Universiteit Leiden

Quantum Computation using Weighted Model Counting

With Applications in Physics

Master Computer Science

Name:	Dirck van den Ende
Student ID:	2541041
Date:	24/07/2025
Specialisation:	Foundations of Computing
1st supervisor:	Dr. Alfons Laarman
2nd supervisor:	Dr. Joon Hyung Lee
3rd supervisor:	Dr. Henning Basold

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Einsteinweg 55
2333 CC Leiden
The Netherlands

Abstract

Quantum physics and, more specifically, quantum computing have the potential to solve a myriad of problems in drug development, traffic optimization, artificial intelligence, and more. However, it is very difficult to solve these problems on a classical computer, and quantum computers are not yet equipped to solve real-world problems. As such, other techniques for solving quantum problems on classical computers have been developed recently. One of these techniques is weighted model counting (WMC), which has proven effective at a range of tasks within computer science, physics, and beyond. However, existing approaches for using WMC in physics only target specific problems, lacking a general framework for expressing problems using WMC. This limits the reusability of these approaches in other applications and means these techniques often lack mathematical rigor. We present an approach for expressing linear algebraic problems, specifically those present in physics and quantum computing, as WMC instances. We provide mathematical rigor by proving the correctness of this approach. We do this by introducing a framework that converts Dirac notation to WMC problems. We build up this framework theoretically, using a type system and denotational semantics, and provide an implementation in Python. We demonstrate the effectiveness of our framework in calculating the partition functions of several physical models: The transverse-field Ising model and the Potts model. As the performance of model counters improves in the future, our framework will provide a bridge between weighted model counting and many real-world problems.

Contents

Notation

Below is a table with the notation used in this thesis, along with the section where the notation is introduced.

Notation	Intr.	Meaning
$\mathbb{B}$	2.1	The set of binary values $\{0, 1\}$.
$\phi[\tau]$	2.1	A Boolean formula ϕ over a set of variables V evaluated for an assignment $\tau : V \to \mathbb{B}$.
$\mathbb{1}\{c\}$	2.1	Indicator function returning 1 if the condition c is true, and 0 otherwise.
$\overline{v}$	2.1	The negation of a Boolean variable v.
$\phi \equiv \psi$	2.1	Logical equivalence of Boolean formulae ϕ and ψ.
$\mathbb{F}$	2.2	Some arbitrary field, which is assumed to be the same field throughout this work.
$\mathrm{WMC}(\phi, W)$	2.2	Weighted model count of ϕ with respect to W.
$\top$	2.2	"Always true" Boolean formula. Often referred to as "top".
$\bot$	2.2	"Always false" Boolean formula. Often referred to as "bottom".
$\langle i \vert_q$, $\langle i \vert$	2.3.1	A row vector of width q with a 1 at the i-th position counting from 0, and 0 everywhere else. The q is left out if it is clear from context.
$\vert i \rangle_q$, $\vert i \rangle$	2.3.1	A column vector of height q with a 1 at the i-th position counting from 0, and 0 everywhere else. The q is left out if it is clear from context.
$A \otimes B$	2.3.2	The Kronecker product of two matrices.
$\mathrm{tr}(M)$	2.3.3	The trace of a matrix.
X, Y, Z	2.3.4	Pauli X, Y, and Z matrices.
H	2.3.4	Hadamard quantum gate/matrix.

Notation	Intr.	Meaning
e^M	2.3.5	Matrix exponential $\sum_{k=0}^{\infty} M^k/k!$.
$\mathcal{S}$	3.1	Scalar type in the language from Chapter 3.
$\mathcal{M}(q, m \to n)$	3.1	Type of a $q^n \times q^m$ matrix in the language from Chapter 3.
$\mathsf{tr}(M)$	3.1	Matrix trace expression in the language from Chapter 3.
$\mathsf{entry}(i, j, M)$	3.1	Expression for the matrix entry at row i and column j, in the language from Chapter 3.
$\mathsf{apply}(f, s)$	3.1	Expression for the application of a field endomorphism on a scalar in the language from Chapter 3.
$\mathsf{bra}(q, i)$	3.1	Expression in the language from Chapter 3 for the length-q row matrix $\langle i|_q$.
$\mathsf{ket}(q, i)$	3.1	Expression in the language from Chapter 3 for the length-q column matrix $|i\rangle_q$.
$\mathsf{trans}(M)$	3.1	Expression for the transpose of a matrix in the language from Chapter 3.
$\mathsf{apply}(f, M)$	3.1	Expression for the entry-wise application of a field endomorphism on a matrix.
$\vdash e : T$	3.2	Expression e has type T.
$[\![e]\!]_v$	3.3	Value denotational semantics of an expression e.
$\mathrm{rep}(\phi, W)$	3.4.1	Value that the tuple (ϕ, W) represents, which is equal to $\mathrm{WMC}(\phi, W)$.
$\mathrm{var}(v)$	3.4.2, A	Set of Boolean variables used in the variable representation v.
$v = n$	3.4.2, A	Formula for equality of a variable encoding v to a value n
val_v	3.4.2, A	Validity formula of a variable encoding v.
$v \leftrightarrow w$	3.4.2, A	Equality formula of two variable encodings v and w.
$\mathrm{rep}(\phi, W, x, y, q)$	3.4.2	Matrix that the tuple (ϕ, W, x, y, q) represents.
Rep	3.4.3	Set of scalar and matrix representations.
$\mathrm{Mat}(\mathbb{F})$, $\mathrm{Mat}(\mathbb{F}, n \times m)$	3.4.3	Set of matrices over a field $\mathbb{F}$, optionally with the given shape.

Notation	Intr.	Meaning
$[r]$	3.4.4	Equivalence class of a representation r under the relation of equal outputs when applying the function rep.
$[\![e]\!]_r$	3.5	Representation denotational semantics of an expression e.
$f_1 \cup f_2$	3.5	Union of two functions with disjoint domains.
$f_1 \cdot f_2$	3.5	Multiplication of functions on where their domains overlap, and the value of one of the functions elsewhere.
Exp	3.6	Set of expressions that have a type.
Λ	4.1, 5.1, 6.1	Set of sites in an Ising model, transverse-field Ising model, or Potts model.
J_{ij}	4.1, 5.1	Interaction strength between sites in an Ising model, transverse-field Ising model.
h_i	4.1	External field strnegth at site i in an Ising model.
σ	4.1	Configuration of spins of an Ising model.
$H_I(\sigma)$	4.1	Hamiltonian of an Ising model with spin configuration σ.
$Z_{\beta,I}$	4.1	Partition function of an Ising model at inverse temperature β.
μ_x, μ_z	5.1	Transverse-field Ising model external field strengths in X and Z directions.
H_Q	5.1	Transverse-field Ising model Hamiltonian matrix.
$Z_{\beta,Q}$	5.1	Partition function of a transverse-field Ising model at inverse temperature β.
$J_{ij}(s_i, s_j)$	6.1	Interaction strength between sites i and j in a generalized Potts model, given their states s_i and s_j.
$h_i(s_i)$	6.1	External field strength at site i in a generalized Potts model, given its state s_i.
s	6.1	Configuration of spins of a Potts model.
$H_P(s)$	6.1	Hamiltonian of a Potts model with spin configuration s.
$Z_{\beta,P}$	6.1	Partition function of a Potts model P at inverse temperature β.

Chapter 1

Introduction

Quantum physics is the theory of the very small, describing the behaviour and interaction of fundamental particles, atoms, and molecules. Quantum physics, and more specifically quantum computing, have the potential to solve a myriad of problems in drug development, traffic optimization, artificial intelligence, and more [25, 38, 47]. Despite this potential, however, it is very difficult to solve these problems with a normal computer, as the space of solutions to check grows exponentially with the problem size. Quantum computers aim to provide a solution to this. However, current quantum computers are not yet powerful enough to deal with these practical problems. As such, other techniques for solving quantum problems on classical computers have been developed recently.

An example of a problem with this exponential blowup is calculating the partition function. This is a physical quantity of a system that can be seen as a measure of the "total energy" in that system. The quantity is generally expressed as an exponentially-sized sum over possible states a system can be in. Using brute-force on even moderately sized partition function problems will yield extremely long runtimes, which means there is a need for tools to speed up this process.

One such promising technique is weighted model counting (WMC), an approach that involves powerful solvers on classical computers. This will investigate the use of weighted model counters to calculate the partition functions of various systems. We will also provide a general framework for solving quantum physics problems using weighted model counting.

1.1. The potential of weighted model counting

Weighted model counting is, at its core, a very basic problem statement: Given some constraint problem and some weights, calculate the weighted sum of all solutions to the constraint problem [4]. We can express this formally as having a Boolean

formula ϕ and a weight function W, where the weighted model count is defined as

$$\mathrm{WMC}(\phi, W) = \sum_{\tau \text{ solves } \phi} W(\tau) \tag{1.1}$$

Generally, the set of solutions τ grows exponentially with the size of ϕ. As such, the problem is #P-hard in general [17], which is a complexity class that can be seen as the "counting equivalent" of NP. Simply put, these problems take a lot of time to solve as the problem size increases.

Despite this complexity, modern model counting tools can achieve an exponential speedup on many real-world problems formulated with WMC. These model counters use advanced techniques such as clause learning, tensor networks, and algebraic decision diagrams to solve problems [14, 15, 16, 40, 43]. This has allowed researchers to apply weighted model counting to a variety of problems and tasks in recent decades, such as in probabilistic inference, statistical physics, and critical infrastructure reliability [8, 12, 28, 30, 34].

1.2. State of the art

More recently, the application of WMC to problems in quantum physics has been explored. Mei et al. [26, 27, 28] showed that WMC can be used for the simulation and analysis of quantum circuits on classical computers. This is done by encoding the gates in a quantum circuit as Boolean formulae. Although quantum simulation at its core involves complex numbers, Mei et al. [28] also showed that weighted model counting problems with complex weights can be reduced to those with real numbers, or even Boolean values $\mathbb{B} = \{0, 1\}$. This allows the simulation to be performed using most existing model counters.

Another area where model counters have shown strength recently is in statistical physics. Nagy et al. [30] showed that the calculation of the partition function of the Ising model, which is a classical model of interactions between particles, can be transformed to a WMC problem. Like WMC, the calculation of this partition function is #P-hard in general, which makes it such a good candidate for this approach. They showed that the TensorOrder [16] model counter, which works with tensor networks, outperforms existing tools like CATN [33] from physics in computing the partition function.

These applications already show the potential that weighted model counting has. However, they lack a more general method for encoding problems from (quantum) physics using weighted model counting.

1.3. Problem statement

Existing approaches for using WMC only target specific problems, lacking a more general framework for applications in physics and beyond. This specifically intro-

duces the following limitations:

- Existing encodings for quantum systems only support operations on qubits, and not more general qudits (qubits with more than two dimensions, see [32]).
- There is no general framework for applying common linear-algebraic operations like matrix multiplication, Kronecker products, or taking the trace of a matrix. These operations often need to be performed when dealing with quantum circuits or quantum systems in general. It is therefore useful to be able to convert a problem written in Dirac notation (a common way of writing linear-algebraic problems in physics) to a weighted model counting problem. Dirac notation has already shown its use in tools like DiracDec [49] and D-Hammer [50], although these tools solve different tasks.
- Existing methods either lack mathematical rigour or involve long proofs. Again, this is mainly caused by the lack of a general framework.
- Weighted model counting has not been applied to physical systems other than the (classical) Ising model.

In this thesis, we propose a general framework for applying WMC to many problems in (quantum) physics. We demonstrate the effectiveness of this framework by using it to calculate the partition functions of several physical models.

1.4. Research questions

In this thesis, we address the following two main research questions:

1. Encoding problems from physics using WMC is hard and currently requires much human insight into the problem at hand. Many of these problems could be formulated more easily when using matrices. This could, for example, be done using Dirac notation. To apply WMC to a wider range of problems from physics, is it possible to encode Dirac notation using weighted model counting, using a general framework?
2. The scope of problems to which WMC has been applied in physics is currently very narrow. In quantum physics specifically, it is limited to the simulation and analysis of circuits. Can we calculate the partition functions of the transverse-field Ising model, which is a quantum model, and the Potts model, which is classical?

1.5. Contributions

We provide three main contributions in this thesis: A theoretical framework for encoding matrices using WMC, an implementation of this framework, and several applications of this framework on problems from physics:

1. We provide a generic encoding of $q^n \times q^m$ matrices using WMC. We provide a method for applying operations such as matrix multiplication, addition, and taking the trace to these encodings. From this, we build a language that can encode Dirac notation using WMC. We provide a formal proof of the correctness of our framework in the appendix.
2. An open-source implementation of this framework in Python is available at [13]. This implementation was used to perform the experiments in Chapters 4, 5, 6.
3. Applications of our framework to the calculation of the partition functions of the (quantum) transverse-field Ising and (classical) Potts models. To our knowledge, this is the first time weighted model counting has been applied to these problems.

Figure 1.1 illustrates the general approach to solving physics problems using our framework. The green boxes indicate choices between physics problems and model counters. In both cases, other options also exist. The user is responsible for the first step of converting the physics problem to a matrix problem. All other steps are automated by the framework.

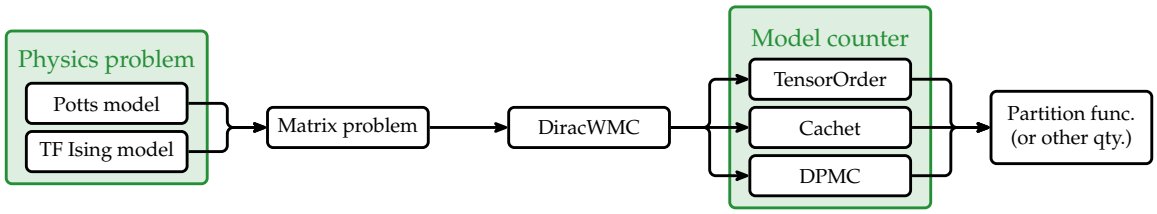


Figure 1.1: Workflow of solving physics problems using our framework.

1.6. Overview

In Chapter 2, we introduce some preliminaries used throughout the thesis. We give some basic definitions from Boolean logic, and build on this to define weighted model counting formally. Furthermore, we introduce some concepts from quantum computing, such as Dirac notation.

Chapter 3 then builds on the definition of weighted model counting to define the matrix encodings used in our framework, which we call "matrix representations". We introduce a function, rep, that maps matrix representations to matrices. We define a formal language and type system with the supported matrix operations (matrix multiplication, addition, taking the trace, etc.) that is similar to DiracDec [49] and D-Hammer [50]. Then, we give procedures for implementing these operations on matrix representations by defining semantics $[\![\cdot]\!]_r$. These semantics map expressions to the matrix representations. At the end of this chapter, we give a theorem showing

the correctness of these operations by showing rep $\circ [\![\cdot]\!]_r$ returns the matrix of the expression.

Chapters 4, 5, and 6 outline several applications of the matrix representations. Chapter 4 discusses the WMC encoding of the Ising model partition function presented by Nagy et al. [30]. The Ising model is a classical model from physics that models interactions between particles. We show that the matrix representations from Chapter 3 can also be used to calculate the partition function. It turns out that these two ways of converting the calculation to WMC result in the same Boolean formula and weight function. The experimental results of Nagy et al. are replicated to show that our framework gives the same results.

Chapter 5 introduces the quantum version of the Ising model: The transverse-field Ising model. We show that, using a technique called Trotterization, the partition function of this model can be approximated using matrix representations. Some experimental results are discussed for small systems.

We introduce a classical generalization of the Ising model, called the Potts model, in Chapter 6. We encode the Potts model partition function and show that the performance of model counters on this problem is comparable to that of the Ising model.

Chapter 2

Preliminaries

In this chapter, we give some preliminaries used throughout this thesis. We introduce basic concepts and notation from Boolean logic, then build on this to define weighted model counting. Lastly, we introduce some concepts from quantum computing.

2.1. Boolean logic

Before introducing model counting problems, we need to establish some basic definitions and notation related to Boolean logic. First, we introduce some notation for the satisfiability of a Boolean formula.

Notation 2.1: Boolean satisfiability

Let ϕ be some Boolean formula over a set of variables V and $\tau : V \to \mathbb{B}$ a function assigning binary values to variables. We write $\phi[\tau] = 1$ if τ **satisfies** ϕ, and $\phi[\tau] = 0$. We will write $\text{sat}(\phi) = 1$ if ϕ is **satisfiable** and $\text{sat}(\phi) = 0$ otherwise, i.e.

$$\text{sat}(\phi) = \mathbb{1}\{\phi \text{ is satisfiable}\} = \max_{\tau : V \to \mathbb{B}} \phi[\tau], \tag{2.1}$$

where $\mathbb{1}$ is an indicator function equal to 1 if ϕ is satisfiable and 0 otherwise.

Model counters (particularly the ones used in this thesis) often require their input to be in conjunctive normal form, which is a conjunction (logical AND) of disjunctions (logical OR) of literals (either a variable or its negation). Although this seems like a major restriction, many problems can be expressed as or rewritten to this form without much effort.

Definition 2.2: Conjunctive normal form

A Boolean formula ϕ is in **conjunctive normal form (CNF)** if it is of the form

$$\phi \equiv \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} x_{ij}, \tag{2.2}$$

where each x_{ij} is a literal (so either v or $\overline{v}$ for a variable v). We use $\phi \equiv \psi$ to denote logical equivalence between two formulae ϕ and ψ.

2.2. Weighted model counting

In this section, we introduce the concept of weighted model counting, in which the problem is to find the weighted sum of all solutions to a Boolean formula. It is a generalization of #SAT, which asks for the number of solutions to a Boolean formula, without weights. This is itself a generalization of SAT, which asks only *if* there is a solution, not how many solutions there are.

Like #SAT problems, weighted model counting is #P-hard in general [17]. However, due to extensive research in recent decades, model counters like Cachet [40], Ganak [43], DPMC [14], TensorOrder [16], and ProCount [15] can solve these problems in a relatively short amount of time. These tools make use of a variety of techniques and concepts, like Tensor networks (TensorOrder), clause learning (Cachet), and algebraic decision diagrams (DPMC), to simplify the problem or otherwise speed up the calculation of the total weight. In many practical applications, this can lead to an exponential speedup compared to brute force. Model counters have proven effective at probabilistic inference, statistical physics, critical infrastructure reliability, and more [8, 12, 28, 30, 34].

In this work, we specifically focus on weighted model counters that have two inputs. First, they accept a Boolean formula in conjunctive normal form (CNF). Second, they take a weight function, mapping literals to weights. Some model counters, like Cachet and TensorOrder, require these weights to be positive. We can still use these solvers to compute the partition functions of the Ising model and Potts model, which are problems we discuss in Chapters 4 and 6. Other problems, like the partition function of the quantum Ising model from Chapter 5, require some of the weights to be negative.

Although the problem using any real weights can be transformed into one using only positive weights, this transformation introduces an XOR between all of the variables with negative weights, which often hurts the runtime of the solver (although solutions to this do exist [2, 44, 51]). Therefore, in this work, we only use solvers that can compute the weight model counting problem at hand directly, without transformations. Similarly, WMC instances with complex weights can also be reduced to instances with real weights [28].

2.2.1. Definition

As mentioned, weighted model counting problems consist of two parts: A Boolean formula over some set of variables V, usually in CNF, and a weight function assigning a weight to every literal from V. Generally, these weights are real numbers when given to a model counter. However, we will work with some general field $\mathbb{F}$ of weights instead, since this generalization does not have any impact on the theory. This could be of particular use when working with problems in quantum computing, where values can be complex numbers.

The weighted model count calculates the sum of weights over all satisfying assignments $\tau : V \to \mathbb{B}$ of the Boolean formula. The weight of an individual assignment is the product of the weights of the variables with their assigned values.

Definition 2.3: Weighted model counting

Let ϕ be a Boolean formula over some finite set of variables V. Let $W : V \times \mathbb{B} \to \mathbb{F}$ be a function, called the **weight function**. The **weighted model count** of ϕ with respect to W is defined as

$$\mathrm{WMC}(\phi, W) = \sum_{\tau : V \to \mathbb{B}} \phi[\tau] \cdot \prod_{v \in V} W(v, \tau(v)) \tag{2.3}$$

This is also often denoted as $W(\phi)$ in literature.

Note that the set of variables V can contain more variables than are in the formula ϕ. However, all variables in ϕ have to be in V. For convenience, we will introduce the following notation related to the weight function:

Notation 2.4

- Denote $\mathrm{dom}(W)$ as the domain V of the weight function W.
- For variables $v \in V$, write $W(v) = W(v, 1)$ and $W(\overline{v}) = W(v, 0)$.

Weighted model counting problems can be particularly useful to calculate probabilities. A Boolean variable can be interpreted as a random variable that only depends on other variables through the formula ϕ. The probabilities of the random variable being true or false are given by the weight function. We demonstrate this in the following example.

Example 2.5

Suppose we have two coins and a die. We would like to know the probability that either of the coins comes up heads, and that we roll a six with the die. We introduce three variables c_1, c_2, and d. The first two represent the first and second coin coming up heads, respectively. The variable d represents the die

rolling a six.

Since the probabilities of heads and tails are equal, we assign the following weights to c_1 and c_2:

$$\begin{aligned} W(\bar{c}_1) &= 1/2, \text{ (coin 1 tails)} & W(\bar{c}_2) &= 1/2 \text{ (coin 2 tails)} \\ W(c_1) &= 1/2, \text{ (coin 1 heads)} & W(c_2) &= 1/2 \text{ (coin 2 heads)} \end{aligned} \tag{2.4}$$

The probability of rolling a six is $1/6$, hence

$$W(\bar{d}) = 5/6, \qquad W(d) = 1/6. \tag{2.5}$$

The event of rolling a six and at least one heads can be expressed as $d \wedge (c_1 \vee c_2)$. To find the probability of this occurring, we sum over all combinations of heads and tails and rolling a six or not rolling a six, such that the formula holds. This is exactly what happens in the weighted model counting definition. Hence, we can calculate the probability of at least one heads and rolling a six as

$$\begin{aligned} &\text{WMC}(d \wedge (c_1 \vee c_2), W) && (2.6)\\ &= W(d)W(c_1)W(\bar{c}_2) + W(d)W(\bar{c}_1)W(c_2) + W(d)W(c_1)W(c_2) && (2.7)\\ &= 1/24 + 1/24 + 1/24 && (2.8)\\ &= 1/8 && (2.9) \end{aligned}$$

Next, we illustrate that the domain of the weight function can contain variables that are not in ϕ using another example.

Example 2.6

Consider $\phi \equiv \top$ ("always true" or "top") and $W : \{x, y\} \times \mathbb{B} \to \mathbb{F}$ constant 2. Then we have

$$\begin{aligned} \text{WMC}(\phi, W) &= W(\bar{x})W(\bar{y}) + W(\bar{x})W(y) + W(x)W(\bar{y}) + W(x)W(y) && (2.10)\\ &= 2 \cdot 2 + 2 \cdot 2 + 2 \cdot 2 + 2 \cdot 2 = 16. && (2.11) \end{aligned}$$

Example 2.7

Changing the formula from Example 2.6 to $\phi \equiv \bot$ ("always false" or "bottom"), we get $\text{WMC}(\phi, W) = 0$.

