Overview; AI Safety via Debate

May 11, 2026

Contents

1 Main Take-Aways

1. There is a difference between syntax and semantics, and since AI cannot fool us on purely syntactic problems (coding, math), semantics is the hard part.
2. Most interesting problems are going to be semantic (physics, social science, AI alignment).
3. We want the AI to prove solutions to semantic problems, which generally would require an unreasonable amount of human judgement calls.
4. **Debate is a form of scalable oversight: it seeks to extend a reward signal for honest solutions to difficult problems while reducing the amount of human judgment needed by an exponential factor.**

2 AI safety via Debate

2.1 Intuition on Scalable Oversight

Imagine you are a manager responsible for reviewing the work of an employee who is, in every relevant technical sense, far more capable than you. She submits a design specification for a critical system — a bridge, a drug, a financial instrument — and you must decide whether to approve it. You cannot check every calculation or verify every assumption. You must either trust her judgment completely, or find some other way to gain confidence that the work is sound. This scenario, which might seem like an ordinary workplace challenge, turns out to be one of the central problems facing the development of advanced artificial intelligence. As AI systems become more capable, a human overseer will face a fundamental epistemic gap. The outputs of these systems may be too complex, too subtle, or simply too voluminous for direct human verification. And yet the consequences of undetected errors, or worse, undetected deception, could be severe.

This problem is known as the *scalable oversight problem*: how do we ensure that human supervision remains meaningful as AI capabilities grow beyond human-level performance in domain after domain? It is not enough to note that a system appears to behave well in tests, or that it has been trained on human preferences. A sufficiently capable system that has learned to appear aligned may behave very differently in deployment, especially in high-stakes situations where the incentives to deceive are highest. What we need is not the appearance of correctness, but a mechanism for verifying it.

Debate is one of the most theoretically principled proposals for solving this problem. The core idea is elegantly simple: instead of asking a human to directly evaluate the output of a powerful AI system

— a task that may be beyond human competence — we ask two AI systems to argue against each other, and have a human judge only the argument. If one system is trying to be honest and the other is trying to deceive, and if the structure of the game is right, honesty should win. The human does not need to understand the full depth of the problem; they need only follow the debate to the point where they can identify which side is telling the truth.

2.2 Interactive Proofs

Interactive proofs are a model of computation consisting of a message exchange between two parties:

1. A computationally powerful but untrustworthy prover, in many cases assumed to have no computational constraints whatsoever.
2. The computationally-bounded but honest verifier, often to polynomial-time algorithms.

Example 2.1 (Graph Isomorphism)**.** Suppose question is whether two given graphs, G_1, G_2 are isomorphic. This is a problem in NP, so the prover can always convince the verifier by sending him the NP-certificate, in this case the permutation π on the vertices that transforms G_1 into G_2. The verifier can checking if $\pi(G_1) = G_2$, which is easily done in polynomial time. If the graphs are non-isomorphic, then nothing the prover says can convince the verifier.

So, does that mean that interactive proofs are basically a fancy way of defining NP? No, they are actually much more powerful! Consider the following example of a problem not believed to be in NP.

Example 2.2 (Graph Non-Isomorphism)**.** Now, the prover wants to convince the verifier that two graphs G_1 and G_2 are not isomorphic. The verifier randomly permutes one of the two graphs and sends the result to the prover. If the graphs are truly non-isomorphic, the prover can always tell which original graph it came from, while if they are isomorphic, no prover can do better than guessing. This is a standard example of an interactive proof that uses randomness in an essential way.

The quality of an interactive proof protocol is generally measured with two values

1. Completeness value c: The probability that the verifier accepts, given that the prover cooperates.
2. Soundness error s: The probability that the verifier accepts, given that the prover tries to fool them.

For the example of Graph Isomorphism, $c = 1$ and $s = 0$, as the verifier will either always accept, or never. For graph non-isomorphism, the values are 1 and 0.5. Note, that whenever there is a reasonable gap between c and s, we can amplify this gap and push it arbitrarily close to 1. In the case of graph non-isomorphism, the verifier can repeat the protocol k times and only accept if the prover got it right every time.

