Does x belongs to L ?

- Verifier
  - An element x
  - Ask questions to prover
  - Gets anwer:
  - Completeness: Is convinced that x in L, if so
  - Soundess: reject « x in L » if not so
- Zero-knowledge:
  - Intuitively: at the end, verifier is convinced that x in L (if so), but *learns nothing else*.

Proof and Interactive proof

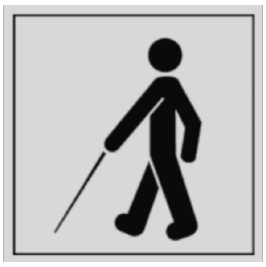
- Two parts in a proof:
  - Prover: knows the proof (-> the secret) [or is intended to know]
  - Verifier: verifies the proof is correct (-> authentication)
- Correctness of a proof system/verifier:
  - **Completeness**: ***every valid proof*** **is accepted** by the verifier
  - **Soundness**: <u>***every invalid proof***</u> **is rejected** by the verifier
- Interactive proof system
  - Protocol (questions/answers) between the verifier and the prover
  - Verifier: **probabilistic** algorithm, **polynomially bounded**
  - Soundness: every invalid proof is rejected with high probability (> 1/2)
  - Completeness: every valid proof is accepted with high probability (>1/2)

Interaction with deterministic verifier and prover

- **Interaction between 2 functions f and g on input x :**
  - $a_1 := f(x)$; $a_2 := g(x, a_1)$; $a_3 := f(x, a_1, a_2)$; …; a_{2i+1}:=$f(x, a_1,…, a_{2i})$; a_{2i+2}:=$g(x, a_1,…,a_{2i+1})$…
  - Notation: $out_f<f , g>(x) = f(x, a_1, … , a_k)$
- **Def**: a language *L* has a *k*-round *deterministic interactive proof system* iff there exists a DTM V that on input $(x,a_1,…, a_i)$ runs in polynomial time $|x|^{O(1)}$ and can have a k-round interaction with any function P such that:
  - Completeness : there exists P such that for any x in L: $Out_V<V,P>(x) = 1$
  - Soundness: iff x not in L then for all P : $Out_V<V,P>(x) = 0$
- Let dIP= { languages L with a k(n)-round deterministic interactive proof system with $k(n)=n^{O(1)}$ }
  - Theorem: dIP = NP. (Proof: 3-SAT)
  - So interaction with deterministic algorithms brings nothing

The power of probabilistic interaction

Prover
(Merlin)



Verifier
(Arthur)

Class IP

- **Def**: a language *L* has a *k*-round *probabilistic interactive proof system* iff there exists a probabilistic polynomial time Turing machine V that that can have a k-round interaction with a function P such that for all input x :
  - Completeness : there exists P such that for any x in L:
    $Prob[\ Out_V\langle V,P\rangle(x) = 1\ ] \geq 2/3$
  - Soundness: iff x not in L then for all P :
    $Prob[\ Out_V\langle V,P\rangle(x) = 1\ ] \leq 1/3$

  Note: all probabilities are on the random choices made by V.

- IP(k) = { L that have a k-round probabilistic interactive proof system}

- IP = $\bigcup_{k \geq 1}$ IP(k)

Example of interactive computation

- Graph isomorphism:
  - Input: G=(V,E) and G'=(V',E')
  - Output: YES iff G == G' (i.e. a permutation of V ->V' makes E=E')
- In NP, not known to be NP-complete,
  not known to be in co-NP.
- Assume an NP Oracle for Graph isomorphism =>
  then a probabilistic verifier can compute Graph isomorphism in polynomial time.
  - Protocol and error probability analysis.
- Theorem [Goldreich&al] :
  - NP included in IP.
  - any language in NP possesses a zero-knowledge protocol.

