Dirace: Practical Proof Automation of Dirac Notation Equations

No Author Given

No Institute Given

Abstract. Dirac notations is the fundamental language in quantum computation and quantum information. Recently, the term rewriting system DiracDec [3] was introduced to automate the equational reasoning with Dirac notations, a critical yet time-intensive task. This work bases upon and extends DiracDec, aiming 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 labelled Dirac notations. Our solver Dirace is implemented in C++ with a Mathematica backend, and demonstrates advantages in decision-making power and time efficiency.

1 Introduction

In 1939, Dirac proposed his notation [2] for quantum mechanics, which is designed to represent linear algebra formulae in a compact and convenient way. For instance, $a\,|\psi\rangle + b\,|\phi\rangle$ indicates the addition of two vectors, i.e., the superposition of two states $|\psi\rangle$ and $|\phi\rangle$. Dirac notation is now widely accepted as the working language in quantum computation and quantum information. The reasonings of Dirac notation play a fundamental role in research and application, just like boolean and integer logic to classical computer science. For example, formalization works of quantum algorithms and quantum programming languages involves plenty of equational proofs of Dirac notation, which are critical, repeating but time-intensive jobs. Dirac notation are also used in quantum programming languages to define the program states, operations and assertions. In order to automate the verification procedure, we need to simplify and check the equivalence of pre-conditions. However, unlike the existence of SAT and SMT solver, we are still in need of a working Dirac notation solver, and this absence has become an obstacle of many areas.

Recently, Xu et al. [3] proposed a theory to decide the equivalence of Dirac notations, as well as a prototype implementation in Mathematica called DiracDec. They proved that the equivalence of basic Dirac notations are decidable. Their algorithm sticks with a pure term rewriting system, and is proved to be confluent and terminating. Even though, there is still a gap between DiracDec and a practical solver for Dirac notation equivalence.

One problem is the algorithm efficiency for equivalence out of the scope of rewritings. DiracDec decides the whole equational theory by rewriting modulo E, where E is a set of equational axioms that cannot be decided by normalization in rewritings, e.g. the associativity and commutativity (AC) of some function symbols. DiracDec uses a direct by inefficient algorithm to decide E, which search through all possible permutations and has factorial complexity. This is witnessed in their evaluation: DiracDec is not good at dealing with "computational examples", which has many AC symbols and are therefore time consuming.

Usability is another problem. DiracDec does not support labelled Dirac notation, which use registers to denote the subsystem, and express the state locally. The typing of DiracDec only deals with Dirac notations, which is not sufficient for a working scenario, where we want to have defined symbols and a context for the typing assumptions of variables. Moreover, to avoid type checking during term rewriting, DiracDec separate the same symbol (e.g. multiplication) for different types to disambiguate, which makes the system bloated. Also, it is hard to integrate the Mathematica implementation as a solvers into other projects.

There is a trade-off between simplicity and efficiency for the system design. The simplicity of term rewriting in DiracDec allows them to reach satisfying theoretical results, but it is also a constraint for optimization. Based on DiracDec, this work aims at building a practical solver. It transforms the term rewriting system into a hybrid algorithm, and overcomes the problems mentioned above. Our main contributions and techniques are:

– An efficient algorithm to decide the equational theories in E. The basic idea is to carry through equivalence checking by normalization, and the normal form for E are obtained by sorting.
– The support for constant register labels. The equivalence decision of labelled Dirac notations is reduced to the unlabelled situation.
– A more usable C++ solver Dirace with abstract language and typing. We also support to define symbols (e.g., transpose and trace) using the functions syntax.

As an evaluation, we tested Dirace on the benchmark of DiracDec and new examples for labelled Dirac notations. The result shows significant improvement in decidability and efficiency over DiracDec.

2 Motivation and Preliminary

One interesting fact about the quantum world is that for two maximally entangled states, applying one operator M on one subsystem is equivalent to applying M^T, the transpose of M, on the other. This property holds no matter how far the two systems are separated, and is expressed as an equation of Dirac notations.

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

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

Here q and r are called labels, denoting the system for Dirac notations. To reason about such equations using computers, we need to formalize the language and work in a proof system.