The power of interactive proofs depends on a number of settings:

1. The computational power of the prover and verifier respectively
2. The number of exchanges between them
3. Whether the verifier can employ randomness
4. How many bits of the provers answer the verifier can access
5. Whether the prover is allowed to learn certain things about the query (zero-knoowledge proofs)

We can take our guidance for Ai safety via debate from complexity theory. Let us consider a sequence of scenarios that builds up to the full debate setup.

2.2.1 One-Round Interactive Proofs: The Merlin-Arthur Protocol

The most base-case scenario for a

Consider a polynomially-bounded verifier, called Arthur, that decides if a word $\mathbf{x}$ belongs to a language L. He gets help from an all-powerful but unreliable prover, Merlin, who always wants to convince Arthur that $\mathbf{x} \in L$. Merlin can send a polynomially long certificate to Arthur to convince him.

A language L belongs to the complexity class MA, if there exists an Arthur $A : \{0,1\}^n \times \{0,1\}^{p(n)} \to \{0,1\}$ and a Merlin M : such that:

1. Completeness Criterion: If $x \in L$, then there exists a certificate $w \in \{0,1\}^{\mathrm{poly}(n)}$ such that Arthur accepts with high probability:

$$x \in L \implies \exists\, w \in \{0,1\}^{\mathrm{poly}(n)} : \Pr\big[V(x,w) = 1\big] \;\geq\; \frac{2}{3}.$$

2. Soundness criterion: If $x \notin L$, then no witness — however cleverly chosen by Merlin — can convince Arthur to accept with non-negligible probability:

$$x \notin L \implies \forall\, w \in \{0,1\}^{\mathrm{poly}(n)} : \Pr\big[V(x,w) = 1\big] \;\leq\; \frac{1}{3}.$$

In other words, for perfect completeness and soundness, then if $x \in L$, then Merlin can convince Arthur that this is indeed the case, if $x \notin L$, there is no certificate that Merlin can produce that would fool Arthur.

It is important to notice that the probabilistic aspect of the criteria enters over a random seed independently of x. This means that these criteria must hold for every x, not for a certain percentage of them. This fact allows for so called *amplification*. As long as the gap between the soundness and completeness probabilities is finite, we can run the verifier multiple times to make the gap as large as possible.

Exercise 2.1. Show that the complexity class NP is contained in MA.

The power of this setup can be extended by increasing the number of interaction rounds between prover and verifier. The class of problems solvable by polynomially many rounds is called IP, short for interactive proofs. A seminal result is that

$$\mathsf{IP} = \mathsf{PSPACE},$$

which means that an interactive protocol with a single prover can resolve very hard computational problems. However, these protocols rely on so called arithmetisation, which translates logical functions into polynomials over a finite field. This technique works doesn't work for realistically powerful provers or for proof steps that need to be judged by a human oracle.

2.3 Debate as a Complexity Class

We define a complexity class capturing the idealized *debate* setup. Intuitively, there are two players, Alice and Bob, who alternately make polynomial-length moves, and at the end a polynomial-time judge decides the winner from the full transcript.

Definition 2.1 (Judge)**.** A *Judge*, or debate verifier, is a deterministic polynomial-time algorithm

$$J(x, a_1, b_1, \ldots, a_{k(n)}, b_{k(n)}) \in \{0,1\},$$

where:

- $x \in \{0,1\}^n$ is the input, representing a question, e.g., "What is the best move in a given game of connect four?",
- $k(n)$ is a polynomially bounded number of rounds,
- each move a_i or b_i is a bit string of length at most $p(n)$ for some polynomial p. These represent arguments by Alice and counter-arguments by Bob, e.g. Alice: "Red cannot play in slot s, because blue playing $s+1$ would win within two moves.", and Bob: "Blue cannot play $s+1$, because red playing $s+1$ thereafter directly wins for red."

The judge outputs 1 if Alice wins and 0 if Bob wins.

