. doi: 10.5281/zenodo.2573505.
- Ville Bergholm, Josh Izaac, Maria Schuld, et al. PennyLane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*, 2022.
- John Azariah. From molecules to qubits: A complete guide to fermion-to-qubit encoding for quantum chemistry simulation, 2026b. URL <https://github.com/johnazariah/encodings/tree/main/docs/from-molecules-to-qubits>. Companion tutorial. Available at <https://github.com/johnazariah/encodings>.