Example 2.8

If we instead change the weight function from Example 2.6 by setting $W(x) = W(\bar{x}) = 0$, we also get $\text{WMC}(\phi, W) = 0$, since there will be a 0 in every

product in the sum from the definition of WMC.

2.3. Quantum computing

We introduce some basic concepts and notation used in quantum computing, which is mostly performed using operations on large ($2^n \times 2^n$) matrices. We will use some of the notation used in quantum computing in the more general context of $q^n \times q^m$ matrices.

2.3.1. Dirac notation

The most basic building blocks are bras and kets, denoted as $\langle i|_q$ and $|i\rangle_q$ respectively. This is called Dirac notation. A bra is a size-q row vector filled with zeros, except at the position i (counted starting from 0), where it has a 1. A ket is defined similarly, but as a column matrix. If the size of the vector is clear from context, the q will often be left out. It is common to compute the scalar $\langle j| M |i\rangle$ for some matrix M. This gives the entry of the matrix at the i-th row and j-th column, written as M_{ij}.

Example 2.9

Below are some examples of operations with bras and kets.

$$\langle 3|_4 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

$$|0\rangle_3 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{2.13}$$

$$\langle 0| \begin{bmatrix} 3 & 4 & 6 \\ 1 & 2 & 0 \end{bmatrix} |2\rangle = 6 \tag{2.14}$$

2.3.2. Kronecker product

The Kronecker product of two matrices is often used in quantum computing to perform a parallel operation on two or more separate subspaces. Written as $A \otimes B$, the Kronecker product is defined as

$$A \otimes B = \begin{bmatrix} A_{0,0}B & \dots & A_{0,r-1}B \\ \vdots & \ddots & \vdots \\ A_{c-1,0}B & \dots & A_{c-1,r-1}B \end{bmatrix} \tag{2.15}$$

where A is an $r \times c$ matrix, and $A_{ij}B$ is the scalar multiplication of the matrix B with A_{ij}. These matrices are filled into the matrix $A \otimes B$.

One reason the Kronecker product is often used in quantum computation is that multiplication can be distributed: $(A_1 \otimes B_1)(A_2 \otimes B_2) = A_1A_2 \otimes B_1B_2$. For this reason, the Kronecker product can describe the parallel application of matrices to different qubits.

Example 2.10

Suppose we have the following matrices:

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \tag{2.16}$$

The Kronecker product of these matrices is

$$A \otimes B = \begin{bmatrix} 2\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} & 1\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \\ 1\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} & 0\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 & 2 & 0 & 1 \\ 4 & 6 & 2 & 3 \\ 0 & 1 & 0 & 0 \\ 2 & 3 & 0 & 0 \end{bmatrix} \tag{2.17}$$

2.3.3. Matrix trace

The trace of a square matrix is the sum of the entries on the diagonal. We write the trace of a matrix as $\mathrm{tr}(M)$. A property of the trace is that $\mathrm{tr}(A \otimes B) = \mathrm{tr}(A) \cdot \mathrm{tr}(B)$.

Example 2.11

The trace of the matrix

$$A = \begin{bmatrix} 2 & 4 \\ 1 & -7 \end{bmatrix} \tag{2.18}$$

is equal to $\mathrm{tr}(A) = -5$.

2.3.4. Pauli and Hadamard operators

Common operations performed in quantum circuits are the Pauli X, Y, and Z operators and the Hadamard H gate, each of which is performed on a single qubit

in quantum computing.

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.19}$$

Each of these matrices is involutive, meaning it is its own inverse. One useful property used when converting the transverse-field Ising model to WMC in Chapter 5 is that $X = HZH$.

2.3.5. Matrix exponential

The exponential of a square matrix is defined similarly to the Taylor expansion of an exponential of a real number:

$$e^M = \exp(M) = \sum_{k=0}^{\infty} \frac{M^k}{k!} \tag{2.20}$$

For a diagonal matrix, this is equal to the matrix with the exponential of each of the entries on the diagonal, and zeros everywhere else. For large non-diagonal matrices, more advanced techniques exist to calculate the matrix exponential relatively quickly [29].

Example 2.12

The matrix exponential of the diagonal matrix

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \log(2) \end{bmatrix} \tag{2.21}$$

is equal to

$$e^A = \begin{bmatrix} e & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \tag{2.22}$$

Chapter 3

Encoding Dirac notation using WMC

We introduce weighted model counting representations of quantum operators. The concept of encoding quantum problems using WMC has been explored before [26, 27, 28], though these works lack a general framework for converting problems from physics to WMC instances. We aim to create a much more general method for encoding matrices, by allowing encodings of any $q^n \times q^m$ matrix, instead of just $2^n \times 2^n$ square matrices and row/column vectors like in previous work. We call this q the base size of the matrix. The framework we present is more generic in that it gives a general method for encoding matrices, as opposed to previous work, which focused on specific applications such as quantum circuits.

We represent these matrices as tuples (ϕ, W, x, y, q) of a Boolean formula ϕ, weight function W, input and output variables x and y, and a base size q. The formula and weight function form the basis of model counting instances, used for every entry in the matrix. The input/output variables act as pointers to the specific entries in the matrix, which are obtained by adding restrictions to the values of these variables to the formula ϕ. Scalars are represented by WMC instances (ϕ, W), where the value of the scalar is $\mathrm{WMC}(\phi, W)$.

Several common matrix operations are introduced that can be performed on these representations directly, such as matrix multiplication, taking the trace, and computing the Kronecker product. To formalize and prove the correctness of these and other operations, we first introduce a language of scalars and matrices, built from scalar constants, bras, and kets. This language is similar to D-Hammer, introduced by Xu et al. [50], as it builds on Dirac notation. However, the language we introduce in this work is less extensive and neither supports contexts nor labeled matrices. We provide an implementation of our approach at [13], which is further discussed in Section 3.7.

We introduce two kinds of denotational semantics on this language: $[\![\cdot]\!]_v$ returns the actual matrix or scalar that an expression represents, while $[\![\cdot]\!]_r$ returns a class of (matrix or scalar) representations that corresponds with the matrix or scalar.

3.1. Language Syntax

We introduce a language that is loosely based on D-Hammer [50]. It has two types of expressions: scalars and matrices. Scalars from a field $\mathbb{F}$ are of type $\mathcal{S}$. Matrices have a type that contains the size of the matrix and its base size. A base size of $q \in \mathbb{Z}_{\geq 2}$ is used to represent $q^n \times q^m$ matrices. The intuition of this number is that it represents the dimension of the smallest vector space that all of our matrices act on. For a system of qubits, for example, a base size of $q = 2$ would be used, since elementary operations on qubits are performed using 2×2 unitary matrices. Any unitary acting on multiple qubits has dimensions that are a power of two.

We write the type of a matrix as $\mathcal{M}(q, m \to n)$, representing a $q^n \times q^m$ matrix, with $n, m \in \mathbb{Z}_{\geq 0}$. Also note the reversal of the order of n and m. We use this notation because a $q^n \times q^m$ matrix (q^n rows and q^m columns) is generally interpreted as a linear map $\mathbb{F}^{q^m} \to \mathbb{F}^{q^n}$.

More formally, for $n, m \in \mathbb{Z}_{\geq 0}$ and $q \in \mathbb{Z}_{\geq 2}$ the type syntax is

$$T ::= \mathcal{S} \mid \mathcal{M}(q, m \to n) \tag{3.1}$$

The syntax of expressions e is split up into scalars s and matrices M.

$$e ::= s \mid M \tag{3.2}$$
$$s ::= \alpha \mid s_1 \cdot s_2 \mid s_1 + s_2 \mid \mathsf{tr}(M) \mid \mathsf{entry}(i, j, M) \mid \mathsf{apply}(f, s) \tag{3.3}$$
$$M ::= \mathsf{bra}(i, q) \mid \mathsf{ket}(i, q) \mid M_2 \cdot M_1 \mid M_1 + M_2 \mid M_1 \otimes M_2 \tag{3.4}$$
$$\mid s \cdot M \mid \mathsf{trans}(M) \mid \mathsf{apply}(f, M) \tag{3.5}$$

Here $\alpha \in \mathbb{F}$ is an arbitrary constant and $f : \mathbb{F} \to \mathbb{F}$ is an arbitrary field endomorphism. Field endomorphisms, by definition, have the properties $f(xy) = f(x)f(y)$ and $f(x + y) = f(x) + f(y)$. The complex conjugate is a notable example of a field endomorphism.

Scalar expressions can be combined using multiplication and addition. Applying a field endomorphism to a scalar also results in another scalar. In addition, taking the trace or getting a specific entry from a matrix gives a scalar.

The most basic matrices are the bra and ket (see Section 2.3.1). From these, operations can be performed, such as adding or multiplying matrices. In addition, we have syntax for taking the Kronecker product, matrix-scalar multiplication, and taking the transpose of a matrix. We also add support for applying a field endomorphism f to every entry of the matrix.

3.2. Type system

The type system associates expressions with types. We say that the expression e has type T if $\vdash e : T$ can be proven using the type rules below.

3.2.1. Scalar type rules

The usual rules for scalars apply: Multiplying or adding two scalars results in a scalar, and applying a field endomorphism to a scalar yields a scalar as well. In addition, any element of $\mathbb{F}$ is a scalar.

$$\frac{\alpha \in \mathbb{F}}{\vdash \alpha : \mathcal{S}}\ (\text{Const}) \qquad \frac{\vdash s_1 : \mathcal{S} \qquad \vdash s_2 : \mathcal{S}}{\vdash s_1 \cdot s_2 : \mathcal{S}}\ (\text{Mul}) \qquad \frac{\vdash s_1 : \mathcal{S} \qquad \vdash s_2 : \mathcal{S}}{\vdash s_1 + s_2 : \mathcal{S}}\ (\text{Add}) \tag{3.6}$$

$$\frac{\vdash s : \mathcal{S} \qquad f : \mathbb{F} \to \mathbb{F} \text{ is a field endomorphism}}{\vdash \mathsf{apply}(f, s) : \mathcal{S}}\ (\text{Apply}) \tag{3.7}$$

Getting an entry from a matrix or calculating the trace of a square matrix also results in a scalar:

$$\frac{\vdash M : \mathcal{M}(q, m \to n)}{\vdash \mathsf{entry}(i, j, M) : \mathcal{S}}\ (\text{Entry}) \qquad \frac{\vdash M : \mathcal{M}(q, n \to n)}{\vdash \mathsf{tr}(M) : \mathcal{S}}\ (\text{Trace}) \tag{3.8}$$

where $0 \leq i < q^n$ and $0 \leq j < q^m$.

3.2.2. Matrix type rules

The bra and ket form the basis for matrix expressions. These have the types $\mathcal{M}(q, 1 \to 0)$ and $\mathcal{M}(0 \to 1)$ respectively, as they can be interpreted as linear maps $\mathbb{F}^q \to \mathbb{F}$ and $\mathbb{F} \to \mathbb{F}^q$. For $0 \leq i < q$ we have

$$\frac{}{\vdash \mathsf{bra}(i, q) : \mathcal{M}(q, 1 \to 0)}\ (\text{Bra}) \qquad \frac{}{\vdash \mathsf{ket}(i, q) : \mathcal{M}(q, 0 \to 1)}\ (\text{Ket}) \tag{3.9}$$

Matrix multiplication is essentially the composition of maps $\mathbb{F}^{q^m} \to \mathbb{F}^{q^k}$ and $\mathbb{F}^{q^k} \to F^{q^n}$ to one map $\mathbb{F}^{q^m} \to \mathbb{F}^{q^n}$. However, do note that the composition is read from right to left. Hence M_2 and

$$\frac{\vdash M_1 : \mathcal{M}(q, m \to k) \qquad \vdash M_2 : \mathcal{M}(q, k \to n)}{\vdash M_2 \cdot M_1 : \mathcal{M}(q, m \to n)}\ (\text{MatMul}) \tag{3.10}$$

Adding two matrices of the same type results in a matrix with that type. Multiplying a matrix by a scalar results in a matrix of the same type, and so does applying a field endomorphism entry-wise.

$$\frac{\vdash M_1 : \mathcal{M}(q, m \to n) \qquad \vdash M_2 : \mathcal{M}(q, m \to n)}{\vdash M_1 + M_2 : \mathcal{M}(q, m \to n)}\ (\text{MatAdd}) \tag{3.11}$$

$$\frac{\vdash s : \mathcal{S} \qquad \vdash M : \mathcal{M}(q, m \to n)}{\vdash s \cdot M : \mathcal{M}(q, m \to n)}\ (\text{ScaMul}) \tag{3.12}$$

$$\frac{\vdash M : \mathcal{M}(q, m \to n) \qquad f : \mathbb{F} \to \mathbb{F} \text{ is a field endomorphism}}{\vdash \mathsf{apply}(f, M) : \mathcal{M}(q, m \to n)}\ (\text{MatApply}) \tag{3.13}$$

Taking the transpose of an $q^n \times q^m$ matrix results in a $q^m \times q^n$ matrix:

$$\frac{\vdash M : \mathcal{M}(q, m \to n)}{\vdash \mathsf{trans}(M) : \mathcal{M}(q, n \to m)} \text{ (Trans)} \tag{3.14}$$

The kronecker product of a $q^{n_1} \times q^{m_1}$ matrix and a $q^{n_2} \times q^{m_2}$ is a $q^{n_1+n_2} \times q^{m_1+m_2}$ matrix:

$$\frac{\vdash M_1 : \mathcal{M}(\mathbb{F}, q, m_1 \to n_1) \qquad \vdash M_2 : \mathcal{M}(\mathbb{F}, q, m_2 \to n_2)}{\vdash M_1 \otimes M_2 : \mathcal{M}(q, m_1 + m_2 \to n_1 + n_2)} \text{ (Kron)} \tag{3.15}$$

Example 3.1

As an example we will prove that $(3 \cdot \mathsf{ket}(0,2) \cdot \mathsf{bra}(1,2)) \otimes \mathsf{ket}(0,2)$ has type $\mathcal{M}(2, 1 \to 2)$, where we have $q = 2$. We will use the field of complex numbers $\mathbb{F} = \mathbb{C}$.

First we prove that $3 \cdot \mathsf{ket}(0,2) \cdot \mathsf{bra}(1,2)$ is of type $\mathcal{M}(2, 1 \to 1)$:

$$\dfrac{\dfrac{3 \in \mathbb{C}}{\vdash 3 : \mathcal{S}} \qquad \dfrac{\dfrac{}{\vdash \mathsf{bra}(1,2) : \mathcal{M}(2, 1 \to 0)} \qquad \dfrac{}{\vdash \mathsf{ket}(0,2) : \mathcal{M}(2, 0 \to 1)}}{\vdash \mathsf{ket}(0,2) \cdot \mathsf{bra}(1,2) : \mathcal{M}(2, 1 \to 1)}}{\vdash 3 \cdot \mathsf{ket}(0,2) \cdot \mathsf{bra}(1,2) : \mathcal{M}(2, 1 \to 1)} \tag{3.16}$$

Applying the Kronecker product and using the type of $\mathsf{ket}(0,2)$ gives

$$\frac{\vdash 3 \cdot \mathsf{ket}(0,2) \cdot \mathsf{bra}(1,2) : \mathcal{M}(2, 1 \to 1) \qquad \dfrac{}{\vdash \mathsf{ket}(0,2) : \mathcal{M}(2, 0 \to 1)}}{\vdash (3 \cdot \mathsf{ket}(0,2) \cdot \mathsf{bra}(1,2)) \otimes \mathsf{ket}(0,2) : \mathcal{M}(2, 1 \to 2)} \tag{3.17}$$

3.3. Value denotational semantics

As a baseline for the representation semantics we introduce later, we define the denotational semantics $[\![\cdot]\!]_v$ as the "value" of an expression, i.e., the concrete scalar or matrix that the expression represents. For example, the value of $\mathsf{ket}(1,2) \cdot \mathsf{bra}(0,2)$ is a 2×2 matrix with a 1 in the bottom-left corner and 0 everywhere else. We define the denotational semantics inductively on the type derivations of the expression, meaning expressions without a type (e.g. $\mathsf{ket}(1,2) \cdot \mathsf{ket}(0,2)$) do not get a value. Note that, due to the way the type system is defined, there is at most one type derivation for each expression.

For scalars, the denotational semantics are defined as follows:

$$\begin{aligned} [\![\alpha]\!]_v &= \alpha & [\![\mathsf{apply}(f,s)]\!]_v &= f([\![s]\!]_v) \\ [\![s_1 + s_2]\!]_v &= [\![s_1]\!]_v + [\![s_2]\!]_v & [\![\mathsf{entry}(i,j,M)]\!]_v &= ([\![M]\!]_v)_{ij} \\ [\![s_1 \cdot s_2]\!]_v &= [\![s_1]\!]_v \cdot [\![s_2]\!]_v & [\![\mathsf{tr}(M)]\!]_v &= \mathsf{tr}([\![M]\!]_v) \end{aligned} \tag{3.18}$$

The semantics of bras and kets are the $1 \times q$ and $q \times 1$ matrices with an entry 1 at the i-th position (counting from zero) and 0 everywhere else. We denote these matrices using Dirac notation with $\langle i|_q$ and $|i\rangle_q$, where the q is left out if it is clear from context.

$$[\![\mathsf{bra}(i,q)]\!]_v = \langle i|_q \qquad [\![\mathsf{ket}(i,q)]\!]_v = |i\rangle_q \tag{3.19}$$

The semantics of other matrix operations are performed simply by evaluating the expression recursively:

$$\begin{aligned} [\![M_2 \cdot M_1]\!]_v &= [\![M_2]\!]_v \cdot [\![M_1]\!]_v & [\![s \cdot M]\!]_v &= [\![s]\!]_v \cdot [\![M]\!]_v \\ [\![M_1 + M_2]\!]_v &= [\![M_1]\!]_v + [\![M_2]\!]_v & [\![\mathsf{trans}(M)]\!]_v &= [\![M]\!]_v^T \\ [\![M_1 \otimes M_2]\!]_v &= [\![M_1]\!]_v \otimes [\![M_2]\!]_v & [\![\mathsf{apply}(f,M)]\!]_v &= f([\![M]\!]_v) \end{aligned} \tag{3.20}$$

Note that the type rules above prohibit any incompatible matrices from being multiplied or added. The value of $f(M)$ for a field endomorphism $f : \mathbb{F} \to \mathbb{F}$ and matrix M is the matrix M with f applied to every entry.

These denotational semantics are in a certain sense valid, because $[\![e]\!]_v \in [\![T]\!]_v$ for any expression e of type T.

Example 3.2

The value semantics of the expression

$$e = (3 \cdot \mathsf{ket}(0,2) \cdot \mathsf{bra}(1,2)) \otimes \mathsf{ket}(0,2) \tag{3.21}$$

from Example 3.1 can be determined as follows:

$$\begin{aligned} [\![e]\!]_v &= [\![3 \cdot \mathsf{ket}(0,2) \cdot \mathsf{bra}(1,2)]\!]_v \otimes [\![\mathsf{ket}(0,2)]\!]_v && (3.22)\\ &= ([\![3]\!]_v \cdot [\![\mathsf{ket}(0,2) \cdot \mathsf{bra}(1,2)]\!]_v) \otimes [\![\mathsf{ket}(0,2)]\!]_v && (3.23)\\ &= (3 \cdot [\![\mathsf{ket}(0,2) \cdot \mathsf{bra}(1,2)]\!]_v) \otimes [\![\mathsf{ket}(0,2)]\!]_v && (3.24)\\ &= (3 \cdot [\![\mathsf{ket}(0,2)]\!]_v \cdot [\![\mathsf{bra}(1,2)]\!]_v) \otimes [\![\mathsf{ket}(0,2)]\!]_v && (3.25)\\ &= (3 \cdot |0\rangle_2 \cdot \langle 1|_2) \otimes |0\rangle_2 && (3.26)\\ &= \left(3 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \end{bmatrix}\right) \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} && (3.27)\\ &= \begin{bmatrix} 0 & 3 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} && (3.28) \end{aligned}$$

The expression is associated with the value we would like it to have.

3.4. Representations

We will introduce representations for both scalars and matrices. Scalars will have a representation that is the solution to a model counting problem (ϕ, W) (i.e., $\mathrm{WMC}(\phi, W)$). Meanwhile, matrices are represented with a longer tuple that also includes input and output variables, and a base size q: (ϕ, W, x, y, q).

3.4.1. Scalar representation

As mentioned above, scalars are represented by model counting instances (ϕ, W). This is formalized in the following definition:

Definition 3.3: Scalar representation

A tuple (ϕ, W) of a Boolean formula ϕ over a set of variables V and a weight function $W : V \times \mathbb{B} \to \mathbb{F}$ uniquely **represents** a constant $\alpha \in \mathbb{F}$ if $\mathrm{WMC}(\phi, W) = \alpha$. For consistency with matrix representations later, we will write rep as a function from Boolean formulae and weight functions to $\mathbb{F}$, defined by

$$\mathrm{rep}(\phi, W) = \mathrm{WMC}(\phi, W) \tag{3.29}$$

Example 3.4

Suppose we have a Boolean formula ϕ and weight function $W : \{x, y\} \times \mathbb{B} \to \mathbb{R}$ given by

$$\phi \equiv x \to y \tag{3.30}$$
$$W(x) = W(\overline{x}) = 1 \tag{3.31}$$
$$W(y) = W(\overline{y}) = 1/2 \tag{3.32}$$

Then $\mathrm{rep}(\phi, W) = \mathrm{WMC}(\phi, W) = 3/2$.

3.4.2. Matrix representation

The definition of a matrix representation extends on that of a scalar representation by adding input and output variables x and y, and a base size q. The input and output variables serve as pointers to the different entries in the matrix. The formula ϕ and weight function W are used as the basis for a set of model counting problems, one for every entry in the matrix. The same weight function is used at every entry, but the Boolean formula ϕ is extended with requirements for the input and output variables: $\phi' \equiv \phi \wedge (x = j) \wedge (y = i)$. The value at the entry is the weighted model count $\mathrm{WMC}(\phi', W)$.

The input and output of a matrix representation consist of Boolean variables that

together represent some number in the range $\{0, \ldots, q^n - 1\}$. We do this by using strings (of length n) of "q-state variable encodings". These encodings use Boolean variables and formulae to represent numbers from $\{0, \ldots, q-1\}$. How these can be implemented is described in Appendix A. However, there are some important properties these encodings need to have. These are outlined below:

- For an encoding v we write $\text{var}(v)$ for the set of all Boolean variables v uses.
- For an encoding v we can write $v = n$ to indicate v is equal to some number n. There should be exactly one assignment $\tau : \text{var}(v) \to \mathbb{B}$ for which $(v = n)[\tau] = 1$.
- Denote $\text{val}_v \equiv \bigvee_{n=0}^{q-1} (v = n)$.
- Write $v \leftrightarrow w$ for the equality of two q-state encodings v and w.