Definition 2.2 (The class Debate)**.** A language $L \subseteq \{0,1\}^*$ is in Debate if there exist polynomials p, k and a debate verifier J such that for every input $x \in \{0,1\}^n$, iff $x \in L$, then

$$\underset{a_1 \in \{0,1\}^{\leq p(n)}}{\exists} \; \underset{b_1 \in \{0,1\}^{\leq p(n)}}{\forall} \cdots \underset{a_{k(n)} \in \{0,1\}^{\leq p(n)}}{\exists} \; \underset{b_{k(n)} \in \{0,1\}^{\leq p(n)}}{\forall} J(x, a_1, b_1, \ldots, a_{k(n)}, b_{k(n)}) = 1.$$

Equivalently, $x \in L$ if and only if Alice has a winning strategy in the polynomial-length debate defined by J.

Remark 2.1. This definition captures the idealized debate protocol used in the complexity-theoretic analysis of AI safety via debate: Alice attempts to defend a claim, Bob attempts to refute it, and the judge only performs a polynomial-time computation on the transcript.

This formulation is very close to a totally quantified Boolean Formula (TQBF), the canonically PSPACE-complete problem. The only differences is that for a TQBF, one would quantify over single bits, and consider a Boolean formula instead of a poly-time algorithm as judge. But these differences are somewhat cosmetic: One can rephrase

$$\exists_{a_i \in \{0,1\}^{\leq p(n)}} \quad \to \quad \exists a_i^1 \ldots \exists a_i^m, \quad \text{where} \quad m \leq p(n)$$

and

$$J(x, a_1, b_1, \ldots, a_{k(n)}, b_{k(n)}) \quad \to \quad \phi(x, a_1, b_1, \ldots, a_{k(n)}, b_{k(n)}),$$

where ϕ is a Boolean formula which has size polynomially in n by the Cook-Levin Theorem[1].

The easiest way to proof that DEBATE = PSPACE is thus showing that this is essentially the same. However, a more insightful way is to prove directly that Debate can solve problems in PSPACE, in a similar way how you show that TQBF is PSPACE complete.

2.4 Theorem: Debate = PSPACE

We now prove that the debate formalism has exactly the power of polynomial space computation.

Theorem 2.1.

$$\text{Debate} = \text{PSPACE}.$$

Proof. We prove both inclusions.

Step 1: PSPACE $\subseteq$ Debate. Let $L \in$ PSPACE. Then there exists a deterministic Turing machine M and a polynomial $s(n)$ such that, on every input x of length n, the machine M decides whether $x \in L$ using at most $s(n)$ tape cells.

Fix an input x. Since M uses only $s(n)$ space, the number of possible configurations of M on input x is at most

$$N = 2^{q(n)}$$

for some polynomial q. Let C_{start} be the start configuration of M on input x, and let C_{acc} denote the accepting configuration. Then $x \in L$ if and only if C_{acc} is reachable from C_{start} in the configuration graph of M.

The key point is that although this graph may have exponentially many nodes, a debate can verify reachability by recursively halving a path.

The reachability predicate. For configurations $C_{\text{left}}, C_{\text{right}}$ and an integer $t \geq 0$, define

$$\text{Reach}(C_{\text{left}}, C_{\text{right}}, t)$$

to mean that there is a path from C_{left} to C_{right} of length at most 2^t in the configuration graph of M.

Since the total number of configurations is at most $N = 2^{q(n)}$, any accepting computation path may be assumed to have length at most N. Thus

$$x \in L \iff \text{Reach}(C_{\text{start}}, C_{\text{acc}}, q(n)).$$

We now describe a debate protocol for $\text{Reach}(C_{\text{left}}, C_{\text{right}}, t)$, where $C_{\text{left}}, C_{\text{right}}$ are arbitrary states of the Turing machine.

Base case. If $t = 0$, then $\text{Reach}(C_{\text{left}}, C_{\text{right}}, 0)$ means that C_{right} is reachable from C_{left} in at most one step. This is equivalent to saying that either $C_{\text{left}} = C_{\text{right}}$, or C_{right} is an immediate successor of C_{left}. Since checking whether one configuration legally follows from another is a local computation, the verifier can decide this in polynomial time.

