Interactive Proofs

Lecture 17

$IP = PSPACE$
So far
So far

IP
So far

- IP
- AM, MA
So far

- IP
- AM, MA
- GNI ∈ IP
So far

- IP
- AM, MA
- GNI ∈ IP
- GNI ∈ AM
So far

- IP
- AM, MA
- GNI ∈ IP
- GNI ∈ AM

Using AM protocol for set lower-bound
So far

- IP
- AM, MA
- GNI ∈ IP
- GNI ∈ AM
  - Using AM protocol for set lower-bound
  - In fact, IP[k] in AM[k+2]
IP = PSPACE
IP = PSPACE

Recall, IP means IP[poly]
IP = PSPACE

Recall, IP means IP[poly]

IP ⊆ PSPACE
IP = PSPACE

- Recall, IP means IP[poly]
- IP ⊆ PSPACE
  - Even though prover unbounded, cannot convince poly time verifier of everything
$IP = \text{PSPACE}$

- Recall, IP means IP[poly]
- $IP \subseteq \text{PSPACE}$
  - Even though prover unbounded, cannot convince poly time verifier of everything
- $\text{PSPACE} \subseteq IP$
IP = PSPACE

- Recall, IP means IP[\text{poly}]
- IP \subseteq PSPACE
  - Even though prover unbounded, cannot convince poly time verifier of everything
- PSPACE \subseteq IP
  - Prover can convince verifier of high complexity statements
\( \text{IP} \subseteq \text{PSPACE} \)
IP $\subseteq$ PSPACE

* Easier direction!
IP ⊆ PSPACE

- Easier direction!

- Plan: For given input calculate \( \Pr[\text{yes}] \) of honest verifier, maximum over all “prover strategies”
IP ⊆ PSPACE

- Easier direction!
- Plan: For given input calculate $\Pr[\text{yes}]$ of honest verifier, maximum over all “prover strategies”
  - Warm-up: public-coins (i.e., AM[poly])
IP ⊆ PSPACE

Easier direction!

Plan: For given input calculate Pr[yes] of honest verifier, maximum over all “prover strategies”

Warm-up: public-coins (i.e., AM[poly])

Could then use the “fact” that IP[poly]=AM[poly]
IP ⊆ PSPACE

- Easier direction!

- Plan: For given input calculate \( \Pr[\text{yes}] \) of honest verifier, maximum over all “prover strategies”

  - Warm-up: public-coins (i.e., AM[poly])

  - Could then use the “fact” that IP[poly]=AM[poly]

  - Or modify the proof (as we’ll do)
AM[poly] \subseteq \text{PSPACE}
AM[\text{poly}] \subseteq \text{PSPACE}

Plan: For given input calculate max Pr[yes] over all "prover strategies"
AM[poly] ⊆ PSPACE

Plan: For given input calculate max Pr[yes] over all "prover strategies"

Assume for convenience (w.l.o.g) each message is a single bit and P, V alternate
AM[poly] ⊆ PSPACE

Plan: For given input calculate max Pr[yes] over all “prover strategies”

Assume for convenience (w.l.o.g) each message is a single bit and P, V alternate

Protocol’s configuration tree: path to a node corresponds to the transcript so far
AM[poly] \subseteq PSPACE

Plan: For given input calculate max Pr[yes] over all “prover strategies”

Assume for convenience (w.l.o.g) each message is a single bit and P, V alternate

Protocol’s configuration tree: path to a node corresponds to the transcript so far
$\text{AM[poly]} \subseteq \text{PSPACE}$
$AM[poly] \subseteq PSPACE$

Plan: For given input calculate maximum value, over all “prover strategies,” of $Pr[\text{yes}]$
AM[\text{poly}] \subseteq \text{PSPACE}

Plan: For given input calculate maximum value, over all "prover strategies," of Pr[yes]

Note that finding the honest prover strategy may require super-PSPACE computation
AM[poly] \subseteq \text{PSPACE}

Plan: For given input calculate maximum value, over all "prover strategies," of Pr[yes]

Note that finding the honest prover strategy may require super-PSPACE computation

Recursively for each node, calculate maximum Pr[yes]
\(\text{AM[poly]} \subseteq \text{PSPACE}\)

Plan: For given input calculate maximum value, over all “prover strategies,” of \(\text{Pr[yes]}\)

Note that finding the honest prover strategy may require super-\(\text{PSPACE}\) computation

