Dirace: Practical Proof Automation of Dirac Notation Equations

No Author Given

No Institute Given

Abstract. Dirac notation is a fundamental language in quantum computation and quantum information. Recently, the term rewriting system DiracDec [4] was introduced to automate equational reasoning with Dirac notation, a critical yet time-intensive task. Building upon DiracDec, this work aims to develop a solver optimized for practical applications. Enhancements include an improved typing system, a simplified language and rewriting system, more efficient algorithms, and added support for labeled Dirac notation. Our solver, Dirace, is implemented in C++ with a Mathematica backend, demonstrating significant improvements in decision-making power and time efficiency.

1 Introduction

In 1939, Dirac introduced his notation for quantum mechanics [2], designed to represent linear algebraic formulas in a compact and convenient form. For example, the expression $a\,|\psi\rangle + b\,|\phi\rangle$ represents the superposition of two quantum states, $|\psi\rangle$ and $|\phi\rangle$. Today, Dirac notation is widely accepted as the standard language in quantum computation and quantum information. Its reasoning forms the foundation of research and applications, much like boolean and integer logic do in classical computer science.

In quantum algorithm formalizations and quantum programming languages, Dirac notation is frequently used in equational proofs, which are critical, repetitive, and often time-intensive. These notations also play a key role in defining program states, operations, and assertions in quantum programming languages. To automate the verification of these programs, we need tools that can simplify and check the equivalence of preconditions. However, unlike the well-established SAT and SMT solvers for classical logic, a practical solver for Dirac notation equivalence remains an unmet need, creating a barrier for progress in several fields.

Recently, Xu et al. [4] proposed a theory for deciding the equivalence of Dirac notation, alongside a prototype implementation in Mathematica called DiracDec. They demonstrated that the equivalence of basic Dirac notation is decidable. Their algorithm, based on a pure term rewriting system, has been proven to be confluent and terminating. Despite this, there remains a gap between DiracDec and a practical solver for Dirac notation equivalence.

One challenge is the efficiency of the algorithm when dealing with equivalences beyond the scope of term rewriting. DiracDec decides the entire equational theory by rewriting modulo E, where E represents a set of axioms that cannot be decided by normalization alone, such as the associativity and commutativity (AC) of certain function symbols. DiracDec uses a direct but inefficient algorithm to decide these axioms, which searches through all possible permutations and exhibits factorial complexity. This inefficiency becomes particularly evident in "computational examples" containing many AC symbols, which are time-consuming to process.

Usability is another area where DiracDec falls short. It does not support labelled Dirac notation, which uses registers to denote subsystems and express states locally. Additionally, DiracDec's typing system only does not provide context for variable typing assumptions or the definition of symbols, which are required in practical scenarios. To avoid type checking during term rewriting, DiracDec separates symbols (e.g., multiplication) for different types, leading to unnecessary complexity. Moreover, integrating the Mathematica implementation into other projects as a solver is challenging.

The system design of DiracDec reflects a trade-off between simplicity and efficiency. While the simplicity of term rewriting allows for strong theoretical results, it limits optimization opportunities. Building on DiracDec, this work aims to develop a practical solver. We transform the term rewriting system into a hybrid algorithm, overcoming the challenges mentioned above. Our main technical contributions are:

- An efficient algorithm for deciding the equational theories in E, based on equivalence checking through normalization, with the normal form for E being obtained via sorting.
- Support for constant register labels, and reducing the equivalence decision of labelled Dirac notation to the unlabelled case.
- A more user-friendly C++ solver, Dirace, featuring an abstract language and typing system. We also support the definition of symbols (e.g., transpose and trace) using function syntax.

We evaluated Dirace against the DiracDec benchmark and new examples involving labelled Dirac notation. The results show significant improvements in both decidability and efficiency compared to DiracDec.

2 Motivation and Preliminary

An interesting property of quantum mechanics is that for two maximally entangled states, applying a quantum operator M to one subsystem is equivalent to applying M^T (the transpose of M) to the other subsystem. This relationship holds regardless of the spatial separation between the two systems, and it can be expressed as an equation in Dirac notation.

Example 1. Let q and r represent two quantum systems in the Hilbert space $\mathcal{H}_T$. Let M be a quantum operation acting on $\mathcal{H}_T$, and let $|\Phi\rangle = \sum_{i\in T} |i\rangle \otimes |i\rangle$ be the maximally entangled state. Then, we have the following equation:

$$M_q \, |\Phi\rangle_{q;r} = M_r^T \, |\Phi\rangle_{q;r} \, .$$

In this equation, q and r are labels denoting the respective subsystems in the Dirac notation. To automate reasoning about such equations, we must formalize the language and develop a proof system to handle it.

Dirac Notation Quantum states are represented as vectors in complex Hilbert spaces, and operations on these states are described by linear transformations. Dirac notation uses the ket $|i\rangle$ and the bra $\langle i|$ to denote basis vectors in a Hilbert space and its dual space, respectively. These symbols can be composed together in various ways to form more complex expressions. The meaning of these compositions depends on the types of the operands involved. For example, the inner product $\langle i|j\rangle$ represents the scalar result of the dot product between $\langle i|$ and $|j\rangle$, while the outer product $|i\rangle \langle j|$ represents an operator. Additionally, Dirac notation uses the tensor product symbol $\otimes$ to describe the combined space of multiple quantum systems.

One of the key features of Dirac notation is its order-independent interpretation. This means that the composition of terms can be written without parentheses, as the interpretation is unaffected by the order of multiplication. For instance, the formula $\langle i| \, |\phi\rangle \, \langle \psi| \, |j\rangle$ can be understood as:

$$\langle i|\phi\rangle \, \langle \psi|j\rangle = \langle i| \, (|\phi\rangle \, \langle \psi|) \, |j\rangle \, ,$$

and these expressions are equivalent for all variables involved.

In practice, Dirac notation is often combined with other syntactic elements, such as the summation symbol $\sum_{i\in S} A$, to enhance expressiveness. Furthermore, labelled Dirac notation, such as $|i\rangle_q \otimes |i\rangle_r$, is used to denote quantum systems in consideration, where subscripts (registers) are added to distinguish different subsystems.

Universal Algebra We use universal algebra and equational logic to formally represent Dirac notation and the reasoning procedure. A universal algebra defines a signature of function symbols, with terms constructed from constants, variables, and function applications. Other basic concepts like substitution of variables or pattern matching are also defined. In our case of Dirac notation, the signature consists of constructors and operations like $|i\rangle$ or $A \otimes B$. The reasoning process is guided by equational logic, which defines an equivalence relation that is compatible with substitution and term construction. This relation formalizes the intuitive concept of equivalence in algebra.

3 Language, Typing and Semantics

The first step is to formalize the language for Dirac notation. In DiracDec, the language has a concrete design, where the same syntax for different types corresponds to distinct symbols. Our goal is to transition from this concrete design to a more abstract one, which aligns more closely with the conventional Dirac notation we use and simplifies the term rewriting system.

To achieve this, we organize our language into three layers: the index, the type, and the term. Terms represent concrete instances such as kets, bras, and operators, which will be typed and checked. The index represents classical data types and appears in type expressions to differentiate between various Hilbert spaces and sets.

Definition 1 (Index Syntax). *The syntax for type indices is:*

$$\sigma ::= x \mid \sigma_1 \times \sigma_2.$$

Here, x is a variable, and $\sigma_1 \times \sigma_2$ represents the product type for tensor product spaces or Cartesian product sets.

Definition 2 (Type Syntax). *The syntax for Dirac notation types is:*

$$T ::= \mathsf{Basis}(\sigma) \mid \mathcal{S} \mid \mathcal{K}(\sigma) \mid \mathcal{B}(\sigma) \mid \mathcal{O}(\sigma_1, \sigma_2) \mid T_1 \to T_2 \mid \forall x.T \mid \mathsf{Set}(\sigma).$$

$\mathsf{Basis}(\sigma)$ denotes the type for basis elements in the index σ. $\mathcal{S}$ represents scalars, while $\mathcal{K}(\sigma)$ and $\mathcal{B}(\sigma)$ refer to ket and bra types in the Hilbert space σ, respectively. $\mathcal{O}(\sigma_1, \sigma_2)$ represents linear operators with σ_2 as the domain and σ_1 as the codomain. $\mathsf{Set}(\sigma)$ refers to the type of subsets of σ, used to denote the values of bound variables in summations. The remaining two constructs define function types: $T_1 \to T_2$ represents normal functions that take a T_1-type argument and return a T_2-type term, while $\forall x.T$ represents index functions that take an index argument x and produce a T-type term, which may depend on x. Index functions are essential for defining polymorphic transformations across Hilbert spaces.

Definition 3 (Term Syntax). *The syntax for Dirac notation terms is:*

$$\begin{aligned} e ::= {} & x \mid \lambda x : T.e \mid \mu x.e \mid e_1\ e_2 \mid (e_1, e_2) \\ & \mid 0 \mid 1 \mid \mathsf{ADDS}(e_1, \cdots, e_n) \mid e_1 \times \cdots \times e_n \mid e^* \mid \delta_{e_1,e_2} \mid \mathsf{DOT}(e_1, e_2) \\ & \mid \mathbf{0}_{\mathcal{K}}(\sigma) \mid \mathbf{0}_{\mathcal{B}}(\sigma) \mid \mathbf{0}_{\mathcal{O}}(\sigma_1, \sigma_2) \mid \mathbf{1}_{\mathcal{O}}(\sigma) \\ & \mid |e\rangle \mid \langle t| \mid e^\dagger \mid e_1.e_2 \mid \mathsf{ADD}(e_1, \cdots, e_n) \mid e_1 \otimes e_2 \\ & \mid \mathsf{MULK}(e_1, e_2) \mid \mathsf{MULB}(e_1, e_2) \mid \mathsf{OUTER}(e_1, e_2) \mid \mathsf{MULO}(e_1, e_2) \\ & \mid \mathbf{U}(e) \mid e_1 \star e_2 \mid \sum_{e_1} e_2. \end{aligned}$$

The terms above are explained in order: $\lambda x : T.e$ represents the abstraction for normal functions, and $\mu x.e$ represents the abstraction for index functions. $e_1\ e_2$

denotes function application. (e_1, e_2) is the basis pair for product types. The constants 0, 1, ADDS, $e_1 \times \cdots \times e_n$, and e^* are symbols for scalars. The next line includes constant symbols for kets, bras, and operators. $|e\rangle$ is a ket, $\langle t|$ is a bra, and $e^\dagger$ denotes the conjugate transpose of e. $e_1.e_2$ represents scaling the term e_2 by scalar e_1. $\mathbf{U}(e)$ denotes the universal set with index e. $e_1 \star e_2$ represents the Cartesian product of e_1 and e_2. $\sum_{e_1} e_2$ is the big operator sum, modeled by folding the function e_2 over the value set e_1. Typically, the sum's body is given by an abstraction. For convenience, we also use the notation $\sum_{x \in s} X$ to represent $\sum_s \lambda x : T.X$. Additionally, ADDS and ADD are distinct AC symbols: ADDS is used for scalar addition and ADD for linear algebra addition. There are five kinds of linear algebraic multiplications among kets, bras, and operators. These are encoded using five different symbols: DOT, MULK, MULB, OUTER, and MULO.

Compared to DiracDec, the symbols for addition, adjoint, scaling, and tensor products have been consolidated. The multiplications are still kept separate due to their differing properties. These operations are denoted as $B \cdot K$, $K_1 \cdot K_2$, $B_1 \cdot B_2$, $K \cdot B$, and $O_1 \cdot O_2$, respectively.

3.1 Typing System

The typing system is responsible for classifying terms within a proof system, using a context that specifies the types of variables and definitions. The context is divided into two components: the environment E, which includes definitions and assumptions, and the context Γ, which tracks bound variables. Both E and Γ consist of sequences of assumptions $x : T$ or definitions $x := t : T$.

Definition 4 (Environment and Context). *The syntax for E and Γ is:*

$$E ::= [] \mid E; x : \mathsf{Index} \mid E; x : T \mid E; x := t : T.$$
$$\Gamma ::= [] \mid \Gamma; x : \mathsf{Index} \mid \Gamma; x : T.$$

Definitions refer to symbols that can be expanded or unfolded, and typically represent abstract concepts, such as transpose or trace in Dirac notation. Assumptions, on the other hand, define the types of variables, and variable assumptions are implicitly universally quantified in the case of equational proofs.

Type checking in our language involves maintaining a well-formed environment and context $E[\Gamma]$. We say an expression t has type X in context $E[\Gamma]$ if the typing judgment $E[\Gamma] \vdash t : X$ can be proven through the rules in Appendix A. The well-formedness of the context $\mathcal{WF}(E)[\Gamma]$ is built incrementally:

$$\frac{}{\mathcal{WF}([])[]} \qquad \frac{\mathcal{WF}(E)[] \qquad x \notin E}{\mathcal{WF}(E; x : \mathsf{Index})[]} \qquad \frac{E[] \vdash t : T \qquad x \notin E}{\mathcal{WF}(E; x := t : T)[]}.$$

Starting with an empty context, new index symbols can be introduced, and symbols with checked types can be defined. Typing judgments are then made inductively using the information from $E[\Gamma]$. For instance, the rule $x : \mathsf{Index} \in$

$E[\Gamma]$ holds if E or Γ contains an assumption for x. And $\mathcal{K}(\sigma)$ is a valid type for kets, if the argument σ is typed as an index.

$$\frac{\mathcal{WF}(E)[\Gamma] \qquad x : \mathsf{Index} \in E[\Gamma]}{E[\Gamma] \vdash x : \mathsf{Index}} \qquad \frac{E[\Gamma] \vdash \sigma : \mathsf{Index}}{E[\Gamma] \vdash \mathcal{K}(\sigma) : \mathsf{Type}}$$

Terms are then typed accordingly. For example, the ket $|t\rangle$ will have the type $\mathcal{K}(\sigma)$ if t is a basis term of index σ. Similarly, the inner product between a bra and a ket of the same index σ is typed as a scalar. It corresponds to the constraint of inner product that vectors should be from the same Hilbert space.

$$\frac{E[\Gamma] \vdash t : \mathsf{Basis}(\sigma)}{E[\Gamma] \vdash |t\rangle : \mathcal{K}(\sigma)} \qquad \frac{E[\Gamma] \vdash B : \mathcal{B}(\sigma) \qquad E[\Gamma] \vdash K : \mathcal{K}(\sigma)}{E[\Gamma] \vdash B \cdot K : \mathcal{S}}$$

The typing for functions and applications follows the standard practice. For example, the index function $\mu x.t$ is typed with x as a bound variable of type Index, and the application $(t\ u)$ has the type $U\{x/u\}$, obtained by replacing x with the index instance u.

$$\frac{E[\Gamma; x : T] \vdash t : U}{E[\Gamma] \vdash (\lambda x : T.t) : T \to U} \qquad \frac{E[\Gamma] \vdash t : U \to T \qquad E[\Gamma] \vdash u : U}{E[\Gamma] \vdash (t\ u) : T}$$

$$\frac{E[\Gamma; x : \mathsf{Index}] \vdash t : U}{E[\Gamma] \vdash (\mu x.t) : \forall x.U} \qquad \frac{E[\Gamma] \vdash t : \forall x.U \qquad E[\Gamma] \vdash u : \mathsf{Index}}{E[\Gamma] \vdash (t\ u) : U\{x/u\}}$$

Finally, the big operator sum is modeled by folding a function over a set, with the typing rule as follows:

$$\frac{E[\Gamma] \vdash s : \mathsf{Set}(\sigma) \qquad E[\Gamma] \vdash f : \mathsf{Basis}(\sigma) \to \mathcal{K}(\tau)}{E[\Gamma] \vdash \sum_s f : \mathcal{K}(\tau)}.$$

Lemma 1. *The typing of expressions is both decidable and unique.*

Proof. The type of an expression can be determined recursively. For any given function symbol and argument types, there is at most one typing rule, ensuring the uniqueness of typing.