Recursive case. Suppose $t > 0$. Then

$$\text{Reach}(C_{\text{left}}, C_{\text{right}}, t)$$

[1] For further reading, see Wikipedia: Cook–Levin theorem.

holds if and only if there exists an intermediate configuration C_{mid} such that

$$\text{Reach}(C_{\text{left}}, C_{\text{mid}}, t-1) \quad \text{and} \quad \text{Reach}(C_{\text{mid}}, C_{\text{right}}, t-1).$$

Indeed, any path of length at most 2^t can be split at its midpoint into two subpaths of length at most 2^{t-1}, and conversely such two subpaths concatenate to a path of length at most 2^t.

This suggests the following debate:

- Alice claims that $\text{Reach}(C_{\text{left}}, C_{\text{right}}, t)$ holds.
- She provides a midpoint configuration C_{mid}.
- Bob then chooses which half of the claim to challenge:

$$\text{Reach}(C_{\text{left}}, C_{\text{mid}}, t-1) \qquad \text{or} \qquad \text{Reach}(C_{\text{mid}}, C_{\text{right}}, t-1).$$

- The debate continues recursively on the challenged subclaim.

Thus Alice defends the existence of a path by naming a midpoint, and Bob attacks by selecting the half he believes is false. Repeating this process recursively drives the dispute down to a base case $t = 0$, which the verifier can check directly.

Why this works. We prove by induction on t that Alice has a winning strategy in the debate for $\text{Reach}(C_{\text{left}}, C_{\text{right}}, t)$ if and only if $\text{Reach}(C_{\text{left}}, C_{\text{right}}, t)$ is true.

For the base case $t = 0$, the verifier checks the claim directly, so the statement is immediate.

For the inductive step, assume the claim holds for $t-1$. If $\text{Reach}(C_{\text{left}}, C_{\text{right}}, t)$ is true, then there exists some midpoint C_{mid} such that both

$$\text{Reach}(C_{\text{left}}, C_{\text{mid}}, t-1) \quad \text{and} \quad \text{Reach}(C_{\text{mid}}, C_{\text{right}}, t-1)$$

are true. Alice names such a C_{mid}. Whatever half Bob chooses to challenge, the challenged subclaim is true, and by the induction hypothesis Alice has a winning strategy in the resulting subdebate.

Conversely, if $\text{Reach}(C_{\text{left}}, C_{\text{right}}, t)$ is false, then for every proposed midpoint C_{mid}, at least one of the two subclaims

$$\text{Reach}(C_{\text{left}}, C_{\text{mid}}, t-1), \qquad \text{Reach}(C_{\text{mid}}, C_{\text{right}}, t-1)$$

must be false. Bob chooses such a false half. By the induction hypothesis, Alice cannot win the resulting subdebate. Hence she has no winning strategy in the original debate.

This completes the induction.

Complexity of the verifier and number of rounds. At each round, Alice provides one configuration E, whose description has polynomial length. Bob responds with one bit indicating which half he wants to challenge. The recursion depth is $q(n)$, which is polynomial in n. At the end, the verifier checks a base case $t = 0$, namely whether one configuration is equal to or an immediate successor of another, which is polynomial-time computable.

Therefore this is a valid polynomial-length debate with a polynomial-time verifier. Since Alice has a winning strategy exactly when

$$\text{Reach}(C_{\text{start}}, C_{\text{acc}}, q(n))$$

is true, the debate decides whether $x \in L$. It follows that $L \in \mathsf{Debate}$, and hence

$$\mathsf{PSPACE} \subseteq \mathsf{Debate}.$$

Step 2: $\mathsf{Debate} \subseteq \mathsf{PSPACE}$**.**

We can reduce Debater to TQBF as discussed before, and TQBF is in PSPACE-complete, thus in PSPACE. □

This challenge-defence recursion gives a good intuition how we expect a debate to play out between AI agents. The question is, how much does the verifier actually have to check to understand that this holds?