Recursively for each node, calculate maximum \(\text{Pr[yes]}\)

Leaves: \(\text{Pr[yes]} = 0\) or \(1\), determined by running verifier’s program
AM[poly] ⊆ PSPACE

Plan: For given input calculate maximum value, over all “prover strategies,” of Pr[yes]

Note that finding the honest prover strategy may require super-PSPACE computation

Recursively for each node, calculate maximum Pr[yes]

Leaves: Pr[yes] = 0 or 1, determined by running verifier’s program

P nodes: max of children
**AM[poly] ⊆ PSPACE**

Plan: For given input calculate maximum value, over all "prover strategies," of $Pr[yes]$

- Note that finding the honest prover strategy may require super-PSPACE computation

- Recursively for each node, calculate maximum $Pr[yes]$

  - Leaves: $Pr[yes] = 0$ or $1$, determined by running verifier's program
  - $P$ nodes: max of children
  - $V$ nodes: average of children
AM[poly] \subseteq \text{PSPACE}

Plan: For given input calculate maximum value, over all "prover strategies," of Pr[yes]

Note that finding the honest prover strategy may require super-PSPACE computation

Recursively for each node, calculate maximum Pr[yes]

Leaves: Pr[yes] = 0 or 1, determined by running verifier's program

P nodes: max of children

V nodes: average of children

In PSPACE: depth polynomial
IP ⊆ PSPACE
IP ⊆ PSPACE

Calculate max Pr[yes] when prover’s strategy can depend only on messages and not private coins
IP ⊆ PSPACE

- Calculate max $\Pr[\text{yes}]$ when prover’s strategy can depend only on messages and not private coins
- Maintain the set of consistent random-tapes at each $V$ node
IP ⊆ PSPACE

- Calculate max $\Pr[\text{yes}]$ when prover’s strategy can depend only on messages and not private coins.
- Maintain the set of consistent random-tapes at each $V$ node.
- Children of $V$ node not always chosen with $1/2$-1/2 probability. Instead weighted by fraction of consistent random-tapes.
IP ⊆ PSPACE

- Calculate max $\Pr[\text{yes}]$ when prover’s strategy can depend only on messages and not private coins.
- Maintain the set of consistent random-tapes at each $V$ node.
- Children of $V$ node not always chosen with $1/2-1/2$ probability. Instead weighted by fraction of consistent random-tapes.
- Leaves: $\Pr[\text{yes}]$ determined by running verifier’s program on all consistent random-tapes of verifier.
IP ⊆ PSPACE

- Calculate max $\Pr[\text{yes}]$ when prover’s strategy can depend only on messages and not private coins.
- Maintain the set of consistent random-tapes at each V node.
- Children of V node not always chosen with 1/2-1/2 probability. Instead weighted by fraction of consistent random-tapes.
- Leaves: $\Pr[\text{yes}]$ determined by running verifier’s program on all consistent random-tapes of verifier.
- P nodes: max of children.
IP ⊆ PSPACE

- Calculate max Pr[yes] when prover’s strategy can depend only on messages and not private coins.
- Maintain the set of consistent random-tapes at each V node.
- Children of V node not always chosen with 1/2-1/2 probability. Instead weighted by fraction of consistent random-tapes.
- Leaves: Pr[yes] determined by running verifier’s program on all consistent random-tapes of verifier.
- P nodes: max of children.
- V nodes: (weighted) average of children.
$\text{PSPACE} \subseteq \text{IP}$
PSPACE \subseteq \text{IP}

Enough to show an IP protocol for TQBF
PSPACE $\subseteq$ IP

- Enough to show an IP protocol for TQBF

- For any $L$ in PSPACE, both prover and verifier can first reduce input to a TQBF instance, and then prover proves its membership
PSPACE \subseteq IP

- Enough to show an IP protocol for TQBF
  - For any L in PSPACE, both prover and verifier can first reduce input to a TQBF instance, and then prover proves its membership

- Recall TQBF
PSPACE $\subseteq$ IP

- Enough to show an IP protocol for TQBF
- For any $L$ in PSPACE, both prover and verifier can first reduce input to a TQBF instance, and then prover proves its membership

Recall TQBF

- Decide whether a QBF is true or not
PSPACE $\subseteq$ IP

- Enough to show an IP protocol for TQBF

- For any $L$ in PSPACE, both prover and verifier can first reduce input to a TQBF instance, and then prover proves its membership