Dirac notation Quantum states live in complex Hilbert spaces. We use vectors in the space to descirbe pure quantum states, and operations are described by linear transformations. It uses the ket $|i\rangle$ and the bra $\langle i|$ to indicate bases of the space and the dual space. Together with other variable symbols, they are composed with each other in sequence, and the composition will be interpreted into different operations, depending on the type of operands. For example, $\langle i|j\rangle$ represents the inner product of $\langle i|$ and $|j\rangle$, which is a scalar, while $|i\rangle\langle j|$ represents the outer product, resulting in an operator. Dirac notation also use $\otimes$ to indicate the vectors and operators in the tensor product space.

Dirac notation further enjoys the property that the interpretation is independent on the order of composition, thus parentheses can be omitted. For example, 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 they are equivalent for all variables.

In practice, Dirac notations are combined with other syntax, such as the big operator sum $\sum_{i\in S} A$, for better expressiveness. We also use labelled Dirac notations like $|i\rangle_q \otimes |i\rangle_r$ to denote the quantum system in consideration, where Dirac notations are subscripted with registers.

Universal algebra We use universal algebra and equational logic to formally represent Dirac notations and the reasoning procedure. A universal algebra defines a signature of function symbols. Terms in the algebra are constructed by constants, variables or function application on terms. 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 $A \otimes B$. Reasonings of Dirac notation terms follow the equational logic. It defines an equaitonal relation compatible with substitution and term construction, which essentially formalizes the common sense of equivalence in algebra.

3 Language, Typing and Semantics

The first step is to formally pin down the language for Dirac notation. As a reference, the language in DiracDec has a concrete design, meaning that the same syntax for different types corresponds to different symbols. Our intention is to move from the concrete design to a more abstract one. This is closer to the casual Dirac notation we use, and simplifies the term rewriting system.

For this purpose, our language involves three layers: the index, the type and the term. Terms describe the concrete instances like ket, bra and operator, which will be checked and typed. Index represents classical datatypes, and they appear in type expressions to distinguish between different 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. $\sigma_1 \times \sigma_2$ is the product type for tensor product space or Cartesian product sets.

Definition 2 (type syntax). *The syntax for Dirac notation types is*

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

$\mathsf{Basis}(\sigma)$ is the type for basis in index σ. $\mathcal{S}$ indicates scalars, $\mathcal{K}(\sigma)$ and $\mathcal{B}(\sigma)$ indicates ket and bra terms in Hilbert space σ, and $\mathcal{O}(\sigma_1, \sigma_2)$ indicates linear operators with Hilbert space σ_2 as domain and σ_1 as codomain. $\mathsf{Set}(\sigma)$ is the type of subsets in σ index, and is used to denote the values of bound variables in summation. The remaining two generators are types for functions. $T_1 \to T_2$ denotes normal functions, which accepts a T_1 type argument and results in a T_2 type term. $\forall x.T$ denotes index functions, which accepts an index argument x and results in a T type term, where T can depend on x. Index functions are necessary for defining transformations that are polymorphic on 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 generators are explained in order. $\lambda x : T.e$ is the abstraction for normal functions, and $\mu x.e$ is the abstraction for index functions. $e_1\ e_2$ is function application. (e_1, e_2) is the basis pair for product types. 0, 1, ADDS, $e_1 \times \cdots \times e_n$ and e^* are symbols for scalars. The next line includes constant symbols for ket, bra and operator. $e^\dagger$ denotes the conjugate transpose of e. $e_1.e_2$ denotes scaling the term e_2 with scalar e_1. $\mathbf{U}(e)$ is the universal set with index e. $e_1 \star e_2$ is the Cartesian product. $\sum_{e_1} e_2$ is the big operator sum, which is modelled by folding the function e_2 over value sets e_1. Usually, the sum body is specified by an abstraction. Therefore we use the notation $\sum_{x \in s} X$ to denote $\sum_s \lambda x : T.X$ as well. Here ADDS and ADD are two different AC symbols representing the scalar addition and the linear algebra addition respectively. There are five kinds of linear algebra multiplications among ket, bra and operator. We encode the typing information by using five different symbols, namely DOT, MULK, MULB, OUTER and MULO.

Compared to DiracDec, the addition, adjoint, scaling and tensor product symbols are merged together. They 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. The multiplications are still separated because their properties do not have to much overlap.

3.1 Typing System

Typing is the procedure to classify the term through a proof system in a typing context. Our context consists of the environment E for definitions and assumptions, and the context Γ for bound variables. E and Γ are sequences of assumptions $x : T$ or definitions $x := t : T$.

