Formula evaluation and span-program viewpoints

Translate a Boolean formula into a question about available vector spans.

Research 21 min lesson Dr. Charith Jayasekara

By the end, you'll be able to

- Verify a two-input AND span program.
- Construct positive and separating negative witnesses.
- Separate witness-size query analysis from gate implementation.

A Boolean formula is a structured question

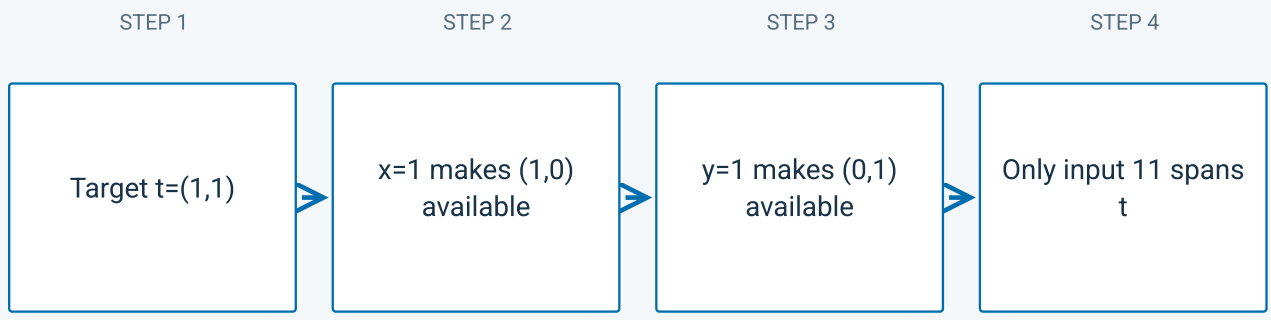
A formula combines input bits through gates such as AND, OR and NOT. Its tree structure can matter: a balanced expression and a long chain may use the same number of leaves but present different evaluation patterns. Quantum formula-evaluation algorithms exploit structure through walks, reflections and linear-algebraic representations called span programs.

A span program replaces a logical question with a geometric one: do the vectors made available by this input span a chosen target vector? If they do, the answer is one; if not, it is zero. This turns a Boolean computation into a statement about linear combinations. It is a representation for designing and analysing algorithms, not a demand that a beginner solve a large matrix problem by hand.

Review vectors and matrices, inner products and query models. The span of a set of vectors is the collection of all their linear combinations.

WORKED DIAGRAM · AA18

AND as a reachability question in vector space



For input 11, coefficients (1,1) combine the available vectors into the target. With only (1,0), every reachable vector has second coordinate zero, so input 10 rejects.

Define a span-program setup

Choose a target vector and a collection of input-labelled vectors. For a given Boolean input, only vectors whose labels are satisfied become available. The program accepts if the target belongs to their span. A positive witness gives coefficients expressing the target as a combination of available vectors. A negative witness separates the target from all available vectors when the program rejects.

Witness sizes influence the quantum query complexity of the associated construction. They are not simply the number of vectors or the formula depth. A useful design must also support the required reflections or linear-algebra operations efficiently. An optimal abstract query representation can still be expensive to compile into gates.

Work an AND example

To compute AND of two bits, use the two-dimensional target (1,1). Make vector (1,0) available when the first bit is one, and vector (0,1) available when the second bit is one. If both bits are one, their sum equals the target, so the program accepts. If only one is available, its span lies on one coordinate axis and cannot reach the target. If neither is available, only the zero vector is reachable.

This gives the correct four-case truth table. For input 11, coefficients one and one form a positive witness. For input 10, the vector (0,1) defines a separating direction: it is orthogonal to the available vector (1,0) but has nonzero inner product with the target. This is the basic idea of a negative witness.

For OR, a one-dimensional target one suffices: provide a copy of the vector one for each input bit that is one. Any available copy spans the target. These simple examples show how different logical gates correspond to different geometry. More elaborate compositions must control witness sizes as well as truth-table correctness.

