# * Interactive proofs and Polynomial hierarchy

امید یعقوبی

دانشکده علوم ریاضی، دانشگاه صنعتی شریف

۲۶ شهریور ۱۴۰۰

**چکیده**

در ۱۹۸۰ مفهوم interactive computation به دو شکل متفاوت توسط Babai به عنوان یک نمونه تغییر یافته از $NP$ که اجازه می‌دهد verifier احتمالاتی باشد [۱] و توسط Goldwasser و Micali به عنوان سیستمی برای اثبات بدون درز دانش [۲] معرفی شد. یک interactive proof system روشی است برای آن‌که *prover* با توانایی بی‌نهایت با گفت‌وگو به *verifier* با توانایی محدود اثبات کند که قضیه‌ای درست است. در این گزارش قدرت توصیفی تصادفی سازی و برهم‌کنش نشان داده می‌شود. یکی از مساله‌های بسیار مهم در نظریه پیچیدگی مساله‌ی ایزومورفیسم بودن گراف‌ها می‌باشد. این‌که مکمل این مساله دارای یک interactive proof است یکی از مهم‌ترین نتایج پیچیدگی محاسبه محسوب می‌شود. سلسله مراتب چندجمله‌ای یکی از مطالب کلیدی این نوشتار است که رابطه‌ی بین توصیف زبان به وسیله‌ی سورهای وجودی و عمومی و پیچیدگی آن‌ها را برای ما روشن می‌سازد. دانشمندان پیچیدگی محاسبه همواره از فروریختن سلسله مراتب چندجمله‌ای به عنوان یک مدرک معتبر برای درست نبودن یک فرض استفاده می‌کنند. در این نوشتار این جمله و سلسله مراتب چندجمله‌ای به طور دقیق مورد بررسی قرار می‌گیرند و رابطه‌ی بین آن و interactive proof system به طور دقیق آنالیز خواهد شد. همچنین نگاه به کلاس‌های پیچیدگی $IP = PSPACE$ و $PH$ و $AM$ به شکل یک بازیِ دونفره از اهداف اصلی این گزارش بوده است.

**کلمات کلیدی:** interactive proofs,polynomial hierarchy,zero-knowledge proofs,graph isomorphism,Arthur-Merlin games

## ۱ معرفی

نگاه سنتی به $NP$ گروهی از زبان‌ها را در بر می‌گیرد که برای عضویت در آن‌ها می‌توان اثباتی [۱] کوتاه داد. یک اثبات برای $x \in L$ همان گواه [۲] برای اثبات عضویت این رشته در زبان است. به بیان دیگر اگر یک *verifier* چندجمله‌ای برای زبان $L$ داشته باشیم که بتوان برای هر $x \in L$ یک گواه چندجمله‌ای با نام $u$ آورد به شکلی که $v(x,u) = 1$ باشد و برای هر $x' \notin L$ نتوان یک گواه تقلبی با نام $u'$ آورد که

*در فارسی برابر Interaction برهم‌کنش است و به تکرار ما Proof system را با *دستگاه اثبات* می‌شناسیم، در نتیجه ترجمه‌ی Interactive Proof Systems را شاید بتوان *دستگاه‌هایِ اثبات برهم‌کنشی* نامید. اما همان‌گونه که کلاس NP را با نام ناآشنایِ *چندجمله‌ای‌ــ غیرقطعی* معرفی نمی‌کنیم و تلاش می‌کنیم تا ترمینولوژی یک علم را امانت‌دارانه در نوشته‌های زبان مادری خود بیاوریم، این‌جا نیز ما برای ترمینولوژی‌های اصلی مانند Interactive Proof System برابر فارسی ناآشنا نخواهیم آورد.

[1] Proof

[2] Witness

$v(x', u') = 1$ آن‌گاه می‌گوییم $L \in NP$ است. یک گواه برای $x \in L$ می‌بایست اندازه‌ای چندجمله‌ای نسبت به $x$ داشته باشد، اما اجباری نیست که بتوان آن را در زمان چندجمله‌ای از روی $x$ ساخت. نگاهی متفاوت‌تر، در نظر گرفتن $NP$ به عنوان کلاسی از زبان‌هاست که برای آن‌ها اثبات‌گری [3] توانمند از نگاه محاسباتی وجود دارد که می‌تواند $x \in L$ را به یک $verifier$ قطعی [4] و چندجمله‌ای اثبات کند. برهم‌کنش بین prover و verifier در این مورد بدیهی است. prover در این‌جا به راحتی certificate را برای verifier ارسال می‌کند و verifer در زمان چندجمله‌ای درست بودن این اثبات را بررسی می‌کند و در صورت درست بودن آن را می‌پذیرد. [۳]