We use the same notation for strings of these variable encodings.

Definition 3.5: Matrix representation

Suppose we have a tuple (ϕ, W, x, y, q) of a Boolean formula ϕ over a set of variables V, a weight function $W : V \times \mathbb{B} \to \mathbb{F}$, two strings of q-state variables x and y over V, and a base size $q \in \mathbb{Z}_{\geq 2}$. This tuple **represents the matrix** $M \in \text{Mat}(\mathbb{F}, q^{|y|} \times q^{|x|})$ if for all $j \in \{0, \ldots, q^{|x|} - 1\}$ and $i \in \{0, \ldots, q^{|y|} - 1\}$ we have

$$\langle j | M | i \rangle = M_{ij} = \text{WMC}(\phi \wedge x = j \wedge y = i, W) \tag{3.33}$$

Again, every tuple represents exactly one matrix, which justifies the notation

$$\text{rep}(\phi, W, x, y, q) = M. \tag{3.34}$$

Example 3.6

We give a representation of the matrix

$$M = \begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} \tag{3.35}$$

with base size $q = 2$. This means we can use Boolean variables as our q-state input and output variables x and y.

Note that any non-zero entry (i, j) in the matrix has $i \neq j$, which means we use the formula $\phi \equiv x \leftrightarrow \overline{y}$. We need the model count to be 2 when x is true and y is false, which is why we set $W(x) = 2$. Any other value of W we set to 1. When determining the weighted model count for every entry of the matrix,

we find that $(\phi, W, x, y, 2)$ is a representation of M:

$$\begin{aligned}
&\mathrm{WMC}((x \leftrightarrow \overline{y}) \wedge \overline{x} \wedge \overline{y}, W) = 0 \\
&\mathrm{WMC}((x \leftrightarrow \overline{y}) \wedge \overline{x} \wedge y, W) = W(\overline{x})W(y) = 1 \\
&\mathrm{WMC}((x \leftrightarrow \overline{y}) \wedge x \wedge \overline{y}, W) = W(x)W(\overline{y}) = 2 \\
&\mathrm{WMC}((x \leftrightarrow \overline{y}) \wedge x \wedge y, W) = 0
\end{aligned} \tag{3.36}$$

The first and last equations are zero since the formulae are unsatisfiable.

Example 3.7

There is no requirement that the input and output variables be different. Such can be the case for diagonal matrices, and the Pauli-Z matrix in particular:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{3.37}$$

This matrix can represented with $(\top, W, x, x, 2)$, using the weight function $W : \{x\} \times \mathbb{B} \to \mathbb{F}$ defined by $W(\overline{x}) = 1$ and $W(x) = -1$.

3.4.3. Representation map

From Definitions 3.3 and 3.5 we introduce the map

$$\mathrm{rep} : \mathrm{Rep} \to \mathbb{F} \cup \mathrm{Mat}(\mathbb{F}), \tag{3.38}$$

where Rep is the set of scalar and matrix representations. This map is neither injective nor surjective. It is not injective because two tuples can represent the same scalar or matrix. Two model counting instances can have the same weighted model count. It is also not surjective because not every matrix shape can be represented. Matrices that can be represented have the shape $q^m \times q^n$, so a 3×2 matrix cannot be represented, for instance. This is a limitation that arises from the Kronecker product operation on matrix representations, defined in Section 3.5.

3.4.4. Equivalence of representations

Checking if two tuples represent the same value is NP-hard in general, since it would require checking $\phi_1 \equiv \phi_2$ for tuples (ϕ_1, W) and (ϕ_2, W). Despite this, it is useful to define the representation denotational semantics as a map to equivalence of representations, rather than the representations themselves. For this, we define the equivalence relation $\sim$ on Rep as follows:

$$r_1 \sim r_2 \quad \Longleftrightarrow \quad \mathrm{rep}(r_1) = \mathrm{rep}(r_2) \tag{3.39}$$

This relation induces an injective map $\text{rep}^{\#} : \text{Rep}/\sim \to \mathbb{F} \cup \text{Mat}(\mathbb{F})$. We denote the equivalence class of a representation r under this relation with $[r]$.

3.4.5. Finding equivalent representations

We will define the representation semantics in the next section as a map from type derivations to classes of representations. This is done inductively. Hence, we can have two classes of representations, and need to combine these in some way to get a new class. We will do this by using representatives of the classes. In the rules in Section 3.5, we introduce two types of constraints: Constraints on the domains of the representations and a constraint $\text{WMC}(\top, W) \neq 0$.

Domain constraints

We put some requirements on the domains of these representatives (e.g., the domains need to be disjoint) to combine them. Finding representations that conform to these restrictions is possible by substituting variables in the representations.

We illustrate this with an example: Suppose we define the representation semantics of $[\![s_1 \cdot s_2]\!]_r$ inductively. Then we already have two representatatives (ϕ_1, W_1) and (ϕ_2, W_2) with $[\![s_1]\!]_r = [(\phi_1, W_1)]$ and $[\![s_2]\!]_r = [(\phi_2, W_2)]$. Now we want to combine them by using $[\![s_1 \cdot s_2]\!]_r = [(\phi_1 \wedge \phi_2, W_1 \cup W_2)]$. A requirement for this to work is that the domains of W_1 and W_2 are disjoint. We can accomplish this by substituting variables in the representation (ϕ_2, W_2) with fresh ones. This can be implemented efficiently.

Requiring $\text{WMC}(\top, W) \neq 0$

Note that $\text{WMC}(\top, W)$ can be written as

$$\text{WMC}(\top, W) = \prod_{v \in V} (W(\overline{v}) + W(v)) \tag{3.40}$$

This quantity can only be 0 if, for some variable $v \in V$, we have $W(\overline{v}) + W(v) = 0$. If we have $W(\overline{v}) = W(v) = 0$, then $\text{WMC}(\phi, W) = 0$ for any Boolean formula ϕ. Hence we have $\text{rep}(\phi, W) = \text{rep}(\bot, W_0)$, with $W_0 : \varnothing \times \mathbb{B} \to \mathbb{F}$. Note that $\text{WMC}(\top, W_0) = 1$.

If $W(v) \neq 0$, we instead introduce a fresh variable v'. We add $v \leftrightarrow v'$ to the formula ϕ, and introduce the weight function $W' : (V \cup \{v'\}) \times \mathbb{B} \to \mathbb{F}$ that is the same as W on V, except for $W'(\overline{v}) = -W(\overline{v})$, $W'(\overline{v}') = -1$, and $W'(v') = 1$.

Using these two methods, for every model counting instance (ϕ, W), we can efficiently find an equivalent instance (ϕ', W') with $\text{WMC}(\top, W') \neq 0$. Note that these methods can also be applied to matrix representations.

3.5. Representation denotational semantics

In this section, we introduce the representation denotational semantics $[\![\cdot]\!]_r$ for all scalar and matrix type expressions. Like with the value semantics, the representation semantics are defined on proof trees of type derivations. We refrain from proving the correctness of these operations here, but proofs can be found in Appendix B.

3.5.1. Scalar representations

Scalar expressions are mapped to classes of equivalent scalar representations, denoted as $[(\phi, W)]$. Scalar constants form the basis of scalar expressions. These are mapped to the classes of representations of the same value α. For the sake of implementation, we give an explicit element of this class:

Rule 3.8: Scalar constant

$$[\![\alpha]\!]_r = [(x, W_\alpha)] \tag{3.41}$$

where $W_\alpha : \{x\} \times \mathbb{B} \to \mathbb{F}$ is a constant function α.

For model counting instances with no variables in common, the model counts can be multiplied using combining them as follows:

Rule 3.9: Scalar multiplication

$$\left.\begin{aligned} &[\![s_1]\!]_r = [(\phi_1, W_1)] \\ &[\![s_2]\!]_r = [(\phi_2, W_2)] \\ &\mathrm{dom}(W_1) \cap \mathrm{dom}(W_2) = \varnothing \end{aligned}\right\} \implies [\![s_1 \cdot s_2]\!]_r = [(\phi_1 \wedge \phi_2, W_1 \cup W_2)] \tag{3.42}$$

Here $W_1 \cup W_2$ indicates the union of two functions with disjoint domains $f_1 : X_1 \to Y_1$ and $f_2 : X_2 \to Y_2$ to a function $f_1 \cup f_2 : X_1 \cup X_2 \to Y_1 \cup Y_2$.

In category-theoretical terms, this is the morphism $f_1 \sqcup f_2$ from the coproduct of X_1 and X_2 to $Y_1 \cup Y_2$. Defining it like this would drop the requirement for the domains to be disjoint. However, there are restrictions in other rules that would make working with this definition difficult.

Adding scalars is more involved, as there is no property of weighted model counting instances that allows for easily adding results. We add a control variable c that points to either ϕ_1 or ϕ_2 as the formula that needs to hold. The other formula does not need to hold, meaning we get a model count that is multiplied by a factor $\mathrm{WMC}(\top, W_i)$. We divide by this quantity by scaling the weights of c appropriately.

As outlined before, equivalent representations with $\mathrm{WMC}(\top, W) \neq 0$ can be found efficiently.

Rule 3.10: Scalar addition

$$\left.\begin{aligned} &[\![s_1]\!]_r = [(\phi_1, W_1)] \\ &[\![s_2]\!]_r = [(\phi_2, W_2)] \\ &\mathrm{dom}(W_1) \cap \mathrm{dom}(W_2) = \varnothing \\ &c \notin \mathrm{dom}(W_1) \cup \mathrm{dom}(W_2) \\ &\mathrm{WMC}(\top, W_1) \neq 0,\ \mathrm{WMC}(\top, W_2) \neq 0 \end{aligned}\right\} \implies \tag{3.43}$$

$$[\![s_1 + s_2]\!]_r = [((\bar{c} \to \phi_1) \wedge (c \to \phi_2), W_1 \cup W_2 \cup W_c)]$$

where $W_c : \{c\} \times \mathbb{B} \to \mathbb{F}$ with $W(c) = 1/\mathrm{WMC}(\top, W_1)$ and $W(\bar{c}) = 1/\mathrm{WMC}(\top, W_2)$.

A field endomorphism f has the property that $\mathrm{WMC}(\phi, f \circ W) = f(\mathrm{WMC}(\phi, W))$ for any weight function W and formula ϕ, which is why it is introduced in the syntax of our language.

Rule 3.11: Field endomorphism on a scalar

$$[\![s]\!]_r = [(\phi, W)] \implies [\![\mathsf{apply}(f, s)]\!]_r = [(\phi, f \circ W)] \tag{3.44}$$

3.5.2. Matrix representations

Matrix typed expressions are mapped to equivalence classes of matrix representations $[(\phi, W, x, y, q)]$, as described in Definition 3.5. Bras and kets form the basis of the matrix-type expressions. These are represented with formulae that fix the values of the input/output variables. The weight function is kept constant 1. This means that there is exactly one input/output index i for which the model count is 1, and it is 0 for all other indices.

Rule 3.12: Bra and ket

$$[\![\mathsf{bra}(i, q)]\!]_r = [(x = i, W_1, x, -, q)] \tag{3.45}$$
$$[\![\mathsf{ket}(i, q)]\!]_r = [(x = i, W_1, -, x, q)] \tag{3.46}$$

where x is a q-state variable and $W_1 : \mathrm{var}(x) \times \mathbb{B} \to \mathbb{F}$ is constant 1 and "$-$" denotes an empty string of variables.

The product of two matrices is essentially the composition of two linear maps. We get the product $M_2 \cdot M_1$ by connecting the output variables of M_1 to the input variables of M_2. Figure 3.1 shows this schematically.

It is also necessary to add val_y for these connected variables y to the formula, since y will no longer be an input or output of the resulting matrix $M_2 \cdot M_1$. If this were not added to the formula, it would allow for values of y outside the range it can represent.

Rule 3.13: Matrix multiplication

$$\left.\begin{aligned} [\![M_1]\!]_r &= [(\phi_1, W_1, x, y, q)] \\ [\![M_2]\!]_r &= [(\phi_2, W_2, y, z, q)] \\ \text{dom}(W_1) \cap \text{dom}(W_2) &= \text{var}(y) \end{aligned}\right\} \implies \tag{3.47}$$

$$[\![M_2 \cdot M_1]\!]_r = [(\phi_1 \wedge \phi_2 \wedge \text{val}_y, W_1 \cdot W_2, x, z, q)]$$

The multiplication of weight functions is using the following rule for multiplying functions $f_1 : X_1 \to \mathbb{F}$ and $f_2 : X_2 \to \mathbb{F}$ to get a function $f_1 \cdot f_2 : X_1 \cup X_2 \to \mathbb{F}$.

$$(f_1 \cdot f_2)(x) = \begin{cases} f_1(x) & \text{if } x \notin X_2 \\ f_2(x) & \text{if } x \notin X_1 \\ f_1(x) \cdot f_2(x) & \text{if } x \in X_1 \cap X_2 \end{cases} \tag{3.48}$$

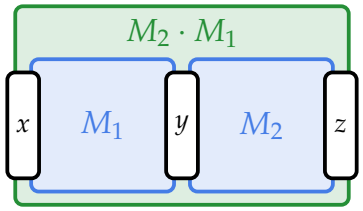


Figure 3.1: Diagram of multiplication operation on matrix representations.

The sum of two matrices is represented in a similar way to scalars, with an extra variable that indicates which matrix should be evaluated. In this case, the input and output variables are also linked with the input and output variables of the respective matrix. Figure 3.2 shows the operation schematically.

Rule 3.14: Matrix addition

$$\left.\begin{aligned}
&[\![M_1]\!]_r = [(\phi_1, W_1, x_1, y_1, q)] \\
&[\![M_2]\!]_r = [(\phi_2, W_2, x_2, y_2, q)] \\
&\mathrm{dom}(W_1) \cap \mathrm{dom}(W_2) = \varnothing \\
&(\{c\} \cup \mathrm{var}(x) \cup \mathrm{var}(y)) \cap (\mathrm{dom}(W_1) \cup \mathrm{dom}(W_2)) = \varnothing \\
&\mathrm{WMC}(\top, W_1) \neq 0,\ \mathrm{WMC}(\top, W_2) \neq 0
\end{aligned}\right\} \implies \tag{3.49}$$

$$[\![M_1 + M_2]\!]_r = [(\phi, W_1 \cup W_2 \cup W_c \cup W_{xy}, x, y, q)]$$

with

$$\begin{aligned}
\phi \equiv\ & (\bar{c} \to ((x \leftrightarrow x_1) \wedge (y \leftrightarrow y_1) \wedge \phi_1)) \\
& \wedge (c \to ((x \leftrightarrow x_2) \wedge (y \leftrightarrow y_2) \wedge \phi_2))
\end{aligned} \tag{3.50}$$

and $W_c : \{c\} \times \mathbb{B} \to \mathbb{F}$ and $W_{xy} : (\mathrm{var}(x) \cup \mathrm{var}(y)) \times \mathbb{B} \to \mathbb{F}$ defined by $W_c(c) = 1/\mathrm{WMC}(\top, W_1)$, $W_c(\bar{c}) = 1/\mathrm{WMC}(\top, W_2)$, and W_{xy} constant function 1.

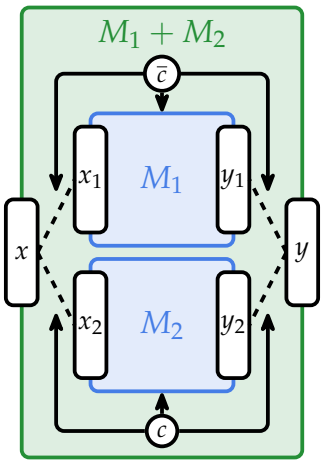


Figure 3.2: Diagram of addition operation on matrix representations.

The Kronecker product representation is constructed from the two independent representations of the matrices M_1 and M_2. The input and output variables of the two matrices are concatenated. Figure 3.3 shows this schematically.

Rule 3.15: Kronecker product

$$\left.\begin{aligned} [\![M_1]\!]_r &= [(\phi_1, W_1, x_1, y_1, q)] \\ [\![M_2]\!]_r &= [(\phi_2, W_2, x_2, y_2, q)] \\ \mathrm{dom}(W_1) \cap \mathrm{dom}(W_2) &= \varnothing \end{aligned}\right\} \implies \tag{3.51}$$

$$[\![M_1 \otimes M_2]\!]_r = [(\phi_1 \wedge \phi_2, W_1 \cup W_2, x_1 x_2, y_1 y_2, q)]$$

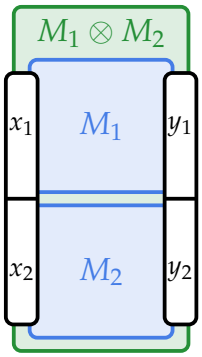


Figure 3.3: Diagram of Kronecker product operation on matrix representations.

The multiplication of a scalar and a matrix can be represented by a conjunction of the two formulae. This operation uses the property that $\mathrm{WMC}(\phi \wedge \psi, W_1 \cup W_2) = \mathrm{WMC}(\phi, W_1) \cdot \mathrm{WMC}(\psi, W_2)$ for two formulae ϕ and ψ for variables in the domains of W_1 and W_2 respectively (such that the domains do not overlap). Figure 3.4 shows the operation schematically.

Rule 3.16: Matrix-scalar multiplication

$$\left.\begin{aligned} [\![s]\!]_r &= [(\phi_s, W_s)] \\ [\![M]\!]_r &= [(\phi, W, x, y, q)] \\ \mathrm{dom}(W_s) \cap \mathrm{dom}(W) &= \varnothing \end{aligned}\right\} \implies [\![s \cdot M]\!]_r = [(\phi \wedge \phi_s, W \cup W_s, x, y, q)] \tag{3.52}$$

The representation of the transpose of a matrix is the same, but with input and output variables swapped. The effect of this operation can be seen directly in (3.33), where swapping the input and output variables replaces $(x = j) \wedge (y = i)$ with $(x = i) \wedge (y = j)$. Figure 3.5 shows the operation schematically.

Rule 3.17: Transpose

$$[\![M]\!]_r = [(\phi, W, x, y, q)] \implies [\![\mathsf{trans}(M)]\!]_r = [(\phi, W, y, x, q)] \tag{3.53}$$

Applying a field endomorphism to a matrix is similar to applying it to a scalar.

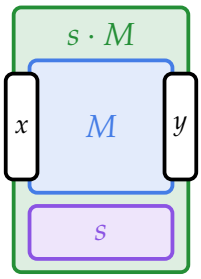


Figure 3.4: Diagram of multiplying a matrix M with a scalar s, using representations for both.

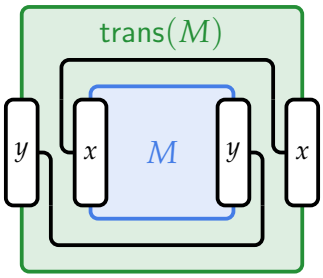


Figure 3.5: Diagram of the transpose of a matrix representation. Input and output variables are swapped.

Rule 3.18: Field endomorphism on a matrix

$$[\![M]\!]_r = [(\phi, W, x, y, q)] \implies [\![\mathsf{apply}(f, M)]\!]_r = [(\phi, f \circ W, x, y, q)] \quad (3.54)$$

The trace of a matrix can be calculated by adding a clause to the conjunction requiring the input and output variables to have the same value. In addition, we need this new input/output to be valid. Figure 3.6 shows the operation schematically.

Rule 3.19: Trace

$$[\![M]\!]_r = [(\phi, W, x, y, q)] \implies [\![\mathsf{tr}(M)]\!]_r = [(\phi \wedge (x \leftrightarrow y) \wedge \mathrm{val}_x, W)] \quad (3.55)$$

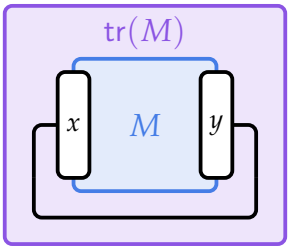


Figure 3.6: Diagram of taking the trace of a matrix, using a matrix representation to get a scalar representation.

An entry in the matrix can be obtained by applying the definition from (3.33) directly.

Instead of returning the quantity $\mathrm{WMC}(\phi \wedge (x = j) \wedge (y = i), W)$, we return the model counting instance.

Rule 3.20: Matrix entry

$$[\![M]\!]_r = [(\phi, W, x, y, q)] \quad \Longrightarrow \quad [\![\mathsf{entry}(i, j, M)]\!] = [(\phi \wedge (x = j) \wedge (y = i), W)] \tag{3.56}$$

3.6. Correctness

To use these semantics effectively, we need to be able to convert an expression to a representation, then use a model counter to get the actual matrix or scalar that is represented. We want the outcome to be the same as evaluating the expression directly (i.e., using $[\![\cdot]\!]_v$). What this means is that we need $\mathrm{rep}^{\#} \circ [\![\cdot]\!]_r = [\![\cdot]\!]_v$. We interpret $[\![\cdot]\!]_r$ and $[\![\cdot]\!]_v$ as maps

$$[\![\cdot]\!]_v : \mathrm{Exp} \to \mathbb{F} \cup \mathrm{Mat}(\mathbb{F}) \tag{3.57}$$

$$[\![\cdot]\!]_r : \mathrm{Exp} \to \mathrm{Rep} \tag{3.58}$$

We define the set Exp of expressions that have a type.

Theorem 3.21

The representation semantics $[\![\cdot]\!]_r$ are well-defined. Furthermore, for the value semantics $[\![\cdot]\!]_v$ and the function $\mathrm{rep}^{\#}$ as defined in Section 3.4.4, we have

$$\mathrm{rep}^{\#} \circ [\![\cdot]\!]_r = [\![\cdot]\!]_v \tag{3.59}$$

Proof. See Appendix B, which uses induction on the proof trees of the expression types. □

3.7. Implementation

We provide a Python implementation of the presented framework, called DiracWMC, at [13]. The implementation supports most of the operations presented in this chapter. In addition, it supports labeling matrices, similar to D-Hammer [50]. Matrices can also be constructed from Boolean formulae and weight functions directly, removing the need to build up matrices from bras and kets.

The subsequent chapters in this thesis use the implementation to test the effectiveness and performance of our framework. The source code for the experiments performed in these chapters can be found in the experiments folder of the implementation.

3.8. Discussion

Although most of the rules from Section 3.5 can be implemented efficiently, yielding a compact CNF formula, the addition rules introduce an extra variable that distributes over the already existing formulae when keeping the formulae in CNF. When doing many additions, this can cause the size of the formula to become quadratic in the number of operations.

An alternative representation (ϕ, W, x, y, q, c) could be introduced, which adds a "conditional variable" c. We can let the representation with $\phi \wedge c$ be of the original matrix, and with $\phi \wedge \bar{c}$ of the matrix with the same shape, but filled with ones. This would make the addition operation result in a more compact formula, namely

$$(c \to (c_1 \vee c_2)) \wedge (\bar{c} \to (\bar{c}_1 \wedge \bar{c}_2)) \wedge (\bar{c}_1 \vee \bar{c}_2) \wedge \phi_1 \wedge \phi_2 \tag{3.60}$$

This requires only a constant amount of extra space per addition. However, it is not certain that the performance of the model counters would increase when using this definition, since the formula $(c \to (c_1 \vee c_2)) \wedge (\bar{c} \to (\bar{c}_1 \wedge \bar{c}_2))$ cannot be simplified easily.