Interactive Algorithm Graph NonIsomorhism

```
AlgoGraphIso(G1=(V1,E1), G2=(V2,E2) ) {
    If (#V1 != #V2) or (#E1 != #E2)
        return "NO : G1 not isomorphic to G2";
    n := #V1 ;
    For (i=1 .. k) {
        P := randompermutation([1, …, n]) ;
        b := random({1,2}) ;
        G' := P(Gb) ;
        ( i, Pi) := Call OracleWhichIsIso(G1, G2, G') ;
        If (Gi ≠ Pi (G') ) FAILURE("Oracle is not reliable") ;
        If ( b ≠ i) return "YES : G1 is isomorphic to G2" ;
    }
    return "NO : G1 not isomorphic to G2";
}
```

```
OracleWhichIsIso(G1, G2, G') {
    // precondition: G' is isomorphic to
    //               G1 or G2 or both.
    // Output: i into {1,2} and a permutation
    //         Pi such that Gi = Pi ( G' )
    … ;
    Return ( i, Pi ) ;
}
```

Theorem: Assuming OracleWhichIsIso of polynomial time, AlgoGraphIso(G_1, G_2) proves in polynomial time $k.n^{O(1)}$ that :

- either G_1 is isomorphic to G_2 (no error)
- or G_1 is not isomorphic with error probability $\leq 2^{-k}$.

Thus, it is a MonteCarlo (randomized) algorithm for GRAPH ISOMORPHISM

Analysis of error probability

***Truth:* $G_1 = G_2$??** \ *Prob(Output of AlgoGraphIso(G_1, G_2))*	***"YES : G_1 is isomorphic to G_2"***	***"NO: G_1 not isomorphic to G_2"***
Case $G_1 = G_2$ (completeness)	Prob = $1 - 2^{-k}$	Prob = 2^{-k}
No: Case $G_1 \neq G_2$ (soundness)	Impossible (Prob = 0)	Always (Prob = 1)

-When the algorithm output YES : G_1 is isomorphic to G_2 then $G_1 = G_2$
=> no error on this output.

-When the algorithm output "NO: G_1 not isomorphic to G_2" then we may have an error (iff $G_1 = G_2$), but with a probability $\leq 2^{-k}$

One-sided error => Monte Carlo algorithm for Graph-Isomorphism

Graph [non]-isomorphism and zero knowledge

- In a zero-knowledge protocol, the verifier learns that G_1 is isomorphic to G_2 but nothing else.

- Previous protocol not known to be zero-knowledge:
  - Prover sends the permutation P_i such that $G_1 = P_i(G_2)$: so the verifier learns not only G_1 isomorphic to G_2 but P_i too.
  - We do not know, given two isomorphic graph, wether there exists a (randomized) polynomial time algorithm that returns a permutation that proves isomorphism.

A zero-knowledge interactive proof for Graph Isomorhism

Verifier

input: $(G_1=(V_1,E_1), G_2=(V_2,E_2))$

Accepts prover if convinced that G1 is isomorphic to G2

2. Receives H;

   Chooses b=random(1,2) and sends b to the prover

4. receives P'' and checks $H = P''(G_b)$

Prover

gets G_1, G_2

private secret perm. P_s: $G_2=P_s(G_1)$;

1. Chooses a random perm. P' and sends to verifier $H=P'(G_2)$

3. Receives b;

   if b=1 sends $P''=P' o P_s$ to the verifier

   else b=2: sends P''=P' to the verifier

Theorem: This is a zero-knowledge, sound and complete, polynomial time interactive proof that the two graphs G_1 and G_2 are isomorph.

#3-SAT in IP

- **Key 1= Arithmetization**:
  a clause c is represented by a polynomial Q(c) as follows:
  - $Q(\text{not}(x)) = 1-x$ $\quad\quad$ $Q(x \text{ and } y) = x.y$
  - $Q(x \text{ or not}(y) \text{ or } z)=Q(\text{not}(\text{not}(x) \text{ and } y \text{ and not}(z)))= 1-((1-x).y.(1-z))$
- Let: $\Phi = (c_1 \text{ and } \dots \text{ and } c_m)$ be a 3-SAT CNF formula,
  and $g(X_1, \dots, X_n) = Q(c_1).Q(c_2). \dots .Q(c_m)$: $\deg(g) \leq 3m$ (small!)
  A circuit that evaluates g at any $(b_1, \dots, b_n)$ has polynomial size.
- To prove $\#SAT(\Phi)=K$ reduces to $K = \Sigma_{b_1=0,1} \dots \Sigma_{b_n=0,1}\, g(b_1, \dots, b_n)$

