TECHNISCHE UNIVERSITÄT DRESDEN

COMPLEXITY THEORY

Lecture 26: Interactive Proof Systems

Markus Krötzsch

Knowledge-Based Systems

TU Dresden, 3rd Feb 2020

Believing without proof?

"Real mathematicians should only believe in mathematical statements that they can prove themselves!"

Believing without proof?

"Real mathematicians should only believe in mathematical statements that they can prove themselves!"

Is this a sensible statement?

In other words: Could a rational mathematician be convinced of a formal claim without having the slightest idea of how to prove it?

Motivation: Provers and Verifiers

Recall: languages in NP admit short, easy-to-check membership certificates

NP membership checking as an interaction of two parties:

- The Prover produces a certificate (proof of membership) that it claims to be valid
- The Verifier validates the certificate to decide upon acceptance

Motivation: Provers and Verifiers

Recall: languages in NP admit short, easy-to-check membership certificates

NP membership checking as an interaction of two parties:

- The Prover produces a certificate (proof of membership) that it claims to be valid
- The Verifier validates the certificate to decide upon acceptance

Can we generalise this idea?

- A (untrusted) Prover tries to convince the Verifier of membership
- Verifier sceptically checks the Prover's arguments before making a decision
- The interaction might involve several rounds of communication
- The Prover might have unbounded computational power,
  but the Verifier should operate in P

Motivation: Provers and Verifiers

Recall: languages in NP admit short, easy-to-check membership certificates

NP membership checking as an interaction of two parties:

- The Prover produces a certificate (proof of membership) that it claims to be valid
- The Verifier validates the certificate to decide upon acceptance

Can we generalise this idea?

- A (untrusted) Prover tries to convince the Verifier of membership
- Verifier sceptically checks the Prover's arguments before making a decision
- The interaction might involve several rounds of communication
- The Prover might have unbounded computational power,
  but the Verifier should operate in P

For which languages can such a polytime Verifier ensure that it can be convinced of membership exactly for the words that really are in the language?

Example: Graph Isomorphism

We consider (undirected) graphs over a set of numbered vertices $1, 2, \ldots, n$.

Two graphs are isomorphic if one can be obtained from the other by a bijective renaming (permutation) of vertices.

GRAPH ISOMORPHISM

Input: Two graphs G_1 and G_2.

Problem: Is G_1 isomorphic to G_2?

Example: Graph Isomorphism

We consider (undirected) graphs over a set of numbered vertices $1, 2, \ldots, n$.

Two graphs are isomorphic if one can be obtained from the other by a bijective renaming (permutation) of vertices.

GRAPH ISOMORPHISM

Input:	Two graphs G_1 and G_2.
Problem:	Is G_1 isomorphic to G_2?

Observations:

- **GRAPH ISOMORPHISM** is in NP (certificate: renaming)
- There are $n!$ many potential permutations, so exhaustive checking requires exponential time

Example: Graph Isomorphism

We consider (undirected) graphs over a set of numbered vertices $1, 2, \ldots, n$.

Two graphs are isomorphic if one can be obtained from the other by a bijective renaming (permutation) of vertices.

Graph Isomorphism

Input:	Two graphs G_1 and G_2.
Problem:	Is G_1 isomorphic to G_2?

Observations:

- **Graph Isomorphism** is in NP (certificate: renaming)
- There are $n!$ many potential permutations, so exhaustive checking requires exponential time

However, **Graph Isomorphism** is not know (or believed) to be NP-hard

Graph Non-Isomorphism

Graph Non-Isomorphism

Input: Two graphs G_1 and G_2.

Problem: Is G_1 not isomorphic to G_2?

There does not seem to be a short certificate for this, but there is an interactive protocol:

Graph Non-Isomorphism

Graph Non-Isomorphism

Input:	Two graphs G_1 and G_2.
Problem:	Is G_1 not isomorphic to G_2?

There does not seem to be a short certificate for this, but there is an interactive protocol:

Protocol: given non-isomorphic graphs G_1 and G_2

- Verifier: randomly select $i \in \{1, 2\}$; randomly permute vertices of G_i to obtain a new graph H; send H to the Prover
- Prover: determine which G_j ($j \in \{1, 2\}$) the graph H is isomorphic to; send j
- Verifier: accept if $i = j$, else reject

Graph Non-Isomorphism

GRAPH NON-ISOMORPHISM

Input: Two graphs G_1 and G_2.

Problem: Is G_1 not isomorphic to G_2?

There does not seem to be a short certificate for this, but there is an interactive protocol:

Protocol: given non-isomorphic graphs G_1 and G_2

- Verifier: randomly select $i \in \{1, 2\}$; randomly permute vertices of G_i to obtain a new graph H; send H to the Prover
- Prover: determine which G_j ($j \in \{1, 2\}$) the graph H is isomorphic to; send j
- Verifier: accept if $i = j$, else reject

Analysis: The Prover can ensure acceptance for non-isomorphic graphs, but for isomorphic graphs it can only achieve acceptance with probability 0.5 (which can be reduced further by repeating the interaction several times) □

Zero-Knowledge Proofs

Running the previous protocol is interestingly uninformative:

- The Verifier can be convinced that $\langle G_1, G_2\rangle \in$ **Graph Non-Isomorphism**
- But the Verifier learns nothing about the reasons
- In particular, the Verifier would not be able to prove this to anybody else

This is called a zero-knowledge proof.