Definition 4 (environment and context).

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

Definitions refer to the symbols that can be unfolded. Definition symbols usually have abstract meanings, and here we use them to encode more operations in Dirac notations, such as transpose and trace. Assumptions are the declarations of the type for variables. In the scenario of equational proofs, variable assumptions implicitly have the universal quantifier.

The type checking of 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 judgement $E[\Gamma] \vdash t : X$ can be proved through the typing rules in Appendix A. Here we present and explain the rules selectively. Firstly, well-formed contexts $\mathcal{WF}(E)[\Gamma]$ are built in the incremental way, e.g.:

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

Starting from an empty context, we can assume new index symbols, and assume or define symbols with checked types. Based on the well-formed context, typing judgements can be proved by information from $E[\Gamma]$, or built inductively. In the following rules, for example, the condition $x : \mathsf{Index} \in E[\Gamma]$ is true if E or Γ has the assumption in their sequences. $\mathcal{K}(\sigma)$ will be a valid type for kets, if the argument σ is typed as the index.

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

The Dirac notations will then be typed accordingly. For example, the ket syntax $|t\rangle$ has type $\mathcal{K}(\sigma)$, if t is typed as a basis term of index σ. Also, the inner product of a bra and a ket with the same type index σ is typed as the scalar. This corresponds to the constraint of inner products that vectors should be in 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 follow the common practice. Take the index function $\mu x.t$ as an example: here x is a bound variable typed as Index, and the type $U\{x/u\}$ of application $(t\ u)$ is obtained by replacing x with the index instance u.

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

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

Lastly, the big operator sum is modelled by folding a function on a set, therefore the typing rule is 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 are decidable and unique.*

Proof. The type can be calculated in the recursive manner. There is at most one typing rule for the same function symbol and argument types, which leads to the uniqueness of typing.

3.2 Semantics

To give meanings to the syntax, we assign the semantics in the denotational way, which is a mapping from syntax to set-theoretical objects.

Denotational semantics The denotational semantics interpret every expression as an object in linear algebra, according to a valuation mapping v from variables to their values. The semantics of e with valuation v is written as $[\![e]\!]_v$, and two expressions e_1 and e_2 are equivalence if they have equal semantics in the mathematical sense for all valuations, written as $[\![e_1]\!] = [\![e_2]\!]$.

The complete interpretation is listed in *[YX] : We need to write this*. Variables typed with Index are interpreted as finite sets, and index product $[\![\sigma_1 \times \sigma_2]\!] = [\![\sigma_1]\!] \times [\![\sigma_2]\!]$ are defined as Cartesian product of sets. Following on, every type is interpreted as a set. For example, the scalar type $[\![\mathcal{S}]\!] = \mathbb{C}$ is interpreted as complex number set, the the ket and bra type $[\![\mathcal{K}(\sigma)]\!] = \mathcal{H}_{[\![\sigma]\!]}$, $[\![\mathcal{K}(\sigma)]\!] = \mathcal{H}^*_{[\![\sigma]\!]}$ are interpreted as the Hilbert space and dual space depending on σ. One special case is the delta symbol $[\![\delta_{s,t}]\!] = \begin{cases} 1, \text{ where } [\![s]\!] = [\![t]\!], \\ 0, \text{ where } [\![s]\!] \neq [\![t]\!]. \end{cases}$.

The idea behind the interpretation of types and terms is to characterize the typing relation with set theory inclusion. Therefore we have the following lemma.

Lemma 2. *For all 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. Directly check all the cases.

This explanation formalizes the common explanation of Dirac notations, and best describes the target of the algorithm. However, computers cannot reason about equivalence through mathematical interpretations. We move on define the proof system as an abstraction.

Axiomatic semantics The proof system for equivalence consists of equational logic as the framework and axioms describing the properties of Dirac notations. The full list is in *[YX] : TODO*. There are axioms for linear space, and other structures like tensor product and inner product. For example, we have the absorption law of zero symbols $X \cdot \mathbf{0} = \mathbf{0}$, and the bilinearity of dot product

$$\begin{array}{rr} (a.X) \cdot Y = a.(X \cdot Y) & X \cdot (Y_1 + Y_2) = X \cdot Y_1 + X \cdot Y_2 \\ X \cdot (a.Y) = a.(X \cdot Y) & (X_1 + X_2) \cdot Y = X_1 \cdot Y + X_2 \cdot Y \end{array}$$

As is mentioned in the introduction, a subset E of the axioms is not decided by term rewriting. The set consists of:

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

The equational theories for scalars are treated as a moduloe, and are not considered here. In the implementation, we use the Mathematica kernel to decide scalar equivalences.

