Ket Notation

We would like to control quantum degrees of freedom in order to implement quantum computing, among other goals that we have. To talk about these degrees of freedom, we're going to use some notation that you might not be familiar with, but it's all very close to linear algebra.

You are likely already familiar with linear algebra, which has column vectors denoted with an arrow, or with boldface type.

It also has a transpose operation that maps column vectors to row vectors, and vice versa.

We can calculate the inner product of two such vectors, either by transposing one of them and multiplying, or by writing the dot product explicitly.

Either way, we get a scalar by taking the inner product of these two vectors.

In addition, we can transform one vector into another using a matrix, which can be written using boldface type or a hat, your preference.

There is a slightly different notation that we use for vectors and matrices in quantum computing, which we often call ket notation.

In ket notation, a quantum state is expressed using a column vector with complex coefficients. For example, we can express the state of a qubit, a two-level system, as a linear combination of two basis vectors, which we call 0 and 1.

In a departure from regular linear algebra, there is a well-defined dual vector for each quantum state, called a bra, which we can obtain by taking the complex conjugate and the transpose of the ket vector. This is important for calculating inner products, which we always do by multiplying the bra for one state by the ket of the other, forming a bra-ket.

A central feature of quantum mechanics is that, when we perform a measurement to determine whether a state is 0 or 1, for example, we get a random answer, and the probability of measuring a state to be 0 is given by the squared magnitude of its 0 coefficient.

One consequence of this is that, since such a measurement on a qubit state must result in 0 or 1, these squared magnitudes must sum to 1, since they are probabilities. This is called Born's rule and the constraint that the probabilities must sum to 1, is called normalization.

We can also express qubit states in different bases. Consider the often-used plus-minus basis, which consists of the normalized sum and difference of the 0 and 1 ket vectors.

Given a state expressed in the 0/1 basis, we can calculate the coefficients required to express the same state in the plus-minus basis, as I have done here.

This leaves us with a small question: if any basis is just as good as any other, is there a basis that we should use as the default?

Fortunately, the devices that we use to store and manipulate qubit states provide us with just such a default basis.

One such device, which we see quite frequently in quantum mechanics, is the harmonic oscillator, shown on the left.

These devices are described by a Hamiltonian, which is a matrix that assigns an energy E_k to each of its eigenstates, or preferred basis states, psi_k.

These states are orthogonal and normalized (so we say that they're orthonormal), so the inner product of psi_j with any psi_k other than itself is 0.

This allows us to use these states as a computational basis, replacing any detailed knowledge of the wavefunction psi_k with a simple label k that indicates which state we're talking about.

Some devices have finite-dimensional state spaces, unlike the harmonic oscillator. This allows us to express the basis states as column vectors without using an infinite amount of space, which is very nice. Such devices include the spin-1/2, which is shown in this figure.

Now that's it for the representation of single-qubit states.

Let's take a look at how ket notation can help us describe the operations that we want to perform on these states, and the measurements we would like to make.

The logic gate is the smallest classical computing system, and all classical computations can be expressed as a large sequence of these gates.

The quantum counterpart to a logic gate is a unitary matrix, which replaces the computational basis with a new basis that depends on the operation that we want to perform.

Probabilistic measurements are also described by matrices, but these matrices are hermitian, so in ket notation, they are just weighted sums of these ket-bra terms, where the psi_k states form an orthonormal basis for the space.

If a measurement results in the state psi_k, the value r_k becomes known to the experimentalist.

The expected value for this measurement is given by sandwiching the measurement operator with the state that we're measuring, and as you can see, the expectation value is just a weighted sum of the r_k terms, with the probabilities given by Born's rule.

That's probably a lot to take in all at once, so let's take a look at a few examples.

Here, we have a unitary operation that exchanges states in the computational basis, which we call X, or the Pauli X if you're already familiar with Pauli matrices.

Here it is decomposed into ket-bra terms, and here's what happens when we use it to transform one of the computational basis states.

We just get the other state. 0 goes to 1, and 1 goes back to 0. Not so exciting.

Now let's take a look at a more interesting operation, the Hadamard gate, H.

As you can see, this changes the basis from the 0/1 basis to the +/- basis that we discussed earlier.

Put in a 0, get out a +, put in a 1, get out a -.

There is another gate, called the phase gate, which only multiplies the one state by a factor of i, changing the basis in a very subtle way.

Now let's take a look at a few measurement operators.

In an interesting coincidence, the Pauli X shows up again, since it's both unitary and hermitian, we can use it both as an operation and a measurement.

Here, we have decomposed it into its eigenbasis, so the ket-bra terms are different than before, and we can see that its output values are +/- 1.

Here's another matrix which is both unitary and hermitian, the pauli Z.