Note: The mathematical property that characterises such proofs formally is that the Verifier could have produced the whole interaction all by itself, without the assistance of a Prover. This would not convince the Verifier, of course, but would not be distinguishable otherwise.

Making interactive proofs formal (1)

The interaction can be viewed as a sequence of messages $m_1, m_2, \ldots, m_k$, followed by the Verifier declaring “accept” or “reject”.

The **Verifier** may consider the following:

- The input string w
- A string r of random bits (certificate-style view of random computation)
- A (partial) message history $m_1\#m_2\#\cdots\#m_i$ of messages exchanged so far (odd-index messages are sent by Verifier, even-index messages by Prover)

$\leadsto$ Verifier can be described by a function $V : \Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma^* \cup \{\text{accept}, \text{reject}\}$

The **Prover** may consider the following:

- The input string w
- A (partial) message history $m_1\#m_2\#\cdots\#m_i$ of messages exchanged so far

$\leadsto$ Prover can be described by a function $P : \Sigma^* \times \Sigma^* \to \Sigma^*$

Making interactive proofs formal (2)

Definition 26.1: A word w is accepted by V and P with random string r if there is a sequence of messages $m_1\#m_2\#\cdots\#m_k$ such that

- for all even $i \geq 0$ we have $m_{i+1} = V(w, r, m_1\#\cdots\#m_i)$
- for all odd $i \geq 0$ we have $m_{i+1} = P(w, m_1\#\cdots\#m_i)$
- $m_k = \text{accept}$ (and in particular k is odd)

In this case, we write $(V \leftrightarrow P)(w, r) = \text{accept}$.

Making interactive proofs formal (2)

Definition 26.1: A word w is accepted by V and P with random string r if there is a sequence of messages $m_1\#m_2\#\cdots\#m_k$ such that

- for all even $i \geq 0$ we have $m_{i+1} = V(w, r, m_1\#\cdots\#m_i)$
- for all odd $i \geq 0$ we have $m_{i+1} = P(w, m_1\#\cdots\#m_i)$
- $m_k =$ accept (and in particular k is odd)

In this case, we write $(V \leftrightarrow P)(w, r) =$ accept.

Definition 26.2: A polynomial verifier V with bound p is a verifier function that ensures that, for all inputs w, random strings r, and provers P, at most $p(|w|)$ computation steps are performed overall (across all interactions).

Note: Polynomial verifiers could, for example, use messages to store the number of available steps that remain, and reject when this is used up.

Making interactive proofs formal (2)

Definition 26.1: A word w is accepted by V and P with random string r if there is a sequence of messages $m_1\#m_2\#\cdots\#m_k$ such that

- for all even $i \geq 0$ we have $m_{i+1} = V(w, r, m_1\#\cdots\#m_i)$
- for all odd $i \geq 0$ we have $m_{i+1} = P(w, m_1\#\cdots\#m_i)$
- $m_k =$ accept (and in particular k is odd)

In this case, we write $(V \leftrightarrow P)(w, r) =$ accept.

Definition 26.2: A polynomial verifier V with bound p is a verifier function that ensures that, for all inputs w, random strings r, and provers P, at most $p(|w|)$ computation steps are performed overall (across all interactions).

Note: Polynomial verifiers could, for example, use messages to store the number of available steps that remain, and reject when this is used up.

Definition 26.3: A polynomial verifier V with bound p and a prover P accept a word w with probability $\Pr[V \leftrightarrow P \text{ accepts } w] = \Pr_{r\in\{0,1\}^{p(|w|)}}[(V \leftrightarrow P)(w, r) = \text{accept}]$.

The class IP

We can now formally define a class of languages that are accepted by polytime Verifiers using interactive proofs:

Definition 26.4: A language **L** is in IP if there is a polynomial verifier V such that, for every word w:

(1) if $w \in \mathbf{L}$ then there is a prover P with $\Pr[V \leftrightarrow P \text{ accepts } w] \geq \frac{2}{3}$,

(2) if $w \notin \mathbf{L}$ then for all provers $\tilde{P}$ we have $\Pr\left[V \leftrightarrow \tilde{P} \text{ accepts } w\right] \leq \frac{1}{3}$.

In words:

- there is a "good" prover P that can convince V to accept $w \in \mathbf{L}$ with high probability

  (note that the existence of one good prover for each $w \in \mathbf{L}$ implies that there is one globally good prover)

- not even a "bad" prover $\tilde{P}$ can convince V to accept words $w \notin \mathbf{L}$ with more than a low probability

Probabilistic interactions

The definition of IP uses probabilistic computations

- The Verifier is a polynomially time-bounded probabilistic TM
- The Prover does not use randomness (and including it would not change IP)
- As discussed for BPP, we can amplify probabilities; in particular, the bounds $\frac{2}{3}$ and $\frac{1}{3}$ are not essential to the definition

Probabilistic interactions

The definition of IP uses probabilistic computations

- The Verifier is a polynomially time-bounded probabilistic TM
- The Prover does not use randomness (and including it would not change IP)
- As discussed for BPP, we can amplify probabilities; in particular, the bounds $\frac{2}{3}$ and $\frac{1}{3}$ are not essential to the definition

The use of randomness in the Verifier is important for expressive power:

Theorem 26.5: Let IP_d be the restriction of IP that is obtained when requiring V to be deterministic (ignoring the random bits). Then $\mathrm{IP}_d = \mathrm{NP}$.