- Recall TQBF

  - Decide whether a QBF is true or not

  - QBF: $Q_1x_1 \ Q_2x_2 \ ... \ Q_nx_n \ F(x_1, ..., x_n)$ for quantifiers $Q_i$ and a formula $F$ on boolean variables
Arithmetization
Arithmetization

- A Boolean formula as a polynomial
Arithmetization

- A Boolean formula as a polynomial
- Arithmetic over a (finite, exponentially large) field
Arithmetization

- A Boolean formula as a polynomial
- Arithmetic over a (finite, exponentially large) field
- 0 and 1 (identities of addition and multiplication) instead of True and False
Arithmetization

A Boolean formula as a polynomial

- Arithmetic over a (finite, exponentially large) field

- 0 and 1 (identities of addition and multiplication) instead of True and False

- For formula F, polynomial P such that for boolean vector \( b \) and corresponding 0-1 vector \( x \) we have \( F(b) = P(x) \)
Arithmetization

- A Boolean formula as a polynomial
  - Arithmetic over a (finite, exponentially large) field
  - 0 and 1 (identities of addition and multiplication) instead of True and False
  - For formula F, polynomial P such that for boolean vector b and corresponding 0-1 vector x we have F(b) = P(x)
  - NOT: (1-x); AND: x.y
Arithmetization

- A Boolean formula as a polynomial
- Arithmetic over a (finite, exponentially large) field
- 0 and 1 (identities of addition and multiplication) instead of True and False

For formula F, polynomial P such that for boolean vector b and corresponding 0-1 vector x we have F(b) = P(x)

- NOT: (1-x); AND: x.y
- OR (as NOT of AND of NOT): 1 - (1-x).(1-y)
Arithmetization

A Boolean formula as a polynomial

Arithmetic over a (finite, exponentially large) field

0 and 1 (identities of addition and multiplication) instead of True and False

For formula F, polynomial P such that for boolean vector b and corresponding 0-1 vector x we have F(b) = P(x)

NOT: (1-x); AND: x.y

OR (as NOT of AND of NOT): 1 - (1-x).(1-y)

Exercise: Arithmetize x=y (now!). Degree? Size?
Arithmetization

A Boolean formula as a polynomial

Arithmetic over a (finite, exponentially large) field

0 and 1 (identities of addition and multiplication) instead of True and False

For formula $F$, polynomial $P$ such that for boolean vector $b$ and corresponding 0-1 vector $x$ we have $F(b) = P(x)$

NOT: $(1-x)$; AND: $x.y$

OR (as NOT of AND of NOT): $1 - (1-x).(1-y)$

Exercise: Arithmetize $x=y$ (now!). Degree? Size?

Can always use a polynomial linear in each variable since $x^n = x$ for $x=0$ and $x=1$
Arithmetization
Arithmetization

- A QBF as a polynomial
Arithmetization

- A QBF as a polynomial
  - TRUE will correspond to $> 0$, and FALSE, $= 0$
Arithmetization

A QBF as a polynomial

TRUE will correspond to $> 0$, and FALSE, $= 0$

Suppose for Boolean formula $F$, polynomial $P$
Arithmetization

A QBF as a polynomial

TRUE will correspond to $> 0$, and FALSE, $= 0$

Suppose for Boolean formula $F$, polynomial $P$

$\exists x \ F(x) \rightarrow P(0) + P(1) > 0 \ (i.e., \ \sum_{x=0,1} P(x) > 0)$
Arithmetization

A QBF as a polynomial

TRUE will correspond to > 0, and FALSE, = 0

Suppose for Boolean formula $F$, polynomial $P$

$$\exists x \ F(x) \rightarrow P(0) + P(1) > 0 \quad (i.e., \ \sum_{x=0,1} P(x) > 0)$$

$$\forall x \ F(x) \rightarrow P(0).P(1) > 0 \quad (i.e., \ \prod_{x=0,1} P(x) > 0)$$
Arithmetization

A QBF as a polynomial

TRUE will correspond to  > 0, and FALSE,  = 0

Suppose for Boolean formula F, polynomial P

\( \exists x \ F(x) \rightarrow P(0) + P(1) > 0 \) (i.e., \( \Sigma_{x=0,1} P(x) > 0 \))

\( \forall x \ F(x) \rightarrow P(0).P(1) > 0 \) (i.e., \( \Pi_{x=0,1} P(x) > 0 \))