3.2 Semantics

The semantics of a language define the meaning of its expressions. In this context, the objective of our algorithm is to determine whether two expressions are semantically equivalent. We define the semantics in a denotational manner, mapping syntax to set-theoretic objects.

Denotational Semantics The denotational semantics interpret every expression as an object in linear algebra, according to a valuation mapping v, which assigns values to the variables in the expression. The semantics of an expression e

with a given valuation v is denoted as $[\![e]\!]_v$. Two expressions e_1 and e_2 are considered equivalent if their semantics are equal for all valuations, i.e., $[\![e_1]\!]_v = [\![e_2]\!]_v$ for all v.

The complete interpretation of terms and types is provided in Appendix B. Variables typed with Index are interpreted as finite sets, and the product of two indices $[\![\sigma_1 \times \sigma_2]\!]$ is defined as the Cartesian product of the sets $[\![\sigma_1]\!]$ and $[\![\sigma_2]\!]$. More generally, each type is interpreted as a set. For example, the scalar type $[\![\mathcal{S}]\!]$ is interpreted as the set of complex numbers $\mathbb{C}$, and the ket and bra types $[\![\mathcal{K}(\sigma)]\!]$ and $[\![\mathcal{B}(\sigma)]\!]$ are interpreted as the Hilbert space $\mathcal{H}_{[\![\sigma]\!]}$ and its dual $\mathcal{H}^*_{[\![\sigma]\!]}$, respectively. Terms are explained as the set elements. For example, the semantics of ket tensor product $[\![K_1 \otimes K_2]\!] \equiv [\![K_1]\!] \otimes [\![K_2]\!]$, is obtained by first calculating the semantics $[\![K_1]\!]$ and $[\![K_2]\!]$ as vectors, and then take the vector tensor product as result.

The idea behind the interpretation of types and terms is to formalize the typing relation using set-theoretic inclusion. Specifically, for a well-formed context $E[\Gamma]$, term t, and type T, if $E[\Gamma] \vdash t : T$, then for any valuation v, the semantics of t must lie within the semantics of T.

Lemma 2. *For any well-formed context $E[\Gamma]$, term t, and type T, if $E[\Gamma] \vdash t : T$, then for all valuations v, $[\![t]\!]_v \in [\![T]\!]_v$.*

Proof. The proof follows directly by checking each case.

This interpretation formalizes the standard understanding of Dirac notation and provides the foundation for the algorithm. However, computers cannot directly reason about equivalence through mathematical interpretations. Therefore, we proceed by defining a proof system that abstracts these concepts.

Axiomatic semantics The proof system for equivalence is based on equational logic, together with axioms that describe the properties of Dirac notation. A full list of these axioms can be found in Appendix C. The axioms cover fundamental aspects of linear spaces, as well as other structures like the tensor and inner products. For example, we have the absorption law for zero symbols: $X \cdot \mathbf{0} = \mathbf{0}$, and the bilinearity of the dot product:

$$(a.X) \cdot Y = a \cdot (X \cdot Y), \quad X \cdot (Y_1 + Y_2) = X \cdot Y_1 + X \cdot Y_2,$$

$$X \cdot (a.Y) = a \cdot (X \cdot Y), \quad (X_1 + X_2) \cdot Y = X_1 \cdot Y + X_2 \cdot Y.$$

As mentioned in the introduction, a subset E of these axioms cannot be decided by term rewriting alone. The axioms in E include:

$$\begin{aligned}
\text{AC-equivalence} \quad & \text{e.g.,} \quad X + Y = Y + X, \quad (X + Y) + Z = X + (Y + Z), \\
\alpha\text{-equivalence} \quad & \lambda x.A = \lambda y.A\{x/y\}, \\
\text{SUM-SWAP} \quad & \sum_{i \in s_1} \sum_{j \in s_2} A = \sum_{j \in s_2} \sum_{i \in s_1} A, \\
\text{scalar theories} \quad & \text{e.g.,} \quad a + 0 = a, \quad a \times (b + c) = a \times b + a \times c.
\end{aligned}$$

The scalar theories are treated separately as a module and are not considered in this work. In the implementation, we use the Mathematica kernel to decide scalar equivalences. These equational axioms provide an operable theory for the proof automation algorithm. Denotational semantics can be seen as one model for this theory, meaning that equivalences derived from the axioms always imply equivalence in the interpretations.

Lemma 3. *For all well-formed contexts $E[\Gamma]$ and terms e_1, e_2, if $\vdash e_1 = e_2$, then $[\![e_1]\!] = [\![e_2]\!]$.*

Proof. This follows directly from checking all cases.

Thus, the axioms are sound with respect to denotational semantics. However, the reverse does not hold: there exist equivalences in denotational semantics that cannot be captured by these axioms. Nevertheless, our work focuses on solving practical examples, which are fully covered by the axioms in the experiments.

To conclude this section, we demonstrate the formalization of the symbols used in the motivating example.

Example 2 (Formalizing the Motivating Example). Definitions and assumptions in the environment E are formalized as follows:

$$\begin{aligned}
&\text{TPO} && := \mu T_1.\mu T_2.\lambda O : \mathcal{O}(T_1,T_2). \sum_{i\in\mathbf{U}(T_1)} \sum_{j\in\mathbf{U}(T_2)} \langle i|\, O\, |j\rangle \cdot |j\rangle\langle i| \\
& && \quad : \forall T_1.\forall T_2.\mathcal{O}(T_1,T_2) \to \mathcal{O}(T_2,T_1); \\
&\text{phi} && := \mu T. \sum_{i\in\mathbf{U}(T)} \sum_{j\in\mathbf{U}(T)} |(i,j)\rangle : \forall T.\mathcal{K}(T\times T); \\
&T && : \mathsf{Index}; \\
&M && : \mathcal{O}(T,T).
\end{aligned}$$

The symbol TPO represents the transpose of an operator, polymorphic on the Hilbert spaces T_1 and T_2. The symbol phi takes the index T and defines the maximally entangled states, summing over all basis elements in T, as indicated by the universal set $\mathbf{U}(T)$. With the assumption of the index T and operator M, we can express the equivalence in the non-labelled version as:

$$(\mathrm{M} \otimes \mathbf{1}_{\mathcal{O}}(\mathrm{T})) \cdot (\mathrm{phi\ T}) = (\mathbf{1}_{\mathcal{O}}(\mathrm{T}) \otimes (\mathrm{TPO\ T\ T\ M})) \cdot (\mathrm{phi\ T}).$$

4 Algorithm for Deciding Dirac Notation Equations

We utilize the term rewriting technique to decide the equivalence of Dirac notation, based on its axiomatic semantics. The process involves normalizing two terms and checking whether they have identical syntax. This normalization is achieved through matching and substitution in universal algebra, using a set of predefined rewriting rules.

As mentioned in the semantics section, the axioms in E cannot be fully decided through term rewriting. DiracDec decides E by examining all possible permutations in the rewriting result. In this work, we introduce a more efficient approach by incorporating sorting algorithms into the normalization procedure.

The normalization procedure consists of the following steps.

1. **First Rewritings**: Expand definitions and simplify expressions.
2. **Variable Expansion**: Consider scalar expressions to ensure completeness.
3. **Second Rewritings**: Normalize terms modulo E.
4. **Sorting Without Bound Variables**: Normalize AC-equivalence.
5. **Swap Successive Summations**: Normalize SUM-SWAP equivalences.
6. **De Bruijn Normalization**: Normalize α-equivalence.

4.1 Normalization Modulo E by Term Rewriting

Term rewriting rules, represented as $l \triangleright r$, are applied recursively to normalize terms. In each step, subterms matching the left-hand side l of a rule are replaced with the corresponding right-hand side r. The procedure terminates when no further rewritings can be made. A comprehensive list of rewriting rules can be found in Appendix D. Below are some key examples to illustrate the design:

One of our optimizations is using functions with indefinite arities. Therefore, we use a flattening rule to handle associativity with AC symbols:

$$a_1 + \cdots + (b_1 + \cdots + b_m) + \cdots + a_n \triangleright a_1 + \cdots + b_1 + \cdots + b_m + \cdots + a_n.$$

Commutativity is handled later in the sorting step. Many of the rewriting rules are directly derived from the equational axioms, such as:

$$\begin{aligned}
&\text{(R-DOT6)} && \langle s| \cdot |t\rangle \triangleright \delta_{s,t},\\
&\text{(R-DELTA0)} && \delta_{s,s} \triangleright 1,\\
&\text{(R-MULK1)} && O : \mathcal{O}(\sigma,\tau) \Rightarrow O \cdot \mathbf{0}_{\mathcal{K}(\tau)} \triangleright \mathbf{0}_{\mathcal{K}(\sigma)},\\
&\text{(R-MULK11)} && (O_1 \otimes O_2) \cdot (K_1 \otimes K_2) \triangleright (O_1 \cdot K_1) \otimes (O_2 \cdot K_2).
\end{aligned}$$

Some of these directions are obvious. For example, (R-DOT6) states that the inner product of two basis vectors is reduced to a delta expression, and (R-DELTA0) transforms a delta function on identical basis to a scalar 1. The rule (R-MULK1) reflects the axiom that multiplying a zero vector results in zero. This rule is conditional on typing, which is checked during rewriting. Some rules, like (R-MULK11), require a deeper understanding, such as the preference for tensor products over multiplication.

As a reference, the term rewriting system in DiracDec has been proven complete for all axioms, except for the sum symbol. The completeness result is derived from checking the confluence and termination of the system. Our rewriting rules are translations from DiracDec into the typed and abstract language, ensuring that the corresponding symbols in our system are also complete.

We also have additional rules that handle summations. For example:

$$i \text{ free in } t \Rightarrow \sum_{i \in \mathbf{U}(\sigma)} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} (\delta_{i,t}.A) \ \triangleright \ \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} A\{i/t\},$$

$$\left(\sum_{i \in M} B\right) \cdot K \ \triangleright \ \sum_{i \in M} (B \cdot K).$$

The first rule eliminates delta expressions in summations, while the second rule pushes summations outside of inner products. While there is no guarantee of completeness for these rules, they work effectively in practice.

Variable Expansion One important technique, revealed in the DiracDec work, is the expansion of variables, which is critical for proofs involving summation. For example:

$$\frac{E[\Gamma] \vdash K : \mathcal{K}(\sigma)}{E[\Gamma] \vdash K \ \triangleright \ \sum_{i \in \mathbf{U}(\sigma)} (\langle i| \cdot K).|i\rangle} \qquad \frac{E[\Gamma] \vdash B : \mathcal{B}(\sigma)}{E[\Gamma] \vdash B \ \triangleright \ \sum_{i \in \mathbf{U}(\sigma)} (B \cdot |i\rangle).\langle i|},$$

$$\frac{E[\Gamma] \vdash O : \mathcal{O}(\sigma, \tau)}{E[\Gamma] \vdash O \ \triangleright \ \sum_{i \in \mathbf{U}(\sigma)} \sum_{j \in \mathbf{U}(\tau)} (\langle i| \cdot O \cdot |j\rangle).(|i\rangle \cdot \langle j|)}.$$

These rules transform variables into their symbolic summations based on their decomposition over the basis. The rules are not terminating, therefore is applied recrusively once in the second step called *variable expansion*. Nevertheless, we have found that applying the expansion only once for all variables is sufficient for normalization.

Lemma 4. *Let expand*(e) *denote the result of expanding all variables in* e *once. For all well-typed terms* e *in* $E[\Gamma]$*,* expand(expand(e)) *and* expand(e) *have the same normal form.*

Proof. Expanding a ket variable twice, for example, results in the following transformation:

$$\sum_{i \in \mathbf{U}(\sigma)} (\langle i| \cdot \sum_{j \in \mathbf{U}(\sigma)} (\langle j| \cdot K) \cdot |j\rangle) \cdot |i\rangle \ \triangleright \ \sum_{i \in \mathbf{U}(\sigma)} \sum_{j \in \mathbf{U}(\sigma)} (\langle j| \cdot K \cdot \langle i|j\rangle) \cdot |i\rangle \,,$$

where the delta symbol elimination rule returns the term to its original form. The same holds for bra and operator terms.

4.2 Deciding Equational Theory E

In the Mathematica implementation of DiracDec, the equational theory E is decided using unification, which attempts to find a substitution for the summation bound variables that makes two expressions syntactically equivalent. This unification process iterates through all permutations of AC symbol arguments, and its complexity is factorial in the maximum number of AC symbol arguments.

As the first improvement, we check α-equivalence using the de Bruijn index [1], which replaces references to bound variable names with the distance from the lambda abstraction to the variable. For instance, the nominal lambda abstraction $\lambda x.x$ is transformed into $\lambda.0$, while $\lambda x.\lambda y.(x\ (y\ x))$ is transformed into $\lambda.\lambda.(1\ (0\ 1))$. This transformation into the de Bruijn index is performed as the final step in the normalization process to check equivalence between terms with different bound variable names.

The remaining axioms, such as AC-equivalence and SUM-SWAP, assert equivalence under permutations. A standard approach for deciding such equivalences is to normalize terms by sorting in a predefined order. For example, given the dictionary order $a < b < c$, the term $b + c + a$ (and any other AC-equivalent term) is normalized into $a + b + c$. However, in our setting, two intertwined difficulties arise: how to assign an order to all terms in the language, and how to simultaneously sort for both axioms.

Consider the following two equivalent terms:

$$\sum_{i\in s_1}\sum_{j\in s_2}\langle i|\,A\,|j\rangle\times\langle j|\,B\,|i\rangle=\sum_{i\in s_2}\sum_{j\in s_1}\langle i|\,B\,|j\rangle\times\langle j|\,A\,|i\rangle$$

While these two terms are equivalent, directly sorting the elements of scalar multiplication using lexical order does not yield the same form.

To address this issue, we propose an algorithm that sorts terms without considering bound variables. The key observation is that in a successive sum expression $\sum_{i\in s_1}\cdots\sum_{j\in s_n}A$, the names and order of the bound variables $i,\ldots,j$ can be freely permuted. Therefore, all bound variables should be treated uniformly during sorting, and the order of summation can then be determined based on the position of the bound variables.

In the example above, we first ignore the bound variables and sort the sum body into $\langle\bullet|\,A\,|\bullet\rangle\times\langle\bullet|\,B\,|\bullet\rangle$. Then, we swap the summations such that the bound variable at the first $\bullet$ position appears at the outermost position. The results will have the same de Bruijn normal form, namely $\sum_{s_1}\sum_{s_2}\langle\$1|\,A\,|\$0\rangle\times\langle\$0|\,B\,|\$1\rangle$.