The equational axioms provide an operable theory for the proof automation algorithm, and the denotational semantics can be considered as one model for the theory. In this sense, the equivalence derived by axioms always imply the equivalence in interpretations.

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

Proof. Direct check all cases.

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

To conclude the design introduction, we demonstrate the formalization of symbols used in the motivating example.

Example 2 (formalizing the motivating example). Definitions and assumptions in the environment E:

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

The symbol TPO is the encoding of operator transpose, which is polymorphic on the Hilbert spaces by T_1 and T_2. phi takes the index T and defines the maximally entangled states. It sums over all basis in T, indicated by the universal set $\mathbf{U}(T)$. With the assumption of index T and operator M, we can write the equivalence in the non-label version, formally as

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

4 Algorithm for Deciding Dirac notation equations

We use the term rewriting technique to decide the equivalence of Dirac notation implied by its axiomatic semantics. Term rewriting decides equivalence by normalizing the two terms, and checking whether they have the same syntax. The normalization is powered by matching and substitution in universal algebra, using a set of rewriting rules. As is mentioned in the sematnics, the axioms in E cannot be decided by term rewriting. DiracDec decides E by examining all possible permutations in the rewriting result. In this work, the general idea is to carry through the normalization procedure using sorting algorithms. The procedure of the normalization is displayed below. The first three steps use term rewriting to work on the structure of Dirac notations, while the last three steps deals with the equational theory E.

1. **first rewritings**: expand definitions and simplify,
2. **variable expansion**: consider scalars expressions for completeness,
3. **second rewritings**: normalize modulo E,
4. **sorting without bound variables**: normalize AC-equivalence,
5. **swap successive summation**: normalize SUM-SWAP,
6. **de Bruijn normalization**: normalize α-equivalence.

4.1 Normalization modulo E by Term Rewriting

Term rewriting rules, written as $l \to r$, are used to normalize terms by recursively matching the subterms of the term with the left hand side l, and replace them with the corresponding right hand side r. The procedure terminates when no more rewritings can be made. The full list of rewriting rules are in Appendix C, and we present some of them to illustrate the design idea.

Firstly, we use the following flattening rule to decide the associative with AC symbols, because we allow functions with indefinite arities.

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

Commutativity is left for the sorting algorithm later on. Many rewriting rules are obtained by assigning directions to the equational axioms, for example:

(R-DOT6) $\langle s| \cdot |t\rangle \ \triangleright \delta_{s,t}$

(R-DELTA0) $\delta_{s,s} \ \triangleright\ 1$

(R-MULK1) $O : \mathcal{O}(\sigma, \tau) \Rightarrow O \cdot \mathbf{0}_{\mathcal{K}(\tau)} \ \triangleright \mathbf{0}_{\mathcal{K}(\sigma)}$

(R-MULK11) $(O_1 \otimes O_2) \cdot (K_1 \otimes K_2) \ \triangleright\ (O_1 \cdot K_1) \otimes (O_2 \cdot K_2)$

Some of them are obvious. The (R-DOT6) says that inner product of two basis will be evaluated to a delta expression, and (R-DELTA0) further transforms delta on identical basis to scalar 1. (R-MULK1) corresponds to the axiom that multiplication on zero vectors results in zero. It has a condition on typing, which is calculated during rewriting. Some rules require the intuition. For example, (R-MULK11) shows that we prefer tensor product than multiplication.

As a reference, the term rewriting system for DiracDec is proved complete for the axioms, except the sum symbol. The result is obtained by checking confluence and termination of the system. Our rewriting rules are translations from DiracDec into the typed and abstract language, therefore completeness for corresponding symbols is also expected to hold.