Extends to more quantifiers: i.e., if F(x) is a QBF above
**Arithmetization**

- **A QBF as a polynomial**
  - TRUE will correspond to $> 0$, and FALSE, $= 0$
  - Suppose for Boolean formula $F$, polynomial $P$
  - $\exists x \; F(x) \rightarrow P(0) + P(1) > 0$ (i.e., $\sum_{x=0,1} P(x) > 0$)
  - $\forall x \; F(x) \rightarrow P(0) \cdot P(1) > 0$ (i.e., $\prod_{x=0,1} P(x) > 0$)

- Extends to more quantifiers: i.e., if $F(x)$ is a QBF above
  - So, how do you arithmetize $\exists x \forall y \; G(x,y)$ and $\forall y \exists x \; G(x,y)$?
Arithmetization

A QBF as a polynomial

TRUE will correspond to $> 0$, and FALSE, $= 0$

Suppose for Boolean formula $F$, polynomial $P$

$\exists x \ F(x) \rightarrow P(0) + P(1) > 0$ (i.e., $\sum_{x=0,1} P(x) > 0$)

$\forall x \ F(x) \rightarrow P(0).P(1) > 0$ (i.e., $\prod_{x=0,1} P(x) > 0$)

Extends to more quantifiers: i.e., if $F(x)$ is a QBF above

So, how do you arithmetize $\exists x \forall y \ G(x,y)$ and $\forall y \exists x \ G(x,y)$?

$\sum_{x=0,1} \prod_{y=0,1} P(x,y) > 0$ and $\prod_{y=0,1} \sum_{x=0,1} P(x,y) > 0$
Arithmetization
Arithmetization

For a protocol for TQBF: Give a protocol for proving that 
$Q_1(x_1=0,1) \land Q_2(x_2=0,1) \land \ldots \land Q_n(x_n=0,1) \land P(x_1,\ldots,x_n) > 0$, where $Q_i$ are 
$\Sigma$ or $\Pi$, and $P$ is a (multi-linear) polynomial
Arithmetization

For a protocol for TQBF: Give a protocol for proving that $Q_1(x_1=0,1) \land Q_2(x_2=0,1) \land \ldots \land Q_n(x_n=0,1) \land P(x_1,\ldots,x_n) > 0$, where $Q_i$ are $\Sigma$ or $\Pi$, and $P$ is a (multi-linear) polynomial.

Instead suppose all $Q_i$ are $\Sigma$. 
For a protocol for TQBF: Give a protocol for proving that
\[ Q_1(x_1=0,1) \land Q_2(x_2=0,1) \land \cdots \land Q_n(x_n=0,1) \land P(x_1,\ldots,x_n) > 0, \]
where \( Q_i \) are \( \Sigma \) or \( \Pi \), and \( P \) is a (multi-linear) polynomial.

Instead suppose all \( Q_i \) are \( \Sigma \)

Counts number of satisfying assignments to an (unquantified) boolean formula \( F \).
Arithmetization

For a protocol for TQBF: Give a protocol for proving that
\[ Q_1(x_1=0,1) \land Q_2(x_2=0,1) \land \ldots \land Q_n(x_n=0,1) \land P(x_1,\ldots,x_n) > 0, \]
where \( Q_i \) are \( \Sigma \) or \( \Pi \), and \( P \) is a (multi-linear) polynomial

Instead suppose all \( Q_i \) are \( \Sigma \)

Counts number of satisfying assignments to an (unquantified) boolean formula \( F \)

Proving \( > 0 \) is trivial
Arithmetization

For a protocol for TQBF: Give a protocol for proving that

\[ Q_1(x_1=0,1) \ Q_2(x_2=0,1) \ \ldots \ Q_n(x_n=0,1) \ P(x_1,\ldots,x_n) > 0, \]

where \( Q_i \) are \( \Sigma \) or \( \Pi \), and \( P \) is a (multi-linear) polynomial

Instead suppose all \( Q_i \) are \( \Sigma \)

Counts number of satisfying assignments to an (unquantified) boolean formula \( F \)

Proving \( > 0 \) is trivial

Consider proving \( = K \) (will be useful in the general case)
Sum-check protocol
Sum-check protocol

To prove: $\Sigma_{x_1} \ldots \Sigma_{x_n} P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$
Sum-check protocol

To prove: $\Sigma_{x_1} \ldots \Sigma_{x_n} P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$
Sum-check protocol

To prove: $\sum_{x_1} \ldots \sum_{x_n} P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$

Note: to evaluate need to add up $2^n$ values
Sum-check protocol

To prove: $\sum_{x_1} \ldots \sum_{x_n} P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$

Note: to evaluate need to add up $2^n$ values

Base case: $n=0$. Verifier will simply use oracle access to $P$. 
Sum-check protocol

To prove: $\sum_{x_1} \ldots \sum_{x_n} P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$

Note: to evaluate need to add up $2^n$ values

Base case: $n=0$. Verifier will simply use oracle access to $P$.