The proof is not hard (exercise; or see Arora & Barak, Lemma 8.4)

Obvious sub-classes of IP

Some observations are straightforward:

Theorem 26.6: NP $\subseteq$ IP.

Proof: Use definition of NP via polynomial-time verifiers. □

Theorem 26.7: BPP $\subseteq$ IP.

Proof: Verifier can solve BPP problems without talking to Prover. □

A superclass of IP

Interestingly, we can use another well-known class to capture IP from above:

Theorem 26.8: IP $\subseteq$ PSpace.

A superclass of IP

Interestingly, we can use another well-known class to capture IP from above:

Theorem 26.8: IP $\subseteq$ PSpace.

Proof: Consider **L** $\in$ IP with polynomial verifier V. For any word w, let

$$\Pr\left[V \text{ accepts } w\right] = \max_{P} \Pr\left[V \leftrightarrow P \text{ accepts } w\right].$$

Then $\Pr\left[V \text{ accepts } w\right] \geq \frac{2}{3}$ if $w \in$ **L** and $\Pr\left[V \text{ accepts } w\right] \leq \frac{1}{3}$ otherwise.

A superclass of IP

Interestingly, we can use another well-known class to capture IP from above:

Theorem 26.8: IP $\subseteq$ PSpace.

Proof: Consider **L** $\in$ IP with polynomial verifier V. For any word w, let

$$\Pr[V \text{ accepts } w] = \max_P \Pr[V \leftrightarrow P \text{ accepts } w].$$

Then $\Pr[V \text{ accepts } w] \geq \frac{2}{3}$ if $w \in$ **L** and $\Pr[V \text{ accepts } w] \leq \frac{1}{3}$ otherwise.

Goal: Compute the value of $\Pr[V \text{ accepts } w]$ in PSpace.

A superclass of IP

Interestingly, we can use another well-known class to capture IP from above:

Theorem 26.8: IP $\subseteq$ PSpace.

Proof: Consider **L** $\in$ IP with polynomial verifier V. For any word w, let

$$\Pr[V \text{ accepts } w] = \max_P \Pr[V \leftrightarrow P \text{ accepts } w].$$

Then $\Pr[V \text{ accepts } w] \geq \frac{2}{3}$ if $w \in$ **L** and $\Pr[V \text{ accepts } w] \leq \frac{1}{3}$ otherwise.

Goal: Compute the value of $\Pr[V \text{ accepts } w]$ in PSpace.

Notation:

- Let M_j abbreviate a message sequence $m_1\#m_2\#\cdots\#m_j$
- $(V \leftrightarrow P)(w, r, M_j) = \text{accept}$ if $(V \leftrightarrow P)(w, r) = \text{accept}$ for a message sequence $m_1\#m_2\#\cdots\#m_k$ that extends M_j (in particular: M_j is possible with r, V and P)
- $\Pr\left[(V \leftrightarrow P) \text{ accepts } w \text{ starting from } M_j\right] = \Pr_{r\in\{0,1\}^{p(|w|)}}[(V \leftrightarrow P)(w, r, M_j) = \text{accept}]$
- $\Pr\left[V \text{ accepts } w \text{ starting from } M_j\right] = \max_P \Pr\left[(V \leftrightarrow P) \text{ accepts } w \text{ starting from } M_j\right]$

IP $\subseteq$ PSpace

Theorem 26.8: IP $\subseteq$ PSpace.

Proof (cont.): What we seek is $\Pr[V \text{ accepts } w] = \Pr[V \text{ accepts } w \text{ starting from } M_0]$, where M_0 is the empty message sequence.

IP $\subseteq$ PSpace

Theorem 26.8: IP $\subseteq$ PSpace.

Proof (cont.): What we seek is $\Pr[V \text{ accepts } w] = \Pr[V \text{ accepts } w \text{ starting from } M_0]$, where M_0 is the empty message sequence.

We define numbers $N[M_j]$ recursively, with the longest possible sequences as base case:

1. If M_j is cannot be produced by V for any r (and P), then $N[M_j] = 0$.
2. Else, if j is odd and $M_j = m_1\#\cdots\#m_j$, then
   - 2.1 If m_j = accept then $N[M_j] = 1$
   - 2.2 If m_j = reject then $N[M_j] = 0$
   - 2.3 If $m_j \notin$ {accept, reject} then $N[M_j] = \max_{m_{j+1}} N[M_j\#m_{j+1}]$
3. Else, if j is even, then $N[M_j] = \text{wt-avg}_{m_{j+1}} N[M_j\#m_{j+1}]$
   where $\text{wt-avg}_{m_{j+1}} N[M_j\#m_{j+1}] = \sum_{m_{j+1}} \Pr_{r\in\{0,1\}^{p(|w|)}}[V(w,r,M_j) = m_{j+1}] \cdot N[M_j\#m_{j+1}]$

In all cases, m_{j+1} ranges over (a superset of) the messages possible at this step (which can be assumed to be of polynomial length, and are therefore bounded).