Exercise 2.2. Let

$$x \in L \iff \exists a_1 \forall b_1 \exists a_2 \forall b_2 \ : \ J(x, a_1, b_1, a_2, b_2) = 1.$$

Explain in plain English what this statement means. In particular, describe what it means for Alice to have a winning strategy in this debate, and how Bob's role is reflected by the universal quantifiers.

Exercise 2.3. Consider the quantified Boolean formula

$$\exists w \forall x \exists y \forall z \ \big((w \vee z) \wedge (x \vee \neg y) \wedge (\neg x \vee y)\big).$$

Interpret this formula as a debate game: Alice chooses the existentially quantified variables, Bob chooses the universally quantified variables. Determine whether Alice has a winning strategy. If she does, describe it explicitly.

Exercise 2.4. Recall the recursive reachability predicate

$$\mathrm{Reach}(C_{\mathrm{left}}, C_{\mathrm{right}}, t) \iff \exists C_{\mathrm{mid}} \ \big(\mathrm{Reach}(C_{\mathrm{left}}, C_{\mathrm{mid}}, t-1) \wedge \mathrm{Reach}(C_{\mathrm{mid}}, C_{\mathrm{right}}, t-1)\big).$$

Turn this recursive definition into a debate protocol. Describe precisely:what Alice claims, what message Alice sends in each round, what Bob sends in response, how the debate proceeds recursively, and what the verifier checks in the base case.

Exercise 2.5. Prove by induction on t that Alice has a winning strategy in the reachability debate for

$$\mathrm{Reach}(C_{\mathrm{left}}, C_{\mathrm{right}}, t)$$

if and only if there is a path from C_{left} to C_{right} of length at most 2^t.

Exercise 2.6. Suppose a deterministic Turing machine M uses at most $s(n)$ space on inputs of length n, and therefore has at most $2^{q(n)}$ configurations for some polynomial q. Analyse the complexity of the reachability debate protocol:

1. How many rounds are needed?
2. How long is Alice's message in each round?
3. How long is Bob's message in each round?
4. Why does this show that the protocol fits the definition of the class Debate?

2.5 Cross-examination raises debate to NEXP

Cross-Examination is an evolution of the debate protocol that includes non-communicating copies of the prover Alice that both Bob and the judge can access. We now prove the key lower bound explaining why cross-examination is more powerful than ordinary debate.

Definition 2.3 (Cross-examination)**.** A language $L \subseteq \{0,1\}^*$ is in CX if there exist a deterministic polynomial-time verifier U and a polynomial r such that for every input $x \in \{0,1\}^n$,

$$x \in L \iff \exists \mathcal{A} \ \forall \mathcal{B}^{\mathcal{A}} \ : \ U^{\mathcal{A},\mathcal{B}}(x) = 1,$$

Here:

- $\mathcal{A}$ is a deterministic Alice strategy that answers queries of length at most $r(n)$ with replies of length at most $r(n)$;
- $\mathcal{B}^{\mathcal{A}}$ is a deterministic Bob strategy which may make at most $r(n)$ adaptive queries of length at most $r(n)$ to fresh independent copies of $\mathcal{A}$;
- the verifier U may also make at most $r(n)$ adaptive queries of length at most $r(n)$ to $\mathcal{A}$ and to $\mathcal{B}$;
- the entire interaction has at most $r(n)$ rounds and all messages have length at most $r(n)$.

Since $\mathcal{A}$ is deterministic, all fresh copies of $\mathcal{A}$ answer the same query in the same way.

Theorem 2.2.

$$\mathsf{NEXP} \subseteq \mathsf{CX\text{-}Debate}.$$

Proof. Let $L \in \mathsf{NEXP}$. Then there exist a nondeterministic Turing machine N and a polynomial p such that N decides L in time

$$T(n) = 2^{p(n)}.$$

Without loss of generality, assume:

- N has a single tape;
- N uses at most $T(n)$ tape cells on inputs of length n;
- once N enters an accepting or rejecting halting state, it stays there forever and leaves the tape unchanged.