#3-SAT in IP

- To prove $\#SAT(\Phi)=K$ reduces to $K = \Sigma_{b_1=0,1} \dots \Sigma_{b_n=0,1}\, g(b_1, \dots, b_n)$
- **Key 2= Recursion .**
  Notation: for $a_1, \dots, a_n$ integers, the following polynomials are defined from g:
  - $\mathbf{g_n}(X_1, \dots, X_n) = g(X_1, \dots, X_n)$; $\mathbf{g_{n-1}}(X_1, \dots, X_{n-1}) = g(X_1, \dots, X_{n-1}, a_n)$
    … $\mathbf{g_k}(X_1, \dots, X_k) = g(X_1, \dots, X_k, a_{k+1}, \dots, a_n)$ …. $\mathbf{g_1}(X_1) = g(X_1, a_2, \dots, a_n)$
  - $\mathbf{S_n}(X)=\Sigma_{b_1=0,1} \dots \Sigma_{b_n=0,1}\, g(b_1, \dots, b_n)$; $\mathbf{S_{n-1}}(X)=\Sigma_{b_1=0,1} \dots \Sigma_{b_{n-1}=0,1} g(b_1,..,b_{n-1},X)$
    $\mathbf{S_{n-2}}(X) =\Sigma_{b_1=0,1} \dots \Sigma_{b_{n-2}=0,1}\, g(b_1, \dots, b_{n-2}, X, a_n)$; … ; $\mathbf{S_2}(X) =g(X, a_2, \dots, a_n)$
- **Recursion** : Proving $\#\Phi=K \Leftrightarrow S_n(X) =K \Leftrightarrow S_{n-1}(0)+S_{n-1}(1)=K$
  To do this, verifier asks to prover $\mathbf{S_{n-1}}(X)$ and checks by random evaluation $\mathbf{S_{n-1}}(X)=\Sigma_{b_1=0,1} \dots \Sigma_{b_{n-1}=0,1} g_n(b_1,..,b_{n-1},X)$;
  this reduces to check $\mathbf{S_{n-1}}(a_n)=\Sigma_{b_1=0,1} \dots \Sigma_{b_{n-1}=0,1} g(b_1,..,b_{n-1},a_n)$
  so $\mathbf{S_{n-1}}(a_n)=\Sigma_{b_1=0,1} \dots \Sigma_{b_{n-1}=0,1} g_{n-1}(b_1,..,b_{n-1})$ => *recursion to n-1*
  Since $\#\Phi=S_n[g]() \leq 2^n$ then for $p>2^n$: $(\#\Phi=K) \Leftrightarrow (\#\Phi=K \bmod p)$
  - To limit to a polynomial number of operations, computation is performed mod a prime p in $2^n .. 2^{2n}$ (provided by prover and checked by verifier)
  - Note: a randomized alternative is that the verifier chooses a *random* smaller prime $p > n^2$.
    (then S_n-K may be multiple of p, which is not possible with $p >2^n$).

#3-SAT in IP

- Error probability: from Schwartz-Zippel:
  - Prob[failure at step k] ≤ $d°(g_k)/p_k \leq d/p$
  - Prob[success at step k] ≥ (1 – d/p)
  - Prob[success for all n steps] ≥ $(1 - d/p)^n$
- Choose p determinstic prime larger than 2^n
  - 2^n = max value for the sum !
  - With p prime, computing mod p makes no error!
- For 3-SAT, d° of each clause ≤ 3, also d°(g)≤3m :

Prob[success] ≥ $(1-3m/p)^n \sim 1-3mn/p$

Choosing p prime larger than $3mn2^n$ (note that p has O(n) bits)

Prob[faiilure] ≤ 2^{-n} (w.h.p.)

Sumcheck protocol in F_p (mod p)

- Input: a circuit $C_n(X_1, \ldots, X_n)$ with n inputs that evaluates a degree d polynomia $g(X_1, \ldots, X_n)$ with coefs in F_p in polynomial time $n^{O(1)}$; and an integer K_n.
- Output: a proof that sum $\Sigma_{b_1=0,1} \ldots \Sigma_{b_n=0,1}\, g(b_1, \ldots, b_n)$ equals K_n *mod* p
  - With notation: this is equivalent to $S_n[g]() = K_n$ *mod* p.
- Verifier: asks prover to send the univariate polynomial $h_n(X)$ of degree ≤ d :

  $h_n(X) = \Sigma_{b_1=0,1} \ldots \Sigma_{b_{n-1}=0,1} g(b_1, b_2, \ldots, b_{n-1}, X)$