IP $\subseteq$ PSpace

Theorem 26.8: IP $\subseteq$ PSpace.

Proof (cont.): What we seek is $\Pr[V \text{ accepts } w] = \Pr[V \text{ accepts } w \text{ starting from } M_0]$, where M_0 is the empty message sequence.

We define numbers $N[M_j]$ recursively, with the longest possible sequences as base case:

1. If M_j is cannot be produced by V for any r (and P), then $N[M_j] = 0$.
2. Else, if j is odd and $M_j = m_1\#\cdots\#m_j$, then
   - 2.1 If m_j = accept then $N[M_j] = 1$
   - 2.2 If m_j = reject then $N[M_j] = 0$
   - 2.3 If $m_j \notin$ {accept, reject} then $N[M_j] = \max_{m_{j+1}} N[M_j\#m_{j+1}]$
3. Else, if j is even, then $N[M_j] = \text{wt-avg}_{m_{j+1}} N[M_j\#m_{j+1}]$
   where $\text{wt-avg}_{m_{j+1}} N[M_j\#m_{j+1}] = \sum_{m_{j+1}} \Pr_{r\in\{0,1\}^{p(|w|)}}[V(w,r,M_j) = m_{j+1}] \cdot N[M_j\#m_{j+1}]$

In all cases, m_{j+1} ranges over (a superset of) the messages possible at this step (which can be assumed to be of polynomial length, and are therefore bounded).

Note 1: Case 2.3 corresponds to best possible answer of any Prover

Note 2: Case 3 corresponds to probability-weighted average for given Verifier

IP $\subseteq$ PSpace

Theorem 26.8: IP $\subseteq$ PSpace.

Proof (cont.): We need to show two claims:

- Claim 1: $N[M_j] = \Pr\left[V \text{ accepts } w \text{ starting from } M_j\right]$
- Claim 2: $N[M_j]$ can be computed in polynomial space

Together, this would show that $N[M_0] = \Pr[V \text{ accepts } w]$ can be computed in polynomial space.

IP $\subseteq$ PSpace

Theorem 26.8: IP $\subseteq$ PSpace.

Proof (cont.): We need to show two claims:

- Claim 1: $N[M_j] = \Pr\left[V \text{ accepts } w \text{ starting from } M_j\right]$
- Claim 2: $N[M_j]$ can be computed in polynomial space

Together, this would show that $N[M_0] = \Pr[V \text{ accepts } w]$ can be computed in polynomial space.

Claim 2 is not hard to see:

- The recursive computation of $N[M_j]$ is of polynomially bounded depth (longer message sequences are never consistent with a polynomial verifier V)
- Checking consistency with some $r \in \{0,1\}^{p(|w|)}$ can be done by iterating over these r
- Computing $\max_{m_{j+1}} N[M_j\#m_{j+1}]$ is similar by iterating over all m_{j+1}
- Computing $\text{wt-avg}_{m_{j+1}} N[M_j\#m_{j+1}]$ is also similar, using two iterations (over m_{j+1} and over all r)

IP $\subseteq$ PSpace

Theorem 26.8: IP $\subseteq$ PSpace.

Proof (cont.): Claim 1 can be shown by induction.

The base cases are when M_j is not consistent (impossible message sequence), or already ends in accept or reject. The claim is clear for these cases.

IP $\subseteq$ PSpace

Theorem 26.8: IP $\subseteq$ PSpace.

Proof (cont.): Claim 1 can be shown by induction.

The base cases are when M_j is not consistent (impossible message sequence), or already ends in accept or reject. The claim is clear for these cases.

For the induction step, assume the claim holds for all $N[M_{j+1}]$ (ind. hypothesis, IH)

- For the case $N[M_j] = \text{wt-avg}_{m_{j+1}} N[M_j\#m_{j+1}]$, we compute:

$$N[M_j] \overset{\text{def}}{=} \sum_{m_{j+1}} \Pr_{r\in\{0,1\}^{p(|w|)}}[V(w,r,M_j) = m_{j+1}] \cdot N[M_j\#m_{j+1}]$$

$$\overset{\text{IH}}{=} \sum_{m_{j+1}} \Pr_{r\in\{0,1\}^{p(|w|)}}[V(w,r,M_j) = m_{j+1}] \cdot \Pr\Big[V \text{ accepts } w \text{ starting from } M_j\#m_{j+1}\Big]$$

$$= \Pr\Big[V \text{ accepts } w \text{ starting from } M_j\Big] \quad \text{(def. of acceptance probability for steps of } V\text{)}$$

IP ⊆ PSpace

Theorem 26.8: IP ⊆ PSpace.

Proof (cont.): Claim 1 can be shown by induction.

The base cases are when M_j is not consistent (impossible message sequence), or already ends in accept or reject. The claim is clear for these cases.