Fix an input $x \in \{0,1\}^n$, and let $T = T(|x|)$.

Step 1: Encode a computation as a tableau. A computation branch of N on input x can be encoded as a tableau

$$\mathcal{T} \in \Gamma^{(T+1)\times(T+1)},$$

where Γ is a constant-size alphabet encoding, for each tape cell and time step:

- the tape symbol in that cell,
- whether the head is on that cell,
- and, if so, the current state.

Row 0 is the initial configuration on input x, and row T is the final configuration after T steps.

We linearize the tableau row-by-row into a string

$$a \in \Gamma^{(T+1)(T+1)}.$$

Step 2: Local consistency conditions. The tableau $\mathcal{T}$ is an accepting computation of N on input x iff all of the following hold:

1. **Initial row condition:** row 0 correctly encodes the start configuration of N on input x.
2. **Accepting row condition:** row T is in an accepting halting configuration.
3. **Local transition conditions:** for every time $1 \le \tau \le T$ and every tape position $1 \le j \le T+1$, the symbol at tableau position (τ, j) is consistent with the machine transition rule applied to a constant-size neighborhood in the previous row, namely

$$(\tau-1, j-1),\ (\tau-1, j),\ (\tau-1, j+1).$$

These are exactly the usual local constraints from the Cook–Levin tableau construction: whether one cell in row τ is correct depends only on a constant-size window in row $\tau - 1$.

Step 3: The cross-examination protocol. We define the following protocol.

1. Alice outputs a string a, intended to be the linearized tableau of an accepting computation branch of N on input x.
2. Bob outputs a challenge

$$t = (\text{type}, \tau, j),$$

where type specifies which constraint is being challenged:

   - type = init: challenge the initial-row condition at cell j;
   - type = acc: challenge the accepting-row condition at cell j;
   - type = step: challenge the local transition condition at spacetime location (τ, j).

3. The verifier computes the queried coordinates $I(x,t)$ as follows:

- for init, query only the j-th cell of row 0;
- for acc, query only the j-th cell of row T;
- for step, query the constant-size neighborhood

$$(\tau-1, j-1),\ (\tau-1, j),\ (\tau-1, j+1),\ (\tau, j).$$

The verifier accepts iff the queried cells satisfy the corresponding local constraint.

Step 4: Completeness. Assume $x \in L$. Then N has some accepting computation branch on input x of length at most T. Let $\mathcal{T}$ be the tableau of that accepting branch, padded after halting so that it has exactly $T+1$ rows, and let a be its linearization.

If Alice outputs this a, then:

- the initial row is correct,
- the final row is accepting,
- every local transition constraint is satisfied.

Hence every challenge Bob can issue is answered correctly by the queried cells, and the verifier always accepts.

Step 5: Soundness. Assume $x \notin L$. Then N has no accepting computation branch on input x of length at most T.

Take any string a output by Alice, and interpret it as a tableau $\mathcal{T}$. Since there is no accepting computation tableau for x, $\mathcal{T}$ must violate at least one of the conditions above:

- either row 0 is not the correct initial configuration,
- or row T is not accepting,
- or some local transition condition fails at some (τ, j).

Bob outputs a challenge t pointing to such a violated condition. By construction, the verifier queries exactly the cells needed to check that local condition, detects the violation, and rejects.

Therefore Bob has a winning strategy whenever $x \notin L$.

Step 6: Verifier complexity. Each challenge $t = (\text{type}, \tau, j)$ uses only

$$O(\log T) = O(p(n)) = \text{poly}(n)$$

bits, since $\tau, j \in [T+1]$. The verifier reads only $O(1)$ tableau entries, and checking the corresponding local constraint is a polynomial-time computation in $|x|$. Hence the verifier runs in polynomial time.

