1 Interactive Proof Systems

We define an interactive proof system (IPS) to be a nondeterministic Turing machine with read-only input tape whose non-halting states are partitioned into two types: existential and coin-flipping. For simplicity, we require that the transition relation δ is such that there are exactly two possible next steps from each coin-flipping state.

Let M be an IPS and let C be a configuration of M. C is either an existential, coin-flipping, accepting, or rejecting configuration depending on its state. We define the probability of reaching an accepting configuration from C, which we denote by $\text{Prob}_a[C]$, recursively as follows:

- If C is a rejecting or accepting configuration then $\text{Prob}_a[C] = 0$ or 1, respectively.
- If C is an existential configuration then $\text{Prob}_a[C] = \max_{C \to C'} \text{Prob}_a[C']$.
- If C is a coin-flipping configuration then $\text{Prob}_a[C] = \frac{1}{2}(\text{Prob}_a[C'] + \text{Prob}_a[C''])$, where C' and C'' are the two configurations reachable from C.

Let $\text{Prob}[M \text{ accepts } x]$ be $\text{Prob}_a[C_0]$, where C_0 is the initial configuration of M on x. We say that the IPS M accepts language L with bounded error if the following holds.

- For all $x \in L$, $\text{Prob}[M \text{ accepts } x] > 2/3$,
- For all $x \notin L$, $\text{Prob}[M \text{ accepts } x] < 1/3$.

Let IP denote the set of languages that have bounded error interactive proof systems M that are polynomial time bounded. The following lemma is not hard to prove:

Lemma 1 *IP is closed under $\leq_m^p$. That is, if $L \in$ IP and $L' \leq_m^p L$, then $L' \in$ IP.*

Interactive proof systems model protocols in which an (existential) prover aims to convince a probabilistic verifier that an input should be accepted. Just as BPP can be considered the class of languages with efficient algorithms, so can IP be viewed as the class of languages that have efficient proofs of membership. Might IP contain more languages than NP? Building on the work of Lund et al., Shamir showed that in fact IP = PSPACE. Showing that IP $\subseteq$ PSPACE is not too difficult - can you see how to do it? We will prove the other direction. In the next sections, we provide some useful background needed to prove this result.

2 More on Quantified Boolean Formulas

We will need to work with quantified Boolean formulas in which quantifiers are not all required to precede a quantifier-free boolean formula.

Example. Two quantified Boolean formulas (the first is slightly modified from an example in Papadimitriou's textbook) are:

$$\phi = \forall x \exists y[(x \vee y) \wedge \forall z[(x \wedge z) \vee (y \wedge \bar{z}) \vee \exists w(z \vee (y \wedge \bar{w}))]], \text{ and} \tag{1}$$

$$\tau = \forall x[(x \vee y) \wedge \forall z[((x \wedge z) \vee (y \wedge \bar{z})) \vee \forall w(z \vee (x \wedge \bar{w}))]]. \tag{2}$$

Each variable is either *free* or *bound* to a quantifier; in ϕ all variables are bound, whereas in τ, y is free. We can associate with every quantified Boolean formula ϕ a set F_ϕ of free variables and a set B_ϕ of bound variables. For simplicity, we will require that each quantifier pertains to a distinct variable.

Formally, we inductively define a quantified Boolean formula (qbf) ϕ over variable set X, and its associated sets F_ϕ and B_ϕ of free and bound variables respectively, as follows. For every $x \in X$, x is a quantified Boolean formula with $F_x = \{x\}$ and $B_x = \emptyset$. Also, if ϕ and ϕ' are quantified Boolean formulas such that $F_\phi \cap B_{\phi'} = \emptyset$ and $F_{\phi'} \cap B_\phi = \emptyset$ then

- $\bar{\phi}$ is a quantified Boolean formula with free variable set F_ϕ and bound variable set B_ϕ;
- $(\phi \vee \phi')$ and $(\phi \wedge \phi')$ are quantified Boolean formulas with free variable set $F_\phi \cup F_{\phi'}$ and bound variable set $B_\phi \cup B_{\phi'}$, and
- if $x \notin B_\phi$ then $(\exists x \phi)$ and $(\forall x \phi)$ are quantified Boolean formulas with free variable set $F_\phi - \{x\}$ and bound literal set $B_\phi \cup \{x\}$.

Sometimes when there is no ambiguity, parentheses are omitted for clarity as in the examples (1) and (2) above. If all variables of a quantified Boolean formula ϕ are bound, then ϕ's truth value can be defined inductively in a natural way. We say that ϕ is *valid* if its value is true. Also we can define in a natural way what is the quantifier to which a particular variable is bound. If a variable x occurs in qbf $(Q_y \rho)$, where $Q \in \{\exists, \forall\}$, then we say that x *is in the scope* of Q.