In our method, the model counter is only called at the end of the process, once one big model counting instance is constructed. It can be beneficial to evaluate scalars and small matrices while constructing the representations. This could reduce the total size of the problems the model counter has to solve.

Chapter 4

Ising model

In this chapter, we introduce the Ising model: A physical model from statistical mechanics, often used to model molecular interactions [6]. It can, for example, be used to model a lattice of magnetic particles with a certain spin where interactions between particles of different spins are different from interactions between particles with the same spin. A quantity that is of interest when studying the Ising model is the partition function $Z_{\beta,I}$, which is a measure of the total energy in the system. It can be used to determine the probability of a certain configuration of states occurring in the system, as it acts as a normalizing factor to determine this probability.

We show a procedure introduced by Nagy et al. for calculating the Ising model partition function using weighted model counting [30]. We also show that there is an alternative way of calculating the partition function using matrices, from which the theory in Chapter 3 can be used to transform the calculation into a weighted model counting problem. It turns out that these two methods result in the same WMC instance. We use the Ising model as a test setting for the implementation of our framework, DiracWMC [13]. In Chapters 5 and 6, we use the same implementation on new applications.

4.1. Definition

The Ising model is represented as a weighted graph with a set of vertices Λ. The weights on the edges of the graph are real numbers J_{ij}, representing interaction strengths between vertices. Vertices are often referred to as sites. Additionally, there is an external field strength h_i (also a real number) at each site, which models external factors that influence the system. Figure 4.1 shows an Ising model on a regular lattice, which is often the case with Ising models of interest. An Ising model is called ferromagnetic if $J_{ij} \geq 0$ for all $i, j \in \Lambda$, and antiferromagnetic if $J_{ij} < 0$ for all $i, j \in \Lambda$. In this work, we consider the general case where J_{ij} can be any real value (i.e., the model does not have to be ferromagnetic or antiferromagnetic).

The partition function is a measure of an Ising model that is the sum of "energies"

over all possible configurations of an Ising model. A configuration is an assignment of values from $\{-1, 1\}$ to every site in the model. We make these concepts explicit in Definition 4.1.

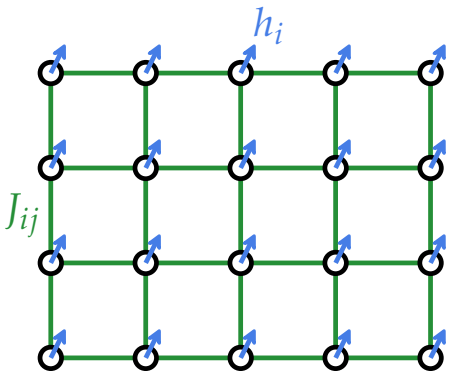


Figure 4.1: A rectangular lattice Ising model. The arrows are the external field strengths h_i at each site, and the connections between the sites are interactions with strengths J_{ij}.

Definition 4.1: Ising model

A **(classical) Ising model** is a tuple $I = (\Lambda, J, h)$ where Λ is a finite set representing vertices (sites), $J : \Lambda^2 \to \mathbb{R}$ is a symmetric (i.e., $J(i,j) = J(j,i)$) function indicating the interaction strengths between sites, and $h : \Lambda \to \mathbb{R}$ a function from sites to external field strengths. As is conventional, we will write $J(i,j)$ as J_{ij} and $h(i)$ as h_i.

A **configuration** is a function $\sigma : \Lambda \to \{-1, 1\}$ that assigns every site a spin. Again, we write $\sigma(i)$ as σ_i.

The **Hamiltonian** $H_I : \mathrm{Map}(\Lambda, \{-1, 1\}) \to \mathbb{R}$ of an Ising model is a function from configurations to energies, defined as

$$H_I(\sigma) = -\sum_{i,j \in \Lambda} J_{ij}\sigma_i\sigma_j - \sum_{i \in \Lambda} h_i\sigma_i \tag{4.1}$$

The **partition function**, at inverse temperature β, is defined as the following sum over all possible configurations:

$$Z_{\beta,I} = \sum_{\sigma:\Lambda \to \{-1,1\}} e^{-\beta H_I(\sigma)} \tag{4.2}$$

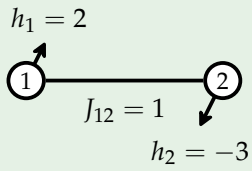
The main use of the partition function is that it is a normalization factor for the Boltzmann distribution $P_{\beta,I}(\sigma) = e^{-\beta H_I(\sigma)}/Z_{\beta,I}$, which calculates the probability of a configuration occuring in the system. At a very high temperature $\beta \to 0$, the Boltzmann distribution approaches the uniform distribution. At very low

temperatures $\beta \to \infty$, the Boltzmann distribution approaches a probability 1 for the configuration with the lowest energy $H_I(\sigma)$. This configuration is known as the ground state, and finding the ground state is a separate, very important problem in physics.

As an example, we show the partition function calculation of an Ising model with two sites.

Example 4.2

Consider an Ising model with two sites, with an interaction strength of 1 between the two sites, and an external field strength of 2 at the first site, and -3 at the second site. Note that this is a ferromagnetic Ising model, because the only interaction J_{12} is positive.



The Hamiltonian of this Ising model is

$$H_I(\sigma) = -J_{12}\sigma_1\sigma_2 - h_1\sigma_1 - h_2\sigma_2 = -\sigma_1\sigma_2 - 2\sigma_1 + 3\sigma_2 \tag{4.3}$$

Calculating the partition function requires summing over four configurations σ, which we will denote as $(-1,-1)$, $(-1,1)$, $(1,-1)$, and $(1,1)$. The partition function as inverse temperature $\beta = 1$ is

$$Z_{\beta,I} = e^{-H((-1,-1))} + e^{-H((-1,1))} + e^{-H((1,-1))} + e^{-H((1,1))} \tag{4.4}$$

$$= e^{-(-2)} + e^{-6} + e^{-(-4)} + e^{-0} \approx 62.99 \tag{4.5}$$

The probability of the configuration $(-1,-1)$ occurring according to the Boltzmann distribution is $P_{\beta,I}((-1,-1)) = e^{-(-2)}/Z_{\beta,I} \approx 0.12$.

The calculation of the partition function is #P-hard in general [3], as the number of terms in the sum is $2^{|\Lambda|}$ and the sum cannot be easily simplified. There are several tools that improve the runtime significantly over brute-force calculation in some cases [33, 41]. However, these tools can still struggle to solve larger problems. This makes the problem a good candidate for weighted model counting tools.

4.2. Conversion to WMC by Nagy et al.

First, we will show how Nagy et al. [30] converted the Ising model partition function calculation to a WMC problem. In the next section, we show how the same can be

accomplished using the matrix representations introduced in Chapter 3.

In the method introduced by Nagy et al. [30], a variable x_i is introduced for each site. For each interaction, there is a variable x_{ij}. A variable x_i being set to false corresponds with $\sigma_i = -1$, and x_i being true corresponds with $\sigma_i = 1$. Likewise the variable x_{ij} should be false if $\sigma_i\sigma_j = -1$ and true if $\sigma_i\sigma_j = 1$. This can be accomplished by introducing the following formula, which can be easily rewritten into CNF:

$$\phi \equiv \bigwedge_{i,j\in\Lambda} (x_{ij} \leftrightarrow (x_i \leftrightarrow x_j)) \tag{4.6}$$

They note that the exponents in the partition function can be split up as follows:

$$Z_{\beta,I} = \sum_{\sigma:\Lambda\to\{-1,1\}} \left(\prod_{i,j\in\Lambda} e^{\beta J_{ij}\sigma_i\sigma_j}\right)\left(\prod_{i\in\Lambda} e^{\beta h_i\sigma_i}\right) \tag{4.7}$$

They then introduce the following weight function:

$$\begin{aligned} W(\overline{x}_{ij}) &= e^{-\beta J_{ij}} & W(x_{ij}) &= e^{\beta J_{ij}} \\ W(\overline{x}_i) &= e^{-\beta h_i} & W(x_i) &= e^{\beta h_i} \end{aligned} \tag{4.8}$$

It follows that $\mathrm{WMC}(\phi, W) = Z_{\beta,I}$.

4.2.1. Reproducing results from Nagy et al.

As a baseline, some of the experiments from Nagy et al. [30] have been reproduced. The experiments in this work are limited to the three model counters Cachet [40], DPMC [14], and TensorOrder [16]. Figure 4.2 shows the runtimes of the three solvers for 2D square lattice Ising models. It shows that the TensorOrder solver is significantly faster than DPMC and Cachet for larger problem sizes. This is in line with effective tools from physics, like CATN [33], using tensor networks to solve the partition function problem, just like TensorOrder does. Figure 4.3 shows the same experiment with Ising models on random graphs of expected degree three. In this case, the performance of the DPMC solver is more comparable to that of TensorOrder. The results are similar to the results from Nagy et al. [30]. The output partition function values have been checked against a brute-force method for square lattices of sizes 2, 3, and 4.

4.3. Conversion to matrix representations

The Ising model can be formulated in an alternative way using matrices. Every site corresponds to a subspace that the matrices can act upon. External fields are modeled using 2×2 diagonal matrices. Interactions are modelled using 4×4 matrices acting on two subspaces.

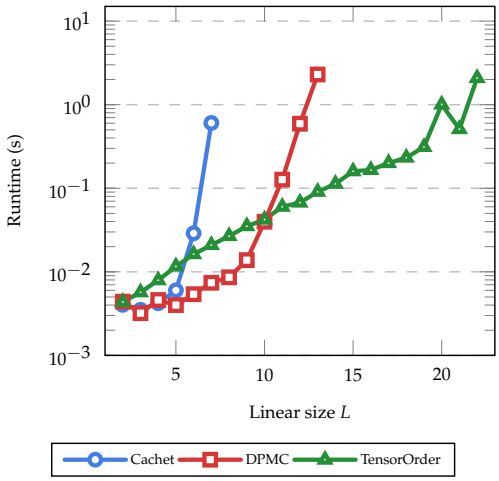


Figure 4.2: Runtime of calculating the partition function of an $L \times L$ square lattice Ising model with interaction strengths and external field strengths from the standard normal distribution, averaged over five runs. Comparison between the model counters Cachet, DPMC, and TensorOrder.

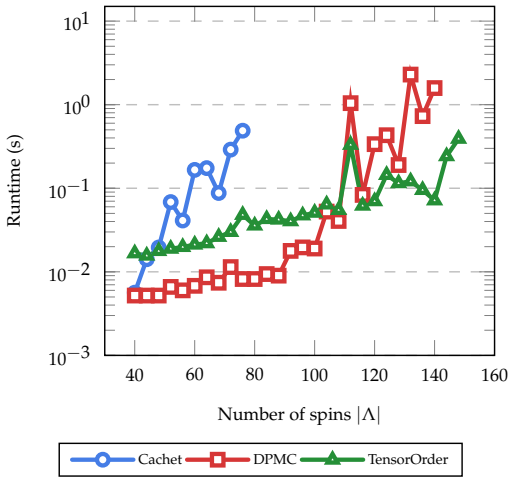


Figure 4.3: Runtime of calculating the partition function of a random graph Ising model for different numbers of spins (nodes), averaged over five runs. The expected degree of each node is three. The interaction strengths are uniformly chosen from $[-1, 1]$, and there is no external field. Comparison between the model counters Cachet, DPMC, and TensorOrder.

This formulation allows for a more general notion of an Ising model using non-diagonal matrices. This is used in the formulation of the quantum Ising model, described in Chapter 5.

4.3.1. Alternative formulation of the Ising model

In this alternative formulation, the Hamiltonian is a diagonal matrix that is the sum of Z and $Z \otimes Z$ Pauli matrices. Recall that the Pauli-Z matrix is defined as

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{4.9}$$

The Hamiltonian is equal to

$$H_I = -\sum_{i,j \in \Lambda} J_{ij} Z_i Z_j - \sum_{i \in \Lambda} h_i Z_i \tag{4.10}$$

Each entry on the diagonal of this matrix corresponds one-to-one with the energy of a configuration. For example, the top-leftmost entry is the energy of the configuration with all spins -1.

The exponential $e^{-\beta H_I} = \sum_{k=0}^{\infty}(-\beta H_I)^k/k!$ is again a diagonal matrix, with every entry on the diagonal now equal to $e^{-\beta E}$, where E is the original entry in the matrix H_I. Taking the trace of this matrix is the same as summing over $e^{-\beta E}$ for all configuration energies E:

$$Z_{I,\beta} = \text{tr}(e^{-\beta H_I}) \tag{4.11}$$

4.3.2. Rewriting the partition function

A property of the matrix exponential is that, if two matrices X and Y commute (i.e., $XY = YX$), then $e^{X+Y} = e^X \cdot e^Y$. Since any two diagonal matrices commute, we can rewrite the matrix exponential to the following:

$$e^{-\beta H_I} = \left(\prod_{i,j\in\Lambda} e^{-J_{ij}Z_iZ_j}\right)\left(\prod_{i\in\Lambda} e^{-h_iZ_i}\right) \tag{4.12}$$

If we can write each of the matrices in this product as a representation using a Boolean formula and weight function, we can use the methods from Chapter 3 to determine the partition function. In the following sections, we describe how to represent matrices $e^{\theta Z}$ and $e^{\theta(Z\otimes Z)}$.

4.3.3. Representing $e^{\theta Z}$

Note that the matrix $e^{\theta Z}$ is equal to

$$e^{\theta Z} = \begin{bmatrix} e^{\theta} & 0 \\ 0 & e^{-\theta} \end{bmatrix} \tag{4.13}$$

We can represent this matrix using a single variable x and no clauses in the Boolean formula. The weight function $W : \{x\} \times \mathbb{B} \to \mathbb{R}$ assigns the weight $e^{-\theta}$ if the variable is set to true, and e^{θ} if it is set to false: $W(\overline{x}) = e^{\theta}$ and $W(x) = e^{-\theta}$. Then we have

$$e^{\theta Z} = \text{rep}(\top, W, x, x, 2) \tag{4.14}$$

4.3.4. Representing $e^{\theta(Z\otimes Z)}$

The matrix $e^{\theta(Z\otimes Z)}$ is equal to

$$e^{\theta(Z\otimes Z)} = \begin{bmatrix} e^{\theta} & 0 & 0 & 0 \\ 0 & e^{-\theta} & 0 & 0 \\ 0 & 0 & e^{-\theta} & 0 \\ 0 & 0 & 0 & e^{\theta} \end{bmatrix} \tag{4.15}$$

We can use an auxiliary variable z in combination with the required input variables x and y to indicate when x and y are equal, since these are exactly the cases corresponding with e^{θ} in the matrix. Hence we introduce the Boolean formula $z \leftrightarrow (x \leftrightarrow y)$ We use the weight function $W : \{x, y, z\} \times \mathbb{B} \rightarrow \mathbb{R}$ which returns 1 for both input variables and $W(\overline{z}) = e^{-\theta}$, $W(z) = e^{\theta}$. Then

$$e^{\theta(\mathrm{Z}\otimes\mathrm{Z})} = \mathrm{rep}(z \leftrightarrow (x \leftrightarrow y), W, xy, xy, 2) \tag{4.16}$$

4.3.5. Comparison with Nagy et al.

When multiplying all matrices from Sections 4.3.3 and 4.3.4 using the matrix representation procedure described in Section 3.5, we get the same formula and weight function as in Nagy et al. [30]. Since the input and output variables of every matrix are the same, the input and output variables of the product representation are also the same. Since there is exactly one input variable per site (as a Kronecker product is taken over all sites), each input/output variable from the matrices $e^{-h_i \mathrm{Z}_i}$ and $e^{-J_{ij}\mathrm{Z}_i\mathrm{Z}_j}$ corresponds with a site in the lattice, and hence with a variable x_i from Section 4.2 (technically, its negation $\overline{x_i}$). The variable z from the representation of $e^{\theta(\mathrm{Z}\otimes\mathrm{Z})}$ corresponds with a variable x_{ij} from Section 4.2.

When calculating the partition function by first converting the Ising model to a matrix representation, and then calculating the trace using a model counter, we get similar runtimes to using the direct method from Section 4.2. Figure 4.4 shows the performance on square lattice Ising models, while Figure 4.5 shows the performance on random graph Ising models. Both experiments were done with the same Ising models as those used in Section 4.2.1. Note that the runtimes include the runtimes of the solvers, but not the time it took to build the matrix representations. These runtimes are not significant for smaller problems, but do become significant as problem sizes increase. We did not include it here, as the methods used to combine the matrices are not well-optimized. An implementation in, for example, C++ could see a significant performance increase here. As mentioned above, the two methods result in the same Boolean formula and weight function, explaining the similar runtimes. Again, the output partition function values have been checked against a brute-force method for square lattices of sizes 2, 3, and 4.

Although both methods yield the same results, the matrix representation method has the advantage that it can be applied to other models similar to this one, without those models having to be expressed in WMC form directly.

In the next chapters, we introduce two other models where this advantage is evident. The transverse-field Ising model in Chapter 5 is not easily convertible to WMC without first looking at matrix representations. The Potts model, introduced in Chapter 6, is similar to the Ising model. Matrix representations have a similar advantage here.

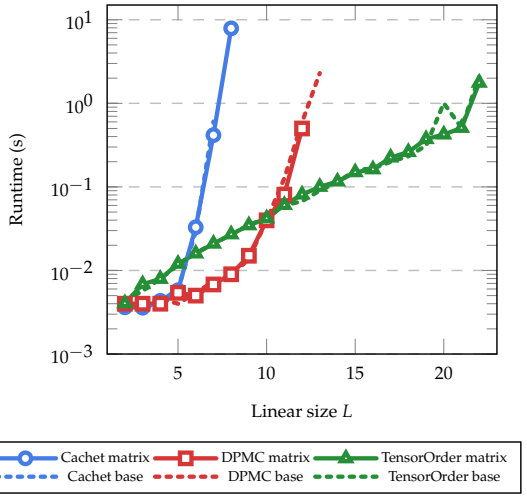


Figure 4.4: Runtime of calculating the partition function of an $L \times L$ square lattice Ising model with interaction strengths and external field strengths from the standard normal distribution, averaged over five runs. The problem is converted to a matrix representation from Chapter 3, after which the trace is calculated using a model counter. Comparison between the model counters Cachet, DPMC, and TensorOrder. Direct method from Nagy et al. in dotted lines [30].

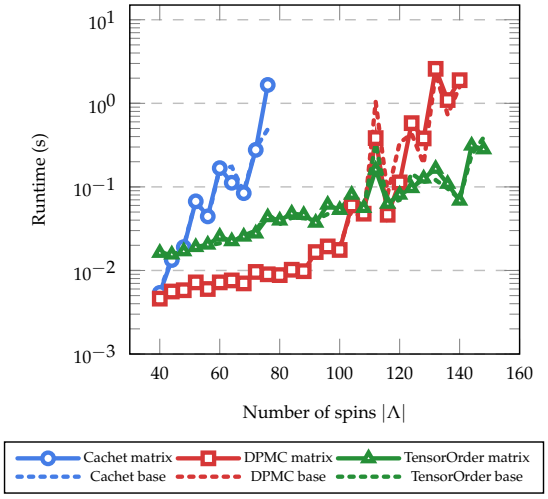


Figure 4.5: Runtime of calculating the partition function of a random graph Ising model for different numbers of spins (nodes), averaged over five runs. The expected degree of each node is three. The interaction strengths are uniformly chosen from $[-1, 1]$, and there is no external field. The problem is converted to a matrix representation from Chapter 3, after which the trace is calculated using a model counter. Comparison between the model counters Cachet, DPMC, and TensorOrder. Direct method from Nagy et al. in dotted lines [30].

Chapter 5

Transverse-field Ising model

We will now move to the quantum version of the Ising model. In this context, the Hamiltonian can be a non-diagonal matrix. The Hamiltonian is used, for example, in the Schrödinger equation. It describes the time evolution of a quantum system

$$H\left|\psi(t)\right\rangle = i\hbar\frac{d}{dt}\left|\psi(t)\right\rangle. \tag{5.1}$$

The quantum partition function is, as can be done in the classical case, defined as the trace of an exponential of the matrix:

$$Z_\beta = \operatorname{tr}\left(e^{-\beta H}\right) = \operatorname{tr}\left(\sum_{k=0}^{\infty}\frac{(-\beta H)^k}{k!}\right) \tag{5.2}$$

The difficulty in calculating this trace lies in the structure of the Hamiltonian for quantum systems. While in the classical Ising model we had a linear combination of Pauli-Z matrices, in the quantum case we may have non-commutative matrices. Recall that, for two matrices A and B, we have $e^{A+B} = e^A \cdot e^B$ if and only if A and B commute.

Applications of the computation of the partition function include finding phase transitions and the calculation of the Helmholtz free energy [37]

$$F = -\frac{1}{\beta}\log Z_\beta. \tag{5.3}$$

5.1. Definition

Finding the partition function, or a quantity related to it, is a common problem in the research of the transverse-field Ising model [7, 22]. This is the model we will discuss in this work. It has many different characterizations based on context. However, we will focus on the definition used by Suzuki [45], which has differing

interaction strengths between spins and constant external field strengths in two different "directions" X and Z. It should be noted that this model is not a complete generalization of the classical Ising model introduced in Chapter 4, since the external field strengths are kept constant at all sites. However, the model is almost a generalization.

Definition 5.1: Transverse-field Ising model

A **transverse-field (quantum) Ising model** is a tuple $Q = (\Lambda, J, \mu_z, \mu_x)$ where Λ is a finite set of vertices (called sites), $J : \Lambda^2 \to \mathbb{R}$ is a symmetric function mapping pairs of sites to interaction strengths, and $\mu_x, \mu_z \in \mathbb{R}$ are constants representing field strengths in two directions.

The **Hamiltonian** of this model is a $2^{|\Lambda|} \times 2^{|\Lambda|}$ matrix given by

$$H_Q = -\sum_{i,j\in\Lambda} J_{ij} Z_i Z_j - \mu_z \sum_{i\in\Lambda} Z_i - \mu_x \sum_{i\in\Lambda} X_i, \tag{5.4}$$

where X_i and Z_i are Pauli matrices (see (4.9)) at site i and identity everywhere else.

The **partition function**, at inverse temperature $\beta > 0$, is defined as

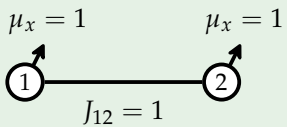
$$Z_{\beta,Q} = \operatorname{tr}\left(e^{-\beta H_Q}\right). \tag{5.5}$$

Recall that the Pauli-X and Pauli-Z matrices are defined as

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{5.6}$$

Example 5.2

Suppose we have a two-spin system $Q = (\Lambda, J, \mu_z, \mu_x)$, where $\Lambda = \{1, 2\}$, $J_{12} = 1$, $\mu_z = 0$ and $\mu_x = 1$.