  The prover sends to verifier a polynomial $s_n(X)$ (a univariate polynomial in $F_p[X]$)
- Verifier receives $s_n(X)$; it checks that $s_n(0)+s_n(1)=K_n$ *mod* p **and** $s_n(X)=h_n(X)$:
  - First of all, if $s_n(0)+s_n(1) \neq K_n$ reject. Else check $s_n(X)=h_n(X)$by random eval:
  - If n=1: $h_1=g$!! => if $g(0)+g(1) \neq K_1$, reject (else accept !)
  - Else verfier picks a random $0 \leq a_n < p$ and computes $K_{n-1} := s_n(a_n)$ mod p ; then, by recursion, it proves:

    $K_{n-1}=h_n(a)$ *mod* p $=\Sigma_{b_1=0,1} \ldots \Sigma_{b_{n-1}=0,1}\, C_{n-1}(b_1, \ldots, b_{n-1})$

    by building the circuit $C_{n-1}(X_1, \ldots, X_{n-1})$ equals to $C_n(X_1, \ldots, X_{n-1}, a)$.

    If $s_n(a) \neq h_n(a)$ *mod* p, the proof is rejected (error detected)
- Error probability [soundness] Prob[reject | sum$\neq K_n$] ≥ $(1-d/p)^n$.

  [by induction: n=1: no error => P[reject | sum≠K]=1 ≥ (1-d/p). Now suppose property true at n-1. At n we have Pr(error)=Pr[$s_n(a)=h_n(a)$ | $s_n \neq h_n$] ≤ d/p => Pr[reject | sum$\neq K_n$] ≥ Pr[reject | $s_n \neq h_n$] ≥ $(1-d/p)^{n-1}.(1-d/p) \geq (1-d/p)^n$.]

#3-SAT: interactive polynomial proof

Verifier

input: $F(X_1, \ldots, X_n) = (c_1 \text{ and } \ldots \text{ and } c_m)$
K_n an integer; let $g(x) = \Pi_{i=1,n} Pol(c_i)$

Accepts iff convinced that $\#F = K_n$.

Preliminar receive p, check p is prime in $\{2^n, 2^{2n}\}$

Compute $g(X_1, \ldots, X_n) = \Pi_{i=1,n} Pol(c_i)$ $deg(g) \leq 3m$

Check $K_n = \Sigma_{X1=0,1} \ldots \Sigma_{Xn=0,1}\, g(X_1, \ldots, X_n)$ [p] :

1. If n=1, if $(g(0)+g(1) = K_n)$ accept ; else reject. If $n \geq 2$, ask $h_n(X)$ to P.

3. Receive $s_n(X)$ of degree $\leq m$. Compute $v_n = s_n(0) + s_n(1)$; if $(v_n \neq K_n)$ reject. else choose $r_n = rand(0, \ldots p-1)$; let $K_{n-1} = s(r_n)$ and use the same protocol to check
$K_{n-1} = \Sigma_{X1=0,1} \ldots \Sigma_{Xn-1=0,1}\, g(X_1, \ldots, X_{n-1}, r_n)$ [p]

Prover

Preliminar: sends p prime in $\{2^n, 2^{2n}\}$

2. Send $s(X)$; [note that if P is not cheating, $s(X) = h_n(X)$]

Theorem: This is a sound and complete, polynomial time randomized interactive proof of #3-SAT.
Moreover, prob(V rejects | $K \neq \#F$) $\geq (1-3m/p)^n$,
also prob(error) $\leq 1-(1-3m/p)^n \leq 3mn2^{-n}$.

Interactive proof of TQBF (1/2)

- Input: quantified boolean formula $F = \forall X_1\ \exists X_2\ \forall X_3 \ldots \exists X_n : \Phi(X_1, \ldots, X_n)$
  Output: Yes if F is true
- Arithmetization: let $P_\Phi(X_1, \ldots, X_n)$ the polynomial that represents Φ.
  - $\exists X_n \in \{0,1\} : Q(X_1, \ldots, X_n)$ is represented by polynomial $Q(X_1, \ldots, X_{n-1}, 0) + Q(X_1, \ldots, X_{n-1}, 1)$
  - $\forall X_n \in \{0,1\} : Q(X_1, \ldots, X_n)$ is represented by polynomial $Q(X_1, \ldots, X_{n-1}, 0).Q(X_1, \ldots, X_{n-1}, 1)$
- With a similar approach to #SAT, arithmetization leads to check s(0).s(1)=K
  But then multiplication makes the degree increase to 2^n (not polynomial !)
- Key: we are only interested by {0,1} values! A polynomial P can be approximated with a multi-linear function with same evaluations at $\{0, 1\}^n$.
  Let $L_i[P]$ be the linearization operator defined as :
  $L_i[P(X_1, \ldots, X_n)] = (1 - X_i)P(X_1, \ldots, X_{i-1}, 0, X_{i+1}, \ldots, X_n) + (X_i)P(X_1, \ldots, X_{i-1}, 1, X_{i+1}, \ldots, X_n)$.
- Linearization of F leads to the expression :
  $\forall X_1\ L_1[\exists X_2\ L_1L_2[\ \forall X_3\ L_1L_2L_3[\ \ldots[\exists X_n\ L_1L_2 \ldots L_n[\ P_\Phi(X_1, \ldots, X_n)\ ]]..]]]$
  whuch is of size $O(1+2+3+\ldots+n) = O(n^2)$ polynomial.