Let ϕ be a quantified Boolean formula. We say that ϕ is *simple* if any variable x of ϕ is in the scope of at most one $\forall$ quantifier that is within the scope of the quantifier of ϕ to which x is bound. We say that ϕ is in *prenex normal form* if ϕ is of the form $(Q_1 x_1 (Q_2 x_2 \ldots (Q_n x_n \phi) \ldots))$, where each Q_i is either $\exists$ or $\forall$, and ϕ does not have quantifiers. We say that ϕ has *negations only over variables* if any expression of the form $\bar{\rho}$ of ϕ is such that ρ is a variable. In what follows, we assume without loss of generality that qbf's have negations only over variables.

Example. The quantified formula ϕ given in (1) is simple. However, the formula τ given in (2) is not simple. (Why?)

Lemma 2 *There is a deterministic polynomial time bounded algorithm that, given as input a quantified Boolean formula ϕ with no free variables, outputs a simple quantified Boolean formula ϕ' such that ϕ is valid if and only if ϕ' is.*

Proof (Sketch) On input ϕ, the algorithm repeats the following until a simple formula is obtained: Let x be a variable that is in the scope of at least two $\forall$ quantifiers that are within the scope of the quantifier to which x is bound. Let Q_z be the leftmost $\forall$ quantifier that contains x in its scope, such that $z \neq x$. Let $(Q_z \rho)$ be a qbf nested in ϕ (i.e. $(Q_z \rho)$ is a quantified Boolean formula). Replace $(Q_z \rho)$ by $(Q_z(\exists x'((x \wedge x') \vee (\bar{x} \wedge \bar{x}')) \wedge \rho')$, where ρ' is obtained from ρ by replacing all occurrences of x by x'. $\square$
Exercise: show that the above algorithm is correct and runs in polynomial time. Can you see how to implement the algorithm in log space?

Example. The formula $\exists y \tau$, where τ is given in (2) has no free variables, but is not simple. Applying the algorithm of Lemma 2, we get

$$\exists y \forall x[(x \vee y) \wedge \forall z((\exists x'((x \wedge x') \vee (\bar{x} \wedge \bar{x}'))) \wedge [((x' \wedge z) \vee (y \wedge \bar{z})) \vee \forall w(z \vee (x' \wedge \bar{w}))])].$$

3 Arithmetizing Quantified Boolean Formulas

We arithmetize a quantified Boolean formula with negations only over variables by replacing each $\vee$ by $+$, each $\wedge$ by $\cdot$, each $\bar{x}$ by $(1-x)$, each $\exists x$ by $\sum_{x\in\{0,1\}}$, and each $\forall$ by $\prod_{x\in\{0,1\}}$.

Example. The arithmetization of ϕ given in Equation (1) is

$$A_\phi = \prod_{x=0}^{1}\sum_{y=0}^{1}[(x+y)\cdot\prod_{z=0}^{1}[(x\cdot z + y\cdot(1-z)) + \sum_{w=0}^{1}(z+y\cdot(1-w))]]. \tag{3}$$

If ϕ has no free variables, then we can associate a value with A_ϕ in a natural way and denote it by $v(A_\phi)$. For A_ϕ given in (3) above, $v(A_\phi) = 96$.

Exercise. Calculate $v(A_\phi)$ for the following ϕ:

$$\exists x\forall y((x\wedge y)\vee(\bar{x}\wedge\bar{y}))$$

$$\forall x_1\forall x_2\ldots\forall x_n\exists y(y\vee\bar{y}).$$

The following lemma can be proved in a straightforward way by induction over the number of $\vee, \wedge, \exists$, and $\forall$ symbols of a formula.

Lemma 3 *For any quantified Boolean formula ϕ with no free variables and negations only over variables, ϕ is valid if and only if $v(A_\phi) > 0$. Also, $v(A_\phi) \le 2^{2^{|A_\phi|}}$.*

If we write A_ϕ given in (3) above as $\prod_x \alpha$, then α is a (univariate) polynomial in x. Let $|\alpha|$ be the number of symbols in the string α. Continuing with our example, $|\alpha|$ is 40 (it would be a little longer if all parentheses were in place) and $\alpha = \alpha(x) = 2x^2 + 8x + 6$. We denote the degree of a univariate polynomial p by $\deg(p)$. In general, the degree of α could be exponential. (Can you see why?) However, when α is simple, this is not the case, as the next lemma shows.