Then the Hamiltonian is the following 4×4 matrix:

$$H_Q = -Z_1Z_2 - X_1 - X_2 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & 0 & -1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{5.7}$$

The partition function at inverse temperature $\beta = 1$ then is

$$Z_{\beta,Q} = \mathrm{Tr}\left(\exp \begin{bmatrix} -1 & -1 & -1 & 0 \\ -1 & 1 & 0 & -1 \\ -1 & 0 & 1 & -1 \\ 0 & -1 & -1 & -1 \end{bmatrix} \right) \tag{5.8}$$

$$\approx \mathrm{Tr} \begin{bmatrix} 1.52 & -2.07 & -2.07 & 1.15 \\ -2.07 & 4.76 & 2.04 & -2.07 \\ -2.07 & 2.04 & 4.76 & -2.07 \\ 1.15 & -2.07 & -2.07 & 1.52 \end{bmatrix} \tag{5.9}$$

$$\approx 12.55. \tag{5.10}$$

It has been shown that finding the partition function of a 2-local Hamiltonian (the above Hamiltonian is 2-local) is QMA-complete [18] in general. However, there are some special cases of Hamiltonians for which the partition function computation is easier. Bravyi et al. showed that the computation of the partition function for Hamiltonians with a specific denseness condition can be achieved in polynomial time in the number of qubits in the system [5]. However, this denseness condition is quite strict and, for example, does not hold in the case of a transverse-field Ising model on a square lattice.

5.2. Trotterization

A common technique that physicists use to solve problems like this is the so-called Trotterization technique [20]. It makes use of the following property for any two matrices A and B:

$$e^{A+B} = \lim_{k \to \infty} \left(e^{\frac{A}{k}} \cdot e^{\frac{B}{k}} \right)^k \tag{5.11}$$

This is true even if A and B do not commute, which is a requirement for the equality $e^{A+B} = e^A e^B$. In our problem, we have two of these non-commuting matrices,

consisting of Pauli-Z and Pauli-X matrices respectively:

$$H_{Q,Z} = -\sum_{i,j\in\Lambda} J_{ij} Z_i Z_j - \mu_z \sum_{i\in\Lambda} Z_i \tag{5.12}$$

$$H_{Q,X} = -\mu_x \sum_{i\in\Lambda} X_i \tag{5.13}$$

Then we have $H_Q = H_{Q,Z} + H_{Q,X}$. Using this, we can approximate the partition function as

$$Z_{\beta,Q} \approx \operatorname{tr}\left(\left(e^{-\beta\frac{H_{Q,Z}}{k}} \cdot e^{-\beta\frac{H_{Q,X}}{k}}\right)^k\right) \tag{5.14}$$

Another useful property of the matrix exponential is that, for an invertible matrix P and any matrix A, we have $e^{P^{-1}AP} = P^{-1}e^A P$. In our problem, the Pauli-X matrix can be diagonalized as $X = HZH$, where H is the involutory Hadamard operator

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{5.15}$$

Define the sum of Pauli-Z matrices

$$H'_{Q,X} = -\mu_x \sum_{i\in\Lambda} Z_i \tag{5.16}$$

This is the same as $H_{Q,X}$, but with all Pauli-X matrices replaced with a Pauli-Z. Therefore, we have $H_{Q,X} = H^{\otimes|\Lambda|} H_{Q,X'} H^{\otimes|\Lambda|}$, which gives

$$Z_{\beta,Q} \approx \operatorname{tr}\left(\left(e^{-\beta\frac{H_{Q,Z}}{k}} \cdot e^{-\beta\frac{H^{\otimes|\Lambda|} H'_{Q,X} H^{\otimes|\Lambda|}}{k}}\right)^k\right) \tag{5.17}$$

$$= \operatorname{tr}\left(\left(e^{-\beta\frac{H_{Q,Z}}{k}} H^{\otimes|\Lambda|} e^{-\beta\frac{H'_{Q,X}}{k}} H^{\otimes|\Lambda|}\right)^k\right) \tag{5.18}$$

Note that we are now left with only Hadamard matrices and exponentials of Pauli-Z matrices.

5.3. Conversion to matrix encodings

The matrices $e^{H_{Q,Z}/k}$ and $e^{H'_{Q,X}/k}$ are products of matrices of the forms $e^{\theta Z}$ and $e^{-\theta Z}$, and can hence be encoded as described in Chapter 4. All that remains is encoding the Hadamard matrices. A single Hadamard can be encoded as the following: (see also [28])

$$(r \leftrightarrow (x \wedge y), W, x, y, 2) \tag{5.19}$$

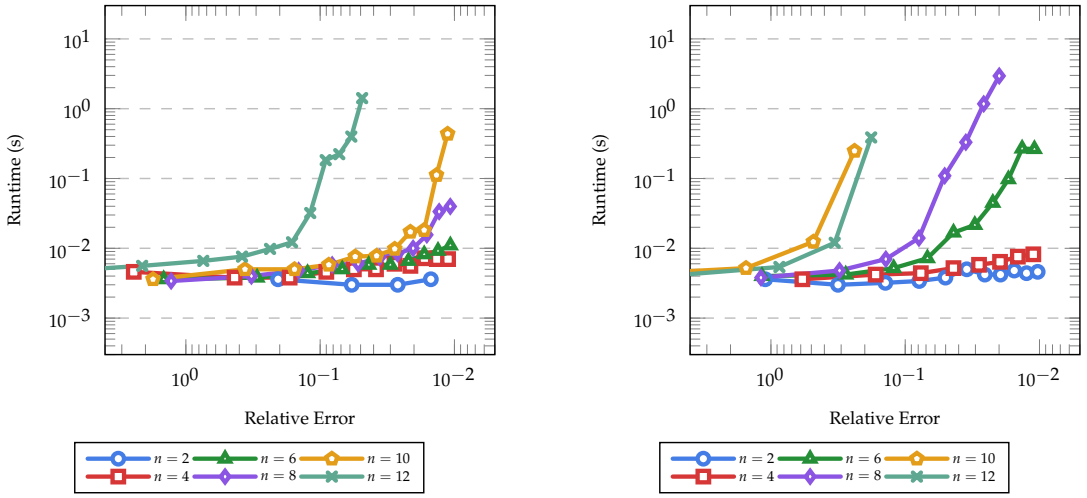
with $W : \{x, y, r\} \times \mathbb{B} \to \mathbb{R}$ constant 1 except for $W(r) = -1$. Multiplications and Kronecker products can be performed using the techniques from Chapter 3.

5.4. Experiments

Figure 5.1 shows the performance of the DPMC model counter when using Trotterization for different numbers of sites. It compares sparse graphs (average degree 1) and denser graphs (average degree 3). On sparser graphs, the performance is much better. This is likely because the graph consists of several separate connected components, which means the Ising model can be seen as the disjoint union of two or more smaller models. In terms of the model counter, this means the formula ϕ can be split up as two formulae $\phi_1 \wedge \phi_2$ with disjoint variables. This performance increase is not inherently something model counters can take into account. This could also be taken into account at the Ising model level.

We compare our method with the SciPy `expm` method [10]. This calculates the exponential of a matrix using a method presented by Al-Mohy and Higham [29]. When using this method, no Trotterization is required, meaning there is no error-runtime tradeoff. There is also no inherent advantage to sparser graphs, as the space used to store the matrix is the same, and the method does not know about the relationships we encoded using Pauli-Z and Pauli-X matrices.

Figure 5.2 shows the performance of the `expm` method for different numbers of qubits. Overall, the performance of our method is generally worse or similar to the `expm` method. However, as the performance of model counters improves in the future, our method may become more useful.



(a) Average degree 1 **(b)** Average degree 3

Figure 5.1: Runtime vs. relative error of approximating the partition function of a random graph transverse-field Ising model for different average degrees, using Trotterization. Interaction strengths and external field strengths are from the standard normal distribution. Results are averaged over five runs. Comparison between different numbers of sites n. Only the runtime of the model counter itself is included. Uses the DPMC model counter.

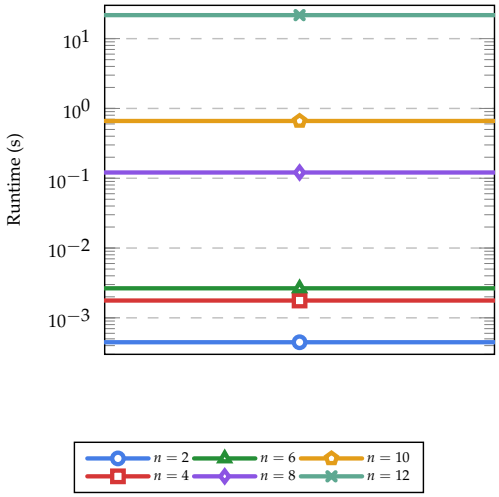


Figure 5.2: Runtime of the SciPy `expm` method when calculating the partition function of a random graph transverse-field Ising model for different average degrees. Drawn using the same scale as in Figure 5.1. Results are averaged over five runs.

Chapter 6

Potts model

We introduce another generalization of the Ising model, known as the Potts model. This generalization allows for more than two states per site, such that interaction strengths are defined separately for each possible pair of states, per pair of sites. The Potts model has applications in image segmentation [11, 21, 35] and has been used to study proteins [23, 24]. In these applications, finding the Gibbs distribution, which is normalized using the partition function, is crucial. In this chapter, we will show that there is a procedure similar to that shown in Chapter 4 for determining the Potts model partition function. We consider two cases: the generalized Potts model, which has a wider range of applications, and the standard Potts model, a more narrow definition that can be described more compactly than the generalized model when converted to a weighted model counting instance.

6.1. Definition

The Potts model, like the Ising model, is defined on some graph with interactions between sites. The difference between the two models lies in the interaction and external field functions J and h. The interaction function J now takes four parameters: Two that indicate the sites that have some interaction, and two that indicate the states of the two sites. The external field function h takes two parameters: The site on which the external field is applied, and the state of the site. Like for the Ising model, we call a Potts model ferromagnetic if J is non-negative and antiferromagnetic if J is strictly negative. Generally, it is assumed that a Potts model is ferromagnetic.

Definition 6.1: (Generalized) Potts model

A **(generalized) Potts model** is a tuple $P = (\Lambda, J, h, q)$ where Λ is a finite set representing vertices (sites), $q \in \mathbb{Z}_{\geq 2}$ is the number of states each site can be in. $J : \Lambda^2 \times \{0, \ldots, q-1\}^2 \to \mathbb{R}$ is a function indicating interaction strengths

between sites for different combinations of states, which is symmetric in the sense that $J(i,j,s_i,s_j) = J(j,i,s_j,s_i)$. $h : \Lambda \times \{0,\dots,q-1\} \to \mathbb{R}$ is a function returning the strength of an external field at each site. Like in the Ising model, it is conventional to write $J(i,j,s_i,s_j)$ as $J_{ij}(s_i,s_j)$ and $h(i,s_i)$ as $h_i(s_i)$.

A **configuration** is a function $s : \Lambda \to \{0,\dots,q-1\}$ that assignments every vertex a state. It is again conventional to write $s(i)$ as s_i.

The **Hamiltonian** $H_P : \mathrm{Map}(\Lambda, \{0,\dots,q-1\}) \to \mathbb{R}$ of a Potts model is a function from configurations to energies, defined as

$$H_P(s) = -\sum_{i,j\in\Lambda} J_{ij}(s_i,s_j) - \sum_{i\in\Lambda} h_i(s_i). \tag{6.1}$$

The **partition function**, at inverse temperature β, is defined as

$$Z_{\beta,P} = \sum_{s:\Lambda\to\{0,\dots,q-1\}} e^{-\beta H_P(s)}. \tag{6.2}$$

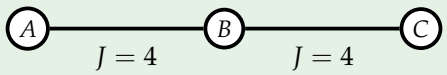
Although the generalized Potts model can be formulated as a weighted model counting problem, we will focus on a simpler version of the Potts model, called the standard Potts model. This is a model without an external field, and with an interaction between two sites only if they are in the same state. Interactions can only occur if two sites are neighbours in some graph over Λ. The interaction strength between connected pairs is kept constant across the model. The Hamiltonian of this model is

$$H_P(s) = -J \sum_{(i,j)\in E} \mathbb{1}\{s_i = s_j\}, \qquad E \subseteq \Lambda^2. \tag{6.3}$$

Here we have $J \geq 0$, i.e., the model is ferromagnetic.

Example 6.2

We will consider a standard Potts model with three sites, labeled A, B, and C. Assume there are three possible states for each site (i.e., $q = 3$). There are interactions between sites 1 and 2, and sites 2 and 3. Suppose the interaction strength J is 4.



The Hamiltonian is

$$H_P(s_A,s_B,s_C) = -4\mathbb{1}\{s_A = s_B\} - 4\mathbb{1}\{s_B = s_C\}. \tag{6.4}$$

The partition function at inverse temperature $\beta = 1$ is

$$Z_{\beta,P} = \sum_{s:\{A,B,C\}\to\{0,1,2\}} e^{-H_P(s_A,s_B,s_C)} \tag{6.5}$$

This sum has $3^3 = 27$ terms. There are three configurations s where all sites are given the same state, in which case $H_P(s) = -8$. There are $3 \cdot 2 \cdot 2 = 12$ configurations where sites A and C are not given the same state as site B, which means $H_P(s) = 0$. There are $3 \cdot 2 = 6$ configurations where sites A and B are assigned the same state, and site C is given a different state. Likewise, there are also 6 configurations where sites B and C are assigned the same state, and site A is given a different state. This makes for 12 configurations with Hamiltonian $H_P(s) = -4$. Adding all of these together to get the partition function yields

$$Z_{\beta,P} = 3e^{-(-8)} + 12e^{-0} + 12e^{-(-4)} \approx 9610.05 \tag{6.6}$$

6.2. Conversion of standard Potts model to WMC

Like in Section 4.3, we can reformulate the problem of finding the partition function to that of finding the trace of a matrix. We can instead write the Hamiltonian as a matrix

$$H_P = -J \sum_{(i,j)\in E} M_{ij} \tag{6.7}$$

The matrix M is a diagonal $q^2 \times q^2$ matrix $M = \sum_{k=0}^{q-1} |k,k\rangle \langle k,k|$. This can be interpreted as the entries on the diagonal being 1 if the two states are the same, and 0 otherwise. Like for the Ising model, the partition function is then equal to

$$Z_{\beta,P} = \operatorname{tr}\left(e^{-\beta H_P}\right) = \operatorname{tr}\left(\prod_{(i,j)\in E} e^{\beta J M_{ij}}\right) \tag{6.8}$$

The matrix $e^{\beta J M_{ij}}$ has entries 1 on the diagonal, except when the two states are the same, in which case there is an entry $e^{\beta J}$. This matrix can be encoded using WMC with $(z \leftrightarrow (x \leftrightarrow y), W, xy, xy)$, where the weight function $W : (\operatorname{var}(x) \cup \operatorname{var}(y) \cup \{z\}) \times \mathbb{B} \to \mathbb{R}$ is constant 1 except for the value $W(z) = e^{\beta J}$.

6.3. Experiments

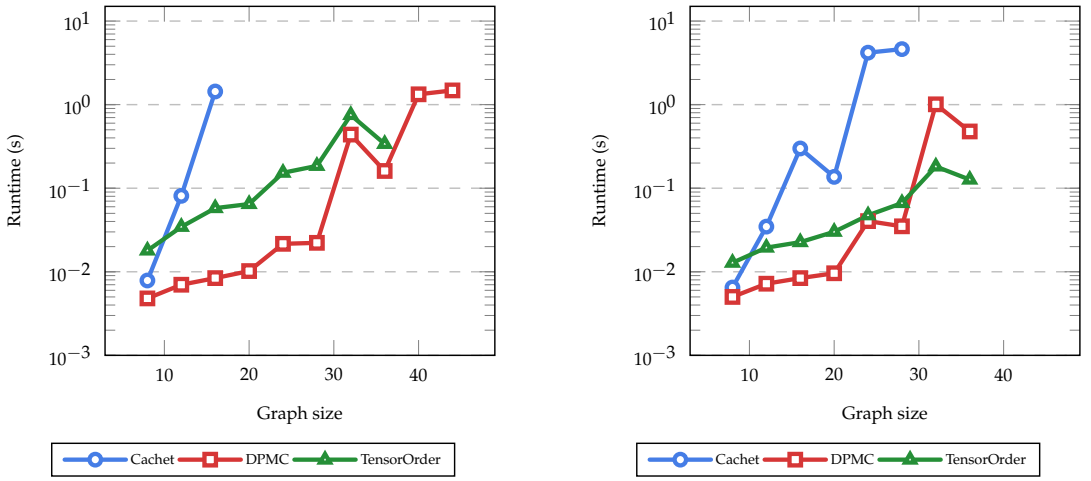
We now present two different experiments relating to the Potts model. First, we compare the performance of different model counters on this problem. Second, we

compare different ways of encoding the input and output variables. Unlike the Ising and transverse-field Ising model, different encodings behave differently in the Potts model, because the base size q of matrices is not necessarily equal to 2.

6.3.1. Comparing model counters

Figure 6.1 shows the performance of the model counters Cachet, DPMC, and TensorOrder when determining the partition function of random graph Potts models for different numbers of states q. The performance is similar to that of the Ising model in Figure 4.5, with DPMC outperforming TensorOrder for smaller problem sizes.

The performance gap between TensorOrder and DPMC is different between $q = 3$ and $q = 4$. For $q = 3$, DPMC outperformed TensorOrder for all graph sizes, while for $q = 4$, TensorOrder outperformed DPMC on larger problems. This may be caused by the logarithmic encoding used in this experiment. In the following, we will compare different encodings of the states in the Potts model using the DPMC solver.



(a) $q = 3$ **(b)** $q = 4$

Figure 6.1: Runtime of calculating the partition function of a random graph standard Potts model for numbers of states $q = 3, 4$. Interaction strength is taken from the standard normal distribution. Results are averaged over five runs. The expected degree of each node is four. Logarithmic variable encoding is used. Comparison between the model counters Cachet, DPMC, and TensorOrder.

6.3.2. Comparing variable encodings

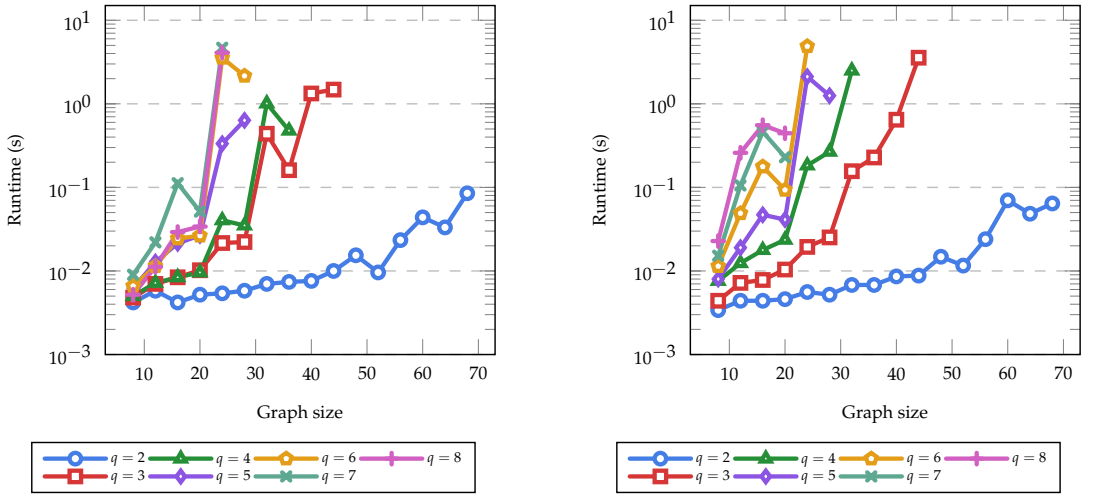
Comparing $q = 3$ with $q = 4$, one can see that all solvers are faster at calculating partition functions of Potts models with four states than at those with three states. This may be caused by the logarithmic encoding used in this experiment (see

Appendix A for more on variable encodings). All states are encoded with two variables v_0 and v_1 in both cases, but in the case of $q = 3$, an extra condition $\overline{v}_0 \vee \overline{v}_1$ is added to the CNF formula. Despite the sum in the partition function definition containing more terms, solvers handle this case better.

Figure 6.2 shows a comparison of the logarithmic encoding with an order encoding, which uses $q - 1$ variables to encode a state in a q-state Potts model (see Appendix A.2). Note that for $q = 2$, these two encodings are the same.

The logarithmic encoding outperforms the order encoding for larger q, although not significantly. Like with the cases $q = 3$ and $q = 4$, the runtimes for $q = 8$ are faster than those for $q = 7$ when using the logarithmic encoding. In the case of the logarithmic encoding, clear groups are also visible: Potts models with $q = 3$ and $q = 4$ have similar runtimes, as do Potts models with $q = 5, 6, 7, 8$. This is because these use the same number of variables per state in the logarithmic encoding, namely $\lceil \log_2 q \rceil$.

These groups are not visible in the order encoding. However, the order encoding still performs similarly to the logarithmic encoding in cases where q is not a power of two. It slightly outperforms the logarithmic encoding for $q = 3$. The gap in performance between the two encodings will likely grow as q increases. However, higher values of q are often of little use in practical applications. Hence, the order encoding remains relevant.



(a) Logarithmic encoding

(b) Order encoding

Figure 6.2: Runtime of calculating the partition function of a random graph standard Potts model for different numbers of states q and different variable encodings. Interaction strength is taken from the standard normal distribution. Results are averaged over five runs. The expected degree of each node is four. The solver used is DPMC.

6.4. Conversion of generalized Potts model to WMC

The calculation of the partition function of the generalized Potts model can be converted to a weighted model counting problem, just like in the case of the standard Potts model. However, because this model is so general, the encoding is much less efficient. We can write the Hamiltonian as a matrix

$$H_P = -\sum_{i,j\in\Lambda}\sum_{s_i,s_j\in\{0,\ldots,q-1\}} M(s_i,s_j)_{ij} - \sum_{i\in\Lambda}\sum_{s_i\in\{0,\ldots,q-1\}} N(s_i)_i. \tag{6.9}$$

The matrix $M(s_i, s_j)$ is a $q^2 \times q^2$ matrix with one non-zero entry, which is the diagonal entry corresponding to the states s_i and s_j at the two sites i and j. This value is equal to $J_{ij}(s_i, s_j)$. Similarly, $N(s_i)$ is a $q \times q$ matrix with one non-zero entry $h_i(s_i)$ on the diagonal, corresponding with the state s_i.