For $n>0$: Let $R(X) := \sum_{x_2} \ldots \sum_{x_n} P(X, x_2, \ldots, x_n)$
Sum-check protocol

To prove: $\sum_{x_1} \ldots \sum_{x_n} P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$

Note: to evaluate need to add up $2^n$ values

Base case: $n=0$. Verifier will simply use oracle access to $P$.

For $n>0$: Let $R(X) := \sum_{x_2} \ldots \sum_{x_n} P(X, x_2, \ldots, x_n)$

$\sum_{x_1} \ldots \sum_{x_n} P(x_1, \ldots, x_n) = R(0) + R(1)$
Sum-check protocol

To prove: $\Sigma_{x_1} \ldots \Sigma_{x_n} P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$

- Note: to evaluate need to add up $2^n$ values

- Base case: $n=0$. Verifier will simply use oracle access to $P$.

- For $n>0$: Let $R(X) := \Sigma_{x_2} \ldots \Sigma_{x_n} P(X, x_2, \ldots, x_n)$

  $\Sigma_{x_1} \ldots \Sigma_{x_n} P(x_1, \ldots, x_n) = R(0) + R(1)$

  - $R$ has only one variable and degree at most $d$
Sum-check protocol

To prove: $\sum_{x_1} \ldots \sum_{x_n} P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$

Note: to evaluate need to add up $2^n$ values

Base case: $n=0$. Verifier will simply use oracle access to $P$.

For $n>0$: Let $R(X) := \sum_{x_2} \ldots \sum_{x_n} P(X, x_2, \ldots, x_n)$

$\sum_{x_1} \ldots \sum_{x_n} P(x_1, \ldots, x_n) = R(0) + R(1)$

$R$ has only one variable and degree at most $d$
Sum-check protocol

To prove: $\Sigma_{x_1} \ldots \Sigma_{x_n} P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$

Note: to evaluate need to add up $2^n$ values

Base case: $n=0$. Verifier will simply use oracle access to $P$.

For $n>0$: Let $R(X) := \Sigma_{x_2} \ldots \Sigma_{x_n} P(X, x_2, \ldots, x_n)$

$\Sigma_{x_1} \ldots \Sigma_{x_n} P(x_1, \ldots, x_n) = R(0) + R(1)$

$R$ has only one variable and degree at most $d$

Prover sends $T=R$ (as $d+1$ coefficients) to verifier

Verifier has only oracle access to $P$
Sum-check protocol

To prove: $\Sigma_{x_1} \ldots \Sigma_{x_n} P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$

Note: to evaluate need to add up $2^n$ values

Base case: $n=0$. Verifier will simply use oracle access to $P$.

For $n>0$: Let $R(X) := \Sigma_{x_2} \ldots \Sigma_{x_n} P(x, x_2, \ldots, x_n)$

$\Sigma_{x_1} \ldots \Sigma_{x_n} P(x_1, \ldots, x_n) = R(0) + R(1)$

$R$ has only one variable and degree at most $d$

Prover sends $T=R$ (as $d+1$ coefficients) to verifier

Verifier has only oracle access to $P$
Sum-check protocol

To prove: $\Sigma x_1 \ldots \Sigma x_n P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$

Note: to evaluate need to add up $2^n$ values

Base case: $n=0$. Verifier will simply use oracle access to $P$.