We have other rules dealing with big operator sum. For example,

(R-SUM-ELIM2) $i \text{ free in } t \Rightarrow \sum_{i \in \mathbf{U}(\sigma)} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} (\delta_{i,t}.A)$

$\triangleright \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} A\{i/t\}$

(R-SUM-PUSH5) $(\sum_{i \in M} B) \cdot K \ \triangleright\ \sum_{i \in M} (B \cdot K)$

The (R-SUM-ELIM2) rule will try to eliminate the delta expression in summations. And the (R-SUM-PUSH5) rule will lift sum to the outside of inner product. There are no completeness guarantee, but the rules work well in practice.

One technique revealed in the DiracDec work is the expansion of variables, critical for proofs with sum:

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

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

These rules transform variables to the symbolic summation of their decomposition on the basis. The rules are not terminating, therefore is applied recrusively once in the second step called *variable expansion.* On the other hand, we discovered that doing the expansion on all variables only once is already sufficient.

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

Proof. If we expand a ket variable twice, the structure will be reduced to

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

the elimination rule of delta symbol returns the term to the one-expansion result. Similar for bra and operator.

5 Deciding Equational Theory E

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

As the first improvement, we check the α-equivalence by de Bruijn index [1], where references to bound variable names are replaced by the distance of the lambda abstraction to the variable. For example, the nominal lambda abstraction $\lambda x.x$ is transformed into $\lambda.0$, while $\lambda x.\lambda y.(x\ (y\ x))$ will is transformed into $\lambda.\lambda.(1\ (0\ 1))$. Transformation into de Bruijn index is at the last step of the normalization to check the equivalence of terms with different bound variable names.

The remaining axioms of AC-equivalence and SUM-SWAP claims equivalence under permutations. A standard approach to decide is normalization by sorting in a given order. For example, given the dictionary order $a < b < c$, the term $b + c + a$ (and any other AC equivalent ones) will be normalized into $a + b + c$. For our setting, there are two difficulties intertwined together: how to assign such an order to all terms in our language, and how to sort for the two axioms simultaneously. As a typical example, the two terms

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

are equivalent. However, directly sorting elemnts of scalar multiplication in the lexical order will not lead to the same form.

We propose an algorithm to solve the problem by sorting without bound variables. The observation is that in a successive sum expression $\sum_{i \in s_1} \cdots \sum_{j \in s_n} A$, the name and order of bound variables $i, \cdots, j$ can be permuted freely. Therefore, all bound variables should be treated uniformly during sorting, and the order of sum can then be decided according to the position of their bound variables.

In the above example, we ignore the bound variables first, and sort the sum body into $\langle \bullet | A | \bullet \rangle \times \langle \bullet | B | \bullet \rangle$. Then we swap the sum, so that the bound variable at first $\bullet$ position appears at the outmost 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 notations: for a term $e = f(a_1, a_2, \cdots, a_n)$, head$(e)$ indicates the function symbol f, and arg(e, i) indicates the i-th argument a_i.

Definition 5 (order without bound variables). *Assume we have a total order of all symbols. Let $\mathcal{B}$ be the set of bound variables. For simplicity, we assume all bound variables are unique. The relation $e_1 =_{\mathcal{B}} e_2$ holds when*

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

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

- *$e_1 \notin \mathcal{B}$ 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 checked that $e_1 =_{\mathcal{B}} e_2$ if and only if neither $e_1 <_{\mathcal{B}} e_2$ or $e_2 <_{\mathcal{B}} e_1$ holds. The idea is to compare the function symbols in the top down order, and omit bound variable occurance. With the order, the sort function normalize the term with respect to 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 $<_{\mathcal{B}}$.*

The order of bound variables can be decided according to the result of the sort transformation, and then successive summations can be normalized using the order.

Definition 7 (swap transformation). *For a term e with sorting result sort(e), we can obtain a total order of all bound variables appearing in the term e. Specifically, they are ordered by their first appearances, except in function definitions λx and μx. The swap transformation then arrange successive summation according to this order.*

The algorithm succeeds in all equivalences of our benchmark, and we formalize its completeness as a conjecture.

Conjecture 1 (completeness of transformation). For any two terms e_1 and e_2 that is 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 equivalence 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 $e_1\downarrow = e_2\downarrow$, then $[\![e_1]\!] = [\![e_2]\!]$.*

Proof. The soundness of term rewriting holds because each rewriting rule preserves the equivalence. The operations in sort and swap transformation follows AC-equivalence and SUM-SWAP axioms. The de Bruijn normal form is sound for α-equivalence.