The trace of the matrix $e^{-\beta H_P}$ is the partition function, which can be written as a trace of a product of matrices:

$$Z_{\beta,P} = \mathrm{tr}\left(\left(\prod_{i,j\in\Lambda}\prod_{s_i,s_j\in\{0,\ldots,q-1\}} e^{\beta M(s_i,s_j)_{ij}}\right)\left(\prod_{i\in\Lambda}\prod_{s_i\in\{0,\ldots,q-1\}} e^{\beta N(s_i)_i}\right)\right) \tag{6.10}$$

The matrices $e^{\beta M(s_i,s_j)_{ij}}$ can be encoded with

$$(z \leftrightarrow (x = s_i \wedge y = s_j), W, xy, xy), \tag{6.11}$$

with $W : (\mathrm{var}(x) \cup \mathrm{var}(y) \cup \{z\}) \times \mathbb{B} \to \mathbb{R}$ constant 1 except for the value $W(z) = e^{\beta J_{ij}(s_i,s_j)}$. The matrices $e^{\beta N(s_i)_i}$ can be encoded similarly with

$$(z \leftrightarrow (x = s_i), W, x, x), \tag{6.12}$$

where $W : (\mathrm{var}(x) \cup \{z\}) \times \mathbb{B} \to \mathbb{R}$ is constant 1 except for the value $W(z) = e^{\beta h_i(s_i)}$.

6.4.1. Discussion

With few non-zero values $J_{ij}(s_i, s_j)$ and $h_i(s_i)$, this could be a fairly efficient encoding of the Potts model. However, in general, this encoding will become very inefficient as clauses are added for every non-zero value. This is in contrast to the standard Potts encoding, which uses only $x \leftrightarrow y$ to encode a sum of matrices $\sum_{s_i,s_j\in\{0,\ldots,q-1\}} M(s_i, s_j)_{ij}$ from the generalized Potts model. This could, depending on the variable encoding chosen, lead to a more compact CNF formula being passed to the model counter.

Chapter 7

Related work

We present several related works, including both theoretical and practical studies.

7.1. D-Hammer

Xu et al. [50] introduced D-Hammer: A tool that can check the equivalence of quantum expressions using labeled Dirac notation. For this, they introduce rewriting rules to normalize terms. The type system and syntax they use are similar to those we introduced in Chapter 3. This work is itself based on their earlier work on DiracDec [49], which uses plain Dirac notation. Their implementation of D-Hammer can be considered a generalization of ZX-calculus [9], extending it with various operations on Hilbert spaces. This comes at a performance cost on problems that can be encoded using both D-Hammer and the ZX-calculus.

The main differences between D-Hammer and our work are that we aim to evaluate an expression written in Dirac notation, rather than checking more general equivalences, and that we use weighted model counting as an intermediate layer to solve problems. Although our implementation does support the labeling of matrices, our theoretical framework does not.

7.2. Category theory

In general, monoidal categories (see [42] for the definition) can be used to provide a syntax for the sequential and parallel composition of matrices (i.e., multiplication and Kronecker product). This allows for the use of the rich field of category theory to be used in the language definition of Chapter 3. Villoria et al. [48] showed that, using enrichment, these operations can be extended to include other algebraic operations like convex combinations. They point out that this can be useful in simulating noise quantum circuits, for example. Using this technique, among others, the language defined in Chapter 3 could be described using category theory. We instead define the syntax explicitly, and then define semantics on this syntax.

7.3. Quantum circuit simulation using WMC

Mei et al. [26, 27] showed that model counting can be used in quantum computing for simulating circuits and equivalence checking. They showed that quantum states can be encoded using variables [28]. Gates can then be encoded by expressing a relationship between input and output variables, which are the states before and after the gate is applied. More recently, Zak et al. [52] extended this work by showing that model counting techniques can be used for the synthesis of quantum circuits. We also build on this work by generalizing the expression of quantum operators using weighted model counting, and consequently applying it to practical applications.

7.4. Ising model partition function

Nagy et al. [30] showed how the Ising model partition function problem can be converted to a WMC instance. They proved that existing model counters like TensorOrder [16] show competitive performance compared to existing techniques. In addition, they relate the problem of calculating the partition function to #CSP, using powerful theoretical tools to gain insights into where the hardness of the problem comes from. We reproduced some of the experimental results and performed experiments on the transverse-field Ising and Potts models. We did this while using the matrix representations instead of converting these problems directly to WMC.

7.5. Hamiltonian simulation using decision diagrams

Sander et al. [39] showed how Hamiltonian simulation can be performed using decision diagrams. At every node in the decision diagram, four outgoing edges represent the four different quadrants of a square $2^n \times 2^n$ matrix. This is then applied recursively to these quadrants. The edges contain weights, such that the value at a specific entry in the matrix can be found by traversing the decision diagram from the root to a leaf. Their technique of splitting up the matrix into quadrants is similar to our technique of using strings of input and output variables. We use a weighted model counting representation, where Sander et al. use decision diagrams.

7.6. Model counters

Model counters can employ several different techniques to calculate weighted model counts efficiently. The three solvers we used in this work are: (1) Cachet [40], an older tool that uses clause learning, (2) DPMC [14], which employs a dynamic programming technique, and (3) TensorOrder [16], which uses tensor-network contraction to solve WMC problems. Other successful model counters include

Ganak [43] and ProCount [15].

Chapter 8

Conclusion

In this work, we presented a framework for encoding quantum operators using Boolean formulae and weight functions. The framework can be used to convert problems written in Dirac notation to weighted model counting problems. The aim was to solve computationally difficult classical and quantum problems using weighted model counters, which have proven effective at a range of tasks in recent decades. We showed the effectiveness of our framework on the (quantum) transverse-field Ising model and the classical Potts model.

Our framework is able to represent arbitrary $q^n \times q^m$ matrices using weighted model counting. This is a significant improvement over previous work, which only provided methods for specific application areas, such as quantum circuit evaluation. For the first time, we formalized the concept of using weighted model counting for matrix calculations by giving a generic framework for performing common matrix operations on matrix representations. We have proven this framework yields correct outcomes mathematically, such that it can be used in future work without risk of erroneous results. In this future work, the focus can now lie more in finding the right matrices or operators to represent problems, rather than converting problems directly to WMC. We validated the effectiveness of our framework by calculating the partition function of the Ising model, as previously achieved by Nagy et al. [30] through direct conversion to WMC. We have shown that these two techniques yield the same result.

To the best of our knowledge, for the first time, we applied WMC to two new partition function problems: the transverse-field Ising model and the (classical) Potts model, by using our framework. The performance of the Potts model partition function calculation was comparable to that of the Ising model. Model counters were also effective at approximating the transverse-field partition function for smaller problem sizes. As model counters improve, we expect our technique to become more relevant in the realm of quantum problems.

8.1. Evaluation

At the start of this thesis, we stated the following research questions:

1. Is it possible to encode Dirac notation using weighted model counting using a general framework?
2. Can we calculate the partition functions of the transverse-field Ising and Potts models?

We answered the first question by providing a framework that can encode arbitrary $q^n \times q^m$ matrices, and providing rules for performing common matrix operations, such as matrix multiplication and addition, on these representations of matrices directly.

We answered the second research question by using our DiracWMC (see [13]) implementation of the framework in these practical applications. We showed that WMC can be used to calculate or approximate the partition functions of both the transverse-field Ising and the Potts model. On the Potts model, the performance was comparable to that of the Ising model, which was shown to be competitive with existing tools by Nagy et al. [30].

8.2. Future work

The matrix representations presented in this paper could be extended to tensors of arbitrary order. This could be achieved by adding more strings of variable encodings to the representations. Most operations presented in this work would be performed similarly, requiring only small extensions for the extra dimensions of the object. This would allow the technique to be applicable in a wider range of problems.

Furthermore, as mentioned in Chapter 3, the current way the matrices are represented does not allow for an addition operation that increases the size of the representation by a constant amount. This could be solved by using an alternative representation. However, it is unclear if an alternative representation would have a significant performance impact.

It may also be possible to build up a similar framework with monoidal categories (see Section 7.2) as the basis. This may provide a rich theoretical foundation to the framework, and as such, may simplify the proofs of its correctness. Although it may be possible to do this by extending the current framework, it is likely very involved. It may therefore be necessary to build a new framework from the ground up.

Further work may also be done on the complexity of the problems passed to the model counters. Some operations may yield a WMC instance that can be solved more efficiently when rewritten in a certain way, for example.

Lastly, similar problems to WMC, like weighted maximum weighted model counting, may also be used to encode matrices. This could pave the way to solving the ground-state problem using matrix representations.

Appendix A

Variable encodings

In this chapter, we describe several ways of encoding a q-state variable (A variable that can take values $\{0, \ldots, q-1\}$) using Boolean variables. There are several ways of doing this, each of which is useful in certain situations [1, 19, 31, 36, 46]. We will introduce several of these variable encodings that can be used in the matrix representations described in Chapter 3. For the matrix representations, we need support for several operations on these variable encodings. Hence, we introduce the following formal definition:

Definition A.1: Variable encoding

A **variable encoding** is a tuple $v = (q, V, =)$, with $q \in \mathbb{Z}_{\geq 2}$ the base of the encoding, V a set of variables the encoding uses, and $=: \{0, \ldots, q-1\} \to \mathrm{Form}(V)$ a function sending a value n to a formula over V that indicates the value of v is equal to n. We use the notation $v = n$. The following should be true:

1. $(v = n) \wedge (v = m) \equiv \bot$ for all $n \neq m$;
2. For every n, there exists exactly one $\tau : V \to \mathbb{B}$ such that $(v = n)[\tau]$ holds.

In addition to this definition, we also introduce notation for formulae that indicate an encoding "has a valid value" and that two encodings "have the same value". These formulae can often be simplified, as will be done in the sections discussing different encodings.

Notation A.2

For encodings $v = (q, V, =)$ and $w = (q, V', =)$, use the following notation:

- $q(v) = q$;

- $\mathrm{var}(v) = V$;
- $\mathrm{val}_v \equiv \bigvee_{n=0}^{q-1}(v = n)$;
- $v \leftrightarrow w \equiv \bigvee_{n=0}^{q-1}(v = n \wedge w = n)$.

Note that a string/tuple of encodings $x = x_{k-1} \dots x_0$ can represent numbers from $\{0, \dots, q^k - 1\}$ by using base-q expansions. We can then use the same notation as defined above for these strings, with:

$$x = n \equiv \bigwedge_{i=0}^{k-1}(x_i = n_i) \qquad \text{for } n = \sum_{i=0}^{k-1} q^i n_i \text{ with } 0 \leq n_i < q \tag{A.1}$$

$$\mathrm{var}(x) = \bigcup_{i=0}^{k-1} \mathrm{var}(x_i) \tag{A.2}$$

$$\mathrm{val}_x \equiv \bigwedge_{i=0}^{k-1} \mathrm{val}_{x_i} \tag{A.3}$$

$$x \leftrightarrow y \equiv \bigwedge_{i=0}^{k-1}(x_i \leftrightarrow y_i) \tag{A.4}$$

Below we list some example encodings. Note that in our implementation, some auxiliary variables may be introduced to make the Boolean formulae more compact in CNF form. However, the structure of the encodings is largely the same.

A.1. Logarithmic encoding

First, we introduce an encoding that uses a logarithmic number of variables relative to the base q [36]. To be precise, we use variables $v_0, \dots, v_{k-1}$, where $k = \lceil \log_2 q \rceil$. Naturally, the set of used variables of an encoding v is

$$\mathrm{var}(v) = \{v_0, \dots, v_{k-1}\}. \tag{A.5}$$

The equality formula for a number n is a cube (conjunction of literals) where v_i or $\overline{v}_i$ is present depending on whether n_i from the binary representation $n = \sum_{i=0}^{k=1} 2^i n_i$ is equal to 1 or 0 respectively.

The formula val_v can be rewritten as follows, making use of the binary representation $q - 1 = \sum_{i=0}^{k-1} 2^i q_i$:

$$\mathrm{val}_v \equiv \bigwedge_{\substack{i \in \{0,\dots,k-1\} \\ q_i = 0}} \left(\overline{v}_i \vee \bigvee_{\substack{j \in \{i+1,\dots,k-1\} \\ q_j = 1}} \overline{v}_j \right). \tag{A.6}$$

For some other base-q encoding w with variables $w_0, \ldots, w_{k-1}$, the formula $v \leftrightarrow w$ can be rewritten by comparing all variables separately:

$$v \leftrightarrow w \equiv \bigwedge_{i=0}^{k-1} (v_i \leftrightarrow w_i). \tag{A.7}$$

A.2. Order encoding

An alternative encoding, which is beneficial in some cases for SAT solvers and model counters, is an order encoding [1]. This encoding uses $q-1$ variables $v_0, \ldots, v_{q-2}$, where each variable v_i should be true if the represented number is strictly larger than i. A big advantage of this representation is the compact formula for val_v, which is a conjunction of implications, making sure no false value comes before a true one:

$$\text{val}_v \equiv \bigwedge_{i=1}^{k-1} (v_i \rightarrow v_{i-1}). \tag{A.8}$$

For another base-q encoding w using variables $w_0, \ldots, w_{k-1}$, we can again check for equality by checking all variables have the same value:

$$v \leftrightarrow w \equiv \bigwedge_{i=0}^{k-1} (v_i \leftrightarrow w_i). \tag{A.9}$$

A.3. One-hot encoding

Finally, we introduce a one-hot encoding, also known as a direct encoding [36]. The encoding requires exactly one out of q variables to be true. The set of variables is $\text{var}(v) = \{v_0, \ldots, v_{q-1}\}$. The validity formula makes sure exactly one variable is true, which can be done with

$$\text{val}_v \equiv \left(\bigvee_{i=0}^{q-1} v_i \right) \wedge \bigwedge_{i \neq j} (\overline{v}_i \vee \overline{v}_j). \tag{A.10}$$

Again, equality of two encodings can be checked by checking if all variables have the same value:

$$v \leftrightarrow w \equiv \bigwedge_{i=0}^{k-1} (v_i \leftrightarrow w_i). \tag{A.11}$$

Note that there are more efficient methods similar to this encoding [19, 31]. From these methods, only the addition of auxiliary variables has been (partially) used in our implementation of the matrix representations.

Appendix B

Correctness of representation denotational semantics

In this chapter we prove Theorem 3.21, which states that the semantics $[\![\cdot]\!]_r$ are well-defined and that $\text{rep}^\# \circ [\![\cdot]\!]_r = [\![\cdot]\!]_v$. We do this using induction on the expression type proof tree. Each separate rule has its own Lemma below. Since $[\![\cdot]\!]_v$ is defined as evaluating the expression using the usual rules, we refrain from mentioning $[\![\cdot]\!]_v$ explicitly in the remainder of this chapter.

Section B.1 lists some general properties of weighted model counting. The proofs in Section B.2 rely on these properties. The proof of Theorem 3.6 is split up into Lemmas, one for every rule in Section 3.5.

B.1. Properties of WMC

Lemma B.1

Let ϕ_1 and ϕ_2 be Boolean formulae over sets of variables V_1 and V_2, respectively. Let $W_1 : V_1 \times \mathbb{B} \to \mathbb{F}$ and $W_2 : V_2 \times \mathbb{B} \to \mathbb{F}$ be weight functions. If $V_1 \cap V_2 = \varnothing$ we have

$$\text{WMC}(\phi_1 \wedge \phi_2, W_1 \cup W_2) = \text{WMC}(\phi_1, W_1) \cdot \text{WMC}(\phi_2, W_2) \tag{B.1}$$

Proof. Write $S = \text{WMC}(\phi_1 \wedge \phi_2, W_1 \cup W_2)$. We have

$$S = \sum_{\tau : V_1 \cup V_2 \to \mathbb{B}} (\phi_1 \wedge \phi_2)[\tau] \prod_{v \in V_1 \cup V_2} (W_1 \cup W_2)(v, \tau(v)) \tag{B.2}$$

The function τ can be split up into two functions τ_1 and τ_2 over the two domains V_1 and V_2, respectively. Since ϕ_1 only contains variables from V_1, we have $\phi_1[\tau] =$

$\phi_1[\tau_1]$, and likewise for ϕ_2.

$$S = \sum_{\tau_1 : V_1 \to \mathbb{B}} \sum_{\tau_2 : V_2 \to \mathbb{B}} \phi_1[\tau_1]\phi_2[\tau_2] \left(\prod_{v \in V_1} W_1(v, \tau_1(v)) \right) \left(\prod_{v \in V_2} W_2(v, \tau_2(v)) \right) \tag{B.3}$$

$$= \left(\sum_{\tau_1 : V_1 \to \mathbb{B}} \phi_1[\tau_1] \prod_{v \in V_1} W_1(v, \tau_1(v)) \right) \left(\sum_{\tau_2 : V_2 \to \mathbb{B}} \phi_2[\tau_2] \prod_{v \in V_2} W_2(v, \tau_2(v)) \right) \tag{B.4}$$

$$= \mathrm{WMC}(\phi_1, W_1) \cdot \mathrm{WMC}(\phi_2, W_2) \tag{B.5}$$

This proves the lemma. □

Lemma B.2

Let ϕ_1 and ϕ_2 be Boolean formulae over a set of variables V, with $\phi_1 \wedge \phi_2 \equiv \bot$ (i.e., $\phi_1 \wedge \phi_2$ is unsatisfiable). Let $W : V \times \mathbb{B} \to \mathbb{F}$ be a weight function. Then

$$\mathrm{WMC}(\phi_1 \vee \phi_2, W) = \mathrm{WMC}(\phi_1, W) + \mathrm{WMC}(\phi_2, W) \tag{B.6}$$

Proof. Writing out the definition, we have

$$\mathrm{WMC}(\phi_1 \vee \phi_2, W) = \sum_{\tau : V \to \mathbb{B}} (\phi_1 \vee \phi_2)[\tau] \prod_{v \in V} W(v, \tau(v)) \tag{B.7}$$

Because ϕ_1 and ϕ_2 cannot be satisfied at the same time, we can write $(\phi_1 \vee \phi_2)[\tau] = \phi_1[\tau] + \phi_2[\tau]$. Taking this outside of the entire sum gives the desired result. □

Lemma B.3

Let ϕ be a Boolean formula, $W : V \times \mathbb{B} \to \mathbb{F}$ a weight function, and $v \in V$ a variable. Then

$$\mathrm{WMC}(\phi, W) = \mathrm{WMC}(\phi \wedge \overline{v}, W) + \mathrm{WMC}(\phi \wedge v, W) \tag{B.8}$$

Proof. This follows from Lemma B.2 by using $\phi \equiv (\phi \wedge \overline{v}) \vee (\phi \wedge v)$ □

Lemma B.4

Let ϕ be a Boolean formula, $W : V \times \mathbb{B} \to \mathbb{F}$ a weight function, and $f : \mathbb{F} \to \mathbb{F}$ a field endomorphism. Then

$$\mathrm{WMC}(\phi, f \circ W) = f(\mathrm{WMC}(\phi, W)) \tag{B.9}$$

Proof. Since f is a field endomorphism, it has properties $f(x + y) = f(x) + f(y)$ and $f(xy) = f(x)f(y)$. This means

$$\mathrm{WMC}(\phi, f \circ W) = \sum_{\tau : V \to \mathbb{B}} \phi[\tau] \prod_{v \in V} (f \circ W)(v, \tau(v)) \tag{B.10}$$

$$= \sum_{\tau: V \to \mathbb{B}} \phi[\tau] \cdot f\left(\prod_{v \in V} W(v, \tau(v))\right). \tag{B.11}$$

Note that $\phi[\tau] \in \{0, 1\}$. If $\phi[\tau] = 0$, for any x we have $f(\phi[\tau] \cdot x) = 0 = f(0) = \phi[\tau] \cdot f(x)$. If $\phi[\tau] = 1$ we have the same property: $f(\phi[\tau] \cdot x) = f(x) = \phi[\tau] \cdot f(x)$. Therefore, we can rewrite the equation above as

$$\begin{aligned}
\mathrm{WMC}(\phi, f \circ W) &= \sum_{\tau: V \to \mathbb{B}} f\left(\phi[\tau] \cdot \prod_{v \in V} W(v, \tau(v))\right) && \text{(B.12)} \\
&= f\left(\sum_{\tau: V \to \mathbb{B}} \phi[\tau] \cdot \prod_{v \in V} W(v, \tau(v))\right) && \text{(B.13)} \\
&= f(\mathrm{WMC}(\phi, W)) && \text{(B.14)}
\end{aligned}$$

□

B.2. Correctness proof

Combining the Lemmas below with induction on type derivation trees proves Theorem 3.21.

Lemma B.5: Scalar constant

Let $\alpha \in \mathbb{F}$. Then $[\![\alpha]\!]_r$ is well-defined and $\mathrm{rep}^{\#}([\![\alpha]\!]_r) = \alpha$.

Proof. The fact that $[\![\alpha]\!]_r$ is well-defined follows directly from the definition. For $W : \{x\} \times \mathbb{B} \to \mathbb{F}$ constant α and $\phi \equiv x$, we have

$$\begin{aligned}
\mathrm{rep}^{\#}([\![\alpha]\!]_r) &= \mathrm{rep}^{\#}[(\phi, W)] && \text{(B.15)} \\
&= \mathrm{rep}(\phi, W) && \text{(B.16)} \\
&= \mathrm{WMC}(\phi, W) && \text{(B.17)} \\
&= \mathrm{sat}(\phi \wedge \overline{x}) \cdot W(\overline{x}) + \mathrm{sat}(\phi \wedge x) \cdot W(x) && \text{(B.18)} \\
&= W(x) && \text{(B.19)} \\
&= \alpha && \text{(B.20)}
\end{aligned}$$

□

Lemma B.6: Scalar multiplication

Let s_1 and s_2 be expressions of type $\mathcal{S}$. Suppose $[\![s_1]\!]_r$ and $[\![s_2]\!]_r$ are well-defined. Then $[\![s_1 \cdot s_2]\!]_r$ is well-defined, and

$$\mathrm{rep}^{\#}([\![s_1 \cdot s_2]\!]_r) = \mathrm{rep}^{\#}([\![s_1]\!]_r) \cdot \mathrm{rep}^{\#}([\![s_2]\!]_r) \tag{B.21}$$

Proof. Suppose $[\![s_1]\!]_r = [(\phi_1, W_1)]$ and $[\![s_2]\!]_r = [(\phi_2, W_2)]$ with the domains of W_1 and W_2 disjoint. Then by definition

$$[\![s_1 \cdot s_2]\!]_r = [(\phi_1 \wedge \phi_2, W_1 \cup W_2)] \tag{B.22}$$

We show that this expression is well-defined by showing the value of $\text{rep}^{\#}([\![s_1 \cdot s_2]\!]_r)$ is independent of the choice of ϕ_1, ϕ_2, W_1, and W_2. We have

$$\text{rep}^{\#}([\![s_1 \cdot s_2]\!]_r) = \text{WMC}(\phi_1 \wedge \phi_2, W_1 \cup W_2) \tag{B.23}$$

Since W_1 and W_2 have non-overlapping domains and ϕ_1 and ϕ_2 have variables in the domains of W_1 and W_2 respectively, Lemma B.1 gives

$$\begin{aligned} \text{rep}^{\#}([\![s_1 \cdot s_2]\!]_r) &= \text{WMC}(\phi_1, W_1) \cdot \text{WMC}(\phi_2, W_2) && \text{(B.24)} \\ &= \text{rep}^{\#}([(\phi_1, W_1)]) \cdot \text{rep}^{\#}([(\phi_2, W_2)]) && \text{(B.25)} \\ &= \text{rep}^{\#}([\![s_1]\!]_r) \cdot \text{rep}^{\#}([\![s_2]\!]_r) && \text{(B.26)} \end{aligned}$$

By assumption, the above expression is well-defined. □

Lemma B.7: Scalar addition

Let s_1 and s_2 be expressions of type $\mathcal{S}$. Suppose $[\![s_1]\!]_r$ and $[\![s_2]\!]_r$ are well-defined. Then $[\![s_1 + s_2]\!]_r$ is well-defined, and

$$\text{rep}^{\#}([\![s_1 + s_2]\!]_r) = \text{rep}^{\#}([\![s_1]\!]_r) + \text{rep}^{\#}([\![s_2]\!]_r) \tag{B.27}$$

Proof. Suppose $[\![s_1]\!]_r = [(\phi_1, W_1)]$ and $[\![s_2]\!]_r = [(\phi_2, W_2)]$ with the domains of W_1 and W_2 disjoint, $\text{WMC}(\top, W_1) \neq 0$, and $\text{WMC}(\top, W_2) \neq 0$. By definition

$$[\![s_1 + s_2]\!]_r = [((\bar{c} \to \phi_1) \wedge (c \to \phi_2), W_1 \cup W_2 \cup W_c)] \tag{B.28}$$

with $W_c : \{c\} \times \mathbb{B} \to \mathbb{F}$ defined by $W_c(c) = 1/\text{WMC}(\top, W_1)$ and $W_c(\bar{c}) = 1/\text{WMC}(\top, W_2)$. Again, we show that this is well-defined by showing $\text{rep}^{\#}([\![s_1 + s_2]\!]_r)$ yields the same value independent of the formulae and weight functions chosen at the start.