Lemma 4 *Let $(Q_x\phi)$ be a simple quantified Boolean formula with no free variables and let $(O_x\alpha)$ be its arithmetization. Then, $deg(\alpha(x)) \le 2|\alpha|$.*

Proof Since $(Q_x\phi)$ is simple, any occurrence of x in α is in the scope of at most one $\prod$. For each subexpression $(\prod_y \beta)$ of α such that x occurs in β, replace the expression $(\prod_y \beta)$ by $(\beta(0)\cdot\beta(1))$, where for $i \in \{0,1\}$, $\beta(i)$ is obtained from β by substituting i for y. Let α' be the resulting expression obtained from α. Then $|\alpha'| \le 2|\alpha|$ and no occurrence of x in α' is in the scope of a $\prod$.

We claim that $\deg(\alpha') \le |\alpha'|$. The proof of this claim is by induction on $|\alpha'|$. The base cases are when α' is a constant, in which case $\deg(\alpha') = 0 \le |\alpha'|$, or $\alpha' = x$ or $\alpha' = (1-x)$, in which case $\deg(\alpha') = 1 \le |\alpha'|$. If $\alpha' = (\beta + \beta')$ then, applying the induction hypothesis to β and β', we have that

$$\deg(\alpha') = \max[\deg(\beta), \deg(\beta')] \le |\beta| + |\beta'| < |\alpha'|.$$

If $\alpha' = \beta\cdot\beta'$ then $\deg(\alpha') = \deg(\beta) + \deg(\beta') \le |\beta| + |\beta'| < |\alpha'|$. Finally, if $\alpha' = (\prod_y \beta)$ then, since x can not occur in β, $\deg(\alpha') = \deg(\beta) = 0 < |\beta| < |\alpha'|$.

Therefore, we have that $\deg(\alpha) = \deg(\alpha') \le |\alpha'| \le 2|\alpha|$. □

Lemma 5 *Let A be the arithmetization of a quantified Boolean formula with no free variables such that $v(A) \neq 0$. Then for sufficiently large $|A|$, $v(A) > 0$ if and only if there is a prime p between $2^{|A|}$ and $2^{2|A|}$ such that $v(A) \neq 0 \mod p$.*

Proof To prove this, we will use two important results from number theory. (Both can be stated more generally than stated here.)

Chinese Remainder Theorem: Let m be the product of distinct primes $p_1, p_2, \ldots, p_k$. Then for any integers $r_1, r_2, \ldots, r_k$, there is a unique r in the range $0 \leq r < m$ such that for all i, $r = r_i \mod p_i$.

Prime Number Theorem: For any sufficiently large positive real number x, the number of primes that are $\leq x$ is at least $x/\ln x$.

From the Prime Number Theorem, the number of primes $\leq 2^{2|A|}$ is at least $2^{|A|+1}$ if $|A|$ is large enough. Also, the number of primes $\leq 2^{|A|}$ is trivially at most $2^{|A|}$. Subtracting the second bound from the first gives us that the number of primes between $2^{|A|}$ and $2^{2|A|}$ is at least $2^{|A|}$, for sufficiently large $|A|$.

Hence, the product of these primes, say m, is greater than $2^{2^{|A|}}$. Using Lemma 3, we have that $v(A) \leq 2^{2^{|A|}} < m$. Therefore by the Chinese Remainder Theorem, it follows that if $v(A) = 0 \mod p_i$ for every prime p_i in the range $2^{|A|} < p_i < 2^{2|A|}$, then $v(A) = 0$. Since $v(A) \neq 0$, we conclude that for some prime p between $2^{|A|}$ and $2^{2|A|}$, $v(A) \neq 0 \mod p$. □

4 PSPACE ⊆ IP

Our goal is to show that PSPACE ⊆ IP, where IP is the class of languages L that have a polynomial-time interactive proof system. That is, a prover can convince a probabilistic verifier that an instance of L is a yes instance, in polynomial time and with bounded error. Let TQBF be the set of valid quantified Boolean formulas in prenex normal form. Since TQBF is PSPACE-complete, from Lemma 1 of the last lecture, it is sufficient to show that TQBF is in IP. By Lemma 2 of the last lecture, it is sufficient to show that the language of all valid simple quantified Boolean formulas (not necessarily in prenex form) is in IP. We describe an IPS for this language.

4.1 The Protocol

Simple-TQBF-IPS(ϕ)
// ϕ is simple, and has m symbols of type $\exists$ or $\forall$

- Guess p and a_0.
  Check that p is prime, that $2^{|A_\phi|} < p < 2^{2|A_\phi|}$ and that $0 < a_0 \leq p - 1$; if not, reject.
  // The goal is to show that $v(A_\phi) = a_0 \mod p$.
  Let $A_0 = A_\phi$.