6 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 notations, and demonstrate how to transform the equivalence problem into that for the Dirac notations 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 notations. 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.*

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

$\mathcal{D}(s_1, s_2)$ is the unified type for all labelled Dirac notations, 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 notations. 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 notations, 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.

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

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 notations 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 notations 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 way, we only need to check the scalars to decide the equivalence of labelled Dirac notations.

7 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 notations and Dirace commands;
- `prover.cpp`: the prover that maintains the context and process commands like definition or equivalence checking.

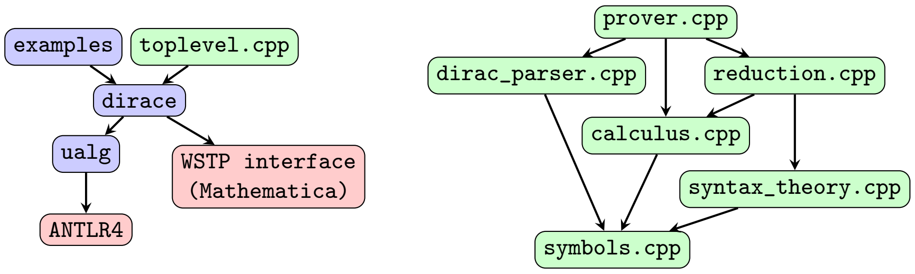

Fig. 1. The structure of Dirace. The left figure illustrates the whole system, and the right figure illustrates the `dirace` module. Blue nodes denote modules, red nodes denote third-party libraries, and green nodes denote files. Arrows represent dependency.

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 notations 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

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 notations
`e1^*`	`CONJ[e1]`	scalar conjugation
`fun i : T => X`	`FUN[i, T, X]`	lambda abstraction

syntacies for terms, and most Dirac notation terms will be encoded in the natural way. Here are some examples for the parsing syntax.

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 notations 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 6.

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.

type	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
Jens2024	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.*

8 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.

Term rewriting.

Other theories and tools are proposed for the formal verification of quantum computations. From the theoretical part, we have ... to verify zx calclus. Other tools for circuits, compilation, program.

9 Conclusion

Based on the first Dirac notation equational reasoning tool DiracDec, this work refines and extends the theory for practical application, and provides the solver Dirace. Experimental results shows that the tool can correctly identify all encoded equivalent expressions efficiently. We expect Dirace will have applications in areas like quantum program verification or proofs of post-quantum cryptography protocols.

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. 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.

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

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

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

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

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

– 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

– Scalar term typing rules.

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

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

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

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

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

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

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

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

– Ket term typing rules.

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

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

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

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

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

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

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

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

– Bra term typing rules.

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

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

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

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

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

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

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

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

– Operator term typing rules.

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

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

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

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

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

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

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

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

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

– 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.

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

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

– Typing rules for labelled Dirac notations.

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

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

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

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

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

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

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

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

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

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

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

B 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)	$\mathbf{1}_\mathcal{O}^\dagger = \mathbf{1}_\mathcal{O} \qquad \mathbf{1}_\mathcal{O} \cdot D = D \qquad \mathbf{1}_\mathcal{O} \otimes \mathbf{1}_\mathcal{O} = \mathbf{1}_\mathcal{O}$
	$\|t\rangle^\dagger = \langle t\| \qquad \|s\rangle \otimes \|t\rangle = \|(s,t)\rangle$

[YX] : to be continued

C 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	$((\lambda x : T.t)\ u) \ \triangleright\ t\{x/u\}$

Rule	Description
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	$a_1 + \cdots + (b_1 + \cdots + b_m) + \cdots + a_n$
	$\triangleright\, a_1 + \cdots + b_1 + \cdots + b_m + \cdots + a_n$
	$a_1 \times \cdots \times (b_1 \times \cdots \times b_m) \times \cdots \times a_n$
	$\triangleright\, a_1 \times \cdots \times b_1 \times \cdots \times b_m \times \cdots \times a_n$
	$X_1 + \cdots + (X'_1 + \cdots + X'_m) + \cdots + X_n$
	$\triangleright\, X_1 + \cdots + X'_1 + \cdots + X'_m + \cdots + X_n$