Write $W = W_1 \cup W_2 \cup W_c$ and $\psi \equiv (\bar{c} \to \phi_1) \wedge (c \to \phi_2)$. Then

$$\text{rep}^{\#}([\![s_1 + s_2]\!]_r) = \text{WMC}(\psi, W) \tag{B.29}$$

By Lemma B.3 we have

$$\text{rep}^{\#}([\![s_1 + s_2]\!]_r) = \text{WMC}(\psi \wedge \bar{c}, W) + \text{WMC}(\psi \wedge c, W) \tag{B.30}$$

We will show that $\text{WMC}(\psi \wedge \bar{c}, W) = \text{rep}^{\#}([\![s_1]\!]_r)$, the case $\text{WMC}(\psi \wedge c, W)$ is symmetric. Note that $\psi \wedge \bar{c} \equiv \phi_1 \wedge \bar{c}$. Since ϕ_1 contains only variables variables in the domain of W_1, Lemma B.1 gives

$$\text{WMC}(\psi \wedge \bar{c}, W) = \text{WMC}(\phi_1 \wedge \bar{c}, W) \tag{B.31}$$

$$= \mathrm{WMC}(\phi_1, W_1) \cdot \mathrm{WMC}(\top, W_2) \cdot \mathrm{WMC}(\bar{c}, W_c) \tag{B.32}$$

$$= \mathrm{WMC}(\phi_1, W_1) \cdot \mathrm{WMC}(\top, W_2) \cdot \frac{1}{\mathrm{WMC}(\top, W_2)} \tag{B.33}$$

$$= \mathrm{WMC}(\phi_1, W_1) \tag{B.34}$$

Similarly it can be shown that $\mathrm{WMC}(\psi \wedge c, W) = \mathrm{WMC}(\phi_2, W_2)$, which means

$$\begin{aligned}
\mathrm{rep}^{\#}([\![s_1 + s_2]\!]_r) &= \mathrm{WMC}(\phi_1, W_1) + \mathrm{WMC}(\phi_2, W_2) \\
&= \mathrm{rep}^{\#}[(\phi_1, W_1)] + \mathrm{rep}^{\#}[(\phi_2, W_2)] \\
&= \mathrm{rep}^{\#}([\![s_1]\!]_r) + \mathrm{rep}^{\#}([\![s_2]\!]_r)
\end{aligned}$$

□

Lemma B.8: Field endomorphism on a scalar

Let s be an expression of type $\mathcal{S}$ and $f : \mathbb{F} \to \mathbb{F}$ a field endomorphism. Suppose $[\![s]\!]_r$ is well-defined. Then $[\![\mathsf{apply}(f, s)]\!]_r$ is well-defined, and

$$\mathrm{rep}^{\#}([\![\mathsf{apply}(f, s)]\!]_r) = f(\mathrm{rep}^{\#}([\![s]\!]_r)) \tag{B.35}$$

Proof. Suppose $[\![s]\!]_r = [(\phi, W)]$. By definition,

$$[\![\mathsf{apply}(f, s)]\!]_r = [(\phi, f \circ W)] \tag{B.36}$$

We show this is well-defined by showing the value of the expression below is independent of the choice of ϕ and W:

$$\mathrm{rep}^{\#}([\![\mathsf{apply}(f, s)]\!]_r) = \mathrm{WMC}(\phi, f \circ W) \tag{B.37}$$

It follows from Lemma B.4 that

$$\mathrm{rep}^{\#}([\![\mathsf{apply}(f, s)]\!]_r) = f(\mathrm{WMC}(\phi, W)) = f(\mathrm{rep}^{\#}([\![s]\!]_r)) \tag{B.38}$$

This completes the proof. □

Lemma B.9: Bra and ket

Let $q \in \mathbb{Z}_{\geq 2}$ and $0 \leq i < q$. Then $[\![\mathsf{bra}(i, q)]\!]_r$ and $[\![\mathsf{ket}(i, q)]\!]_r$ are well-defined and

$$\mathrm{rep}^{\#}([\![\mathsf{bra}(i, q)]\!]_r) = \langle i|_q \tag{B.39}$$

$$\mathrm{rep}^{\#}([\![\mathsf{ket}(i, q)]\!]_r) = |i\rangle_q \tag{B.40}$$

Proof. We will prove the correctness of the bra. The case of ket is symmetric. The semantics are well-defined by definition.

We need to show that $\text{rep}^{\#}([\![\text{bra}(i,q)]\!]_r)$ is a row vector with a 1 at entry i and 0 everywhere else. This can be done by verifying that, for every $0 \leq j < q$, we have

$$\text{rep}^{\#}([\![\text{bra}(i,q)]\!]_r)\,|j\rangle = \mathbb{1}\{i = j\} \tag{B.41}$$

Note that we have

$$[\![\text{bra}(i,q)]\!]_r = [(x = i, W_1, x, -, q)] \tag{B.42}$$

with x a q-state variable encoding and $W : \text{var}(x) \times \mathbb{B} \to \mathbb{F}$ constant 1. Using the definition of matrix representations, we have

$$\text{rep}^{\#}([\![\text{bra}(i,q)]\!]_r)\,|j\rangle = \text{WMC}(x = i \wedge x = j, W_1) = \mathbb{1}\{i = j\} \tag{B.43}$$

This proves the lemma. □

Lemma B.10: Matrix product

Let M_1 and M_2 be expressions with types $\mathcal{M}(q, m \to k)$ and $\mathcal{M}(q, k \to n)$ respectively. Suppose $[\![M_1]\!]_r$ and $[\![M_2]\!]_r$ are well-defined. Then $[\![M_2 \cdot M_1]\!]_r$ is well-defined, and

$$\text{rep}^{\#}([\![M_2 \cdot M_1]\!]_r) = \text{rep}^{\#}([\![M_2]\!]_r) \cdot \text{rep}^{\#}([\![M_1]\!]_r) \tag{B.44}$$

Proof. Suppose we have

$$[\![M_1]\!]_r = [(\phi_1, W_1, x, y, q)] \tag{B.45}$$
$$[\![M_2]\!]_r = [(\phi_2, W_2, y, z, q)] \tag{B.46}$$

with $\text{dom}(W_1) \cap \text{dom}(W_2) = \text{var}(y)$. We will show that $[\![M_2 \cdot M_1]\!]_r$ is well-defined by showing that the result of $\text{rep}^{\#}([\![M_2 \cdot M_1]\!]_r)$ is independent of the choice of formulae and weight functions earlier. We can prove the lemma by showing that, for all $0 \leq i < q^n$ and $0 \leq j < q^m$, we have

$$\langle i|\,\text{rep}^{\#}([\![M_2 \cdot M_1]\!]_r)\,|j\rangle = \langle i|\,\text{rep}^{\#}([\![M_2]\!]_r) \cdot \text{rep}^{\#}([\![M_1]\!]_r)\,|j\rangle \tag{B.47}$$

By definition, we have

$$[\![M_2 \cdot M_1]\!]_r = [\phi_1 \wedge \phi_2 \wedge \text{val}_y, W_1 \cdot W_2, x, z, q] \tag{B.48}$$

Furthermore,

$$\langle i|\,\text{rep}^{\#}([\![M_2 \cdot M_1]\!]_r)\,|j\rangle = \text{WMC}(\phi_1 \wedge \phi_2 \wedge \text{val}_y \wedge x = j \wedge z = i, W_1 \cdot W_2) \tag{B.49}$$

Using the property $\text{val}_v \equiv \bigwedge_{a=0}^{q^k-1}(v = a)$ and Lemma B.2, we have

$$\langle i|\,\text{rep}^{\#}([\![M_2 \cdot M_1]\!]_r)\,|j\rangle = \sum_{a=0}^{q^k-1} \text{WMC}(\phi_1 \wedge \phi_2 \wedge y = a \wedge x = j \wedge z = i, W_1 \cdot W_2) \tag{B.50}$$

Since the only overlap $\phi_1 \wedge x = j$ and $\phi_2 \wedge z = i$ have is $\text{var}(y)$, which is the only overlap in the domains of W_1 and W_2, and all variables in $\text{var}(y)$ are fixed by $y = a$, we can rewrite this as

$$\begin{aligned}
\langle i|\,\text{rep}^{\#}([\![M_2 \cdot M_1]\!]_r)\,|j\rangle &= \sum_{a=0}^{q^k-1} \text{WMC}(\phi_1 \wedge x = j \wedge y = a, W_1) && \text{(B.51)}\\
&\quad \cdot \text{WMC}(\phi_2 \wedge z = i \wedge y = a, W_2) && \text{(B.52)}\\
&= \sum_{a=0}^{q^k-1} \langle a|\,\text{rep}^{\#}([\![M_1]\!]_r)\,|j\rangle\,\langle i|\,\text{rep}^{\#}([\![M_2]\!]_r)\,|a\rangle && \text{(B.53)}\\
&= \sum_{a=0}^{q^k-1} \langle i|\,\text{rep}^{\#}([\![M_2]\!]_r)\,|a\rangle\,\langle a|\,\text{rep}^{\#}([\![M_1]\!]_r)\,|j\rangle && \text{(B.54)}\\
&= \langle i|\,\text{rep}^{\#}([\![M_2]\!]_r) \cdot \text{rep}^{\#}([\![M_1]\!]_r)\,|j\rangle && \text{(B.55)}
\end{aligned}$$

This proves the lemma. □

Lemma B.11: Matrix sum

Let M_1 and M_2 be expressions with type $\mathcal{M}(q, m \to n)$. Suppose $[\![M_1]\!]_r$ and $[\![M_2]\!]_r$ are well-defined. Then $[\![M_1 + M_2]\!]_r$ is also well-defined, and

$$\text{rep}^{\#}([\![M_1 + M_2]\!]_r) = \text{rep}^{\#}([\![M_1]\!]_r) + \text{rep}^{\#}([\![M_2]\!]_r) \tag{B.56}$$

Proof. Suppose that

$$[\![M_1]\!]_r = [(\phi_1, W_1, x_1, y_1, q)] \tag{B.57}$$
$$[\![M_2]\!]_r = [(\phi_2, W_2, x_2, y_2, q)] \tag{B.58}$$

with the domains of W_1 and W_2 disjoint, $\text{WMC}(\top, W_1) \neq 0$, and $\text{WMC}(\top, W_2) \neq 0$. Let c be a Boolean variable and x and y strings of variable encodings, of the same lengths as x_1 (or x_2) and y_1 (or y_2), respectively. Let these be chosen in such a way that neither c nor the variables in x and y are contained in either of the domains of W_1 and W_2. We have defined

$$[\![M_1 + M_2]\!]_r = [(\phi, W_1 \cup W_2 \cup W_c \cup W_{xy}, x, y, q)] \tag{B.59}$$

with

$$\begin{aligned}
\phi \equiv\ & (\bar{c} \to ((x \leftrightarrow x_1) \wedge (y \leftrightarrow y_1) \wedge \phi_1))\\
& \wedge (c \to ((x \leftrightarrow x_2) \wedge (y \leftrightarrow y_2) \wedge \phi_2))
\end{aligned} \tag{B.60}$$

and $W_c : \{c\} \times \mathbb{B} \to \mathbb{F}$ and $W_{xy} : (\text{var}(x) \cup \text{var}(y)) \times \mathbb{B} \to \mathbb{F}$ defined by $W_c(c) = 1/\text{WMC}(\top, W_1)$, $W_c(\bar{c}) = 1/\text{WMC}(\top, W_2)$, and W_{xy} constant 1.

We will show that $[\![M_1 + M_2]\!]_r$ is well-defined by showing $\text{rep}^\#([\![M_1 + M_2]\!]_r)$ yields the same value, independent of the choice of representations at the start of the proof. Let $0 \leq i < q^n$ and $0 \leq j < q^m$. Write $W = W_1 \cup W_2 \cup W_c \cup W_{xy}$ and $\psi \equiv \phi \wedge x = j \wedge y = i$. We have

$$\langle i| \text{rep}^\#([\![M_1 + M_2]\!]_r) |j\rangle = \text{WMC}(\psi, W) \tag{B.61}$$

We will show that $\text{WMC}(\psi \wedge \bar{c}, W) = \langle i| \text{rep}^\#([\![M_1]\!]_r) |j\rangle$, the case $\text{WMC}(\psi \wedge c, W)$ is symmetric. Note that

$$\psi \wedge \bar{c} \equiv (x \leftrightarrow x_1) \wedge (y \leftrightarrow y_1) \wedge \phi_1 \wedge x = j \wedge y = i \wedge \bar{c} \tag{B.62}$$

Since ϕ_1 does not contain the variable c, Lemma B.1 gives

$$\begin{aligned}
\text{WMC}(\psi \wedge \bar{c}, W) &= \text{WMC}(\psi, W_1 \cup W_{xy}) \cdot \text{WMC}(\top, W_2) \cdot \text{WMC}(\bar{c}, W_c) && \text{(B.63)}\\
&= \text{WMC}(\psi, W_1 \cup W_{xy}) \cdot \text{WMC}(\top, W_2) \cdot \frac{1}{\text{WMC}(\top, W_2)} && \text{(B.64)}\\
&= \text{WMC}(\psi, W_1 \cup W_{xy}) && \text{(B.65)}
\end{aligned}$$

We can rewrite ψ to

$$\psi \equiv \phi_1 \wedge x = j \wedge x_1 = j \wedge y = i \wedge y_1 = i \tag{B.66}$$

Since ψ_1 only contains variables in the domain of W_1, we can rewrite further to

$$\text{WMC}(\psi \wedge \bar{c}, W) = \text{WMC}(\phi_1 \wedge x_1 = j \wedge y_1 = i, W_1) \cdot \text{WMC}(x = j \wedge y = i, W_{xy}) \tag{B.67}$$

Note that there is exactly one satisfying assignment of $x = j \wedge y = i$, which means the term on the right is 1, which means

$$\begin{aligned}
\text{WMC}(\psi \wedge \bar{c}, W) &= \text{WMC}(\phi \wedge x_1 = j \wedge y_1 = i, W_1) && \text{(B.68)}\\
&= \langle i| \text{rep}^\#[(\phi, W, x_1, y_1, q)] |j\rangle && \text{(B.69)}\\
&= \langle i| \text{rep}^\#([\![M_1]\!]_r) |j\rangle && \text{(B.70)}
\end{aligned}$$

Similarly, it can be proven that

$$\text{WMC}(\psi \wedge c, W) = \langle i| \text{rep}^\#([\![M_2]\!]_r) |j\rangle \tag{B.71}$$

Combining these with Lemma B.3 gives

$$\begin{aligned}
\langle i| \text{rep}^\#([\![M_1 + M_2]\!]_r) |j\rangle &= \text{WMC}(\psi, W) && \text{(B.72)}\\
&= \text{WMC}(\psi \wedge c, W) + \text{WMC}(\psi \wedge \bar{c}) && \text{(B.73)}\\
&= \langle i| \text{rep}^\#([\![M_1]\!]_r) |j\rangle + \langle i| \text{rep}^\#([\![M_2]\!]_r) |j\rangle && \text{(B.74)}\\
&= \langle i| (\text{rep}^\#([\![M_1]\!]_r) + \text{rep}^\#([\![M_2]\!]_r)) |j\rangle && \text{(B.75)}
\end{aligned}$$

This proves the lemma. □

Lemma B.12: Kronecker product

Let M_1 and M_2 be expressions with types $\mathcal{M}(q, m_1 \to n_1)$ and $\mathcal{M}(q, m_2 \to m_1)$ respectively. Suppose $[\![M_1]\!]_r$ and $[\![M_2]\!]_r$ are well-defined. Then $[\![M_1 \otimes M_2]\!]_r$ is well-defined, and

$$\operatorname{rep}^{\#}([\![M_1 \otimes M_2]\!]_r) = \operatorname{rep}^{\#}([\![M_1]\!]_r) \otimes \operatorname{rep}^{\#}([\![M_2]\!]_r) \tag{B.76}$$

Proof. Suppose we have

$$[\![M_1]\!]_r = [(\phi_1, W_1, x_1, y_1, q)] \tag{B.77}$$
$$[\![M_2]\!]_r = [(\phi_2, W_2, x_2, y_2, q)] \tag{B.78}$$

with $\operatorname{dom}(W_1) \cap \operatorname{dom}(W_2) = \varnothing$. Then, by definition,

$$[\![M_1 \otimes M_2]\!]_r = [(\phi_1 \wedge \phi_2, W_1 \cup W_2, x_1 x_2, y_1 y_2, q)] \tag{B.79}$$

We show that this is well-defined by showing that the value of $\operatorname{rep}^{\#}([\![M_1 \otimes M_2]\!]_r)$ is independent of the choice of the representations at the start of the proof. We need to show that, for all $0 \le i_1 < q^{n_1}$, $0 \le i_2 < q^{n_2}$, $0 \le j_1 < q^{m_1}$, and $0 \le j_2 < q^{m_2}$:

$$\langle i_1 i_2 | \operatorname{rep}^{\#}([\![M_1 \otimes M_2]\!]_r) | j_1 j_2 \rangle = \langle i_1 i_2 | (\operatorname{rep}^{\#}([\![M_1]\!]_r) \otimes \operatorname{rep}^{\#}([\![M_2]\!]_r)) | j_1 j_2 \rangle \tag{B.80}$$

We have

$$\langle i_1 i_2 | \operatorname{rep}^{\#}([\![M_1 \otimes M_2]\!]_r) | j_1 j_2 \rangle \tag{B.81}$$
$$= \operatorname{WMC}(\phi_1 \wedge \phi_2 \wedge x_1 x_2 = j_1 j_2 \wedge y_1 y_2 = i_1 i_2, W_1 \cup W_2) \tag{B.82}$$
$$= \operatorname{WMC}(\phi_1 \wedge \phi_2 \wedge x_1 = j_1 \wedge x_2 = j_2 \wedge y_1 = i_1 \wedge y_2 = i_2, W_1 \cup W_2) \tag{B.83}$$

Lemma B.1 gives

$$\langle i_1 i_2 | \operatorname{rep}^{\#}([\![M_1 \otimes M_2]\!]_r) | j_1 j_2 \rangle \tag{B.84}$$
$$= \operatorname{WMC}(\phi_1 \wedge x_1 = j_1 \wedge y_1 = i_1, W_1) \operatorname{WMC}(\phi_2 \wedge x_2 = j_2 \wedge y_2 = i_2, W_2) \tag{B.85}$$
$$= \langle i_1 | \operatorname{rep}^{\#}([\![M_1]\!]_r) | j_1 \rangle \langle i_2 | \operatorname{rep}^{\#}([\![M_2]\!]_r) | j_2 \rangle \tag{B.86}$$
$$= \langle i_1 i_2 | (\operatorname{rep}^{\#}([\![M_1]\!]_r) \otimes \operatorname{rep}^{\#}([\![M_2]\!]_r)) | j_1 j_2 \rangle \tag{B.87}$$

This proves the lemma. □

Lemma B.13: Matrix-scalar multiplication

Let s be an expression of type $\mathcal{S}$ and M an expression of type $\mathcal{M}(q, m \to n)$. Suppose $[\![s]\!]_r$ and $[\![M]\!]_r$ are well-defined. Then $[\![s \cdot M]\!]_r$ is also well-defoned, and

$$\operatorname{rep}^{\#}([\![s \cdot M]\!]_r) = \operatorname{rep}^{\#}([\![s]\!]_r) \cdot \operatorname{rep}^{\#}([\![M]\!]_r) \tag{B.88}$$

Proof. We prove that $[\![s \cdot M]\!]_r$ is well-defined by showing the result of $\text{rep}^\#([\![s \cdot M]\!]_r)$ is independent of the choices of representations

$$[\![s]\!]_r = [(\phi_s, W_s)] \tag{B.89}$$
$$[\![M]\!]_r = [(\phi, W, x, y, q)] \tag{B.90}$$

with $\text{dom}(W) \cap \text{dom}(W_s) = \varnothing$. For every $0 \leq i < q^n$ and $0 \leq j < q^m$, we have

$$\langle i| \text{rep}^\#([\![s \cdot M]\!]_r) |j\rangle = \langle i| \text{rep}^\#[(\phi \wedge \phi_s, W \cup W_s, x, y, q)] |j\rangle \tag{B.91}$$
$$= \text{WMC}(\phi \wedge \phi_s \wedge x = j \wedge y = i, W \cup W_s) \tag{B.92}$$

Lemma B.1 gives

$$\langle i| \text{rep}^\#([\![s \cdot M]\!]_r) |j\rangle = \text{WMC}(\phi_s, W_s) \cdot \text{WMC}(\phi \wedge x = j \wedge y = i, W) \tag{B.93}$$
$$= \text{rep}^\#([\![s]\!]_r) \cdot \langle i| \text{rep}^\#([\![M]\!]_r) |j\rangle \tag{B.94}$$

Since this is true for any i and j, this proves the lemma. □

Lemma B.14: Matrix transpose

Let M be an expression of type $\mathcal{M}(q, m \to n)$ and suppose $[\![M]\!]_r$ is well-defined. Then $[\![\text{trans}(M)]\!]_r$ is well-defined, and

$$\text{rep}^\#([\![\text{trans}(M)]\!]_r) = (\text{rep}^\#([\![M]\!]_r))^T \tag{B.95}$$

Proof. Suppose $[\![M]\!]_r = [(\phi, W, x, y, q)]$. By definition,

$$[\![\text{trans}(M)]\!]_r = [(\phi, W, y, x, q)] \tag{B.96}$$

We show that this is well-defined by showing $\text{rep}^\#([\![\text{trans}(M)]\!]_r)$ yields the same value, independent of the choice of the representation before.