To describe the algorithm in the following, we introduce two key notations. For a term $e = f(a_1, a_2, \ldots, a_n)$, $\mathrm{head}(e)$ denotes the function symbol f, while $\arg(e, i)$ refers to the i-th argument a_i of the term. In this context, variables and constants are treated as functions with zero arguments.

Definition 5 (Order Without Bound Variables). *Let $\mathcal{B}$ represent the set of bound variables, with the assumption that all bound variables are unique. We also assume that a total order exists over all symbols. The relation $e_1 =_{\mathcal{B}} e_2$ holds if:*

- *$head(e_1) = head(e_2)$, and for all i, $arg(e_1, i) =_{\mathcal{B}} arg(e_2, i)$, or*
- *$e_1 \in \mathcal{B}$ and $e_2 \in \mathcal{B}$.*

The relation $e_1 <_{\mathcal{B}} e_2$ holds between two terms if:

- *$e_1 \notin \mathcal{B}$ and $e_2 \in \mathcal{B}$, or*
- *$head(e_1) < head(e_2)$, or*

- $head(e_1) = head(e_2)$, *and there exists* n *with* $arg(e_1, n) <_{\mathcal{B}} arg(e_2, n)$, *where* $arg(e_1, i) =_{\mathcal{B}} arg(e_2, i)$ *for all* $i < n$.

It can be shown that $e_1 =_{\mathcal{B}} e_2$ if and only if neither $e_1 <_{\mathcal{B}} e_2$ nor $e_2 <_{\mathcal{B}} e_1$ holds. The purpose of this ordering is to compare function symbols in a top-down manner while ignoring bound variables. The sorted order enables normalization of terms in terms of AC equivalence.

Definition 6 (Sort Transformation). *For a term* e *with bound variable set* $\mathcal{B}$, *The sort transformation* $sort(e)$ *is defined as*

- e, *if* e *is a variable or constant;*
- $\lambda x : T.sort(e')$, *if* $e \equiv \lambda x : T.e'$;
- $\mu x.sort(e')$, *if* $e \equiv \mu x.sort(e')$; *or*
- $f(sort(a_1), \cdots, sort(a_n))$, *if* $e \equiv f(a_1, \cdots, a_n)$. *If* f *is an AC symbol, then the order of* $sort(a_i)$ *is sorted ascendingly according to* $<_{\mathcal{B}}$.

After sorting, the next step is the *swap transformation*, which arranges successive summations based on the order of bound variables.

Definition 7 (Swap Transformation). *For a term* e *with a sorting result* $sort(e)$, *the swap transformation proceeds by ordering all bound variables according to their first appearances, except in function definitions* λx *and* μx. *The swap transformation then reorders the successive summations accordingly.*

The algorithm that applies the sort and swap transformations, followed by de Bruijn normalization, successfully handles all equivalences in our benchmark. We formalize the completeness of this transformation in the following conjecture:

Conjecture 1 (Completeness of Transformation). For any two terms e_1 and e_2 that are equivalent under AC equivalence, SUM-SWAP, and α-equivalence, they will have the same form after sort, swap, and de Bruijn transformations.

Lastly, we can prove that the equivalence established by this normalization procedure is sound with respect to the semantics.

Theorem 1 (Soundness). *For any well-formed context* $E[\Gamma]$ *and well-typed expressions* e_1 *and* e_2, *if* e_1 *and* e_2 *have the same normal form, then* $[\![e_1]\!] = [\![e_2]\!]$.

Proof. The soundness of the term rewriting procedure follows from the fact that each rewriting rule preserves equivalence. Furthermore, the operations in the sort and swap transformations respect the AC-equivalence and SUM-SWAP axioms. Finally, the de Bruijn normalization ensures soundness for α-equivalence.

5 Labelled Dirac Notation

Labelled Dirac notation uses register names to indicate the quantum system of vectors and operators. This allows us to express and reason aboud the states and

operations locally, without referring to the whole system. For instance, assume Q and R are two registers, we have

$$M_Q \cdot |\Phi\rangle_{(Q,R)} = ((M \otimes I) \cdot |\Phi\rangle)_{(Q,R)}.$$

On the left hand side, the state of two subsystems is $|\Phi\rangle$, and we apply quantum operation M on the system Q. It is equivalent to extend the operation using identity operators on other subsystems (i.e. the cylinder extension), and consider the application in the whole system.

In this section, we introduce the language and typing of registers and labelled Dirac notation, and demonstrate how to transform the equivalence problem into that for the Dirac notation studied above. The first new notion is quantum registers, and we assume they are constructed from a set $\mathcal{R}$.

Definition 8 (quantum registers).

$$R ::= r \in \mathcal{R} \mid (R, R).$$

Registers can be composed by pairs of (R, R), and this structure corresponds to the structure of tensor product spaces in Dirac notation. To reason about the registers, we define their variable set as the enumeration of all register symbols involved.

Definition 9 (register variable set). *The variable set of a register is defined inductively by:*

- $\mathsf{var}(R) = \{r\}$, *if* $R \equiv r$; *or*
- $\mathsf{var}(R) = \mathsf{var}(R_1) \cup \mathsf{var}(R_2)$, *if* $R \equiv (R_1, R_2)$.

Definition 10 (labelled Dirac notation). *The* ***labelled Dirac notation*** *includes all Dirac notation symbols and the generators defined below. Here,* $s \subseteq \mathcal{R}$ *is a register variable set.*

$$T ::= \mathcal{D}(s, s) \mid \mathsf{Reg}(\sigma)$$
$$e ::= R \mid |i\rangle_r \mid {}_r\langle i| \mid e_R \mid e_{R;R} \mid e \otimes e \otimes \cdots \otimes e \mid e \cdot e$$

$\mathcal{D}(s_1, s_2)$ is the unified type for all labelled Dirac notation, where s_1 indicates the codomain systems and s_2 indicates the domain systems. For instance, labelled ket has type $\mathcal{D}(s_1, \emptyset)$, and labelled bra has type $\mathcal{D}(\emptyset, s_2)$. $\mathsf{Reg}(\sigma)$ are types for registers R, and the index σ indicates the type of Hilbert space represented by the register. Terms also include the labelled notation e_R for bra, ket and $e_{R;R}$ for operators. We introduce new dot and tensor product symbols for labelled Dirac notation. In labelled Dirac notation, the structure of tensor product does not matter, therefore $\otimes$ is an AC symbol. Following the unified type $\mathcal{D}(s, s)$, all kinds of multiplications are represented by the same dot product $e \cdot e$. Finally, $|i\rangle_r$ and ${}_r\langle i|$ are labelled basis for the normal form of labelled Dirac notation, where r are registers symbols in $\mathcal{R}$.

Typing rules Some typing rules are introduced here. The rule for $\mathcal{D}(s_1, s_2)$ requires that all registers in variable set s_1 and s_2 are well-typed. The rule for K_R demonstrates how a register label is added to the Dirac notation, and the rule for $D^\dagger$ shows that labelled Dirac notation also have symbols for calculation, such as the adjoint.

$$\frac{E[\Gamma] \vdash \sigma : \mathsf{Index}}{E[\Gamma] \vdash \mathsf{Reg}(\sigma) : \mathsf{Type}} \qquad \frac{E[\Gamma] \vdash r : \mathsf{Reg}(\sigma_r) \text{ for all } r \text{ in } s_1 \text{ and } s_2}{E[\Gamma] \vdash \mathcal{D}(s_1, s_2) : \mathsf{Type}}$$

$$\frac{E[\Gamma] \vdash R : \mathsf{Reg}(\sigma) \qquad E[\Gamma] \vdash K : \mathcal{K}(\sigma)}{E[\Gamma] \vdash K_R : \mathcal{D}(\mathsf{var}R, \emptyset)} \qquad \frac{E[\Gamma] \vdash D : \mathcal{D}(s_1, s_2)}{E[\Gamma] \vdash D^\dagger : \mathcal{D}(s_2, s_1)}$$

The dot and the tensor product symbols are different from those in unlabelled Dirac notation. Since the goal of labels is to replace the order and structure of tensor products by the reference to registers, the tensor product becomes an AC symbol. The typing still checks whether the component subsystems are disjoint with each other.

$$\frac{E[\Gamma] \vdash D_i : \mathcal{D}(s_i, s_i') \qquad \bigcap_i s_i = \emptyset \qquad \bigcap_i s_i' = \emptyset}{E[\Gamma] \vdash D_1 \otimes \cdots \otimes D_i : \mathcal{D}(\bigcup_i s_i, \bigcup_i s_i')}.$$

As for the dot product, the disjointness is considered except registers contracted by multiplication.

$$\frac{\begin{array}{cc} E[\Gamma] \vdash D_1 : \mathcal{D}(s_1, s_1') & s_1 \cap s_2 \backslash s_1' = \emptyset \\ E[\Gamma] \vdash D_2 : \mathcal{D}(s_2, s_2') & s_2' \cap s_1' \backslash s_2 = \emptyset \end{array}}{E[\Gamma] \vdash D_1 \cdot D_2 : \mathcal{D}(s_1 \cup (s_2 \backslash s_1'), s_2' \cup (s_1' \backslash s_2))}.$$

Normalization The normalization procedure of Dirac notation is extended to check equivalence with labels. We add rules to the term rewriting system, which in general try to represent labelled Dirac notation with labelled basis and scalar coefficients. The first step is the label elimination. Take operator as an example:

$$O_{R,R'} \ \triangleright \sum_{i_{r_1} \in \mathbf{U}(\sigma_{r_1})} \cdots \sum_{i_{r_n} \in \mathbf{U}(\sigma_{r_n})} \sum_{i_{r_1'} \in \mathbf{U}(\sigma_{r_1'})} \cdots \sum_{i_{r_{n'}'} \in \mathbf{U}(\sigma_{r_{n'}'})} (\langle i_R| \cdot O \cdot |i_{R'}\rangle).(|i_{r_1}\rangle_{r_i} \otimes \cdots \otimes |i_{r_n}\rangle_{r_n} \otimes {}_{r_1'}\langle i_{r_1'}| \otimes \cdots \otimes {}_{r_{n'}'}\langle i_{r_{n'}'}|).$$

The rules for e_R (ket and bra) are similar. This step reduces all labelled terms e_R or $e_{R;R}$. Other symbols on labelled Dirac notation are also reduced by rules like $(D_1 \cdot D_2)^\dagger \ \triangleright \ D_2^\dagger \cdot D_1^\dagger$. The final step operates on sum and dot product. They will lift summation to the outside, and eliminate the bra-ket pairs whenever possible.

(R-SUM-PUSHD0) $X_1 \otimes \cdots (\sum_{i \in M} D) \cdots \otimes X_2 \ \triangleright \ \sum_{i \in M} (X_1 \otimes \cdots D \cdots \otimes X_n)$

(R-L-SORT0) $A : \mathcal{D}(s_1, s_2), B : \mathcal{D}(s_1', s_2'), s_2 \cap s_1' = \emptyset \Rightarrow A \cdot B \ \triangleright \ A \otimes B$

(R-L-SORT1) ${}_r\langle i| \cdot |j\rangle_r \ \triangleright \ \delta_{i,j}$

(R-L-SORT2) ${}_r\langle i| \cdot (Y_1 \otimes \cdots \otimes |j\rangle_r \otimes \cdots \otimes Y_m) \ \triangleright \ \delta_{i,j}.(Y_1 \otimes \cdots \otimes Y_m)$

In the end, if there are no variables of $\mathcal{D}(s_1, s_2)$, the expression will be reduced to the addition of big operator sum, and each sum body is labelled basis with Dirac notation scalar coefficients:

$$\sum_i \cdots \sum_j a_1.(|i\rangle_p \otimes \cdots \otimes \langle j|_q) + \cdots + \sum_k \cdots \sum_l a_m.(|k\rangle_r \otimes \cdots \otimes \langle l|_s)$$

In this stage, we only need to check the scalars to decide the equivalence of labelled Dirac notation.

6 Implementation and Case Study

The main purpose of this work is to build a practical tool that works well in checking Dirac notation equations. The refinements and extensions above concludes in our implementation called Dirace, a solver written in C++. It has a parser built by ANTLR4, and scalar reasonings are powered by a Mathematica kernel. The user can use commands to make definitions and assumptions in the maintained context, conduct the normalization and equivalence checking, and obtain the rewriting trace output. This implementation is tested on the benchmark of the DiracDec work, and succeeds in proving most of them efficiently. It can be used from the command line interactively, or can be integrated into other C++ projects as a library.

Project Structure The project structure is illustrated in Figure 1. `ualg` is the module for universal algebra, defining basic concepts like terms and substitutions. It serves as the library for `dirace`, which are then utilized in the example benchmarks and the toplevel command line application. The components of `dirace` are as follows:

- `symbols.cpp`: the reserved symbols and AC symbols;
- `syntax_theory.cpp`: syntax related algorithms, such as de Bruijn normalization and freeness of variables;
- `calculus.cpp`: type checker and intergration with Mathematica;
- `reduction.cpp`: all the rewriting rules and transformations;
- `dirac_parser.cpp`: parser for Dirac notation and Dirace commands;
- `prover.cpp`: the prover that maintains the context and process commands like definition or equivalence checking.

The internal data structure for terms is a pointer-based syntax tree following the function application style:

```
term ::= ID | ID [term (, term)*].
```

The syntax tree can be an identifier, or an application with an identifier as the function head, and several syntax trees as arguments. There are several Dirac

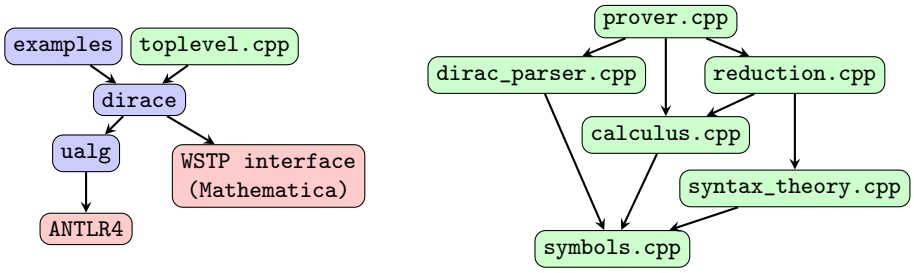

Fig. 1. The structure of Dirace. The left figure illustrates the whole system, and the right figure illustrates the `dirace` module. Blue nodes denote modules, red nodes denote third-party libraries, and green nodes denote files. Arrows represent dependency.

notation terms and their corresponding syntax trees.

$X_1 + X_2 + X_3$	`ADD[X1, X2, X3]`
$\lambda x : \mathcal{O}(T_1, T_2).x^\dagger$	`FUN[x, OTYPE[T1, T2], ADJ[x]]`
$\sum_{i \in \mathbf{U}(T)} \|i\rangle \langle i\|$	`SUM[USET[T], FUN[i, BASIS[T], OUTER[KET[i], BRA[i]]]]`

The syntax tree structure is also compatible with the datatype of Mathematica. This improves the interoperability between Dirace and the Mathematica system, enabling them to work interleavingly. To improve usability, Dirace also supports many special syntacies for terms, and most Dirac notation terms will be encoded in the natural way. Here are some examples for the parsing syntax.

syntax	parsing result	explanation
`|e>`	`KET[e]`	the ket basis
`e1 + ... + en`	`ADD[e1, ..., en]`	the addition
`e1 e2`	`COMPO[e1, e2]`	composition in Dirac notation
`e1^*`	`CONJ[e1]`	scalar conjugation
`fun i : T => X`	`FUN[i, T, X]`	lambda abstraction

Finally, Dirace uses a prover to host the computation. The prover maintains a well-formed context $E[\Gamma]$, and processes commands to modify the context and conduct calculations. The commands are listed below.