For $n>0$: Let $R(X) := \Sigma x_2 \ldots \Sigma x_n P(X, x_2, \ldots, x_n)$

$\Sigma x_1 \ldots \Sigma x_n P(x_1, \ldots, x_n) = R(0) + R(1)$

$R$ has only one variable and degree at most $d$

Prover sends $T=R$ (as $d+1$ coefficients) to verifier

Verifier checks $K = T(0) + T(1)$. Still needs to check $T=R$
Sum-check protocol
Sum-check protocol

To prove: $\Sigma_{x_1} \ldots \Sigma_{x_n} P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$
Sum-check protocol

To prove: $\Sigma x_1 \ldots \Sigma x_n P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$

Verifier wants to check $T(X) = R(X) := \Sigma x_2 \ldots \Sigma x_n P(X, x_2, \ldots, x_n)$
Sum-check protocol

To prove: \( \Sigma x_1 \ldots \Sigma x_n P(x_1, \ldots, x_n) = K \) for some degree d polynomial \( P \)

Verifier wants to check \( T(X) = R(X) := \Sigma x_2 \ldots \Sigma x_n P(X, x_2, \ldots, x_n) \)

Picks random field element \( a \) (large enough field)
Sum-check protocol

To prove: $\Sigma_{x_1} \ldots \Sigma_{x_n} P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$

Verifier wants to check $T(X) = R(X) := \Sigma_{x_2} \ldots \Sigma_{x_n} P(X, x_2, \ldots, x_n)$

Picks random field element $a$ (large enough field)

Asks prover to prove that $T(a) = R(a) = \Sigma_{x_2} \ldots \Sigma_{x_n} P(a, x_2, \ldots, x_n)$
Sum-check protocol

- To prove: $\Sigma x_1 \ldots \Sigma x_n P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$

- Verifier wants to check $T(X) = R(X) := \Sigma x_2 \ldots \Sigma x_n P(X, x_2, \ldots, x_n)$

- Picks random field element $a$ (large enough field)

- Asks prover to prove that $T(a) = R(a) = \Sigma x_2 \ldots \Sigma x_n P(a, x_2, \ldots, x_n)$

- Recurse on $P_1(x_2, \ldots, x_n) = P(a, x_2, \ldots, x_n)$ of one variable less
Sum-check protocol

To prove: \( \Sigma_{x_1} \ldots \Sigma_{x_n} P(x_1, \ldots, x_n) = K \) for some degree \( d \) polynomial \( P \)

Verifier wants to check \( T(X) = R(X) := \Sigma_{x_2} \ldots \Sigma_{x_n} P(X, x_2, \ldots, x_n) \)

Picks random field element \( a \) (large enough field)

Asks prover to prove that \( T(a) = R(a) = \Sigma_{x_2} \ldots \Sigma_{x_n} P(a, x_2, \ldots, x_n) \)

Recurse on \( P_1(x_2, \ldots, x_n) = P(a, x_2, \ldots, x_n) \) of one variable less

i.e., Recurse to prove \( \Sigma_{x_2} \ldots \Sigma_{x_n} P_1(x_2, \ldots, x_n) = T(a) \)
Sum-check protocol

To prove: $\Sigma_{x_1} \ldots \Sigma_{x_n} P(x_1, \ldots, x_n) = K$ for some degree $d$ polynomial $P$

Verifier wants to check $T(X) = R(X) := \Sigma_{x_2} \ldots \Sigma_{x_n} P(X, x_2, \ldots, x_n)$

Picks random field element $a$ (large enough field)

Asks prover to prove that $T(a) = R(a) = \Sigma_{x_2} \ldots \Sigma_{x_n} P(a, x_2, \ldots, x_n)$

Recurse on $P_1(x_2, \ldots, x_n) = P(a, x_2, \ldots, x_n)$ of one variable less

i.e., Recurse to prove $\Sigma_{x_2} \ldots \Sigma_{x_n} P_1(x_2, \ldots, x_n) = T(a)$

Note: $P_1$ has degree at most $d$; verifier has oracle access to $P_1$ (as it knows $a$, and has oracle access to $P$)
Sum-check protocol
Sum-check protocol

Why does sum-check protocol work?
Sum-check protocol

Why does sum-check protocol work?

Instead of checking $T(X) = R(X)$, simply checks (recursively) if $T(a) = R(a)$ for a single random $a$ in the field
Sum-check protocol

Why does sum-check protocol work?

Instead of checking $T(X) = R(X)$, simply checks (recursively) if $T(a) = R(a)$ for a single random $a$ in the field.
Why does sum-check protocol work?

Instead of checking $T(X) = R(X)$, simply checks (recursively) if $T(a)=R(a)$ for a single random $a$ in the field.

Completeness is obvious.
Sum-check protocol

Why does sum-check protocol work?