For the induction step, assume the claim holds for all $N[M_{j+1}]$ (ind. hypothesis, IH)

- For the case $N[M_j] = \max_{m_{j+1}} N[M_j\#m_{j+1}]$, we compute:

$$\begin{aligned} N[M_j] &\overset{\text{IH}}{=} \max_{m_{j+1}} \Pr\left[V \text{ accepts } w \text{ starting from } M_j\#m_{j+1}\right] \\ &= \Pr\left[V \text{ accepts } w \text{ starting from } M_j\right] \end{aligned}$$

  The second equality follows since this probability can be achieved by a Prover that sends the message m_{j+1} that maximises $N[M + j]$, and no higher probability can be achieved by any other message.

This finishes the proof of the theorem. □

The power of the prover

Our definition of IP allows Prover to have unlimited computational power (possibly even uncomputable behaviour).

However, our proof of IP $\subseteq$ PSpace showed that the optimal Prover output for any given Verifier can be computed in polynomial space, so we get:

Corollary 26.9: The class IP remains the same if the Prover is required to compute its responses in polynomial space.

The power of IP

So far, we know that IP contains NP, BPP, but also **Graph Non-Isomorphism**, which is not known to be in either class.

The power of IP

So far, we know that IP contains NP, BPP, but also **Graph Non-Isomorphism**, which is not known to be in either class.

As we will see, IP can do much more. We start with the following problem:

#SAT

Input:	A propositional logic formula φ.
Problem:	The number of satisfying assignments of φ

Note:

- **#SAT** is not a decision problem. Let **#SAT**$_D = \{\langle\varphi, k\rangle \mid k$ is the solution of **#SAT** on $\varphi\}$ be the corresponding decision problem
- Computing **#SAT** solves propositional satisfiability as well as unsatisfiability.
- Indeed, it is complete for the powerful class #P

Solving **#SAT**$_D$ in IP

Theorem 26.10: #SAT$_D$ $\in$ IP

We consider a formula φ of size n and with m propositional variables $x_1, \ldots, x_m$.

For $1 \leq i \leq m$, let $f_i : \{0,1\}^i \to \mathbb{N}$ be the function that maps $\langle a_1, \ldots, a_m \rangle$ to the number of satisfying assignments of φ with $x_1 = a_1, \ldots, x_i = a_i$.

- Then $f_0()$ is the solution to **#SAT**
- We find $f_i(a_1, \ldots, a_i) = f_{i+1}(a_1, \ldots, a_i, 0) + f_{i+1}(a_1, \ldots, a_i, 1)$

#SAT$_D$ $\in$ IP: first attempt

Protocol: to check if $\langle \varphi, k \rangle \in$ **#SAT**$_D$

- P: send $f_0()$ to V
- V: check if $f_0() = k$ and reject if this fails
- For $i = 1, \ldots, m$:
  - P: send $f_i(a_1, \ldots, a_i)$ to V for all $\langle a_1, \ldots, a_i \rangle \in \{0, 1\}^i$
  - V: check, for all $\vec{a} \in \{0, 1\}^{i-1}$, if $f_{i-1}(\vec{a}) = f_i(\vec{a}, 0) + f_i(\vec{a}, 1)$, reject if not
- V: check if, for all $\langle a_1, \ldots, a_m \rangle \in \{0, 1\}^m$, $f_m(a_1, \ldots, a_m) = 1$ if and only if $\{x_1 \mapsto a_1, \ldots, x_m \mapsto a_m\}$ is a satisfying assignment for φ; accept iff

#SAT$_D$ $\in$ IP: first attempt

Protocol: to check if $\langle \varphi, k\rangle \in$ **#SAT**$_D$

- P: send $f_0()$ to V
- V: check if $f_0() = k$ and reject if this fails
- For $i = 1, \ldots, m$:
  - P: send $f_i(a_1, \ldots, a_i)$ to V for all $\langle a_1, \ldots, a_i\rangle \in \{0, 1\}^i$
  - V: check, for all $\vec{a} \in \{0, 1\}^{i-1}$, if $f_{i-1}(\vec{a}) = f_i(\vec{a}, 0) + f_i(\vec{a}, 1)$, reject if not
- V: check if, for all $\langle a_1, \ldots, a_m\rangle \in \{0, 1\}^m$, $f_m(a_1, \ldots, a_m) = 1$ if and only if $\{x_1 \mapsto a_1, \ldots, x_m \mapsto a_m\}$ is a satisfying assignment for φ; accept iff

This protocol does not show **#SAT**$_D$ $\in$ IP:

- it requires exponential time to perform exponentially many checks.

#SAT$_D$ ∈ IP: first attempt

Protocol: to check if $\langle\varphi, k\rangle \in$ **#SAT**$_D$

- P: send $f_0()$ to V
- V: check if $f_0() = k$ and reject if this fails
- For $i = 1, \ldots, m$:
  - P: send $f_i(a_1, \ldots, a_i)$ to V for all $\langle a_1, \ldots, a_i\rangle \in \{0, 1\}^i$
  - V: check, for all $\vec{a} \in \{0, 1\}^{i-1}$, if $f_{i-1}(\vec{a}) = f_i(\vec{a}, 0) + f_i(\vec{a}, 1)$, reject if not
- V: check if, for all $\langle a_1, \ldots, a_m\rangle \in \{0, 1\}^m$, $f_m(a_1, \ldots, a_m) = 1$ if and only if $\{x_1 \mapsto a_1, \ldots, x_m \mapsto a_m\}$ is a satisfying assignment for φ; accept iff

This protocol does not show **#SAT**$_D$ ∈ IP:

- it requires exponential time to perform exponentially many checks.