- `Def ID := term.` It defines the `ID` as the `term`, using the **W-Def** typing rule.
- `Var ID := term.` It make an assumption of `ID` with the `term` as type, using the **W-AssumeE** typing rules.
- `Check term.` Type checking the `term` and output the result.
- `Normalize term.` Normalize the `term` using the algorithm introduced in Section 4.
- `CheckEq term with term.` Check the equivalence of the two terms calculating and comparing their normal forms.

The prover will type check the terms for each command. We can also use `Normalize term with trace.` to output the proof trace during normalization. The proof trace

is a sequence of records, including the rule or transformation appied, the position of application, and the pre- and post-transformation terms. The record helps understand the normalization procedure better, and can be turned into verified proofs in theorem provers in the future.

Use Case Here we encode the motivating Example 1, examine and explain how it is checked in Dirace. The encoding is shown below.

```
Var T : INDEX. Var M : OTYPE[T, T].
Def phi := idx T => Sum nv in USET[T], |(nv, nv)>.
Var r1 : REG[T]. Var r2 : REG[T].
CheckEq M_r1;r1 (phi T)_(r1, r2) with (TPO T T M)_r2;r2 (phi T)_(r1, r2).
```

The first three lines use the `Var` and `Def` commands to set up the context for the Dirac notation. `T` is a type index, representing arbitrary Hilbert space types. `M` is assume to be an operator in the Hilbert space with type `T`. `phi` is defined as the maximally entangled state, depending on the bound variable `T` as index. `r1` and `r2` are register names for the two subsystems.

In the left hand side of `CheckEq` command, `M_r1;r1` denotes the labelled notation $M_{r_1;r_1}$, and `(phi T)_(r1, r2)` denotes the entangled state $|\Phi\rangle_{(r_1,r_2)}$. They are connected by a white space, which is parsed into the composition of Dirac notation, and will be reduced into the operator-ket multiplication after typing. The right hand side is interpreted similarly, except the defined symbol `TPO` in the environment:

```
Def TPO := idx sigma => idx tau => fun O : OTYPE[sigma, tau] => Sum i in
    USET[sigma], Sum j in USET[tau], (<i| O |j>).(|j> <i|).
```

The `TPO` symbol represents the transpose of operators, and encodes the formalization in Example 2. Thanks to the design of environment and functions, many other commonly used symbols in Dirac notation are encoded and provided as defined symbols in Dirace.

Within one second, the prover reports the result of equivalence with their common normal form:

```
The two terms are equal.
[Normalized Term] SUM[USET[T], FUN[BASIS[T], SUM[USET[T], FUN[BASIS[T],
    SCR[DOT[BRA[$1], MULK[M, KET[$0]]], LTSR[LKET[$1, r1], LKET[$0, r2
    ]]]]]]] : DTYPE[RSET[r1, r2], RSET]
```

The normal form is in the internal syntax tree format mentioned above. A more readable interpretation is:

$$\sum_{\mathbf{U}(T)} \sum_{\mathbf{U}(T)} \langle \$1|\, M\, |\$0\rangle \,.\, |\$1\rangle_{r_1} \otimes |\$0\rangle_{r_2} : \mathcal{D}(\{r_1, r_2\}, \emptyset).$$

Here \$0 and \$1 are de Bruijn indices. The result is a ket on the $\{r_1, r_2\}$ system as expected, and follows pattern proposed in Section 5.

Benchmark performance To evaluate Dirace, we first test the examples from DiracDec benchmark and make a comparison. The experiments are carried out using a MacBook Pro with M3 Max chip.

source	DiracDec			Dirace		
	expressable	success	time(s)	expressable	success	time(s)
textbook(QCQI)	18	18	1.02	18	18	0.82
CoqQ	162	156	48.69	158	158	9.74
circuits	2	2	17.67	3	2	1.4
research paper	4	4	59.53	4	4	0.73

Fig. 2. For DiracDec, examples that cannot be decided within 60 seconds are not included.

The timing of Dirace does not include the initialization of Mathematica kernel link, which takes about 3 seconds in the beginning. As to expressibility, the language for DiracDec has the support for projectors `fst` and `snd` on basis pairs, satisfying $\texttt{fst}(s,t) = s$ and $\texttt{snd}(s,t) = t$. We found this feature is almost not used, so we removed the support. As a result, Dirace encodes 158 examples for the CoqQ part, 4 less than DiracDec. For decidability, we improved the rewriting rules about sum for Dirace, and proved several more examples failed by DiracDec. The main difference is about their time efficiency. Because of our algorithm to decide AC-equivalence and SUM-SWAP, Dirace has a significant efficiency improvement, especially on those "computational examples" mentioned in the DiracDec paper. One typical example comes from the paper by Jens about the equivalence of operators for qubits. The system has to decompose the term on the concrete $|0\rangle$ and $|1\rangle$ basis, resulting a lot of addition elements. It takes DiracDec about one minute, but Dirace solves it within one second.

We also built an example benchmark for labelled Dirac notation. *[YX] : We need examples.*

7 Related Work

Automated theorem proving has seen significant advancements in recent years, particularly through the development of satisfiability modulo theories (SMT) solvers, including prominent tools like Z3. These solvers have become essential in various fields such as formal verification, synthesis, and model checking. Equational reasoning is another crucial area of research within automated theorem proving, which focuses on solving problems that involve equations between terms in an algebraic structure. Equational provers, such as Vampire and E, have played a pivotal role in addressing the challenge of proving equations in first-order logic, employing sophisticated algorithms like superposition and term rewriting.

Formal verification of quantum computation is receiving increasing attentions during these years. See [3] for a comprehensive review. Verification frameworks in Coq include the foundational formalization CoqQ and quantum circuit language QWIRE. Verifications of quantum programs are also considered, such as the Hoare logic based methods [] and model checking based methods []. The equational reasoning of Dirac notation is crucial for establishing property proofs in these works. Verification theoreis and tools based on other languages are also proposed, such as PyZX for reasoning and simplification of ZX-calculus.

8 Conclusion and Future Work

Based on the first equational reasoning tool for Dirac notation called DiracDec, this work improves and extends the theory for practical applications, and provides the solver Dirace. Experiments show that the tool demonstrates advantages in decidability, efficiency and usability.

We expect Dirace to have applications in areas like quantum program verification or proofs of post-quantum cryptography protocols in the future. One promising following up is to connect Dirace with theorem provers like Coq. It involves transforming theorem prover expressions into Dirace, and verify the proof trace of Dirace in theorem provers. Besids, most quantum program verifiers nowadays depend on matrix calculations. Dirace can serve as the replacement for matrix methods to enable symbolic deductions.

References

1. de Bruijn, N.: Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. Indagationes Mathematicae (Proceedings) **75**(5), 381–392 (1972). https://doi.org/https://doi.org/10.1016/1385-7258(72)90034-0, https://www.sciencedirect.com/science/article/pii/1385725872900340
2. Dirac, P.A.M.: A new notation for quantum mechanics. In: Mathematical proceedings of the Cambridge philosophical society. vol. 35, pp. 416–418. Cambridge University Press (1939). https://doi.org/10.1017/S0305004100021162
3. Lewis, M., Soudjani, S., Zuliani, P.: Formal verification of quantum programs: Theory, tools, and challenges. ACM Transactions on Quantum Computing **5** (12 2023). https://doi.org/10.1145/3624483
4. Xu, Y., Barthe, G., Zhou, L.: Automating equational proofs in dirac notation. Proc. ACM Program. Lang. **9**(POPL) (Jan 2025). https://doi.org/10.1145/3704878, https://doi.org/10.1145/3704878

A Full Typing Rules

This section includes the full list of typing rules.

– Rules for a well-formed environment and context.

$$\textbf{W-Empty}\quad \frac{}{\mathcal{WF}([])[]}$$

$$\textbf{W-AssumE-Index}\quad \frac{\mathcal{WF}(E)[] \qquad x \notin E}{\mathcal{WF}(E; x : \mathsf{Index})[]}$$

$$\textbf{W-AssumE-Term}\quad \frac{E[] \vdash T : \mathsf{Type} \qquad x \notin E}{\mathcal{WF}(E; x : T)[]}$$

$$\textbf{W-Def-Term}\quad \frac{E[] \vdash t : T \qquad x \notin E}{\mathcal{WF}(E; x := t : T)[]}$$

$$\textbf{W-AssumC-Index}\quad \frac{\mathcal{WF}(E)[\Gamma]}{\mathcal{WF}(E)[\Gamma; x : \mathsf{Index}]}$$

$$\textbf{W-AssumC-Term}\quad \frac{E[\Gamma] \vdash T : \mathsf{Type}}{\mathcal{WF}(E)[\Gamma; x : T]}$$

– Rules for type indices.

$$\textbf{Index-Var}\quad \frac{\mathcal{WF}(E)[\Gamma] \qquad x : \mathsf{Index} \in E[\Gamma]}{E[\Gamma] \vdash x : \mathsf{Index}}$$

$$\textbf{Index-Prod}\quad \frac{E[\Gamma] \vdash \sigma : \mathsf{Index} \qquad E[\Gamma] \vdash \tau : \mathsf{Index}}{E[\Gamma] \vdash \sigma \times \tau : \mathsf{Index}}$$

$$\textbf{Index-Qudit}\quad \frac{\mathcal{WF}(E)[\Gamma]}{E[\Gamma] \vdash \mathsf{bool} : \mathsf{Index}}$$

– Rules for types.

$$\textbf{Type-Lam}\quad \frac{E[\Gamma] \vdash T : \mathsf{Type} \qquad E[\Gamma] \vdash U : \mathsf{Type}}{E[\Gamma] \vdash T \to U : \mathsf{Type}}$$

$$\textbf{Type-Index}\quad \frac{E[\Gamma; x : \mathsf{Index}] \vdash U : \mathsf{Type}}{E[\Gamma] \vdash \forall x.U : \mathsf{Type}}$$

$$\textbf{Type-Basis}\quad \frac{E[\Gamma] \vdash \sigma : \mathsf{Index}}{E[\Gamma] \vdash \mathsf{Basis}(\sigma) : \mathsf{Type}}$$

$$\textbf{Type-Ket}\quad \frac{E[\Gamma] \vdash \sigma : \mathsf{Index}}{E[\Gamma] \vdash \mathcal{K}(\sigma) : \mathsf{Type}}$$

$$\textbf{Type-Bra}\quad \frac{E[\Gamma] \vdash \sigma : \mathsf{Index}}{E[\Gamma] \vdash \mathcal{B}(\sigma) : \mathsf{Type}}$$

$$\textbf{Type-Opt}\quad \frac{E[\Gamma] \vdash \sigma : \mathsf{Index} \qquad E[\Gamma] \vdash \tau : \mathsf{Index}}{E[\Gamma] \vdash \mathcal{O}(\sigma, \tau) : \mathsf{Type}}$$

$$\textbf{Type-Scalar}\quad \frac{\mathcal{WF}(E)[\Gamma]}{E[\Gamma] \vdash \mathcal{S} : \mathsf{Type}}$$

$$\textbf{Type-Set}\quad \frac{E[\Gamma] \vdash \sigma : \mathsf{Index}}{E[\Gamma] \vdash \mathsf{Set}(\sigma) : \mathsf{Type}}$$

$$\textbf{Type-Register}\quad \frac{E[\Gamma] \vdash \sigma : \mathsf{Index}}{E[\Gamma] \vdash \mathsf{Reg}(\sigma) : \mathsf{Type}}$$

$$\textbf{Type-Labelled}\quad \frac{E[\Gamma] \vdash r : \mathsf{Reg}(\sigma_r) \text{ for all } r \text{ in } s_1 \text{ and } s_2}{E[\Gamma] \vdash \mathcal{D}(s_1, s_2) : \mathsf{Type}}$$

– Rules for variable and function typings. Here $U\{x/u\}$ means replacing the bound variable x with u in U.

$$\textbf{Term-Var}\quad \frac{\mathcal{WF}(E)[\Gamma] \quad (x:T)\in E[\Gamma] \text{ or } (x:=t:T)\in E \text{ for some } t}{E[\Gamma]\vdash x:T}$$

$$\textbf{Lam}\quad \frac{E[\Gamma;x:T]\vdash t:U}{E[\Gamma]\vdash(\lambda x:T.t):T\to U}$$

$$\textbf{Index}\quad \frac{E[\Gamma;x:\mathsf{Index}]\vdash t:U}{E[\Gamma]\vdash(\mu x.t):\forall x.U}$$

$$\textbf{App-Lam}\quad \frac{E[\Gamma]\vdash t:U\to T \qquad E[\Gamma]\vdash u:U}{E[\Gamma]\vdash(t\ u):T}$$

$$\textbf{App-Index}\quad \frac{E[\Gamma]\vdash t:\forall x.U \qquad E[\Gamma]\vdash u:\mathsf{Index}}{E[\Gamma]\vdash(t\ u):U\{x/u\}}$$

– Basis term typing rules.

$$\textbf{Basis-0}\quad \frac{\mathcal{WF}(E[\Gamma])}{E[\Gamma]\vdash 0:\mathsf{Basis}(\mathsf{bool})}$$

$$\textbf{Basis-1}\quad \frac{\mathcal{WF}(E[\Gamma])}{E[\Gamma]\vdash 1:\mathsf{Basis}(\mathsf{bool})}$$

$$\textbf{Basis-Pair}\quad \frac{E[\Gamma]\vdash s:\mathsf{Basis}(\sigma) \qquad E[\Gamma]\vdash t:\mathsf{Basis}(\tau)}{E[\Gamma]\vdash(s,t):\mathsf{Basis}(\sigma\times\tau)}$$

– Composition typing rules.