تلاش‌های بسیاری برای توصیف پروسه‌ی اثبات [5] به صورت فرمال پیش از این صورت گرفته است. $NP$ تا به امروز یک توصیف بسیار موفق از این مفهوم به حساب می‌آید. به بیان غیردقیق، یک قضیه در $NP$ قابل اثبات است اگر بتوان در صورت از پیش دادن اثباتی برای آن، به سادگی درستی این اثبات را بررسی کرد. در حقیقت کوک کلاس $NP$ را این‌گونه تعریف می‌کند: " یک $NP$ proof-system دارای دو ماشین تورینگ گفت‌وگو کننده با نام‌های $A$ و $B$ است که به آن‌ها به ترتیب *prover* و *verifier* می‌گوییم. prover دارای توان نمایی [6] بوده و verifier توان چندجمله‌ای دارد. هر دوی $A$ و $B$ قطعی هستند. آن‌ها یک نوار را برای ارتباط با یکدیگر به اشتراک گذاشته‌اند. روی ورودی $x$ در زبانی که در کلاس $NP$ است، $A$ ابتدا یک $y$ که اندازه‌ای چندجمله‌ای نسبت به $x$ دارد را محاسبه می‌کند. حال آن را روی نواری که بین آن‌ها مشترک است می‌نویسد تا $B$ بتواند آن را بخواند. حال $B$ بررسی می‌کند که $f_L(y) = x$ باشد. $f_L$ یک تابع قابل محاسبه در زمان چندجمله‌ای [7] نسبت به زبان $L$ است، اگر این گونه بود آن را می‌پذیرد. "[۲] در این‌جا $A$ در واقع به $B$ شاخه‌ای از محاسبه را نشان می‌دهد که به *accept* می‌رسد. در تعریف پذیرفته شده‌ی امروزی، $A$ همان *certificate* را برای $x \in L$ محاسبه کرده و آن را به $B$ می‌دهد تا آن را *verify* کند.

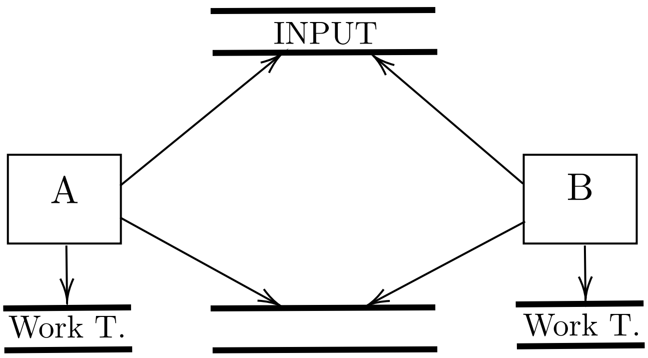


Figure 1: The $NP$ proof-system

چه چیز به شکل شهودی برای یک پروسه‌ی اثبات اساسی است؟ ابتدا انتظار داریم که یک قضیه‌ی درست را بتوان اثبات کرد. پس از آن می‌خواهیم که به هیچ شکلی نتوان اثباتی برای یک قضیه‌ی غلط آورد.

[3] Prover

[4] Deterministic

[5] Theorem-proving procedure

[6] Exponential-time

[7] Polynomial-time computable function

در آخر انتظار می‌رود که بررسی درستی اثبات یا به اسطلاح *verify* کردن آن به صورت کارا [۸] انجام پذیرد. اگر بخواهیم شرط سوم را بیشتر توضیح دهیم، این‌که چقدر زمان می‌برد که prover اثبات خود را از نگاه محاسبه‌ای بسازد مهم نیست، ولی اهمیت دارد که verifier بتواند درستی آن را به سادگی بررسی کند. [۲]