However, the protocol is otherwise correct:

- if k is the correct result, a truthful Prover can convince the Verifier
- if k is not correct, not even a mischievous Prover can convince the verifier (exercise: why?)

Arithmetisation

To reduce the number of messages and checks, we use arithmetisation.

φ is transformed into an arithmetic expression Φ by replacing subexpressions:

- $\alpha \wedge \beta$ becomes $\alpha\beta$
- $\neg\alpha$ becomes $(1 - \alpha)$
- $\alpha \vee \beta$ becomes $\alpha * \beta = 1 - (1 - \alpha)(1 - \beta)$

Arithmetisation

To reduce the number of messages and checks, we use arithmetisation.

φ is transformed into an arithmetic expression Φ by replacing subexpressions:

- $\alpha \wedge \beta$ becomes $\alpha\beta$
- $\neg\alpha$ becomes $(1 - \alpha)$
- $\alpha \vee \beta$ becomes $\alpha * \beta = 1 - (1 - \alpha)(1 - \beta)$

Some observations:

- Φ is a multivariate polynomial function over variables $x_1, \ldots, x_n$
- The degree of Φ is bounded by the size n of φ
- The value of Φ for inputs $x_i \in \{0, 1\}$ is also in $\{0, 1\}$, and corresponds to the valuation of φ on the corresponding truth values
- We can evaluate Φ oven an arbitrary field

Example 26.11: For a prime number p, the algebra of natural numbers $\{0, 1, \ldots, p - 1\}$ and where $+$ and $\cdot$ are addition and multiplication modulo p is a finite field. This field is denoted GF(p).

#SAT$_D$ ∈ IP

Theorem 26.10: #SAT$_D$ ∈ IP

Proof: By our prior observation, k is a solution to **#SAT** exactly if

$$k = \sum_{a_1 \in \{0,1\}} \ldots \sum_{a_m \in \{0,1\}} \Phi(a_1, \ldots, a_m) \tag{1}$$

The Prover tries to convince the Verifier of this.
We are looking for a protocol to verify this property of a polynomial Φ.

#SAT$_D$ ∈ IP

Theorem 26.10: #SAT$_D$ ∈ IP

Proof: By our prior observation, k is a solution to **#SAT** exactly if

$$k = \sum_{a_1 \in \{0,1\}} \cdots \sum_{a_m \in \{0,1\}} \Phi(a_1, \ldots, a_m) \tag{1}$$

The Prover tries to convince the Verifier of this.
We are looking for a protocol to verify this property of a polynomial Φ.

Initialisation:

The Prover sends a prime number p with $2^n < p \leq 2^{2n}$ (n: size of φ). All calculations will be performed in GF(p).
Note: The right side of (1) is at most $2^m \leq 2^n$, so the value is unaffected by this restriction.

The Verifier checks that p is really prime (primality is known to be in P)

#SAT$_D$ ∈ IP

Theorem 26.10: #SAT$_D$ ∈ IP

Proof (cont.): Given a multi-variate polynomial $g(x_1, \ldots, x_\ell)$, let $h(x_1)$ denote the (univariate) polynomial $\sum_{a_2 \in \{0,1\}} \cdots \sum_{a_m \in \{0,1\}} g(x_1, a_2, \ldots, a_m)$.

Protocol: to check $K = \sum_{a_1 \in \{0,1\}} \cdots \sum_{a_m \in \{0,1\}} g(a_1, \ldots, a_m) \mod p$ for multi-variate polynomial g that has a polynomial-size representation and polynomial degree

- V: if $m = 1$, verify $g(0) + g(1) = K$ and reject or accept accordingly;
  if $m \geq 2$, ask P to send a polynomial-size representation of $h(x_1)$
- P: send a polynomial $\tilde{h}(x_1)$ (if P is truthful, it sends $\tilde{h} = h$)
- V: check if $\tilde{h}$ is polynomially sized and of degree $\leq n$;
  check if $K = \tilde{h}(0) + \tilde{h}(1)$; reject if any of these fail;
  pick a random $b \in$ GF(p) and send b to P
- Recursively use the same protocol to verify
  $\tilde{h}(b) = \sum_{a_2 \in \{0,1\}} \cdots \sum_{a_m \in \{0,1\}} g(b, a_2, \ldots, a_m) \mod p$

#SAT$_D$ ∈ IP

Theorem 26.10: #SAT$_D$ ∈ IP

Proof (cont.): It is not hard to verify that the protocol can be implemented by a polynomial verifier:

- All polynomials are given by polynomial representations and have polynomial degree
- They can therefore be evaluated in polynomial time (using binary encoding of numbers)
- The random number $b \leq p \leq 2^{2n}$ consists of $2n$ random bits
- There are $\leq m$ recursive applications of the protocol

#SAT$_D$ $\in$ IP

Theorem 26.10: #SAT$_D$ $\in$ IP

Proof (cont.): If the claim is true, a truthful Prover can ensure that V accepts.