$$\textbf{Compo-SS}\quad \frac{E[\Gamma]\vdash x:\mathcal{S} \qquad E[\Gamma]\vdash y:\mathcal{S}}{E[\Gamma]\vdash x\circ y:\mathcal{S}}$$

$$\textbf{Compo-SK}\quad \frac{E[\Gamma]\vdash x:\mathcal{S} \qquad E[\Gamma]\vdash y:\mathcal{K}(\sigma)}{E[\Gamma]\vdash x\circ y:\mathcal{K}(\sigma)}$$

$$\textbf{Compo-SB}\quad \frac{E[\Gamma]\vdash x:\mathcal{S} \qquad E[\Gamma]\vdash y:\mathcal{B}(\sigma)}{E[\Gamma]\vdash x\circ y:\mathcal{B}(\sigma)}$$

$$\textbf{Compo-SO}\quad \frac{E[\Gamma]\vdash x:\mathcal{S} \qquad E[\Gamma]\vdash y:\mathcal{O}(\sigma,\tau)}{E[\Gamma]\vdash x\circ y:\mathcal{O}(\sigma,\tau)}$$

$$\textbf{Compo-KS}\quad \frac{E[\Gamma]\vdash x:\mathcal{K}(\sigma) \qquad E[\Gamma]\vdash y:\mathcal{S}}{E[\Gamma]\vdash x\circ y:\mathcal{K}(\sigma)}$$

$$\textbf{Compo-KK}\quad \frac{E[\Gamma]\vdash x:\mathcal{K}(\sigma) \qquad E[\Gamma]\vdash y:\mathcal{K}(\tau)}{E[\Gamma]\vdash x\circ y:\mathcal{K}(\sigma\times\tau)}$$

$$\textbf{Compo-KB}\quad \frac{E[\Gamma]\vdash x:\mathcal{K}(\sigma) \qquad E[\Gamma]\vdash y:\mathcal{B}(\tau)}{E[\Gamma]\vdash x\circ y:\mathcal{O}(\sigma,\tau)}$$

$$\textbf{Compo-BS}\quad \frac{E[\Gamma]\vdash x:\mathcal{B}(\sigma) \qquad E[\Gamma]\vdash y:\mathcal{S}}{E[\Gamma]\vdash x\circ y:\mathcal{B}(\sigma)}$$

$$\textbf{Compo-BK}\quad \frac{E[\Gamma]\vdash x:\mathcal{B}(\sigma) \qquad E[\Gamma]\vdash y:\mathcal{K}(\sigma)}{E[\Gamma]\vdash x\circ y:\mathcal{S}}$$

$$\textbf{Compo-BB}\quad \frac{E[\Gamma]\vdash x:\mathcal{B}(\sigma) \qquad E[\Gamma]\vdash y:\mathcal{B}(\tau)}{E[\Gamma]\vdash x\circ y:\mathcal{B}(\sigma\times\tau)}$$

$$\textbf{Compo-BO}\quad \frac{E[\Gamma]\vdash x:\mathcal{B}(\sigma)\qquad E[\Gamma]\vdash y:\mathcal{O}(\sigma,\tau)}{E[\Gamma]\vdash x\circ y:\mathcal{B}(\tau)}$$

$$\textbf{Compo-OS}\quad \frac{E[\Gamma]\vdash x:\mathcal{O}(\sigma,\tau)\qquad E[\Gamma]\vdash y:\mathcal{S}}{E[\Gamma]\vdash x\circ y:\mathcal{O}(\sigma,\tau)}$$

$$\textbf{Compo-OK}\quad \frac{E[\Gamma]\vdash x:\mathcal{O}(\sigma,\tau)\qquad E[\Gamma]\vdash y:\mathcal{K}(\tau)}{E[\Gamma]\vdash x\circ y:\mathcal{K}(\sigma)}$$

$$\textbf{Compo-OO}\quad \frac{E[\Gamma]\vdash x:\mathcal{O}(\sigma,\tau)\qquad E[\Gamma]\vdash y:\mathcal{O}(\sigma',\tau')}{E[\Gamma]\vdash x\circ y:\mathcal{O}(\sigma\times\sigma',\tau\times\tau')}$$

$$\textbf{Compo-DD}\quad \frac{\begin{array}{cc}E[\Gamma]\vdash x:\mathcal{D}(s_1,s_1')\qquad s_1\cap s_2\backslash s_1'=\emptyset\\ E[\Gamma]\vdash y:\mathcal{D}(s_2,s_2')\qquad s_2'\cap s_1'\backslash s_2=\emptyset\end{array}}{E[\Gamma]\vdash x\circ y:\mathcal{D}(s_1\cup(s_2\backslash s_1'),s_2'\cup(s_1'\backslash s_2))}$$

– Scalar term typing rules.

$$\textbf{Sca-0}\quad \frac{\mathcal{WF}(E)[\Gamma]}{E[\Gamma]\vdash 0:\mathcal{S}}$$

$$\textbf{Sca-1}\quad \frac{\mathcal{WF}(E)[\Gamma]}{E[\Gamma]\vdash 1:\mathcal{S}}$$

$$\textbf{Sca-Delta}\quad \frac{E[\Gamma]\vdash s:\mathsf{Basis}(\sigma)\qquad E[\Gamma]\vdash t:\mathsf{Basis}(\sigma)}{E[\Gamma]\vdash \delta_{s,t}:\mathcal{S}}$$

$$\textbf{Sca-Add}\quad \frac{E[\Gamma]\vdash a_i:\mathcal{S}\text{ for all }i}{E[\Gamma]\vdash a_1+\cdots+a_n:\mathcal{S}}$$

$$\textbf{Sca-Mul}\quad \frac{E[\Gamma]\vdash a_i:\mathcal{S}\text{ for all }i}{E[\Gamma]\vdash a_1\times\cdots\times a_n:\mathcal{S}}$$

$$\textbf{Sca-Conj}\quad \frac{E[\Gamma]\vdash a:\mathcal{S}}{E[\Gamma]\vdash a^*:\mathcal{S}}$$

$$\textbf{Sca-Dot}\quad \frac{E[\Gamma]\vdash B:\mathcal{B}(\sigma)\qquad E[\Gamma]\vdash K:\mathcal{K}(\sigma)}{E[\Gamma]\vdash B\cdot K:\mathcal{S}}$$

$$\textbf{Sca-Sum}\quad \frac{E[\Gamma]\vdash s:\mathsf{Set}(\sigma)\qquad E[\Gamma]\vdash f:\mathsf{Basis}(\sigma)\to\mathcal{S}}{E[\Gamma]\vdash \sum_s f:\mathcal{S}}$$

– Ket term typing rules.

$$\textbf{Ket-0}\quad \frac{E[\Gamma]\vdash \sigma:\mathsf{Index}}{E[\Gamma]\vdash \mathbf{0}_{\mathcal{K}}(\sigma):\mathcal{K}(\sigma)}$$

$$\textbf{Ket-Basis}\quad \frac{E[\Gamma]\vdash t:\mathsf{Basis}(\sigma)}{E[\Gamma]\vdash |t\rangle:\mathcal{K}(\sigma)}$$

$$\textbf{Ket-Adj}\quad \frac{E[\Gamma]\vdash B:\mathcal{B}(\sigma)}{E[\Gamma]\vdash B^\dagger:\mathcal{K}(\sigma)}$$

$$\textbf{Ket-Scr}\quad \frac{E[\Gamma]\vdash a:\mathcal{S}\qquad E[\Gamma]\vdash K:\mathcal{K}(\sigma)}{E[\Gamma]\vdash a.K:\mathcal{K}(\sigma)}$$

$$\textbf{Ket-Add}\quad \frac{E[\Gamma]\vdash K_i:\mathcal{K}(\sigma)\text{ for all }i}{E[\Gamma]\vdash K_1+\cdots+K_n:\mathcal{K}(\sigma)}$$

$$\textbf{Ket-MulK}\quad \frac{E[\Gamma]\vdash O:\mathcal{O}(\sigma,\tau)\qquad E[\Gamma]\vdash K:\mathcal{K}(\tau)}{E[\Gamma]\vdash O\cdot K:\mathcal{K}(\sigma)}$$

$$\textbf{Ket-Tsr}\quad \frac{E[\Gamma] \vdash K_1 : \mathcal{K}(\sigma) \qquad E[\Gamma] \vdash K_2 : \mathcal{K}(\tau)}{E[\Gamma] \vdash K_1 \otimes K_2 : \mathcal{K}(\sigma \times \tau)}$$

$$\textbf{Ket-Sum}\quad \frac{E[\Gamma] \vdash s : \mathsf{Set}(\sigma) \qquad E[\Gamma] \vdash f : \mathsf{Basis}(\sigma) \to \mathcal{K}(\tau)}{E[\Gamma] \vdash \sum_s f : \mathcal{K}(\tau)}$$

– Bra term typing rules.

$$\textbf{Bra-0}\quad \frac{E[\Gamma] \vdash \sigma : \mathsf{Index}}{E[\Gamma] \vdash \mathbf{0}_{\mathcal{B}}(\sigma) : \mathcal{B}(\sigma)}$$

$$\textbf{Bra-Basis}\quad \frac{E[\Gamma] \vdash t : \mathsf{Basis}(\sigma)}{E[\Gamma] \vdash \langle t| : \mathcal{B}(\sigma)}$$

$$\textbf{Bra-Adj}\quad \frac{E[\Gamma] \vdash K : \mathcal{K}(\sigma)}{E[\Gamma] \vdash K^\dagger : \mathcal{B}(\sigma)}$$

$$\textbf{Bra-Scr}\quad \frac{E[\Gamma] \vdash a : \mathcal{S} \qquad E[\Gamma] \vdash B : \mathcal{B}(\sigma)}{E[\Gamma] \vdash a.B : \mathcal{B}(\sigma)}$$

$$\textbf{Bra-Add}\quad \frac{E[\Gamma] \vdash B_i : \mathcal{B}(\sigma) \text{ for all } i}{E[\Gamma] \vdash B_1 + \cdots + B_n : \mathcal{B}(\sigma)}$$

$$\textbf{Bra-MulB}\quad \frac{E[\Gamma] \vdash B : \mathcal{K}(\sigma) \qquad E[\Gamma] \vdash O : \mathcal{O}(\sigma, \tau)}{E[\Gamma] \vdash B \cdot O : \mathcal{B}(\tau)}$$

$$\textbf{Bra-Tsr}\quad \frac{E[\Gamma] \vdash B_1 : \mathcal{B}(\sigma) \qquad E[\Gamma] \vdash B_2 : \mathcal{B}(\tau)}{E[\Gamma] \vdash B_1 \otimes B_2 : \mathcal{B}(\sigma \times \tau)}$$

$$\textbf{Bra-Sum}\quad \frac{E[\Gamma] \vdash s : \mathsf{Set}(\sigma) \qquad E[\Gamma] \vdash f : \mathsf{Basis}(\sigma) \to \mathcal{B}(\tau)}{E[\Gamma] \vdash \sum_s f : \mathcal{B}(\tau)}$$

– Operator term typing rules.

$$\textbf{Opt-0}\quad \frac{E[\Gamma] \vdash \sigma : \mathsf{Index} \qquad E[\Gamma] \vdash \tau : \mathsf{Index}}{E[\Gamma] \vdash \mathbf{0}_{\mathcal{O}}(\sigma, \tau) : \mathcal{O}(\sigma, \tau)}$$

$$\textbf{Opt-1}\quad \frac{E[\Gamma] \vdash \sigma : \mathsf{Index}}{E[\Gamma] \vdash \mathbf{1}_{\mathcal{O}}(\sigma) : \mathcal{O}(\sigma, \sigma)}$$

$$\textbf{Opt-Adj}\quad \frac{E[\Gamma] \vdash O : \mathcal{O}(\sigma, \tau)}{E[\Gamma] \vdash O^\dagger : \mathcal{O}(\tau, \sigma)}$$

$$\textbf{Opt-Scr}\quad \frac{E[\Gamma] \vdash a : \mathcal{S} \qquad E[\Gamma] \vdash O : \mathcal{O}(\sigma, \tau)}{E[\Gamma] \vdash a.O : \mathcal{O}(\sigma, \tau)}$$

$$\textbf{Opt-Add}\quad \frac{E[\Gamma] \vdash O_i : \mathcal{O}(\sigma, \tau) \text{ for all } i}{E[\Gamma] \vdash O_1 + \cdots + O_n : \mathcal{O}(\sigma, \tau)}$$

$$\textbf{Opt-Outer}\quad \frac{E[\Gamma] \vdash K : \mathcal{K}(\sigma) \qquad E[\Gamma] \vdash B : \mathcal{B}(\tau)}{E[\Gamma] \vdash K \cdot B : \mathcal{O}(\sigma, \tau)}$$

$$\textbf{Opt-Mulo}\quad \frac{E[\Gamma] \vdash O_1 : \mathcal{O}(\sigma, \tau) \qquad E[\Gamma] \vdash O_2 : \mathcal{O}(\tau, \rho)}{E[\Gamma] \vdash O_1 \cdot O_2 : \mathcal{O}(\sigma, \rho)}$$

$$\textbf{Opt-Tsr}\quad \frac{E[\Gamma] \vdash O_1 : \mathcal{O}(\sigma_1, \tau_1) \qquad E[\Gamma] \vdash O_2 : \mathcal{O}(\sigma_2, \tau_2)}{E[\Gamma] \vdash O_1 \otimes O_2 : \mathcal{O}(\sigma_1 \times \sigma_2, \tau_1 \times \tau_2)}$$

$$\textbf{Opt-Sum}\quad \frac{E[\Gamma] \vdash s : \mathsf{Set}(\sigma) \qquad E[\Gamma] \vdash f : \mathsf{Basis}(\sigma) \to \mathcal{O}(\tau, \rho)}{E[\Gamma] \vdash \sum_s f : \mathcal{O}(\tau, \rho)}$$

– Set term typing rules.

$$\textbf{Set-U} \quad \frac{E[\Gamma] \vdash \sigma : \mathsf{Index}}{E[\Gamma] \vdash \mathbf{U}(\sigma) : \mathsf{Set}(\sigma)}$$

$$\textbf{Set-Prod} \quad \frac{E[\Gamma] \vdash A : \mathsf{Set}(\sigma) \qquad E[\Gamma] \vdash B : \mathsf{Set}(\tau)}{E[\Gamma] \vdash A \star B : \mathsf{Set}(\sigma \times \tau)}$$

– Register term typing rules.

$$\textbf{Reg-Var} \quad \frac{\mathcal{WF}(E[\Gamma]) \qquad r : \mathsf{Reg}(\sigma) \in E}{E[\Gamma] \vdash r : \mathsf{Reg}(\sigma)}$$

$$\textbf{Reg-Pair} \quad \frac{\begin{array}{l} E[\Gamma] \vdash R : \mathsf{Reg}(\sigma) \\ E[\Gamma] \vdash Q : \mathsf{Reg}(\tau) \end{array} \quad \mathsf{var}(R) \cap \mathsf{var}(Q) = \emptyset}{E[\Gamma] \vdash (R, Q) : \mathsf{Reg}(\sigma \times \tau)}$$

– Typing rules for labelled Dirac notation.