Therefore the above is a valid cross-examination debate protocol for L, and so

$$L \in \mathsf{CX\text{-}Debate}.$$

Since $L \in \mathsf{NEXP}$ was arbitrary, we conclude that

$$\mathsf{NEXP} \subseteq \mathsf{CX\text{-}Debate}.$$

□

Exercise 2.7 (From ordinary debate to cross-examination)**.** Explain in your own words why the following ordinary debate protocol does *not* suffice to verify an exponentially long computation:

> Alice claims that an exponential-time machine M accepts x. Bob points to a suspicious time step t. Alice explains what happens at time t. The verifier checks the explanation.

What goes wrong if Alice is allowed to answer each question separately without being forced to remain globally consistent?

Exercise 2.8 (Why copies matter)**.** Suppose Alice is asked two separate questions about a claimed computation tableau:

- What symbol appears in row 17, column 5?
- What are the symbols in the local neighborhood around row 17, column 5?

Give an example of how Alice could answer these two questions inconsistently if she is not forced to commit to a single global tableau in advance.

Then explain how querying independent copies of Alice can be viewed as enforcing consistency with one fixed underlying object.

Exercise 2.9 (Reading a tableau)**.** Consider the following toy computation tableau:

$\tau \backslash j$	1	2	3	4	5
0	$[q_0, 1]$	0	1	$\sqcup$	$\sqcup$
1	1	$[q_0, 0]$	1	$\sqcup$	$\sqcup$
2	1	1	$[q_0, 1]$	$\sqcup$	$\sqcup$
3	1	1	1	$[q_0, \sqcup]$	$\sqcup$
4	1	0	1	$[q_{\text{acc}}, \sqcup]$	$\sqcup$

Answer the following:

1. What is the start configuration?
2. At which time step does the head first move onto the blank symbol?
3. Why is the final row accepting?

Exercise 2.10 (Local checks)**.** In the tableau above, Bob challenges the transition from row 2 to row 3 at column 4.

1. Which entries of the tableau should the verifier inspect in order to perform a local check?
2. Why is it enough to inspect only a constant-size neighborhood rather than the whole tableau?
3. Did Alice smuggle in a mistake into the table?

Exercise 2.11 (Designing a Bob challenge)**.** Suppose Alice presents a tableau $\mathcal{T}$ for an accepting computation. List the three main types of challenge Bob can make:

1. an initial-row challenge,
2. an accepting-row challenge,
3. a transition challenge.

For each type, explain exactly what Bob must specify, and exactly what the verifier checks.

Exercise 2.12 (Why this reaches NEXP)**.** Suppose a nondeterministic Turing machine N runs in time $T(n) = 2^{p(n)}$.

1. How large is a full accepting computation tableau for N?
2. Why can Alice not simply write the entire tableau down in an ordinary polynomial-length debate?
3. Why can Bob nevertheless challenge one local location of the tableau using only polynomially many bits?
4. Why can the verifier check that challenge in polynomial time?

Use your answers to explain why cross-examination can verify an exponentially long computation even though the verifier never reads the whole computation.

Exercise 2.13 (Cross-examination as precommitment)**.** Let Q be the set of all possible local queries Bob might ask about a tableau, for example:

$$Q = \{(\tau, j, \text{type})\}.$$

Explain why Alice's behavior under cross-examination can be modeled as a function

$$A_x : Q \to \Sigma,$$

where Σ is the set of possible local answers.

Why does this function behave like an exponentially large precommitment table?

Exercise 2.14 (Compare PSPACE and NEXP intuitions)**.** Write a short paragraph comparing the following two pictures:

- In the PSPACE reachability proof, Alice repeatedly gives a midpoint configuration and Bob chooses which half to challenge.
- In the NEXP cross-examination proof, Alice implicitly commits to a full exponentially large tableau and Bob challenges one local location.

What is the key conceptual difference between these two protocols?

Exercise 2.15 (Find the flaw)**.** A student says:

> "Cross-examination is unnecessary. Bob can just ask Alice for the value at position (τ, j), then ask her for the value at position $(\tau, j+1)$, and then reconstruct the whole tableau bit by bit."

Explain why this does not work in a polynomial-length protocol.