#SAT$_D$ ∈ IP

Theorem 26.10: #SAT$_D$ ∈ IP

Proof (cont.): If the claim is true, a truthful Prover can ensure that V accepts.

If the claim is false, the probability that V accepts is very small:

- For $m = 1$, the probability is 0 (V will just check directly)
- For $m > 1$, P must send some $\tilde{h} \neq h$ in order to pass the check $K = \tilde{h}(0) + \tilde{h}(1)$
  If V selects b such that $\tilde{h}(b) = \sum_{a_2 \in \{0,1\}} \cdots \sum_{a_m \in \{0,1\}} g(b, a_2, \ldots, a_m) \mod p$, then P con continue to play truthfully and V will eventually accept
- Overall, there are $m - 1$ opportunities for P to be lucky in this sense.
- But if $\tilde{h} \neq h$, then the chance of a random $b \in \{0, \ldots, p\} \supseteq \{0, \ldots, 2^m\}$ to be such that $\tilde{h}(b) - h(b) = 0$ is $\leq d/2^n$, where d is the degree of $\tilde{h} - h$ (Schwartz-Zippel Lemma).
- The degree of h and any reasonable $\tilde{h}$ is bounded by the size n of φ (linear), while 2^n is exponential, hence the success rate is small for sufficiently large φ.
- The overall chance of P tricking V to accept a wrong claim is $\leq 1 - (1 - n/2^n)^{m-1}$, which is $\leq 1/n$ for $n \geq 10$. □

Main result

The main insight about IP is as follows:

Theorem 26.12: IP = PSpace

Proof: We have already shown IP $\subseteq$ PSpace. For the converse, we adopt our proof of **#SAT**$_D$ $\in$ IP to show that **TrueQBF** $\in$ IP. This suffices (why?).

Main result

The main insight about IP is as follows:

Theorem 26.12: IP = PSpace

Proof: We have already shown IP $\subseteq$ PSpace. For the converse, we adopt our proof of **#SAT**$_D$ $\in$ IP to show that **TrueQBF** $\in$ IP. This suffices (why?).

Consider a QBF of the form $\psi = \forall x_1.\exists x_2.\forall x_3.\cdots\exists x_m.\varphi[x_1,\ldots,x_n]$ (this is w.l.o.g. – why?).

Using the arithmetisation Ψ of φ, we find $\psi \in$ **TrueQBF** iff

$$\overset{*}{\sum_{a_1\in\{0,1\}}}\ \prod_{a_2\in\{0,1\}}\ \overset{*}{\sum_{a_3\in\{0,1\}}}\cdots\overset{*}{\sum_{a_m\in\{0,1\}}}\Phi(a_1,\ldots,a_m)=1 \tag{2}$$

where $\overset{*}{\sum}_{a\in\{0,1\}}P(a) = P(0) * P(1) = 1-(1-P(0))(1-P(1))$.

We would like to verify (2) using similar ideas as for **#SAT**$_D$ $\in$ IP.

Showing IP = PSpace

We would like to verify (2) using similar ideas as for **#SAT**$_D \in$ IP.

Showing IP = PSpace

We would like to verify (2) using similar ideas as for **#SAT**$_D \in$ IP.

Problem: The degree of polynomials such as
$h(x_1) = \prod_{a_2\in\{0,1\}} \sum^*_{a_3\in\{0,1\}} \cdots \sum^*_{a_m\in\{0,1\}} \Phi(x_1, a_2, \ldots, a_m)$ can be as large as 2^n
$\rightsquigarrow$ no polynomial-size description, no polytime evaluation

Showing IP = PSpace

We would like to verify (2) using similar ideas as for **#SAT**$_D \in$ IP.

Problem: The degree of polynomials such as
$h(x_1) = \prod_{a_2\in\{0,1\}} \sum^*_{a_3\in\{0,1\}} \cdots \sum^*_{a_m\in\{0,1\}} \Phi(x_1, a_2, \ldots, a_m)$ can be as large as 2^n
$\leadsto$ no polynomial-size description, no polytime evaluation

Solution: Reduce degrees of all relevant polynomials in a way that preserves truth values

- Idea: if $x \in \{0,1\}$, then $x^d = x$ and $P(x) = xP(1) + (1-x)P(0)$
- We define an operator R with $Rx.P(x) = xP(1) + (1-x)P(0)$
- Then the degree of x in $Rx.P(x)$ is always 1

Showing IP = PSpace

We would like to verify (2) using similar ideas as for **#SAT**$_D$ $\in$ IP.

Problem: The degree of polynomials such as
$h(x_1) = \prod_{a_2 \in \{0,1\}} \sum^*_{a_3 \in \{0,1\}} \cdots \sum^*_{a_m \in \{0,1\}} \Phi(x_1, a_2, \ldots, a_m)$ can be as large as 2^n
$\rightsquigarrow$ no polynomial-size description, no polytime evaluation

Solution: Reduce degrees of all relevant polynomials in a way that preserves truth values

- Idea: if $x \in \{0, 1\}$, then $x^d = x$ and $P(x) = xP(1) + (1 - x)P(0)$
- We define an operator R with $Rx.P(x) = xP(1) + (1 - x)P(0)$
- Then the degree of x in $Rx.P(x)$ is always 1

We redefine the polynomial we want evaluate as follows:

$$\exists x_1.Rx_1 \forall x_2.Rx_1.Rx_2.\exists x_3.Rx_1.Rx_2.Rx_3. \cdots \exists x_m.Rx_1. \cdots Rx_m.\Phi(x_1, \ldots, x_m)$$

where $\exists x.P(x) = P(0) * P(1)$ and $\forall x.P(x) = P(0) \cdot P(1)$.
Note: This expression is of quadratic size compared to ψ.