$$\textbf{L-Basis-Ket} \quad \frac{r : \mathsf{Reg}(\sigma) \in E \qquad E[\Gamma] \vdash i : \mathsf{Basis}(\sigma)}{E[\Gamma] \vdash |i\rangle_r : \mathcal{D}(\{r\}, \emptyset)}$$

$$\textbf{L-Basis-Bra} \quad \frac{r : \mathsf{Reg}(\sigma) \in E \qquad E[\Gamma] \vdash i : \mathsf{Basis}(\sigma)}{E[\Gamma] \vdash {}_r\langle i| : \mathcal{D}(\emptyset, \{r\})}$$

$$\textbf{L-Ket} \quad \frac{E[\Gamma] \vdash R : \mathsf{Reg}(\sigma) \qquad E[\Gamma] \vdash K : \mathcal{K}(\sigma)}{E[\Gamma] \vdash K_R : \mathcal{D}(\mathsf{var}R, \emptyset)}$$

$$\textbf{L-Bra} \quad \frac{E[\Gamma] \vdash R : \mathsf{Reg}(\sigma) \qquad E[\Gamma] \vdash B : \mathcal{B}(\sigma)}{E[\Gamma] \vdash B_R : \mathcal{D}(\emptyset, \mathsf{var}R)}$$

$$\textbf{L-Opt} \quad \frac{\begin{array}{l} E[\Gamma] \vdash R_1 : \mathsf{Reg}(\sigma_1) \\ E[\Gamma] \vdash R_2 : \mathsf{Reg}(\sigma_2) \end{array} \quad E[\Gamma] \vdash O : \mathcal{O}(\sigma_1, \sigma_2)}{E[\Gamma] \vdash O_{R_1;R_2} : \mathcal{D}(\mathsf{var}R_1, \mathsf{var}R_2)}$$

$$\textbf{L-Conj} \quad \frac{E[\Gamma] \vdash D : \mathcal{D}(s_1, s_2)}{E[\Gamma] \vdash D^\dagger : \mathcal{D}(s_2, s_1)}$$

$$\textbf{L-Scl} \quad \frac{E[\Gamma] \vdash S : \mathcal{S} \qquad E[\Gamma] \vdash D : \mathcal{D}(s_1, s_2)}{E[\Gamma] \vdash S.D : \mathcal{D}(s_1, s_2)}$$

$$\textbf{L-Add} \quad \frac{E[\Gamma] \vdash D_i : \mathcal{D}(s_1, s_2) \quad \text{forall } i}{E[\Gamma] \vdash D_1 + \cdots + D_n : \mathcal{D}(s_1, s_2)}$$

$$\textbf{L-Tsr} \quad \frac{E[\Gamma] \vdash D_i : \mathcal{D}(s_i, s_i') \qquad \bigcap_i s_i = \emptyset \qquad \bigcap_i s_i' = \emptyset}{E[\Gamma] \vdash D_1 \otimes \cdots \otimes D_i : \mathcal{D}(\bigcup_i s_i, \bigcup_i s_i')}$$

$$\textbf{L-Dot} \quad \frac{\begin{array}{ll} E[\Gamma] \vdash D_1 : \mathcal{D}(s_1, s_1') & s_1 \cap s_2 \backslash s_1' = \emptyset \\ E[\Gamma] \vdash D_2 : \mathcal{D}(s_2, s_2') & s_2' \cap s_1' \backslash s_2 = \emptyset \end{array}}{E[\Gamma] \vdash D_1 \cdot D_2 : \mathcal{D}(s_1 \cup (s_2 \backslash s_1'), s_2' \cup (s_1' \backslash s_2))}$$

$$\textbf{L-Sum} \quad \frac{E[\Gamma] \vdash s : \mathsf{Set}(\sigma) \qquad E[\Gamma] \vdash f : \mathsf{Basis}(\sigma) \to \mathcal{D}(s_1, s_2)}{E[\Gamma] \vdash \sum_s f : \mathcal{D}(s_1, s_2)}$$

B Denotational Semantics

Definition 11 (Interpretation of indices). *The interpretation $[\![\sigma]\!]$ of a index is defined inductively as follows:*

$$\textit{(Basis types)} \qquad [\![\sigma_1 \times \sigma_2]\!] \equiv [\![\sigma_1]\!] \times [\![\sigma_2]\!].$$

Definition 12 (Interpretation of types). *The interpretation $[\![T]\!]$ of a type is defined inductively as follows:*

$$\begin{array}{ll}
\textit{(Basis types)} & [\![\mathsf{Basis}(\sigma)]\!] \equiv [\![\sigma]\!], \\
\textit{(Dirac types)} & [\![\mathcal{S}]\!] \equiv \mathbb{C}, \qquad [\![\mathcal{K}(\sigma)]\!] \equiv \mathcal{H}_{[\![\sigma]\!]}, \qquad [\![\mathcal{B}(\sigma)]\!] \equiv \mathcal{H}^*_{[\![\sigma]\!]}, \\
& [\![\mathcal{O}(\sigma,\tau)]\!] \equiv \mathcal{L}(\mathcal{H}_{[\![\tau]\!]}, \mathcal{H}_{[\![\sigma]\!]}) \\
\textit{(Set types)} & [\![\mathsf{Set}(\sigma)]\!] = \mathcal{P}([\![\sigma]\!])
\end{array}$$

We now turn to the interpretation of expressions. As usual, the interpretation is parametrized by a valuation v, which maps all variables x to their value $v(x)$.

Definition 13 (Semantics of expressions). *The interpretation of e under valuation v, written as $[\![e]\!]_v$, is defined by the clauses of*

$$\begin{array}{ll}
\textit{(Scalars)} & [\![0]\!] \equiv 0, \qquad [\![1]\!] \equiv 1, \qquad [\![a+b]\!] \equiv [\![a]\!] + [\![b]\!], \qquad [\![a \times b]\!] \equiv [\![a]\!] \times [\![b]\!], \\
& [\![a^*]\!] \equiv [\![a]\!]^*, \qquad [\![\delta_{s,t}]\!] \equiv \begin{cases} 1, \textit{ where } [\![s]\!] = [\![t]\!], \\ 0, \textit{ where } [\![s]\!] \neq [\![t]\!], \end{cases} \qquad [\![B \cdot K]\!] \equiv [\![B]\!] \cdot [\![K]\!], \\
\textit{(Constants)} & [\![\mathbf{0}_{\mathcal{K}}(\sigma)]\!] \equiv \mathbf{0}, \qquad [\![\mathbf{0}_{\mathcal{B}}(\sigma)]\!] \equiv \mathbf{0}, \qquad [\![\mathbf{0}_{\mathcal{O}}(\sigma,\tau)]\!] \equiv \mathbf{0}, \qquad [\![\mathbf{1}_{\mathcal{O}}(\sigma)]\!] \equiv \mathbf{I}, \\
\textit{(Basis)} & [\![|t\rangle]\!] \equiv |[\![t]\!]\rangle\,, \qquad [\![\langle t|]\!] \equiv \langle[\![t]\!]|\,, \\
\textit{(Shared symbols)} & [\![D^\dagger]\!] \equiv [\![D]\!]^\dagger, \qquad [\![a.D]\!] \equiv [\![a]\!][\![D]\!], \qquad [\![D_1 + D_2]\!] \equiv [\![D_1]\!] + [\![D_2]\!], \\
& [\![D_1 \cdot D_2]\!] \equiv [\![D_1]\!] \cdot [\![D_2]\!], \qquad [\![D_1 \otimes D_2]\!] \equiv [\![D_1]\!] \otimes [\![D_2]\!]. \\
\textit{(set terms)} & [\![\mathbf{U}(\sigma)]\!] \equiv [\![\sigma]\!], \qquad [\![M_1 \times M_2]\!] \equiv [\![M_1]\!] \times [\![M_2]\!], \\
\textit{(sum)} & [\![\sum_{i \in M} X]\!]_v \equiv \sum_{m \in [\![M]\!]} [\![X]\!]_{v[i \mapsto m]}.
\end{array}$$

Denotational semantics of expressions. Symbol D represents appropriate terms from the ket, bra, or operator sorts. States in $\mathcal{H}$ are represented by column vector, co-states in $\mathcal{H}^*$ by row vector, then all $\cdot$ above are interpreted as matrix multiplications, while $\otimes$ as Kronecker products. We omit the semantics of functions.

C Axiomatic Semantics

The full list of equational axioms are provided below.

(Ax-Scalar) $(B \cdot K)^* = K^\dagger \cdot B^\dagger$

(Ax-Delta) $\delta^*_{s,t} = \delta_{s,t} \qquad \langle s| \cdot |t\rangle = \delta_{s,t}$

$\delta_{s,s} = 1 \qquad s \neq t \vdash \delta_{s,t} = 0 \qquad \delta_{s,t} = \delta_{t,s}$

(Ax-Linear) $\mathbf{0} + D = D \qquad D_1 + D_2 = D_2 + D_1$

$(D_1 + D_2) + D_3 = D_1 + (D_2 + D_3)$

$0.D = \mathbf{0} \qquad a.\mathbf{0} = \mathbf{0} \qquad 1.D = D$

$a.(b.D) = (a \times b).D \qquad (a + b).D = a.D + b.D$

$a.(D_1 + D_2) = a.D_1 + a.D_2$

(Ax-Bilinear) $D \cdot \mathbf{0} = \mathbf{0} \qquad D_1 \cdot (a.D_2) = a.(D_1 \cdot D_2)$

$D_0 \cdot (D_1 + D_2) = D_0 \cdot D_1 + D_0 \cdot D_2$

$\mathbf{0} \cdot D = \mathbf{0} \qquad (a.D_1) \cdot D_2 = a.(D_1 \cdot D_2)$

$(D_1 + D_2) \cdot D_0 = D_1 \cdot D_0 + D_2 \cdot D_0$

$D \otimes \mathbf{0} = \mathbf{0} \qquad D_1 \otimes (a.D_2) = a.(D_1 \otimes D_2)$

$D_0 \otimes (D_1 + D_2) = D_0 \otimes D_1 + D_0 \otimes D_2$

$\mathbf{0} \otimes D = \mathbf{0} \qquad (a.D_1) \otimes D_2 = a.(D_1 \otimes D_2)$

$(D_1 + D_2) \otimes D_0 = D_1 \otimes D_0 + D_2 \otimes D_0$

(Ax-Adjoint) $\mathbf{0}^\dagger = \mathbf{0} \qquad (D^\dagger)^\dagger = D \qquad (a.D)^\dagger = a^*.(D^\dagger)$

$(D_1 + D_2)^\dagger = D_1^\dagger + D_2^\dagger$

$(D_1 \cdot D_2)^\dagger = D_2^\dagger \cdot D_1^\dagger \qquad (D_1 \otimes D_2)^\dagger = D_1^\dagger \otimes D_2^\dagger$

(Ax-Comp) $D_0 \cdot (D_1 \cdot D_2) = (D_0 \cdot D_1) \cdot D_2$

$(D_1 \otimes D_2) \cdot (D_3 \otimes D_4) = (D_1 \cdot D_3) \otimes (D_2 \cdot D_4)$

$(K_1 \cdot B) \cdot K_2 = (B \cdot K_2).K_1 \qquad B_1 \cdot (K \cdot B_2) = (B_1 \cdot K).B_2$

$(B_1 \otimes B_2) \cdot (K_1 \otimes K_2) = (B_1 \cdot K_1) \times (B_2 \cdot K_2)$

(Ax-Ground) $\mathbf{1}_{\mathcal{O}}^\dagger = \mathbf{1}_{\mathcal{O}} \qquad \mathbf{1}_{\mathcal{O}} \cdot D = D \qquad \mathbf{1}_{\mathcal{O}} \otimes \mathbf{1}_{\mathcal{O}} = \mathbf{1}_{\mathcal{O}}$

$|t\rangle^\dagger = \langle t| \qquad |s\rangle \otimes |t\rangle = |(s,t)\rangle$

(Sum) $\displaystyle\sum_{i \in s} \mathbf{0} = \mathbf{0} \qquad \sum_{i \in \mathbf{U}(\sigma)} |i\rangle \cdot \langle i| = \mathbf{1}_{\mathcal{O}}(\sigma)$

$\displaystyle i \text{ free in } t \Rightarrow \sum_{i \in \mathbf{U}(\sigma)} \delta_{i,t} = 1$

$\displaystyle i \text{ free in } t \Rightarrow \sum_{i \in \mathbf{U}(\sigma)} \delta_{i,t}.A = A\{i/t\}$

$\displaystyle\sum_{i \in M} \sum_{j \in M} \delta_{i,j} = \sum_{j \in M} 1$

$\displaystyle\sum_{i \in M} \sum_{j \in M} \delta_{i,j}.A = \sum_{j \in M} A\{i/j\}$

$$b_1 \times \cdots \times (\sum_{i\in M} a) \times \cdots \times b_n \triangleright \sum_{i\in M} (b_1 \times \cdots \times a \times \cdots \times b_n)$$

$$(\sum_{i\in M} a)^* = \sum_{i\in M} a^* \qquad (\sum_{i\in M} A)^\dagger = \sum_{i\in M} A^\dagger$$

$$a.(\sum_{i\in M} A) = \sum_{i\in M} a.A \qquad (\sum_{i\in M} a).A = \sum_{i\in M} a.A$$

$$X \cdot (\sum_{i\in M} Y) = \sum_{i\in M} X \cdot Y \qquad (\sum_{i\in M} X) \cdot Y = \sum_{i\in M} X \cdot Y$$

$$X \otimes (\sum_{i\in M} Y) = \sum_{i\in M} X \otimes Y \qquad (\sum_{i\in M} X) \otimes Y = \sum_{i\in M} X \otimes Y$$

$$\sum_{i\in M} (a_1 + \cdots + a_n) = (\sum_{i\in M} a_1) + \cdots + (\sum_{i\in M} a_n)$$

$$\sum_{i\in M} (X_1 + \cdots + X_n) = (\sum_{i\in M} X_1) + \cdots + (\sum_{i\in M} X_n)$$

$$\sum_{i\in \mathbf{U}(\sigma\times\tau)} A = \sum_{j\in \mathbf{U}(\sigma)} \sum_{k\in \mathbf{U}(\tau)} A\{i/(j,k)\}$$

$$\sum_{i\in M_1 \star M_2} A = \sum_{j\in M_1} \sum_{k\in M_2} A\{i/(j,k)\}$$

(α-equivalence) $\lambda x.A = \lambda y.A\{x/y\}$

(Sum-Swap) $\sum_{i\in s_1} \sum_{j\in s_2} A = \sum_{j\in s_2} \sum_{i\in s_1} A$

D Rewriting Rules

This section includes all the rewriting rules used in the system. Related rules are collected in the same table.

Table 1: Reductions for the definitions and function applications.

Rule	Description
BETA-ARROW	$((\lambda x : T.t)\ u) \ \triangleright\ t\{x/u\}$
BETA-INDEX	$((\mu x.t)\ u) \ \triangleright\ t\{x/u\}$
DELTA	$(c := t : T) \in E \Rightarrow c \ \triangleright\ t$

Table 2: The special to flatten all AC symbols within one call.

Rule	Description
R-FLATTEN	$a_1 + \cdots + (b_1 + \cdots + b_m) + \cdots + a_n$
	$\triangleright\ a_1 + \cdots + b_1 + \cdots + b_m + \cdots + a_n$
	$a_1 \times \cdots \times (b_1 \times \cdots \times b_m) \times \cdots \times a_n$
	$\triangleright\ a_1 \times \cdots \times b_1 \times \cdots \times b_m \times \cdots \times a_n$

Rule	Description
	$X_1 + \cdots + (X'_1 + \cdots + X'_m) + \cdots + X_n$
	$\triangleright\ X_1 + \cdots + X'_1 + \cdots + X'_m + \cdots + X_n$

Table 3: Rules for scalar symbols.