Interactive proof of TQBF (2/2)

- Recursive protocol. Suppose for any polynomial $g(X_1, \ldots, X_k)$ the prover is able to convince the verifier that
  - $g(a_1, \ldots, a_k)=C$ with prob=1 for any $a_1, \ldots, a_k, C$ when it is true
  - and prob≤ε when it is false.
- Let U be the polynomial of degree d :
  - Case 1: $U(X_1, \ldots, X_{k-1})$ = « $\exists X_k \in \{0,1\} : g(X_1, \ldots, X_k)$ » = $g(X_1, \ldots, X_{k-1},0) + g(X_1, \ldots, X_{k-1},1)$
    => The prover provides a polynomial $s(X_k)$ supposed to be $g(a_1, \ldots, a_{k-1}, X_k)$
    Verifier checks if $s(0)+s(1) = C$. If not reject;
    else verifier picks a random $0 \leq \alpha < p$ and asks prover to prove « $s(\alpha) = g(a_1, \ldots, a_{k-1}, \alpha)$ ».
  - Case 2: $U(X_1, \ldots, X_{k-1})$ = « $\forall X_k \in \{0,1\} : g(X_1, \ldots, X_k)$ » = $g(X_1, \ldots, X_{k-1},0).g(X_1, \ldots, X_{k-1},1)$
    Same as case 1 but verifier checks if $s(0).s(1) = C$ [instead of $s(0)+s(1)=C$]
  - Case 3: $U(X_1, \ldots, X_k)$ = « $L_k[g(X_1, \ldots, X_k)]$ » = $(1-X_k)g(X_1, \ldots, X_{k-1},0) + X_k.g(X_1, \ldots, X_{k-1},1)$
    => The prover provides a polynomial $s(X_k)$ supposed to be $g(a_1, \ldots, a_{k-1}, X_k)$
    Verifier checks $(1-a_k)s(0) + a_k.s(1) = C$. If not reject;
    else verifier picks a random $0 \leq \alpha < p$ and asks prover to prove « $s(\alpha) = g(a_1, \ldots, a_{k-1}, \alpha)$ ».
- Error analysis

Complexity classes

Decision problems (1 output bit: YES/ NO)

Deterministic polynomial time:
- P : both Yes/No sides
- NP : certification for the Yes side
- co-NP: certification for the No side

Randomized polynomial time:
- BPP: Atlantic City: prob(error) < 1/2
- RPP: Monte Carlo: prob(error YES side)=0 ; prob(error NO side)< 1/2
- ZPP: Las Vegas: prob(failure)<1/2 but prob(error)=0

IP Interactive proof
- Verifier: randomized polynomial time
- Prover: interactive (dynamic), unbound power
  - F(x) = YES => it exists a correct prover Π such that Prob[Verifier (Π, x) accepts] = 1;
  - F(x) = NO => for all prover Π: Prob[Verifier (Π, x) accepts] < 1/2.
- Theorem: IP = PSPACE (interaction with randomized algorithms helps!)

PCP: Probabilistiic Checkable Proofs (static proof)
- PCP(r, q) : the verifier uses random bits and reads q bits of the proof only.
- Theorem: NP=PCP(log n, O(1))

Application in cryptology: zero-knowledge [wikipedia]

- Importance of « proof » in crypto: eg. identity proof=authentication
- Ali Baba (Peggy) knows the secret
  - "iftaH ya simsim" («Open Sesame»)
  - "Close, Simsim" («Close Sesame»).
- Bob (Victor) and Ali Baba design a protocol to prove that Ali Baba has the secret without revealing it
  - Ali Baba is *the prover*
  - Bob is *the verifier*
  - Ali Baba leaks no information