این‌که چگونه *proof* یا اثبات را تعریف کنیم در آن‌چه در مورد پروسه‌ی اثبات می‌گوییم تاثیر مهمی خواهد داشت. مفهوم اثبات نیز مانند مفهوم محاسبه [۹] یک مفهوم ذاتا شهودی است. اگر بخواهیم در مورد این‌گونه مفاهیم در جهان ریاضیات سخن بگوییم، مجبوریم تا آن‌ها را به شکل قابل قبولی فرمال سازیم. محاسبه‌پذیری توسط ماشین تورینگ یک نمونه‌ی بسیار جالب از فرمال‌سازی [۱۰] مفهوم شهودی محاسبه است. با این وجود هیچ فرمال‌سازی نخواهد توانست آن‌چه در شهود خود داریم را بازنمایی کند، چراکه آن‌چه به آن شهود می‌گوییم یک امر درونی است. $NP$ نیز یک فرمال‌سازی موفق از مفهوم شهودی پروسه‌ی اثبات به شمار می‌آید. با این حال $NP$ تنها نوع خاصی از این پروسه‌ی اثبات را فرمال کرده است. آن دسته از اثبات‌ها که می‌توان آن‌ها را در دفتری نوشت و آن را تحویل داد. برای فرمال سازی دسته‌ی گسترده‌تری از اثبات‌ها که شامل برهم‌کنش بین prover و verifier است، *interactive proof-systems* معرفی می‌شوند. ما در این‌جا با آن دسته از اثبات‌ها سر و کار داریم که می‌توان آن‌ها در " *کلاس درس* " توضیح داد. در کلاس درس، مدرس، از قابلیت گفت‌وگو کردن با دانشجویانی که اثبات به آن‌ها داده می‌شود به شکل کارایی بهره می‌برد تا اثبات مورد قبول آن‌ها قرار گیرد. آن‌ها ممکن است در بین اثبات سوال‌هایی بپرسند و جواب‌هایی بگیرند. به این ترتیب پروسه‌ی اثبات ساده‌تر از زمانی خواهد بود که مجبوریم کل اثبات را در دفتری نوشته و بی هیچ گفت‌وگویی آن را تحویل دهیم. این به آن دلیل است که نوشتن این اثبات در دفتر باید به شکلی باشد که هرگونه سوال احتمالی را پیشبینی و به آن پاسخ گوید که این می‌تواند اندازه‌ی اثبات را طولانی سازد. [۲]

پس از مطرح کردن این مدل، ممکن است این سوال پیش آید که پس از تکمیل فرایند اثبات که در آن prover به verifier یک قضیه را اثبات می‌کند، verifier چه میزان بیشتر از قبل می‌داند؟ یا به بیانی چه مقدار دانش [۱۱] در این فرایند از prover به verifier منتقل شده است؟ آیا روشی وجود دارد که توسط آن prover بتواند قضیه‌ای را به verifier اثبات کند بی‌آنکه verifier پس از اتمام این فرایند چیزی بیش از درستی [۱۲] این اثبات بداند؟ برای فهم بهتر این مساله در نظر بگیرید که یک دانشجوی تازه ورود علوم‌کامپیوتر به ناگهان توانسته است مساله $P$ vs $NP$ را حل کند. او در واقع اثبات کرده است که $P = NP$ برقرار است. حال می‌خواهد این ادعای خود را پیش یکی از اساتید که با او آشنایی ندارد اثبات کند. استاد نمی‌تواند ادعای عجیب این دانشجوی جوان را باور کند و همچنین باورش بر این است که $P \neq NP$ برقرار است. حال از او می‌خواهد که اثبات خود را به او بیان کند. اما دانشجوی جوان و گمنام از این واهمه دارد که استاد اثبات او را به نام خود در مجلات علمی منتشر سازد و به همین رو این درخواست را نمی‌پذیرد. استاد که دانش و تجربه‌ی فراوان در رشته‌ی خود دارد به او می‌گوید اگر ادعای شما راست باشد، از آن‌جا که می‌دانیم $FACTORING$ یک مساله $NP$ است و شما ادعا می‌کنید $P = NP$ در نتیجه اگر من یک نمونه از این مساله را به شما بدهم می‌توانید در مدت زمان قابل قبولی جواب آن را به من بدهید. سپس او به صفحه‌ی ویکی‌پدیای *RSA-numbers* مراجعه می‌کند و عدد زیر را روی کاغذی نوشته و به دانشجو می‌دهد:

RSA-2048= 2519590847565789349402718324004839857142928212620403202

7777137836043662020707595556264018525880784406918290641249515082‌1

8929855914917618450280848912007284499268739280728777673597141834‌7

---

[8] Efficient

[9] Computation

[10] Formalization

[11] Knowledge

[12] Validity

2702618963750149718246911650776133798590957000973304597488084284017974291006424586918171951187461215151726546322822168699875491824224336372590851418654620435767984233871847744479207399342365848238242811981638150106748104516603773060562016196762561338441436038339044149526344321901146575444541784240209246165157233507787077498171257724679629263863563732899121548314381678998850404453640235273819513786365643912120103971228221207203557