It returns the exact same values as X, just for states in the 0/1 basis instead of the +/- basis.

And finally we have the identity, which is also unitary and hermitian.

We can note that the identity doesn't change the basis at all, so that indeed it can be expressed in any basis.

Also when used as a measurement operator, it always returns 1, no matter what state is input.

Now that we have a few examples done, I would like to focus a little on a useful geometric representation of qubit states, called the Bloch Sphere.

To show how qubit states can be mapped to the surface of a sphere, let's start by expressing the coefficients alpha and beta in polar co-ordinates.

Next, we can note that we can get rid of the phase on the alpha coefficient, since a ket multiplied by a phase produces the same measurement results as the ket itself, for any potential experiment, so these are not different in any physical sense.

This results in a simpler expression for the state, but it's not yet as simple as it can be.

To get it even simpeler, we need to recall that these states have to be normalized, so the sum of the squares of the two radii involved has to be 1.

This implies that we can express them as the cosine and sine of an angle theta, since cos(theta)^2 + sin(theta)^2 is always 1.

This family of angles theta and phi also describes the set of points on the surface of a sphere of unit radius in 3d space.

We can see that, setting theta to 0, and phi to whatever we want (here it's 0), we get cos(theta) = 1, sin(theta) = 0, so the corresponding state is simply the 0 basis state.

If we set theta to pi however, we get the 1 state regardless of the setting of phi.

Likewise if we set theta to pi/2, and phi to either 0 or pi, we get one of the states from the +/- basis on the equator of the sphere.

Unitary operations, which change the basis we are working in, effectively rotate the sphere.

For example, the Hadamard operation from earlier rotates the +/- states on the equator by 90 degrees until they're at the poles.

Also note that the states on opposite sides of the sphere are actually orthogonal, so this mapping does not preserve the angle between states, but it's still useful for describing single-qubit states and operations.

But how do we describe multi-qubit states and operations?

Specifically, how do we build them up from operations on smaller subsystems of a many-qubit state?

To accomplish this, we use a different kind of matrix product, called the tensor product, or Kronecker product.

To take the tensor product of two matrices A and B, we write out a block matrix where each block is equal to B times the appropriate element of A.

The upper left block is B times the upper left element of A, and so on.

The interesting thing about the tensor product is that it's compatible with the regular matrix product.

That is to say if I take the matrix product of two tensor products, I get the same matrix as if I took the matrix products first, then evaluated the tensor product.

This is also easier to understand if we take a look at a few examples.

Here we see the tensor product of two states, + and 0, resulting in a state which we often call +0.

Each 2-by-1 block of the state vector we're calculating is proportional to the vector for the zero state, and it's multiplied by one of the elements of the plus state.

We can also take tensor products of operations and observables, here we take the tensor product of X and the identity, making a block matrix whose blocks are all equal to the identity, multiplied by one of the elements of X.

Now that we have seen the basic formalism and notation, we're ready to calculate the results of a small sequence of quantum operations, which prepares a Bell state.

First, we introduce a two-qubit gate which cannot be written as a tensor product of one-qubit gates, the controlled not, or cnot for short.

As we can see, it can be decomposed into a sum of two tensor products that says if the first qubit is in the zero state, do the identity, and if the first qubit is in the one state, perform an X on the second qubit.

Here we see the Bell state, which we're going to prepare using the cnot. We can write it out using either ket notation, or as a column vector.

Note that the Bell state, just like the cnot, cannot be written as a tensor product.

The first step in preparing the Bell state is to perform a Hadamard operation on the first qubit, which has been initiliazed in the zero state. So we calculate the tensor product of the Hadamard with the identity, since we're not going to do anything to the second qubit immediately.

Now we're ready to set up our sequence of operations.

First, we prepare the state 00, then we apply the Hadamard, then the cnot.

This results in a Bell state, which is written out here as a column vector.

Now this is the correct answer, but it was a little tedious to come to, and the matrices involved are a little large. 3x3 matrices are typically big enough for any phisicist, so 4x4 is overdoing it a little.

Let's try and do it the easy way, using some more ket notation.

First, we use the compatibility of the Kronecker and regular products to show that the state we get from executing the Hadamard on the first qubit is simply +0, without having to use any matrices.

Now, all we have to show is that the cnot will take our state, which is 00 + 10, to the 00 + 11 state that we're after.

If we insert the decomposition of the cnot into tensor products that we saw earlier, we can see that the resulting state has coefficients proportional to these inner products of the 0 and 1 states.

Of course, the 0/1 basis is orthonormal, so the products 00 and 11 evaluate to 1, and the products 01 and 10 evaluate to 0, leaving us with two remaining terms.

These terms are exactly the 00 and 11 terms that we were after.

And that's how ket notation van help us to describe the effects of operations and measurements on quantum states.