Showing IP = PSpace

We write $\exists x_1.Rx_1\forall x_2.Rx_1.Rx_2.\exists x_3.Rx_1.Rx_2.Rx_3.\cdots\exists x_m.Rx_1.\cdots Rx_m.\Phi(x_1,\ldots,x_m)$ as $O_1y_1.O_2y_2.\cdots.O_ky_k.\Phi(x_1,\ldots,x_m)$, where $O_i\in\{\exists,\forall,R\}$ and $y_i\in\{x_1,\ldots,x_m\}$.

Verifier picks a prime $p > n^4$ (for n the size of ψ); we calculate in GF(p).

Showing IP = PSpace

We write $\exists x_1.\mathsf{R}x_1\forall x_2.\mathsf{R}x_1.\mathsf{R}x_2.\exists x_3.\mathsf{R}x_1.\mathsf{R}x_2.\mathsf{R}x_3.\cdots\exists x_m.\mathsf{R}x_1.\cdots\mathsf{R}x_m.\Phi(x_1,\ldots,x_m)$ as $O_1y_1.O_2y_2.\cdots.O_ky_k.\Phi(x_1,\ldots,x_m)$, where $O_i \in \{\exists,\forall,\mathsf{R}\}$ and $y_i \in \{x_1,\ldots,x_m\}$.

Verifier picks a prime $p > n^4$ (for n the size of ψ); we calculate in GF(p).

Protocol: to check $K = O_1y_1.O_2y_2.\cdots.O_ky_k.g(b_1,\ldots,b_\ell) \mod p$ where $O_1y_1.O_2y_2.\cdots.O_ky_k.g$ is a polynomial in ℓ variables that has a polynomial-size representation and polynomial degree

- V: if $k=0$, verify $g(b_1,\ldots,b_\ell)=K$ and reject or accept accordingly;
  else, ask P for a representation of $O_2y_2.\cdots.O_ky_k.g(b_1,\ldots,b_\ell)[y_1 \mapsto \text{undef}]$
- P: send a polynomial $\tilde{h}(y_1)$
- V: check if $\tilde{h}$ is polynomially sized and of degree $\leq m$;
  check if $K = O_1y_1.\tilde{h}(y_1)$; reject if any of these fail;
  pick a random $b \in$ GF(p) and send b to P
- Recursively use the same protocol to verify
  $\tilde{h}(b) = O_2y_2.\cdots.O_ky_k.g(b_1,\ldots,b_\ell)[y_1 \mapsto b] \mod p$

Explanations

The following notes may help to understand the protocol.

- The function $O_1 y_1. \cdots .O_k y_k.g$ is a function on variables $x_1, \ldots, x_\ell$
  - Variables x_i $(i > \ell)$ are bound by $\exists$ or $\forall$, hence eliminated
  - Variables x_i $(i \leq \ell)$ may still occur in R operators, but they do not remove them
- $O_2 y_2. \cdots .O_k y_k.g$ is a function on variables $x_1, \ldots, x_\ell, x_{\ell+1}$ if $O_1 \in \{\exists, \forall\}$
- $O_2 y_2. \cdots .O_k y_k.g$ is a function on variables $x_1, \ldots, x_\ell$ if $O_1 = R$
- $O_2 y_2. \cdots .O_k y_k.g(b_1, \ldots, b_\ell)[y_1 \mapsto \text{undef}]$ denotes the function over y_1 obtained by ignoring the binding b_i for $y_1 = x_i$ (only relevant if $O_1 = R$)
- $O_2 y_2. \cdots .O_k y_k.g(b_1, \ldots, b_\ell)[y_1 \mapsto b]$ denotes the function over y_1 obtained by redefining the binding b_i for $y_1 = x_i$ to be b (only relevant if $O_1 = R$)
- The check $K = O_1 y_1.\tilde{h}(y_1)$ is evaluated as required for O_1

Finishing the proof

Theorem 26.12: IP = PSpace

Proof: Summary of approach:

- The problem is arithmetised and extended with degree-reduction operators
- A prime $p > n^4$ is chosen to define a filed GF(p) for calculations
- A protocol is followed to verify the arithmetisation yields 1 =

As in the case of **#SAT**$_D$, the Prover's chances of fooling the Verifier as small:

- Wrong claims require to send wrong polynomials $\tilde{h}(y_1)$
- It is unlikely that V picks a random value b on which $\tilde{h}(p)$ agrees with the correct function's value ($p > n^4$ suffices here since the degree of the functions are small)

This finishes the proof. □

Summary and Outlook

Interactive proofs enable probabilistic machines to solve problems beyond NP

Graph Non-Isomorphism has an interesting interactive zero-knowledge proof protocol

IP = PSpace

What's next?

- Summary & consultation
- Examinations