اگر دانشجو پس از مدتی بازگشت و فاکتورهای اول این عدد بزرگ را به استاد داد، استاد می‌تواند به راحتی درستی آن را با ضرب کردن فاکتورها بررسی کند و تا میزان قابل قبولی متقاعد می‌شود که او راست می‌گوید ولی نمی‌تواند صد در صد اطمینان داشته باشد، چراکه $FACTORING$ مساله‌ای نیست که $NPC$ بودن آن ثابت شده باشد و در واقع بسیار نامحتمل است که چنین مساله‌ای $NPC$ باشد (*چراکه این مساله در* $NP \cap coNP$ *است و اگر این‌طور باشد سلسله‌مراتب چندجمله‌ای فرو خواهد ریخت.*) و همچنین احتمال دارد که دانشجو فاکتورها را از روشی دیگر پیدا کرده باشد، یا ممکن است این مساله از ابتدا در کلاس $P$ بوده باشد. ولی او تا حدودی متقاعد شده است چون می‌داند که فاکتورهای این عدد بزرگ پیدا نشده‌اند و برای یابنده آن نیز ۲۰۰ هزار دلار جایزه‌ی نقدی گذاشته‌اند. پس از این رو او احتمال بالایی می‌دهد که دانشجو راست می‌گوید و برای آن‌که این احتمال را بالاتر هم ببرد می‌تواند دانشجو را با تعداد بیشتری از این اعداد بزرگ RSA ارزیابی کند. هرچند بازهم این احتمال وجود دارد که دانشجو به روش‌های دیگری این فاکتورها را به دست بیاورد و به هیچ رو مساله‌ی $P$ vs $NP$ را حل نکرده باشد، ولی حالا تا میزان بالایی استاد متقاعد شده است. حال بررسی کنیم که استاد چقدر بیشتر از آن‌چه قبل از اثبات می‌دانست، می‌داند. او حالا فاکتورهای این عددهای بزرگ را می‌داند، پس میزان دانش او پس از اثبات احتمالاتیِ دانشجو صفر نیست. مثال دیگر بازی سودوکو است، آلیس یک مساله سودوکو را حل کرده است ولی باب که نتوانسته است آن را حل کند به آلیس می‌گوید که این جدول حل نشدنی است. آلیس ادعا می‌کند که آن را حل کرده است ولی اگر جواب خود را به باب بدهد، بازی برای باب بی‌مزه خواهد شد. آیا روشی وجود دارد که بدون دادن هیچ دانشی نسبت به جواب جدول، آلیس به باب با احتمال بالا اثبات کند که جدول راه حل دارد؟ این‌ها نمونه مساله‌هایی است که پس از تعاریف به آن‌ها خواهیم پرداخت.

## ۲ Interactive proofs

از آن‌جا در آینده خواهیم دید قدرت محاسباتی کلاس تعریف شده توسط Interactive Proofs با PSPACE برابر است، استفاده از تعریف مبتنی بر بازیِ دونفره برای آن دور از ذهن نخواهد بود. به همین رو تعریف خود را بر پایه‌ی منبع [۴] تنظیم می‌کنیم.

هر مساله $A$ را می‌توان به شکل یک بازیِ دونفره تصور کرد. بازیکن اول prover سعی دارد بازیکن دوم یا همان verifier را متقاعد کند که $x \in A$ برقرار است. روی هر نمونه $x$ هر بازیکن در نوبت خود رشته‌ی $y_i$ در $i$ امین مرحله از بازی به دیگری ارسال می‌کند. ساخت $y_i$ از روی $Partial\ message$ $history$ که برابر است با پیام‌های تا این‌جا تبادل شده به شکل $(y_1, y_2, \ldots, y_{i-1})$ و ورودی $x$ توسط بازیکن‌ها انجام می‌شود. بعد از $k$ حرکت prover برنده است اگر که verifier رشته‌ی $x$ را در نهایت verify کند یا به بیانی اگر تابع مربوط به verifier را با $f$ نشان دهیم و $y = y_1, y_2, \ldots, y_k$ آن‌گاه $f(x, y) = 1$ باشد. در غیر این صورت می‌گوییم verifier برنده است. همچنین تعداد راندهای بازی و همچنین طول هر پیام ارسال شده می‌بایست نسبت به $x$ چندجمله‌ای باشد. [۴]

در حقیقت پس از یک دوره سوال/جواب بین prover و verifier در نهایت این verifier است که تصمیم می‌گیرد $x \in L$ برقرار است یا نه، توجه شود که prover ممکن است برای برنده شدن بخواهد تقلب کند، یعنی بخواهد با دادن یکسری جواب به دروغ verifier را متقاعد کند که $x \in L$ برقرار است در صورتی که $x \notin L$ برقرار است.