Instead of checking $T(X) = R(X)$, simply checks (recursively) if $T(a) = R(a)$ for a single random $a$ in the field.

Completeness is obvious.

Soundness: Since $T(X)$ and $R(X)$ are of degree $d$, if $T \neq R$, at most $d$ points where they agree.
Sum-check protocol

Why does sum-check protocol work?

- Instead of checking $T(X) = R(X)$, simply checks (recursively) if $T(a) = R(a)$ for a single random $a$ in the field.

  Completeness is obvious.

  Soundness: Since $T(X)$ and $R(X)$ are of degree $d$, if $T \neq R$, at most $d$ points where they agree.

  Error (picking a bad $a$), with probability $\leq d/p$, where field is of size $p$.
Sum-check protocol

Why does sum-check protocol work?

Instead of checking $T(X) = R(X)$, simply checks (recursively) if $T(a) = R(a)$ for a single random $a$ in the field.

Completeness is obvious.

Soundness: Since $T(X)$ and $R(X)$ are of degree $d$, if $T \neq R$, at most $d$ points where they agree.

Error (picking a bad $a$), with probability $\leq \frac{d}{p}$, where $p$ is the size of the field.

Also possible error in recursive step (despite good $a$).
Sum-check protocol

Why does sum-check protocol work?

- Instead of checking $T(X) = R(X)$, simply checks (recursively) if $T(a) = R(a)$ for a single random $a$ in the field

- Completeness is obvious

- Soundness: Since $T(X)$ and $R(X)$ are of degree $d$, if $T \neq R$, at most $d$ points where they agree

  - Error (picking a bad $a$), with probability $\leq d/p$, where field is of size $p$

  - Also possible error in recursive step (despite good $a$)

    - At most $nd/p$ if $n$ variables. Can take $p$ exponential.
IP Protocol for TQBF
For a protocol for TQBF: Give a protocol for proving that $Q_1(x_1=0,1) Q_2(x_2=0,1) \ldots Q_n(x_n=0,1) P(x_1,\ldots,x_n) > 0$, where $Q_i$ are $\Sigma$ or $\Pi$ and $P$ is a multi-linear polynomial.
IP Protocol for TQBF

For a protocol for TQBF: Give a protocol for proving that $Q_1(x_1=0,1) \ Q_2(x_2=0,1) \ ... \ Q_n(x_n=0,1) \ P(x_1,\ldots,x_n) > 0$, where $Q_i$ are $\Sigma$ or $\Pi$ and $P$ is a multi-linear polynomial

In fact a protocol to prove: $Q_1 \ x_1 \ ... \ Q_n \ x_n \ P(x_1,\ldots,x_n) = K$
IP Protocol for TQBF

- For a protocol for TQBF: Give a protocol for proving that
  \[ Q_1(x_1=0,1) \cdot Q_2(x_2=0,1) \cdots Q_n(x_n=0,1) \cdot P(x_1,\ldots, x_n) > 0, \text{ where } Q_i \text{ are } \Sigma \text{ or } \Pi \text{ and } P \text{ is a multi-linear polynomial} \]

- In fact a protocol to prove: \[ Q_1 \cdot x_1 \cdots Q_n \cdot x_n \cdot P(x_1,\ldots, x_n) = K \]

- Problem with generalizing sum-check protocol: the univariate poly
  \[ R(X) := Q_2(x_2) \cdots Q_n(x_n) \cdot P(X, x_2,\ldots, x_n) \] has exponential degree. Verifier can’t read \( T(X)=R(X) \)
IP Protocol for TQBF

For a protocol for TQBF: Give a protocol for proving that

\[ Q_1(x_1=0,1) \ Q_2(x_2=0,1) \ \ldots \ Q_n(x_n=0,1) \ P(x_1,\ldots,x_n) > 0, \] where \( Q_i \) are \( \Sigma \) or \( \Pi \) and \( P \) is a multi-linear polynomial.

In fact a protocol to prove: \( Q_1 x_1 \ldots Q_n x_n P(x_1,\ldots,x_n) = K \)

Problem with generalizing sum-check protocol: the univariate poly

\[ R(X) := Q_2 x_2 \ldots Q_n x_n P(X,x_2,\ldots,x_n) \] has exponential degree. Verifier can't read \( T(X)=R(X) \)

Instead of \( T \), can work with "linearization" of \( T \)
For a protocol for TQBF: Give a protocol for proving that

\[ Q_1(x_1=0,1) \ Q_2(x_2=0,1) \ldots \ Q_n(x_n=0,1) \ P(x_1,\ldots,x_n) > 0, \]

where \( Q_i \) are \( \Sigma \) or \( \Pi \) and \( P \) is a multi-linear polynomial.