We want to show that $\langle i| \text{rep}^\#([\![\text{trans}(M)]\!]_r) |j\rangle = \langle j| \text{rep}^\#([\![M]\!]_r)) |i\rangle$. Using the formula above, we get

$$\langle i| \text{rep}^\#([\![\text{trans}(M)]\!]_r) |j\rangle = \text{WMC}(\phi \wedge y = j \wedge x = i, W) \tag{B.97}$$
$$= \text{WMC}(\phi \wedge x = i \wedge y = j, W) \tag{B.98}$$
$$= \langle j| \text{rep}^\#([\![M]\!]_r) |i\rangle \tag{B.99}$$

This proves the lemma. □

Lemma B.15: Field endomorphism on a matrix

Let M be an expression of type $\mathcal{M}(q, m \to n)$ and $f : \mathbb{F} \to \mathbb{F}$ a field endomorphism. Suppose $[\![M]\!]_r$ is well-defined. Then $[\![\text{apply}(f, M)]\!]_r$ is well-defined,

and

$$\text{rep}^{\#}([\![\mathsf{apply}(f, M)]\!]_r) = f(\text{rep}^{\#}([\![M]\!]_r)) \tag{B.100}$$

We interpret f being applied to a matrix as it being applied to every entry in the matrix, as it is done for the value semantics $[\![\cdot]\!]_v$.

Proof. For $[\![M]\!]_r = [(\phi, W, x, y, q)]$ we have defined

$$[\![\mathsf{apply}(f, M)]\!]_r = [(\phi, f \circ W, x, y, q)] \tag{B.101}$$

We prove this is well-defined by showing that $\text{rep}^{\#}([\![\mathsf{apply}(f, M)]\!]_r)$ has the same value, independent of the choice of the representation at the start of the proof. Using Lemma B.4, we get

$$\begin{aligned}
\langle i|\, \text{rep}^{\#}([\![\mathsf{apply}(f, M)]\!]_r)\, |j\rangle &= \text{WMC}(\phi \wedge x = j \wedge y = i, f \circ W) && \text{(B.102)}\\
&= f(\text{WMC}(\phi \wedge x = j \wedge y = i, W)) && \text{(B.103)}\\
&= f(\langle i|\, \text{rep}^{\#}([\![M]\!]_r)\, |j\rangle) && \text{(B.104)}
\end{aligned}$$

Since this is true for any $0 \leq i < q^n$ and $0 \leq j < q^m$, this proves the lemma. □

Lemma B.16: Trace

Let M be an expression of type $\mathcal{M}(q, n \to n)$ and suppose $[\![M]\!]_r$ is well-defined. Then $[\![\text{tr}(M)]\!]_r$ is also well-defined, and

$$\text{rep}^{\#}([\![\text{tr}(M)]\!]_r) = \text{tr}(\text{rep}^{\#}([\![M]\!]_r)) \tag{B.105}$$

Proof. Suppose that $[\![M]\!]_r = [(\phi, W, x, y, q)]$, then

$$[\![\text{tr}(M)]\!]_r = [(\phi \wedge (x \leftrightarrow y) \wedge \text{val}_x, W)] \tag{B.106}$$

We prove that this is well-defined by showing the result of $\text{rep}^{\#}([\![\text{tr}(M)]\!]_r)$ is independent of the choice of the representation $[\![M]\!]_r$. We have

$$\begin{aligned}
\text{rep}^{\#}([\![\text{tr}(M)]\!]_r) &= \text{rep}^{\#}[(\phi \wedge (x \leftrightarrow y) \wedge \text{val}_x, W)] && \text{(B.107)}\\
&= \text{WMC}(\phi \wedge (x \leftrightarrow y) \wedge \text{val}_x, W) && \text{(B.108)}
\end{aligned}$$

Using the fact that $\text{val}_x \equiv \bigwedge_{a=0}^{q^n-1}(x = a)$ and Lemma B.2, we get

$$\begin{aligned}
\text{rep}^{\#}([\![\text{tr}(M)]\!]_r) &= \sum_{a=0}^{q^n-1} \text{WMC}(\phi \wedge (x \leftrightarrow y) \wedge x = a, W) && \text{(B.109)}\\
&= \sum_{a=0}^{q^n-1} \text{WMC}(\phi \wedge x = a \wedge y = a, W) && \text{(B.110)}
\end{aligned}$$

$$= \sum_{a=0}^{q^n-1} \langle a| \operatorname{rep}^{\#}([\![M]\!]_r) |a\rangle \tag{B.111}$$

$$= \operatorname{tr}(\operatorname{rep}^{\#}([\![M]\!]_r)) \tag{B.112}$$

This proves the lemma. □

Lemma B.17: Matrix entry

Let M be an expression of type $\mathcal{M}(q, m \to n)$. Suppose we have indices i and j with $0 \leq i < q^n$ and $0 \leq j < q^m$. Furthermore, suppose $[\![M]\!]_r$ is well-defined. Then $[\![\mathsf{entry}(i, j, M)]\!]_r$ is well-defined, and

$$\operatorname{rep}^{\#}([\![\mathsf{entry}(i, j, M)]\!]_r) = (\operatorname{rep}^{\#}([\![M]\!]_r))_{ij} \tag{B.113}$$

Proof. Suppose $[\![M]\!]_r = [(\phi, W, x, y, q)]$. Then, by definition, we have

$$[\![\mathsf{entry}(i, j, M)]\!]_r = [(\phi \wedge x = j \wedge y = i, W)] \tag{B.114}$$

We prove this is well-defined by showing that $\operatorname{rep}^{\#}([\![\mathsf{entry}(i, j, M)]\!]_r)$ yields the same result, independent of the choice of the representation before. From the definition of matrix representations, we get

$$(\operatorname{rep}^{\#}([\![M]\!]_r)) = \operatorname{WMC}(\phi \wedge x = j \wedge y = i, W) \tag{B.115}$$

$$= \operatorname{rep}^{\#}[(\phi \wedge x = j \wedge y = i, W)] \tag{B.116}$$

$$= [\![\mathsf{entry}(i, j, M)]\!]_r \tag{B.117}$$

This proves the lemma. □

References

[1] Ignasi Abío and Peter J. Stuckey. "Encoding Linear Constraints into SAT". In: *Principles and Practice of Constraint Programming*. Ed. by Barry O'Sullivan. Cham: Springer International Publishing, 2014, pp. 75–91. ISBN: 978-3-319-10428-7.

[2] Bernhard Andraschko, Julian Danner, and Martin Kreuzer. "SAT solving using XOR-OR-AND normal forms". en. In: *Math. Comput. Sci.* 18.4 (Dec. 2024).

[3] F Barahona. "On the computational complexity of Ising spin glass models". In: *Journal of Physics A: Mathematical and General* 15.10 (Oct. 1982), p. 3241. DOI: 10.1088/0305-4470/15/10/028. URL: https://dx.doi.org/10.1088/0305-4470/15/10/028.

[4] A Biere, M Heule, and H van Maaren. *Handbook of Satisfiability*. en. Amsterdam, NY: IOS Press, Jan. 2009, p. 650.

[5] Sergey Bravyi, Anirban Chowdhury, David Gosset, and Pawel Wocjan. "Quantum Hamiltonian complexity in thermal equilibrium". In: *Nature Physics* 18.11 (Oct. 2022), pp. 1367–1370. ISSN: 1745-2481. DOI: 10.1038/s41567-022-01742-5. URL: http://dx.doi.org/10.1038/s41567-022-01742-5.

[6] Stephen G Brush. "History of the Lenz-Ising model". In: *Reviews of modern physics* 39.4 (1967), p. 883.

[7] O. Canko and E. Albayrak. "Pair-approximation method for the quantum transverse spin-2 Ising model with a trimodal-random field". In: *Physics Letters A* 340.1 (2005), pp. 18–30. ISSN: 0375-9601. DOI: https://doi.org/10.1016/j.physleta.2005.04.025. URL: https://www.sciencedirect.com/science/article/pii/S0375960105005694.

[8] Mark Chavira and Adnan Darwiche. "On probabilistic inference by weighted model counting". In: *Artificial Intelligence* 172.6 (2008), pp. 772–799. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2007.11.002. URL: https://www.sciencedirect.com/science/article/pii/S0004370207001889.

[9] Bob Coecke and Ross Duncan. "Interacting quantum observables: categorical algebra and diagrammatics". In: *New Journal of Physics* 13.4 (Apr. 2011), p. 043016. DOI: 10.1088/1367-2630/13/4/043016. URL: https://dx.doi.org/10.1088/1367-2630/13/4/043016.

[10] The SciPy community. *scipy.linalg.expm*. 2008. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.expm.html (visited on 06/27/2025).

[11] Christoph Dann, Peter Gehler, Stefan Roth, and Sebastian Nowozin. "Pottics – The Potts Topic Model for Semantic Image Segmentation". In: *Proceedings of 34th DAGM Symposium*. Lecture Notes in Computer Science. Springer, Aug. 2012, pp. 397–407.

[12] Paulius Dilkas and Vaishak Belle. "Weighted model counting with conditional weights for Bayesian networks". In: *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*. Ed. by Cassio de Campos and Marloes H. Maathuis. Vol. 161. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 386–396. URL: https://proceedings.mlr.press/v161/dilkas21a.html.

[13] Dirck van den Ende. *DiracWMC*. URL: https://github.com/System-Verification-Lab/DiracWMC.

[14] Jeffrey M. Dudek, Vu H. N. Phan, and Moshe Y. Vardi. "DPMC: Weighted Model Counting by Dynamic Programming on Project-Join Trees". In: *Principles and Practice of Constraint Programming: 26th International Conference, CP 2020, Louvain-La-Neuve, Belgium, September 7–11, 2020, Proceedings*. Louvain-la-Neuve, Belgium: Springer-Verlag, 2020, pp. 211–230. ISBN: 978-3-030-58474-0. DOI: 10.1007/978-3-030-58475-7_13. URL: https://doi.org/10.1007/978-3-030-58475-7_13.

[15] Jeffrey M. Dudek, Vu H. N. Phan, and Moshe Y. Vardi. "ProCount: Weighted Projected Model Counting with Graded Project-Join Trees". In: *Theory and Applications of Satisfiability Testing – SAT 2021*. Ed. by Chu-Min Li and Felip Manyà. Cham: Springer International Publishing, 2021, pp. 152–170. ISBN: 978-3-030-80223-3.

[16] Jeffrey M. Dudek and Moshe Y. Vardi. *Parallel Weighted Model Counting with Tensor Networks*. 2021. arXiv: 2006.15512 [cs.DS]. URL: https://arxiv.org/abs/2006.15512.

[17] H. B. Hunt and R. E. Stearns. "On the complexity of satisfiability problems for algebraic structures (preliminary report)". In: *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*. Ed. by Teo Mora. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 250–258. ISBN: 978-3-540-46152-4.

[18] Julia Kempe, Alexei Kitaev, and Oded Regev. "The Complexity of the Local Hamiltonian Problem". In: *SIAM Journal on Computing* 35.5 (2006), pp. 1070–1097. DOI: 10.1137/S0097539704445226. eprint: https://doi.org/10.1137/S0097539704445226. URL: https://doi.org/10.1137/S0097539704445226.

[19] William Klieber and Gihwon Kwon. "Efficient CNF Encoding for Selecting 1 from N Objects". In: 2007. URL: https://api.semanticscholar.org/CorpusID:165159977.

[20] Physics Claire Kluber. *Trotterization in Quantum Theory*. 2025. arXiv: 2310.13296 [quant-ph]. URL: https://arxiv.org/abs/2310.13296.

[21] Philipp Krähenbühl and Vladlen Koltun. "Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials". In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger. Vol. 24. Curran Associates, Inc., 2011. URL:

https://proceedings.neurips.cc/paper_files/paper/2011/file/beda24c1e1b46055dff2c39c98fd6fc1-Paper.pdf.

[22] Johannes Lang, Bernhard Frank, and Jad C. Halimeh. "Concurrence of dynamical phase transitions at finite temperature in the fully connected transverse-field Ising model". In: *Phys. Rev. B* 97 (17 May 2018), p. 174401. DOI: 10.1103/PhysRevB.97.174401. URL: https://link.aps.org/doi/10.1103/PhysRevB.97.174401.

[23] Ronald M Levy, Allan Haldane, and William F Flynn. "Potts Hamiltonian models of protein co-variation, free energy landscapes, and evolutionary fitness". en. In: *Curr Opin Struct Biol* 43 (Nov. 2016), pp. 55–62.

[24] Alex J. Li, Mindren Lu, Israel Desta, Vikram Sundar, Gevorg Grigoryan, and Amy E. Keating. "Neural network-derived Potts models for structure-based protein design using backbone atomic coordinates and tertiary motifs". In: *Protein Science* 32.2 (2023), e4554. DOI: https://doi.org/10.1002/pro.4554. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/pro.4554. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/pro.4554.

[25] Weikang Li, Zhide Lu, and Dong-Ling Deng. "Quantum Neural Network Classifiers: A Tutorial". In: *SciPost Phys. Lect. Notes* (2022), p. 61. DOI: 10.21468/SciPostPhysLectNotes.61. URL: https://scipost.org/10.21468/SciPostPhysLectNotes.61.

[26] Jingyi Mei, Marcello Bonsangue, and Alfons Laarman. "Simulating Quantum Circuits by Model Counting". In: *Computer Aided Verification*. Ed. by Arie Gurfinkel and Vijay Ganesh. Cham: Springer Nature Switzerland, 2024, pp. 555–578. ISBN: 978-3-031-65633-0.

[27] Jingyi Mei, Tim Coopmans, Marcello Bonsangue, and Alfons Laarman. "Equivalence Checking of Quantum Circuits by Model Counting". In: *Automated Reasoning*. Ed. by Christoph Benzmüller, Marijn J.H. Heule, and Renate A. Schmidt. Cham: Springer Nature Switzerland, 2024, pp. 401–421. ISBN: 978-3-031-63501-4.

[28] Jingyi Mei, Jan Martens, and Alfons Laarman. "Disentangling the Gap Between Quantum and #SAT". In: *Theoretical Aspects of Computing – ICTAC 2024: 21st International Colloquium, Bangkok, Thailand, November 25–29, 2024, Proceedings*. Bangkok, Thailand: Springer-Verlag, 2024, pp. 17–40. ISBN: 978-3-031-77018-0. DOI: 10.1007/978-3-031-77019-7_2. URL: https://doi.org/10.1007/978-3-031-77019-7_2.

[29] Awad H. Al-Mohy and Nicholas J. Higham. "A New Scaling and Squaring Algorithm for the Matrix Exponential". In: *SIAM Journal on Matrix Analysis and Applications* 31.3 (2010), pp. 970–989. DOI: 10.1137/09074721X. eprint: https://doi.org/10.1137/09074721X. URL: https://doi.org/10.1137/09074721X.

[30] Shaan Nagy, Roger Paredes, Jeffrey M. Dudek, Leonardo Dueñas-Osorio, and Moshe Y. Vardi. "Ising model partition-function computation as a weighted counting problem". In: *Phys. Rev. E* 109 (5 May 2024), p. 055301. DOI: 10.1103/PhysRevE.109.055301.

[31] Van-Hau Nguyen, Van-Quyet Nguyen, Kyungbaek Kim, and Pedro Barahona. "Empirical Study on SAT-Encodings of the At-Most-One Constraint". In: *The 9th International Conference on Smart Media and Applications*. SMA 2020. Jeju, Republic of Korea: Association for Computing Machinery, 2021, pp. 470–475. ISBN: 9781450389259. DOI: 10.1145/3426020.3426170. URL: https://doi.org/10.1145/3426020.3426170.

[32] Peter B R Nisbet-Jones, Jerome Dilley, Annemarie Holleczek, Oliver Barter, and Axel Kuhn. "Photonic qubits, qutrits and ququads accurately prepared and delivered on demand". In: *New Journal of Physics* 15.5 (May 2013), p. 053007. DOI: 10.1088/1367-2630/15/5/053007. URL: https://dx.doi.org/10.1088/1367-2630/15/5/053007.

[33] Feng Pan, Pengfei Zhou, Sujie Li, and Pan Zhang. "Contracting Arbitrary Tensor Networks: General Approximate Algorithm and Applications in Graphical Models and Quantum Circuit Simulations". In: *Phys. Rev. Lett.* 125 (6 Aug. 2020), p. 060503. DOI: 10.1103/PhysRevLett.125.060503. URL: https://link.aps.org/doi/10.1103/PhysRevLett.125.060503.

[34] Paredes, Dueñas-Osorio, Meel, and Vardi. *A weighted model counting approach for critical infrastructure reliability*. May 2019.

[35] Nara M. Portela, George D.C. Cavalcanti, and Tsang Ing Ren. "Contextual Image Segmentation Based on the Potts Model". In: *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*. 2013, pp. 256–261. DOI: 10.1109/ICTAI.2013.47.

[36] Steven Prestwich. "Chapter 2. CNF Encodings". In: *Frontiers in Artificial Intelligence and Applications*. Frontiers in artificial intelligence and applications. IOS Press, Feb. 2021.

[37] Mario Reis. "Chapter 8 - Paramagnetism". In: *Fundamentals of Magnetism*. Ed. by Mario Reis. Boston: Academic Press, 2013, p. 94. ISBN: 978-0-12-405545-2. DOI: https://doi.org/10.1016/B978-0-12-405545-2.00008-4. URL: https://www.sciencedirect.com/science/article/pii/B9780124055452000084.

[38] Sebastián V. Romero, Alejandro Gomez Cadavid, Pavle Nikačević, Enrique Solano, Narendra N. Hegade, Miguel Angel Lopez-Ruiz, Claudio Girotto, Masako Yamada, Panagiotis Kl. Barkoutsos, Ananth Kaushik, and Martin Roetteler. *Protein folding with an all-to-all trapped-ion quantum computer*. 2025. arXiv: 2506.07866 [quant-ph]. URL: https://arxiv.org/abs/2506.07866.

[39] Aaron Sander, Lukas Burgholzer, and Robert Wille. "Towards Hamiltonian Simulation with Decision Diagrams". In: *2023 International Conference on Quantum Computing and Engineering*. May 2023. DOI: 10.1109/QCE57702.2023.00039. arXiv: 2305.02337 [quant-ph].

[40] Tian Sang, Paul Beame, and Henry A. Kautz. "Heuristics for Fast Exact Model Counting". In: *International Conference on Theory and Applications of Satisfiability Testing*. 2005. URL: https://api.semanticscholar.org/CorpusID:137080.

[41] Nicol Schraudolph and Dmitry Kamenetsky. "Efficient Exact Inference in Planar Ising Models". In: *Advances in Neural Information Processing Systems*. Ed. by D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou. Vol. 21. Curran

Associates, Inc., 2008. URL: https://proceedings.neurips.cc/paper_files/paper/2008/file/816b112c6105b3ebd537828a39af4818-Paper.pdf.

[42] P. Selinger. "A Survey of Graphical Languages for Monoidal Categories". In: *New Structures for Physics*. Springer Berlin Heidelberg, 2010, pp. 289–355. ISBN: 9783642128219. DOI: 10.1007/978-3-642-12821-9_4. URL: http://dx.doi.org/10.1007/978-3-642-12821-9_4.

[43] Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S Meel. "GANAK: A Scalable Probabilistic Exact Model Counter." In: *IJCAI*. Vol. 19. 2019. 2019, pp. 1169–1176.

[44] Mate Soos, Stephan Gocht, and Kuldeep S. Meel. "Tinted, Detached, and Lazy CNF-XOR Solving and Its Applications to Counting and Sampling". In: *Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part I*. Los Angeles, CA, USA: Springer-Verlag, 2020, pp. 463–484. ISBN: 978-3-030-53287-1. DOI: 10.1007/978-3-030-53288-8_22. URL: https://doi.org/10.1007/978-3-030-53288-8_22.

[45] Masuo Suzuki. "Relationship between d-Dimensional Quantal Spin Systems and (d+1)-Dimensional Ising Systems: Equivalence, Critical Exponents and Systematic Approximants of the Partition Function and Spin Correlations". In: *Progress of Theoretical Physics* 56.5 (Nov. 1976), pp. 1454–1469. ISSN: 0033-068X. DOI: 10.1143/PTP.56.1454. eprint: https://academic.oup.com/ptp/article-pdf/56/5/1454/5264429/56-5-1454.pdf. URL: https://doi.org/10.1143/PTP.56.1454.

[46] Tomoya Tanjo, Naoyuki Tamura, and Mutsunori Banbara. "Proposal of a compact and efficient SAT encoding using a numeral system of any base". In: 2011. URL: https://api.semanticscholar.org/CorpusID:15006712.

[47] Jedwin Villanueva, Gary J Mooney, Bhaskar Roy Bardhan, Joydip Ghosh, Charles D Hill, and Lloyd C L Hollenberg. *Hybrid quantum optimization in the context of minimizing traffic congestion*. 2025. arXiv: 2504.08275 [quant-ph]. URL: https://arxiv.org/abs/2504.08275.

[48] Alejandro Villoria, Henning Basold, and Alfons Laarman. "Enriching Diagrams with Algebraic Operations". In: *Foundations of Software Science and Computation Structures*. Ed. by Naoki Kobayashi and James Worrell. Cham: Springer Nature Switzerland, 2024, pp. 121–143. ISBN: 978-3-031-57228-9.

[49] Yingte Xu, Gilles Barthe, and Li Zhou. "Automating Equational Proofs in Dirac Notation". In: *Proc. ACM Program. Lang.* 9.POPL (Jan. 2025). DOI: 10.1145/3704878. URL: https://doi.org/10.1145/3704878.

[50] Yingte Xu, Li Zhou, and Gilles Barthe. *D-Hammer: Efficient Equational Reasoning for Labelled Dirac Notation*. 2025. arXiv: 2505.08633 [cs.PL]. URL: https://arxiv.org/abs/2505.08633.

[51] Jiong Yang and Kuldeep S. Meel. "Engineering an Efficient PB-XOR Solver". In: *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Ed. by Laurent D. Michel. Vol. 210. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 58:1–58:20. ISBN: 978-3-95977-211-2.

DOI: 10.4230/LIPIcs.CP.2021.58. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2021.58.

[52] Dekel Zak, Jingyi Mei, Jean-Marie Lagniez, and Alfons Laarman. "Reducing Quantum Circuit Synthesis to #SAT". In: *31th International Conference on Principles and Practice of Constraint Programming (CP 2025)*. Accepted for publication. 2025.