Rule	Description
R-CONJ5	$\delta^*_{s,t} \ \triangleright\ \delta_{s,t}$
R-CONJ6	$(B \cdot K)^* \ \triangleright\ K^\dagger \cdot B^\dagger$
R-DOT0	$\mathbf{0}_{\mathcal{B}}(\sigma) \cdot K \ \triangleright\ 0$
R-DOT1	$B \cdot \mathbf{0}_{\mathcal{K}}(\sigma) \ \triangleright\ 0$
R-DOT2	$(a.B) \cdot K \ \triangleright\ a \times (B \cdot K)$
R-DOT3	$B \cdot (a.K) \ \triangleright\ a \times (B \cdot K)$
R-DOT4	$(B_1 + \cdots + B_n) \cdot K \ \triangleright\ B_1 \cdot K + \cdots + B_n \cdot K$
R-DOT5	$B \cdot (K_1 + \cdots + K_n) \ \triangleright\ B \cdot K_1 + \cdots + B \cdot K_n$
R-DOT6	$\langle s \vert \cdot \vert t \rangle \ \triangleright\ \delta_{s,t}$
R-DOT7	$(B_1 \otimes B_2) \cdot \vert (s,t) \rangle \ \triangleright\ (B_1 \cdot \vert s \rangle) \times (B_2 \cdot \vert t \rangle)$
R-DOT8	$\langle (s,t) \vert \cdot (K_1 \otimes K_2) \ \triangleright\ (\langle s \vert \cdot K_1) \times (\langle t \vert \cdot K_2)$
R-DOT9	$(B_1 \otimes B_2) \cdot (K_1 \otimes K_2) \ \triangleright\ (B_1 \cdot K_1) \times (B_2 \cdot K_2)$
R-DOT10	$(B \cdot O) \cdot K \ \triangleright\ B \cdot (O \cdot K)$
R-DOT11	$\langle (s,t) \vert \cdot ((O_1 \otimes O_2) \cdot K) \ \triangleright\ ((\langle s \vert \cdot O_1) \otimes (\langle t \vert \cdot O_2)) \cdot K$
R-DOT12	$(B_1 \otimes B_2) \cdot ((O_1 \otimes O_2) \cdot K) \ \triangleright\ ((B_1 \cdot O_1) \otimes (B_2 \cdot O_2)) \cdot K$
R-DELTA0	$\delta_{a,a} \ \triangleright\ 1$
R-DELTA1	$\delta_{(a,b),(c,d)} \ \triangleright\ \delta_{a,c} \times \delta_{b,d}$

Table 4: Rules for scaling.

Rule	Description
R-SCR0	$1.X \ \triangleright\ X$
R-SCR1	$a.(b.X) \ \triangleright\ (a \times b).X$
R-SCR2	$a.(X_1 + \cdots + X_n) \ \triangleright\ a.X_1 + \cdots + a.X_n$
R-SCRK0	$K : \mathcal{K}(\sigma) \Rightarrow 0.K \ \triangleright\ \mathbf{0}_{\mathcal{K}}(\sigma)$
R-SCRK1	$a.\mathbf{0}_{\mathcal{K}}(\sigma) \ \triangleright\ \mathbf{0}_{\mathcal{K}}(\sigma)$
R-SCRB0	$B : \mathcal{B}(\sigma) \Rightarrow 0.B \ \triangleright\ \mathbf{0}_{\mathcal{B}}(\sigma)$
R-SCRB1	$a.\mathbf{0}_{\mathcal{B}}(\sigma) \ \triangleright\ \mathbf{0}_{\mathcal{B}}(\sigma)$
R-SCRO0	$O : \mathcal{O}(\sigma,\tau) \Rightarrow 0.O \ \triangleright\ \mathbf{0}_{\mathcal{O}}(\sigma,\tau)$
R-SCRO1	$a.\mathbf{0}_{\mathcal{O}}(\sigma,\tau) \ \triangleright\ \mathbf{0}_{\mathcal{O}}(\sigma,\tau)$

Table 5: Rules for addition.

Rule	Description
R-ADDID	$+(X) \triangleright X$
R-ADD0	$Y_1 + \cdots + X + \cdots + X + \cdots + Y_n \triangleright Y_1 + \cdots + Y_n + \cdots + (1+1).X$
R-ADD1	$Y_1 + \cdots + X + \cdots + a.X + \cdots + Y_n \triangleright Y_1 + \cdots + Y_n + (1+a).X$
R-ADD2	$Y_1 + \cdots + a.X + \cdots + X + \cdots + Y_n \triangleright Y_1 + \cdots + Y_n + (a+1).X$
R-ADD3	$Y_1 + \cdots + a.X + \cdots + b.X + \cdots + Y_n \triangleright Y_1 + \cdots + Y_n + (a+b).X$
R-ADDK0	$K_1 + \cdots + \mathbf{0}_{\mathcal{K}}(\sigma) + \cdots + K_n \triangleright K_1 + \cdots + K_n$
R-ADDB0	$B_1 + \cdots + \mathbf{0}_{\mathcal{B}}(\sigma) + \cdots + B_n \triangleright B_1 + \cdots + B_n$
R-ADDO0	$O_1 + \cdots + \mathbf{0}_{\mathcal{O}}(\sigma,\tau) + \cdots + O_n \triangleright O_1 + \cdots + O_n$

Table 6: Rules for adjoint.

Rule	Description
R-ADJ0	$(X^\dagger)^\dagger \triangleright X$
R-ADJ1	$(a.X)^\dagger \triangleright (a^*).(X^\dagger)$
R-ADJ2	$(X_1 + \cdots + X_n)^\dagger \triangleright X_1^\dagger + \cdots + X_n^\dagger$
R-ADJ3	$(X \otimes Y)^\dagger \triangleright X^\dagger \otimes Y^\dagger$
R-ADJK0	$\mathbf{0}_{\mathcal{B}}(\sigma)^\dagger \triangleright \mathbf{0}_{\mathcal{K}}(\sigma)$
R-ADJK1	$\langle t|^\dagger \triangleright |t\rangle$
R-ADJK2	$(B \cdot O)^\dagger \triangleright O^\dagger \cdot B^\dagger$
R-ADJB0	$\mathbf{0}_{\mathcal{K}}(\sigma)^\dagger \triangleright \mathbf{0}_{\mathcal{B}}(\sigma)$
R-ADJB1	$|t\rangle^\dagger \triangleright \langle t|$
R-ADJB2	$(O \cdot K)^\dagger \triangleright K^\dagger \cdot O^\dagger$
R-ADJO0	$\mathbf{0}_{\mathcal{O}}(\sigma,\tau)^\dagger \triangleright \mathbf{0}_{\mathcal{O}}(\tau,\sigma)$
R-ADJO1	$\mathbf{1}_{\mathcal{O}}(\sigma)^\dagger \triangleright \mathbf{1}_{\mathcal{O}}(\sigma)$
R-ADJO2	$(K \cdot B)^\dagger \triangleright B^\dagger \cdot K^\dagger$
R-ADJO3	$(O_1 \cdot O_2)^\dagger \triangleright O_2^\dagger \cdot O_1^\dagger$

Table 7: Rules for tensor product.

Rule	Description
R-TSR0	$(a.X_1) \otimes X_2 \triangleright a.(X_1 \otimes X_2)$
R-TSR1	$X_1 \otimes (a.X_2) \triangleright a.(X_1 \otimes X_2)$
R-TSR2	$(X_1 + \cdots + X_n) \otimes X' \triangleright X_1 \otimes X' + \cdots + X_n \otimes X'$
R-TSR3	$X' \otimes (X_1 + \cdots + X_n) \triangleright X' \otimes X_1 + \cdots + X' \otimes X_n$
R-TSRK0	$K : \mathcal{K}(\tau) \Rightarrow \mathbf{0}_{\mathcal{K}}(\sigma) \otimes K \triangleright \mathbf{0}_{\mathcal{K}}(\sigma \times \tau)$
R-TSRK1	$K : \mathcal{K}(\tau) \Rightarrow K \otimes \mathbf{0}_{\mathcal{K}}(\sigma) \triangleright \mathbf{0}_{\mathcal{K}}(\tau \times \sigma)$
R-TSRK2	$|s\rangle \otimes |t\rangle \triangleright |(s,t)\rangle$

Rule	Description
R-TSRB0	$B : \mathcal{B}(\tau) \Rightarrow \mathbf{0}_{\mathcal{B}}(\sigma) \otimes B \ \triangleright\ \mathbf{0}_{\mathcal{B}}(\sigma \times \tau)$
R-TSRB1	$B : \mathcal{B}(\tau) \Rightarrow B \otimes \mathbf{0}_{\mathcal{B}}(\sigma) \ \triangleright\ \mathbf{0}_{\mathcal{B}}(\tau \times \sigma)$
R-TSRB2	$\langle s\vert \otimes \langle t\vert \ \triangleright\ \langle (s,t)\vert$
R-TSRO0	$O : \mathcal{O}(\sigma,\tau) \Rightarrow O \otimes \mathbf{0}_{\mathcal{O}}(\sigma',\tau') \ \triangleright\ \mathbf{0}_{\mathcal{O}}(\sigma \times \sigma', \tau \times \tau')$
R-TSRO1	$O : \mathcal{O}(\sigma,\tau) \Rightarrow \mathbf{0}_{\mathcal{O}}(\sigma',\tau') \otimes O \ \triangleright\ \mathbf{0}_{\mathcal{O}}(\sigma' \times \sigma, \tau' \times \tau)$
R-TSRO2	$\mathbf{1}_{\mathcal{O}}(\sigma) \otimes \mathbf{1}_{\mathcal{O}}(\tau) \ \triangleright\ \mathbf{1}_{\mathcal{O}}(\sigma \times \tau)$
R-TSRO3	$(K_1 \cdot B_1) \otimes (K_2 \cdot B_2) \ \triangleright\ (K_1 \otimes K_2) \cdot (B_1 \otimes B_2)$

Table 8: Rule for $O \cdot K$.

Rule	Description
R-MULK0	$\mathbf{0}_{\mathcal{O}}(\sigma,\tau) \cdot K \ \triangleright\ \mathbf{0}_{\mathcal{K}}(\sigma)$
R-MULK1	$O : \mathcal{O}(\sigma,\tau) \Rightarrow O \cdot \mathbf{0}_{\mathcal{K}}(\tau) \ \triangleright\ \mathbf{0}_{\mathcal{K}}(\sigma)$
R-MULK2	$\mathbf{1}_{\mathcal{O}}(\sigma) \cdot K \ \triangleright K$
R-MULK3	$(a.O) \cdot K \ \triangleright\ a.(O \cdot K)$
R-MULK4	$O \cdot (a.K) \ \triangleright\ a.(O \cdot K)$
R-MULK5	$(O_1 + \cdots + O_n) \cdot K \ \triangleright\ O_1 \cdot K + \cdots + O_n \cdot K$
R-MULK6	$O \cdot (K_1 + \cdots + K_n) \ \triangleright\ O \cdot K_1 + \cdots + O \cdot K_n$
R-MULK7	$(K_1 \cdot B) \cdot K_2 \ \triangleright\ (B \cdot K_2).K_1$
R-MULK8	$(O_1 \cdot O_2) \cdot K \ \triangleright\ O_1 \cdot (O_2 \cdot K)$
R-MULK9	$(O_1 \otimes O_2) \cdot ((O_1' \otimes O_2') \cdot K) \ \triangleright\ ((O_1 \cdot O_1') \otimes (O_2 \cdot O_2')) \cdot K$
R-MULK10	$(O_1 \otimes O_2) \cdot \vert (s,t)\rangle \ \triangleright\ (O_1 \cdot \vert s\rangle) \otimes (O_2 \cdot \vert t\rangle)$
R-MULK11	$(O_1 \otimes O_2) \cdot (K_1 \otimes K_2) \ \triangleright\ (O_1 \cdot K_1) \otimes (O_2 \cdot K_2)$

Table 9: Rule for $B \cdot O$.

Rule	Description
R-MULB0	$B \cdot \mathbf{0}_{\mathcal{O}}(\sigma,\tau) \ \triangleright\ \mathbf{0}_{\mathcal{B}}(\tau)$
R-MULB1	$O : \mathcal{O}(\sigma,\tau) \Rightarrow \mathbf{0}_{\mathcal{B}}(\sigma) \cdot O \ \triangleright\ \mathbf{0}_{\mathcal{B}}(\tau)$
R-MULB2	$B \cdot \mathbf{1}_{\mathcal{O}}(\sigma) \ \triangleright B$
R-MULB3	$(a.B) \cdot O \ \triangleright\ a.(B \cdot O)$
R-MULB4	$B \cdot (a.O) \ \triangleright\ a.(B \cdot O)$
R-MULB5	$(B_1 + \cdots + B_n) \cdot O \ \triangleright\ B_1 \cdot O + \cdots + B_n \cdot O$
R-MULB6	$B \cdot (O_1 + \cdots + O_n) \ \triangleright\ B \cdot O_1 + \cdots + B \cdot O_n$
R-MULB7	$B_1 \cdot (K \cdot B_2) \ \triangleright\ (B_1 \cdot K).B_2$
R-MULB8	$B \cdot (O_1 \cdot O_2) \ \triangleright\ (B \cdot O_1) \cdot O_2$
R-MULB9	$(B \cdot (O_1' \otimes O_2')) \cdot (O_1 \otimes O_2) \ \triangleright\ B \cdot ((O_1' \otimes O_2') \cdot (O_1 \otimes O_2))$
R-MULB10	$\langle (s,t)\vert \cdot (O_1 \otimes O_2) \ \triangleright\ (\langle s\vert \cdot O_1) \otimes (\langle t\vert \cdot O_2)$
R-MULB11	$(B_1 \otimes B_2) \cdot (O_1 \otimes O_2) \ \triangleright\ (B_1 \cdot O_1) \otimes (B_2 \cdot O_2)$

Table 10: Rules for $K \cdot B$.

Rule	Description
R-OUTER0	$B : \mathcal{B}(\tau) \Rightarrow \mathbf{0}_{\mathcal{K}}(\sigma) \cdot B \ \triangleright\ \mathbf{0}_{\mathcal{O}}(\sigma, \tau)$
R-OUTER1	$K : \mathcal{K}(\sigma) \Rightarrow K \cdot \mathbf{0}_{\mathcal{B}}(\tau) \ \triangleright\ \mathbf{0}_{\mathcal{O}}(\sigma, \tau)$
R-OUTER2	$(a.K) \cdot B \ \triangleright\ a.(K \cdot B)$
R-OUTER3	$K \cdot (a.B) \ \triangleright\ a.(K \cdot B)$
R-OUTER4	$(K_1 + \cdots + K_n) \cdot B \ \triangleright\ K_1 \cdot B + \cdots + K_n \cdot B$
R-OUTER5	$K \cdot (B_1 + \cdots + B_n) \ \triangleright\ K \cdot B_1 + \cdots + K \cdot B_n$

Table 11: Rules for $O_1 \cdot O_2$.

Rule	Description
R-MULO0	$O : \mathcal{O}(\tau, \rho) \Rightarrow \mathbf{0}_{\mathcal{O}}(\sigma, \tau) \cdot O \ \triangleright\ \mathbf{0}_{\mathcal{O}}(\sigma, \rho)$
R-MULO1	$O : \mathcal{O}(\sigma, \tau) \Rightarrow O \cdot \mathbf{0}_{\mathcal{O}}(\tau, \rho) \ \triangleright\ \mathbf{0}_{\mathcal{O}}(\sigma, \rho)$
R-MULO2	$\mathbf{1}_{\mathcal{O}}(\sigma) \cdot O \ \triangleright\ O$
R-MULO3	$O \cdot \mathbf{1}_{\mathcal{O}}(\sigma) \ \triangleright\ O$
R-MULO4	$(K \cdot B) \cdot O \ \triangleright\ K \cdot (B \cdot O)$
R-MULO5	$O \cdot (K \cdot B) \ \triangleright\ (O \cdot K) \cdot B$
R-MULO6	$(a.O_1) \cdot O_2 \ \triangleright\ a.(O_1 \cdot O_2)$
R-MULO7	$O_1 \cdot (a.O_2) \ \triangleright\ a.(O_1 \cdot O_2)$
R-MULO8	$(O_1 + \cdots + O_n) \cdot O' \ \triangleright\ O_1 \cdot O' + \cdots + O_n \cdot O'$
R-MULO9	$O' \cdot (O_1 + \cdots + O_n) \ \triangleright\ O' \cdot O_1 + \cdots + O' \cdot O_n$
R-MULO10	$(O_1 \cdot O_2) \cdot O_3 \ \triangleright\ O_1 \cdot (O_2 \cdot O_3)$
R-MULO11	$(O_1 \otimes O_2) \cdot (O_1' \otimes O_2') \ \triangleright\ (O_1 \cdot O_1') \otimes (O_2 \cdot O_2')$
R-MULO12	$(O_1 \otimes O_2) \cdot ((O_1' \otimes O_2') \cdot O_3) \ \triangleright\ ((O_1 \cdot O_1') \otimes (O_2 \cdot O_2')) \cdot O_3$

Table 12: Rules for sets.