The target lies in the available span exactly when both required input vectors are present.

$$t = \begin{pmatrix}1\\1\end{pmatrix}, \quad v_1 = \begin{pmatrix}1\\0\end{pmatrix}, \quad v_2 = \begin{pmatrix}0\\1\end{pmatrix}, \qquad t \in \operatorname{span}\{v_i : x_i = 1\} \iff x_1 \wedge x_2 = 1.$$

When both bits are one, coefficients one and one give $v_1 + v_2 = t$. With only one vector available, its coordinate-axis span misses the target. Witness sizes and implementation costs matter when this geometric test becomes a query algorithm.

From witnesses to quantum queries

The associated quantum algorithm uses input-dependent and input-independent reflections. The input determines which subspace is available, while the fixed representation determines another geometric operation. Spectral behaviour distinguishes accepting and rejecting inputs. This is related to Grover's two-reflection geometry but applies much more generally than unstructured search.

Reichardt's work connects span programs with the general adversary bound and quantum query algorithms. A later reflection-based construction establishes a tight characterisation of bounded-error Boolean quantum query complexity through that bound. These are access-model results; they do not imply a universal cheap gate implementation for every Boolean function. Read the span-program connection and the reflection-based result.

Why formulas are an important example

An OR of N input bits is unstructured search: the answer is one if any leaf is one. Grover gives a square-root query scale. Composed formulas combine simpler functions, and span programs provide a principled way to combine their query

behaviour. For suitable formula families, this leads to strong evaluation algorithms with matching or near-matching lower bounds.

The exact theorem depends on the gate set, formula structure and error model. Do not generalise a result for balanced formulas to arbitrary circuits with shared subcomputations without analysis. A formula is a tree; a circuit can reuse intermediate values and form a directed acyclic graph. Those are different computational objects.

Follow a design workflow

Write the Boolean function and its input-access model. Build a span representation whose truth table is correct. Analyse positive and negative witness sizes for all relevant inputs. Construct the quantum reflections or walk and state the resulting query guarantee. Then count gate synthesis, workspace and input-lookup costs if an implementation is required.

For a small function, enumerate all inputs to check the representation before studying asymptotics. The AND example's target must be two-dimensional: using a one-dimensional target with either available input vector would accidentally compute OR. This illustrates why an appealing geometric picture still needs a complete correctness check.

Limits and learning value

Span programs are often best understood first as an analysis language. They can reveal why a query algorithm is possible and how lower bounds guide design. They do not replace all algorithmic creativity with a practical button that takes a truth table and emits a short circuit. The representation and required fixed operations may grow very large.

The deeper lesson is that lower bounds and algorithms can meet in the same mathematical object. Rather than guessing a circuit and hoping it is optimal, one can study a complexity measure that both limits and enables quantum querying. This connection prepares readers for the adversary-method chapter without requiring the full semidefinite-program machinery immediately.

Practice with answers

Exercise 1. Why does making only (1,0) available fail to reach target (1,1), even if arbitrary real coefficients are allowed?

> Check the worked answer
>
> Every multiple of (1,0) has second coordinate zero. The target's second coordinate is one, so it lies outside that span. Coefficients can scale available directions but cannot invent a missing independent direction.

Exercise 2. Does a correct span-program truth table prove an efficient hardware algorithm?

> Check the worked answer
>
> No. We also need witness-size bounds, a valid query construction and efficient implementation of its fixed and input-dependent operations. Truth-table correctness establishes the represented function, not the total physical resource cost.

Check your understanding

PAUSE & THINK

Why can multiples of (1,0) not reach target (1,1)?

- [ ] Their second coordinate remains zero
- [ ] Their norm is always one
- [ ] Negative coefficients are forbidden

Answer: Their second coordinate remains zero. Scaling an available direction cannot create a missing independent direction. The target lies outside that one-dimensional span.

Sources

- Read the span-program connection
- the reflection-based result