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	$\langle s\| \cdot \|t\rangle \ \triangleright\ \delta_{s,t}$
R-DOT7	$(B_1 \otimes B_2) \cdot \|(s,t)\rangle \ \triangleright\ (B_1 \cdot \|s\rangle) \times (B_2 \cdot \|t\rangle)$
R-DOT8	$\langle (s,t)\| \cdot (K_1 \otimes K_2) \ \triangleright\ (\langle s\| \cdot K_1) \times (\langle t\| \cdot K_2)$
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	$\langle (s,t)\| \cdot ((O_1 \otimes O_2) \cdot K) \ \triangleright\ ((\langle s\| \cdot O_1) \otimes (\langle t\| \cdot O_2)) \cdot K$
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	$\mathbf{1}_\mathcal{O}(\sigma)^\dagger \ \triangleright\ \mathbf{1}_\mathcal{O}(\sigma)$

Rule	Description
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	$\|s\rangle \otimes \|t\rangle \ \triangleright\ \|(s,t)\rangle$
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	$\langle s\| \otimes \langle t\| \ \triangleright\ \langle (s,t)\|$
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	$(O_1 \otimes O_2) \cdot \|(s,t)\rangle \ \triangleright\ (O_1 \cdot \|s\rangle) \otimes (O_2 \cdot \|t\rangle)$
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	$(O_1 \otimes O_2) \cdot (O'_1 \otimes O'_2) \ \triangleright\ (O_1 \cdot O'_1) \otimes (O_2 \cdot O'_2)$

Rule	Description
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	$\mathbf{1}_{\mathcal{O}}(\sigma) \ \triangleright\ \sum_{i \in \mathbf{U}(\sigma)} \vert i \rangle \cdot \langle i \vert$

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	$\sum_{i \in M} \sum_{j \in M} \sum_{k_1 \in s_1} \cdots \sum_{k_n \in s_n} (a_1 \times \cdots \times \delta_{i,j} \times \cdots \times a_n)$

Rule	Description
	$\rhd \sum_{j\in M}\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n}(a_1\{j/i\}\times\cdots\times a_n\{j/i\})$
R-SUM-ELIM6	$\sum_{i\in M}\sum_{j\in M}\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n}(\delta_{i,j}.A)$ $\rhd \sum_{j\in M}\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n}A\{j/i\}$
R-SUM-ELIM7	$\sum_{i\in M}\sum_{j\in M}\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n}(a_1\times\cdots\times\delta_{i,j}\times\cdots\times a_n).A$ $\rhd \sum_{j\in M}\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n}(a_1\{j/i\}\times\cdots\times a_n\{j/i\}).A\{j/i\}$
R-SUM-ELIM8	$\sum_{i\in M}\sum_{j\in M}\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n}((a_1\times\cdots\times\delta_{i,j}\times\cdots\times a_n)+\cdots+(b_1\times\cdots\times\delta_{i,j}\times\cdots\times b_n)).A$ $\rhd \sum_{j\in M}\sum_{k_1\in s_1}\cdots\sum_{k_n\in s_n}((a_1\{j/i\}\times\cdots\times a_n\{j/i\})+\cdots+(b_1\{j/i\}\times\cdots\times b_n\{j/i\})).A\{j/i\}$

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

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

Rule	Description
R-L-EXPAND	$K_R \ \triangleright\ \sum_{i_{r_1}\in\mathbf{U}(\sigma_{r_1})}\cdots\sum_{i_{r_n}\in\mathbf{U}(\sigma_{r_n})}((\langle i_R\rvert\cdot K).(\lvert i_{r_1}\rangle_{r_i}\otimes\cdots\otimes\lvert i_{r_n}\rangle_{r_n})$

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

Table 20: Rules for labelled Dirac notations.

Rule	Description
R-ADJDK	$({}_r\langle i\|)^\dagger \triangleright \|i\rangle_r$
R-ADJDB	$(\|i\rangle_r)^\dagger \triangleright {}_r\langle i\|$
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	$(D_1 + \cdots + D_n) \cdot D \triangleright D_1 \cdot D + \cdots + D_n \cdot D$
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 notations.

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	$(X_1 \otimes \cdots \otimes {}_r\langle i\| \otimes \cdots \otimes X_n) \cdot (Y_1 \otimes \cdots \otimes \|j\rangle_r \otimes \cdots \otimes Y_m)$
	$\triangleright \delta_{i,j}.(X_1 \otimes \cdots \otimes X_n) \cdot (Y_1 \otimes \cdots \otimes Y_m)$