- For i from 1 to m do

  1. Let A_{i-1} be of the form $c_i + c_i' O_u A_i(u)$, where O_u is the first $\sum$ or $\prod$ symbol of A_{i-1}.
     // Here, $A_i(u)$ refers to the string of symbols following the O_u operator.
     // Later in the analysis, we will also use $A_i(u)$ to refer to the polynomial in u

2. Guess the coefficients of a polynomial $\alpha_i(u)$ of degree at most $2|A_\phi|$.
   // The claim is that the polynomials $A_i(u)$ and $\alpha_i(u)$ are the same.
3. Check that $c_i + c_i' O_u \alpha_i(u) = a_{i-1} \mod p$; if not, reject.
4. Choose $r_i \in [0 \ldots p-1]$ randomly and uniformly.
5. Let $a_i = \alpha_i(r_i) \mod p$.
6. Let A_i be the expression $A_i(r_i)$.

- Check that $v(A_m) = a_m$; if so, accept, and otherwise reject.

4.2 Example

The following example is from Papadimitriou's text. Let ϕ be as in Equation (1), and let A_0 be the arithmetization A_ϕ, given in Equation (3).

Suppose that the prime p guessed in the first step is 13. (Let's ignore the fact that 13 is less than $2^{|A_\phi|}$.) Also, let $a_0 = 5$ ($= 96 \mod 13 = v(A_0) \mod 13$). The rounds of the protocol proceed as follows, for particular "correct" guesses and random choices of the r_i:

Round 1.

1. $A_0 = \prod_{x=0}^{1} A_1(x)$, where
$$A_1(x) = \sum_{y=0}^{1}[(x+y) \cdot \prod_{z=0}^{1}[x \cdot z + y \cdot (1-z) + \sum_{w=0}^{1}(z + y \cdot (1-w))]].$$
2. $\alpha_1(x) = 2x^2 + 8x + 6$.
3. $\alpha_1(0)\alpha_1(1) = 6 \cdot 16 = 5 \mod 13 = a_0 \mod 13$.
4. $r_1 = 9$.
5. $a_1 = \alpha_1(r_1) \mod 13 = 2 \cdot 9^2 + 8 \cdot 9 + 6 \mod 13 = 6 \mod 13$.
6. $A_1 = \sum_{y=0}^{1}[(9+y) \cdot \prod_{z=0}^{1}[9 \cdot z + y \cdot (1-z) + \sum_{w=0}^{1}(z + y \cdot (1-w))]]$.

Round 2.

1. $A_1 = \sum_{y=0}^{1} A_2(y)$, where
$$A_2(y) = (9+y) \cdot \prod_{z=0}^{1}[9 \cdot z + y \cdot (1-z) + \sum_{w=0}^{1}(z + y \cdot (1-w))].$$
2. $\alpha_2(y) = 2y^3 + y^2 + 3y$.
3. $\alpha_2(0) + \alpha_2(1) = 6 \mod 13 = a_1 \mod 13$.
4. $r_2 = 3$.
5. $a_2 = \alpha_2(3) \mod 13 = (2 \cdot 3^3 + 3^2 + 3 \cdot 3) \mod 13 = 7 \mod 13$.
6. $A_2 = (9+3) \cdot \prod_{z=0}^{1}[9 \cdot z + 3 \cdot (1-z) + \sum_{w=0}^{1}(z + 3 \cdot (1-w))]$.

Round 3.

1. $A_2 = (9+3)\prod_{z=0}^{1} A_3(z)$, where $A_3(z) = 9 \cdot z + 3 \cdot (1-z) + \sum_{w=0}^{1}(z + 3 \cdot (1-w))$.
2. $\alpha_3(z) = 8z + 6$.
3. $12\alpha_3(0)\alpha_3(1) = 12 \cdot 6 \mod 13 = 7 \mod 13 = a_2 \mod 13$.
4. $r_3 = 7$.
5. $a_3 = \alpha_3(r_3) \mod 13 = 8 \cdot 7 + 6 \mod 13 = 10 \mod 13$.
6. $A_3 = 9 \cdot 7 + 3 \cdot (1-7)) + \sum_{w=0}^{1}(7 + 3 \cdot (1-w)) = 45 + \sum_{w=0}^{1}(7 + 3 \cdot (1-w)) = 6 + \sum_{w=0}^{1}(7 + 3 \cdot (1-w)) \mod 13$.

Round 4.