In fact a protocol to prove:

\[ Q_1 x_1 \ldots Q_n x_n P(x_1,\ldots,x_n) = K \]

Problem with generalizing sum-check protocol: the univariate poly

\[ R(X) := Q_2 x_2 \ldots Q_n x_n P(X,x_2,\ldots,x_n) \]

has exponential degree. Verifier can't read \( T(X)=R(X) \).

Instead of \( T \), can work with "linearization" of \( T \).

Prover sends \( L(X) = ( T(1)-T(0) ) \ X + T(0) \).
IP Protocol for TQBF

For a protocol for TQBF: Give a protocol for proving that
\[ Q_1(x_1=0,1) \ Q_2(x_2=0,1) \ \ldots \ Q_n(x_n=0,1) \ \ P(x_1,\ldots,x_n) > 0, \]
where \( Q_i \) are \( \Sigma \) or \( \Pi \)
and \( P \) is a multi-linear polynomial.

In fact a protocol to prove: \[ Q_1 \ x_1 \ldots \ Q_n \ x_n \ P(x_1,\ldots,x_n) = K \]

Problem with generalizing sum-check protocol: the univariate poly \( R(X) := Q_2 x_2 \ldots Q_n x_n P(X,x_2,\ldots,x_n) \) has exponential degree. Verifier can't read \( T(X) = R(X) \).

Instead of \( T \), can work with “linearization” of \( T \)

Prover sends \( L(X) = ( T(1) - T(0) ) \ X + T(0) \)
Verifier picks random \( a \), and asks prover to show \( R'(a) = L(a) \)
For a protocol for TQBF: Give a protocol for proving that
\[ Q_1(x_1=0,1) \land Q_2(x_2=0,1) \land \ldots \land Q_n(x_n=0,1) \land P(x_1,\ldots,x_n) > 0, \]
where \( Q_i \) are \( \Sigma \) or \( \Pi \) and \( P \) is a multi-linear polynomial.

In fact a protocol to prove: \( Q_1 \land \ldots \land Q_n \land P(x_1,\ldots,x_n) = K \)

Problem with generalizing sum-check protocol: the univariate poly
\[ R(X) := Q_2(x_2) \land \ldots \land Q_n(x_n) \land P(X,x_2,\ldots,x_n) \]
has exponential degree. Verifier can't read \( T(X)=R(X) \)

Instead of \( T \), can work with "linearization" of \( T \)

Prover sends \( L(X) = (T(1)-T(0)) \times X + T(0) \)
Verifier picks random \( a \), and asks prover to show \( R'(a) = L(a) \)
IP Protocol for TQBF

For a protocol for TQBF: Give a protocol for proving that $Q_1(x_1=0,1) Q_2(x_2=0,1) ... Q_n(x_n=0,1) P(x_1, ..., x_n) > 0$, where $Q_i$ are $\Sigma$ or $\Pi$ and $P$ is a multi-linear polynomial

In fact a protocol to prove: $Q_1 x_1 ... Q_n x_n P(x_1, ..., x_n) = K$

Problem with generalizing sum-check protocol: the univariate poly $R(X) := Q_2 x_2 ... Q_n x_n P(X,x_2, ..., x_n)$ has exponential degree. Verifier can’t read $T(X)=R(X)$

Instead of $T$, can work with “linearization” of $T$

- Prover sends $L(X) = ( T(1)-T(0) ) X + T(0)$
- Verifier picks random $a$, and asks prover to show $R'(a) = L(a)$
- Verifier checks (as appropriate) $L(1).L(0) = K$ or $L(1)+L(0) = K$
IP Protocol for TQBF
IP Protocol for TQBF

IP = PSPACE
IP Protocol for TQBF

- IP = PSPACE
- Protocol is public-coin
IP Protocol for TQBF

- $\text{IP} = \text{PSPACE}$
- Protocol is public-coin
- $\text{IP} = \text{AM[poly]} = \text{PSPACE}$
IP Protocol for TQBF

- $IP = PSPACE$
- Protocol is public-coin
  - $IP = AM[poly] = PSPACE$
- Protocol has perfect completeness