Rule	Description
R-SET0	$\mathbf{U}(\sigma) \star \mathbf{U}(\tau) \ \triangleright\ \mathbf{U}(\sigma \times \tau)$

Table 13: Rules for sum operators.

Rule	Description
R-SUM-CONST0	$\sum_{x \in s} 0 \ \triangleright\ 0$
R-SUM-CONST1	$\sum_{x \in s} \mathbf{0}_{\mathcal{K}}(\sigma) \ \triangleright\ \mathbf{0}_{\mathcal{K}}(\sigma)$
R-SUM-CONST2	$\sum_{x \in s} \mathbf{0}_{\mathcal{B}}(\sigma) \ \triangleright\ \mathbf{0}_{\mathcal{B}}(\sigma)$
R-SUM-CONST3	$\sum_{x \in s} \mathbf{0}_{\mathcal{O}}(\sigma, \tau) \ \triangleright\ \mathbf{0}_{\mathcal{O}}(\sigma, \tau)$
R-SUM-CONST4	$\mathbf{1}_{\mathcal{O}}(\sigma) \ \triangleright\ \sum_{i \in \mathbf{U}(\sigma)} \lvert i\rangle \cdot \langle i\rvert$

Table 14: Rules for eliminating $\delta_{s,t}$. These rules match the δ operator modulo the commutativity of its arguments.

Rule	Description
R-SUM-ELIM0	i free in $t \Rightarrow \sum_{i \in \mathbf{U}(\sigma)} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} \delta_{i,t}$ $\triangleright \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} 1$
R-SUM-ELIM1	i free in $t \Rightarrow$ $\sum_{i \in \mathbf{U}(\sigma)} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} (a_1 \times \cdots \times \delta_{i,t} \times \cdots \times a_n)$ $\triangleright \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} a_1\{i/t\} \times \cdots \times a_n\{i/t\}$
R-SUM-ELIM2	i free in $t \Rightarrow \sum_{i \in \mathbf{U}(\sigma)} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} (\delta_{i,t}.A)$ $\triangleright \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} A\{i/t\}$
R-SUM-ELIM3	i free in $t \Rightarrow$ $\sum_{i \in \mathbf{U}(\sigma)} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} (a_1 \times \cdots \times \delta_{i,t} \times \cdots \times a_n).A$ $\triangleright \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} (a_1\{i/t\} \times \cdots \times a_n\{i/t\}).A\{i/t\}$
R-SUM-ELIM4	$\sum_{i \in M} \sum_{j \in M} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} \delta_{i,j}$ $\triangleright \sum_{j \in M} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} 1$
R-SUM-ELIM5	$\sum_{i \in M} \sum_{j \in M} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} (a_1 \times \cdots \times \delta_{i,j} \times \cdots \times a_n)$ $\triangleright \sum_{j \in M} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} (a_1\{j/i\} \times \cdots \times a_n\{j/i\})$
R-SUM-ELIM6	$\sum_{i \in M} \sum_{j \in M} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} (\delta_{i,j}.A)$ $\triangleright \sum_{j \in M} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} A\{j/i\}$
R-SUM-ELIM7	$\sum_{i \in M} \sum_{j \in M} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} (a_1 \times \cdots \times \delta_{i,j} \times \cdots \times a_n).A$ $\triangleright \sum_{j \in M} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} (a_1\{j/i\} \times \cdots \times a_n\{j/i\}).A\{j/i\}$
R-SUM-ELIM8	$\sum_{i \in M} \sum_{j \in M} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} ((a_1 \times \cdots \times \delta_{i,j} \times \cdots \times a_n) +$ $\cdots + (b_1 \times \cdots \times \delta_{i,j} \times \cdots \times b_n)).A$ $\triangleright \sum_{j \in M} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} ((a_1\{j/i\} \times \cdots \times a_n\{j/i\}) +$ $\cdots + (b_1\{j/i\} \times \cdots \times b_n\{j/i\})).A\{j/i\}$

Table 15: Rules for pushing terms into sum operators. Because we apply type checking on variables, and stick to unique bound variables, these operations are always sound.

Rule	Description
R-SUM-PUSH0	$b_1 \times \cdots \times (\sum_{i \in M} a) \times \cdots \times b_n$

Rule	Description
	$\triangleright\ \sum_{i\in M}(b_1 \times \cdots \times a \times \cdots \times b_n)$
R-SUM-PUSH1	$(\sum_{i\in M} a)^* \ \triangleright\ \sum_{i\in M} a^*$
R-SUM-PUSH2	$(\sum_{i\in M} X)^\dagger \ \triangleright\ \sum_{i\in M} X^\dagger$
R-SUM-PUSH3	$a.(\sum_{i\in M} X) \ \triangleright\ \sum_{i\in M}(a.X)$
R-SUM-PUSH4	$(\sum_{i\in M} a).X \ \triangleright\ \sum_{i\in M}(a.X)$
R-SUM-PUSH5	$(\sum_{i\in M} B)\cdot K \ \triangleright\ \sum_{i\in M}(B\cdot K)$
R-SUM-PUSH6	$(\sum_{i\in M} O)\cdot K \ \triangleright\ \sum_{i\in M}(O\cdot K)$
R-SUM-PUSH7	$(\sum_{i\in M} B)\cdot O \ \triangleright\ \sum_{i\in M}(B\cdot O)$
R-SUM-PUSH8	$(\sum_{i\in M} K)\cdot B \ \triangleright\ \sum_{i\in M}(K\cdot B)$
R-SUM-PUSH9	$(\sum_{i\in M} O_1)\cdot O_2 \ \triangleright\ \sum_{i\in M}(O_1\cdot O_2)$
R-SUM-PUSH10	$B\cdot(\sum_{i\in M} K) \ \triangleright\ \sum_{i\in M}(B\cdot K)$
R-SUM-PUSH11	$O\cdot(\sum_{i\in M} K) \ \triangleright\ \sum_{i\in M}(O\cdot K)$
R-SUM-PUSH12	$B\cdot(\sum_{i\in M} O) \ \triangleright\ \sum_{i\in M}(B\cdot O)$
R-SUM-PUSH13	$K\cdot(\sum_{i\in M} B) \ \triangleright\ \sum_{i\in M}(K\cdot B)$
R-SUM-PUSH14	$O_1\cdot(\sum_{i\in M} O_2) \ \triangleright\ \sum_{i\in M}(O_1\cdot O_2)$
R-SUM-PUSH15	$(\sum_{i\in M} X_1)\otimes X_2 \ \triangleright\ \sum_{i\in M}(X_1\otimes X_2)$
R-SUM-PUSH16	$X_1\otimes(\sum_{i\in M} X_2) \ \triangleright\ \sum_{i\in M}(X_1\otimes X_2)$

Table 16: Rules for addition and index in sum.

Rule	Description
R-SUM-ADDS0	$\sum_{i\in M}(a_1+\cdots+a_n) \ \triangleright\ (\sum_{i\in M} a_1)+\cdots+(\sum_{i\in M} a_n)$
R-SUM-ADD0	$\sum_{i\in M}(X_1+\cdots+X_n) \ \triangleright\ (\sum_{i\in M} X_1)+\cdots+(\sum_{i\in M} X_n)$
R-SUM-INDEX0	$\sum_{i\in \mathbf{U}(\sigma\times\tau)} A \ \triangleright\ \sum_{j\in\mathbf{U}(\sigma)}\sum_{k\in\mathbf{U}(\tau)} A\{i/(j,k)\}$
R-SUM-INDEX1	$\sum_{i\in M_1\star M_2} A \ \triangleright\ \sum_{j\in M_1}\sum_{k\in M_2} A\{i/(j,k)\}$

Table 17: Rules for bool index.

Rule	Description
R-BIT-DELTA	$\delta_{0,1} \ \triangleright\ 0$
R-BIT-ONEO	$\mathbf{1}_{\mathcal{O}}(\mathsf{bool}) \ \triangleright\ \lvert 0\rangle\langle 0\rvert + \lvert 1\rangle\langle 1\rvert$
R-BIT-SUM	$\sum_{i\in\mathbf{U}(\mathsf{bool})} A \ \triangleright\ A\{i/0\}+A\{i/1\}$

Table 18: Rules about addition and sum.

Rule	Description
R-MULS2	$b_1\times\cdots\times(a_1+\cdots+a_n)\times\cdots\times b_m$ $\triangleright\ (b_1\times\cdots\times a_1\times\cdots\times b_m)+\cdots+(b_1\times\cdots\times a_n\times\cdots\times b_m)$

Rule	Description
R-SUM-ADD1	$Y_1 + \cdots + Y_n + \sum_{i\in M}(a+b).X$
	$\triangleright\ Y_1 + \cdots + \sum_{i\in M}(a.X) + \cdots + \sum_{i\in M}(b.X) + Y_n$
R-SUM-FACTOR	$X_1 + \cdots + (\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n} A)$
	$+(\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n} A) + \cdots + X_n$
	$X_1 + \cdots + (\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n}(1+1).A) + \cdots + X_n$
	$X_1 + \cdots + (\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n} a.A)$
	$+(\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n} A) + \cdots + X_n$
	$X_1 + \cdots + (\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n}(a+1).A) + \cdots + X_n$
	$X_1 + \cdots + (\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n} a.A)$
	$+(\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n} b.A) + \cdots + X_n$
	$X_1 + \cdots + (\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n}(a+b).A) + \cdots + X_n$

Table 19: Rules to eliminate labels in Dirac notation.

Rule	Description
R-L-EXPAND	$K_R\ \triangleright\ \sum_{i_{r_1}\in\mathbf{U}(\sigma_{r_1})}\cdots\sum_{i_{r_n}\in\mathbf{U}(\sigma_{r_n})}(\langle i_R\vert\cdot K).(\vert i_{r_1}\rangle_{r_i}\otimes\cdots\otimes\vert i_{r_n}\rangle_{r_n})$
	$B_R\ \triangleright\ \sum_{i_{r_1}\in\mathbf{U}(\sigma_{r_1})}\cdots\sum_{i_{r_n}\in\mathbf{U}(\sigma_{r_n})}(B\cdot\vert i_R\rangle).({}_{r_1}\langle i_{r_1}\vert\otimes\cdots\otimes{}_{r_n}\langle i_{r_n}\vert)$
	$O_{R,R'}\ \triangleright\ \sum_{i_{r_1}\in\mathbf{U}(\sigma_{r_1})}\cdots\sum_{i_{r_n}\in\mathbf{U}(\sigma_{r_n})}\sum_{i_{r'_1}\in\mathbf{U}(\sigma_{r'_1})}\cdots\sum_{i_{r'_{n'}}\in\mathbf{U}(\sigma_{r'_{n'}})}$
	$(\langle i_R\vert\cdot O\cdot\vert i_{R'}\rangle).(\vert i_{r_1}\rangle_{r_i}\otimes\cdots\otimes\vert i_{r_n}\rangle_{r_n}\otimes{}_{r'_1}\langle i_{r'_1}\vert\otimes\cdots\otimes{}_{r'_{n'}}\langle i_{r'_{n'}}\vert)$

Table 20: Rules for labelled Dirac notation.

Rule	Description
R-ADJDK	$({}_r\langle i\vert)^\dagger\ \triangleright\ \vert i\rangle_r$
R-ADJDB	$(\vert i\rangle_r)^\dagger\ \triangleright\ {}_r\langle i\vert$
R-ADJD0	$(D_1\otimes\cdots\otimes D_n)^\dagger\ \triangleright\ D_1^\dagger\otimes\cdots\otimes D_n^\dagger$
R-ADJD1	$(D_1\cdot D_2)^\dagger\ \triangleright\ D_2^\dagger\cdot D_1^\dagger$
R-SCRD0	$D_1\otimes\cdots\otimes(a.D_n)\otimes\cdots\otimes D_m\ \triangleright\ a.(D_1\otimes\cdots\otimes D_m)$
R-SCRD1	$(a.D_1)\cdot D_2\ \triangleright\ a.(D_1\cdot D_2)$
R-SCRD2	$D_1\cdot(a.D_2)\ \triangleright\ a.(D_1\cdot D_2)$
R-TSRD0	$X_1\otimes\cdots\otimes(D_1+\cdots+D_n)\otimes\cdots X_m$
	$\triangleright\ X_1\otimes\cdots D_1\cdots\otimes X_m+\cdots+X_1\otimes\cdots D_n\cdots\otimes X_m$
R-DOTD0	$(D_1+\cdots+D_n)\cdot D\ \triangleright\ D_1\cdot D+\cdots+D_n\cdot D$

Rule	Description
R-DOTD1	$D \cdot (D_1 + \cdots + D_n) \ \triangleright \ D \cdot D_1 + \cdots + D \cdot D_n$
R-SUM-PUSHD0	$X_1 \otimes \cdots (\sum_{i \in M} D) \cdots \otimes X_2 \ \triangleright \ \sum_{i \in M} (X_1 \otimes \cdots D \cdots \otimes X_n)$
R-SUM-PUSHD1	$(\sum_{i \in M} D_1) \cdot D_2 \ \triangleright \ \sum_{i \in M} (D_1 \cdot D_2)$
R-SUM-PUSHD2	$D_1 \cdot (\sum_{i \in M} D_2) \ \triangleright \ \sum_{i \in M} (D_1 \cdot D_2)$

Table 21: Rules to simplify dot product in labelled Dirac notation.

Rule	Description
R-L-SORT0	$A : \mathcal{D}(s_1, s_2), B : \mathcal{D}(s_1', s_2'), s_2 \cap s_1' = \emptyset \Rightarrow A \cdot B \ \triangleright \ A \otimes B$
R-L-SORT1	${}_r\langle i| \cdot |j\rangle_r \ \triangleright \ \delta_{i,j}$
R-L-SORT2	${}_r\langle i| \cdot (Y_1 \otimes \cdots \otimes |j\rangle_r \otimes \cdots \otimes Y_m) \ \triangleright \ \delta_{i,j}.(Y_1 \otimes \cdots \otimes Y_m)$
R-L-SORT3	$(X_1 \otimes \cdots \otimes {}_r\langle i| \otimes \cdots \otimes X_n) \cdot |j\rangle_r \ \triangleright \ \delta_{i,j}.(X_1 \otimes \cdots \otimes X_n)$
R-L-SORT1	$(X_1 \otimes \cdots \otimes {}_r\langle i| \otimes \cdots \otimes X_n) \cdot (Y_1 \otimes \cdots \otimes |j\rangle_r \otimes \cdots \otimes Y_m)$ $\triangleright \ \delta_{i,j}.(X_1 \otimes \cdots \otimes X_n) \cdot (Y_1 \otimes \cdots \otimes Y_m)$