1. $A_3 = 6 + \sum_{w=0}^{1} A_4(w)$, where $A_4(w) = 7 + 3 \cdot (1-w)$.
2. $\alpha_4(w) = 10 - 3w$.
3. $6 + \alpha_4(0) + \alpha_4(1) = 6 + 10 + 7 \mod 13 = 10 \mod 13 = a_3$.
4. $r_4 = 2$.
5. $a_4 = \alpha_4(r_4) \mod 13 = 10 - 6 \mod 13 = 4 \mod 13$.
6. $A_4 = 7 + 3 \cdot (1-2)) = 4 \mod 13$.

Upon exiting the repeat loop, $A_4 = 4 = a_4$, and so the input is accepted.

4.3 Proof of Correctness

In proving correctness of the protocol, the more interesting case is when the input ϕ is not valid. The following lemma will be useful for this case.

For a given input ϕ, let $S = S(\phi)$ be a *strategy* of the IPS on ϕ, i.e., sequence of nondeterministic transition choices that determine the polynomials $\alpha_i(u)$ at step 2, plus the initial choices of p and a_0. For an execution of the protocol with this strategy, let $E_i = E_i(\phi, S)$ be the event that $a_i \neq v(A_i)$ either the verifier rejects before the end of round i is reached, or $a_i \neq v(A_i)$. Intuitively, event E_i corresponds to the cases where either the verified has determined before the end of step i that the prover is trying to cheat, and so has rejected, or that the prover has to continue to prove something that's not true, namely that $a_i = v(A_i)$.

Lemma 6 *Let ϕ be a simple quantified Boolean formula with $A_\phi = 0$. Let S be any strategy of the IPS on ϕ. Then for all $i, 1 \leq i \leq m$, Prob$[E_i] \geq (1 - \frac{2n}{2^n})^i$, where $n = |A_\phi|$ and the probability is taken over the random choices of the r_i during the run of the protocol.*

Proof The proof is by induction on i. The lemma is clearly true when $i = 0$, since by hypothesis, either $v(A_\phi) \neq a_0$ or $a_0 = 0$, and in the latter case the protocol does not reach round 1. Suppose the claim is true for $i-1$. Then, event E_{i-1} occurs with probability at least $(1 - \frac{2n}{2^n})^{i-1}$. Suppose that indeed event E_{i-1} occurs and that the end of round i of the protocol is reached. Then considering $A_i(u)$ as a polynomial, it must be the case that $\alpha_i(u) \neq A_i(u)$ (otherwise, the run would have rejected at step 3 of the ith iteration). Since the polynomial $\alpha_i(u) - A_i(u)$ has degree at most $2n$, it has at most $2n$ roots. Hence, the probability that r_i (chosen randomly and uniformly from the range $[0, \ldots, p-1]$) is a root of $\alpha_i(u) - A_i(u)$ is at most $2n/2^n$, since $p > 2^n$. Therefore,

Prob$[E_i \mid E_{i-1}] \geq$ Prob[r_i is not a root of $\alpha_i(u) - A_i(u) \mid E_{i-1}$ and round $i+1$ is reached] $\geq 1 - 2n/2^n$.

As a result,

$$\text{Prob}[E_i] \geq \text{Prob}[E_i \mid E_{i-1}]\text{Prob}[E_{i-1}] \geq \left(1 - \frac{2n}{2^n}\right)\left(1 - \frac{2n}{2^n}\right)^{i-1} = \left(1 - \frac{2n}{2^n}\right)^{i}.$$

□

Theorem 1 *PSPACE ⊆ IP.*

Proof (Sketch) It is not hard to show that the protocol runs in polynomial time, and if $A_\phi \neq 0$ then there is a sequence of guesses that causes the protocol to accept with probability 1.

We will show that for sufficiently large inputs ϕ, if $A_\phi = 0$ then no matter what the strategy is, the probability that the protocol accepts is small. First, no matter what p is chosen, $A_\phi = 0 \mod p$. If a_0 is chosen to be 0, then the protocol rejects; hence suppose that $a_0 \neq 0$, in which case $A_\phi \neq a_0 \mod p$. By Lemma 6, the probability that the run rejects before exiting the for loop, or that the for loop is exited with $v(A_m) \neq a_m$ (in which case the protocol rejects upon exiting the for loop), is at least $(1 - \frac{2n}{2^n})^m$. (Here, m is the number of quantifiers of ϕ.) Therefore, the protocol rejects with probability at least $(1 - 2n/2^n)^m \geq (1 - 2n/2^n)^n$. This quantity goes to 1 as n goes to infinity. □