حالت‌های متفاوتی را می‌توان برای prover و verifier تصور کرد. ابتدا حالتی را تصور می‌کنیم که هردوی آن‌ها Deterministic باشند. برای نمونه فرض کنید می‌خواهیم برای $3SAT$ یک Interactive Proof ساده بدهیم. verifier در هر راند برای هر clause با نام $C_j$ لیترال True کننده‌ی آن clause را از prover می‌پرسد. در یک مرحله، بدون از دست دادن عمومیت $(wlog)$ فرض کنیم prover یک لیترال به فرم منفی مانند $\neg x$ را به verifier تحویل داده است. حال verifier در پیام‌های تا به حال تبادل شده یا Partial message history بررسی می‌کند که پیش از این prover فرم مثبت آن، یعنی $x$ را تحویل نداده باشد. اگر فرم مثبت آن وجود داشت، یعنی prover می‌خواهد تقلب کند و در نتیجه بی‌درنگ verifier آن را reject می‌کند. اما اگر فرم مثبت آن وجود نداشت چک می‌کند که به راستی لیترال درون آن clause ظاهر شده باشد یا به عبارتی $\neg x \in C_j$ باشد. سپس به سراغ clause بعدی می‌رود. اگر برای همه‌ی clause ها این شرطها برقرار بود، verifier ورودی را Accept می‌کند. به این ترتیب به نظر می‌رسد که برهم‌کنش در مورد زبان‌های $L \in NP$ بی‌فایده است. چراکه هر دوی prover و verifier از پروتوکول آگاهند و درنتیجه prover می‌تواند در همان پیام اول، در یک راند، کل جواب‌ها را بدون دریافت هیچ سوالی از verifier برای او ارسال کند. پیام او بازهم چندجمله‌ای نسبت به $x$ باقی خواهد ماند. همچنین verifier نیز در زمان چندجمله‌ای نسبت به ورودی $x$ می‌تواند آن را verify کند.

**مشاهده ۱.۲.** *اگر هر دوی prover و verifier قطعی (Deterministic) باشند، به بیش از یک راند نیازی نیست.*

با همان تکنیکی که برای مساله‌ی $3SAT$ معرفی کردیم prover می‌تواند در یک راند، همه‌ی پیام‌هایی که قرار است ارسال شود را در راند اول ارسال کند چراکه از پروتوکول آگاهی دارد و رفتار verifier نیز برای هر ورودی قطعی است.

**تعریف ۱.** *(Interaction بین دو تابع قطعی) اگر $f$ و $g$ دو تابع قطعی و $k$ یک عدد طبیعی مثبت به*

*شکل زیر باشند:*

$$f, g : \{0,1\}^* \to \{0,1\}^*, k \in \mathbb{N}^+$$

*حال یک Interaction با $k$ راند بین $f$ و $g$ روی ورودی $x \in \{0,1\}^*$ را با نماد $\langle f, g\rangle(x)$ نشان می‌دهیم و دنباله‌ای از رشته‌ها به شکل $y_1, y_2, \ldots, y_k \in \{0,1\}^*$ می باشد که در ادامه تعریف می‌شود:*

$$\begin{aligned} y_1 &= f(x, \epsilon) \\ y_2 &= g(x, y_1) \\ y_3 &= f(x, y_1y_2) \\ &\vdots \\ y_k &= Q(x, y_1y_2\ldots y_{k-1}), Q \in \{f, g\} \end{aligned}$$

*همچنین خروجی تابع $f$ را در انتهای interaction با $out_f\langle f, g\rangle(x)$ نشان می‌دهیم و برابر با $f(x, y_1y_2\ldots y_k) \in \{0,1\}$ می‌باشد.*

**تعریف ۲.** *(Deterministic proof systems) می‌گوییم زبان $L$ دارای یک Deterministic proof system با $k$ راند است، اگر TM با نام $V$ برای آن وجود داشته باشد که روی ورودی $x, y_1y_2\ldots y_i$ در زمان چندجمله‌ای نسبت به $x$ اجرا شود و بتواند یک Interaction، $k$ رانده با هر تابع $P$ داشته باشد، به شکلی که شرایط زیر برقرار باشند:*

$$\begin{aligned} &(Completeness) \quad x \in L \Rightarrow \exists P : \{0,1\}^* \to \{0,1\}^*, out_v\langle V, P\rangle(x) = 1 \\ &(Soundness) \quad x \notin L \Rightarrow \forall P : \{0,1\}^* \to \{0,1\}^*, out_v\langle V, P\rangle(x) = 0 \end{aligned}$$

ما از یک دستگاه اثبات انتظار داریم که در آن بتوان هر قضیه‌ی درست را اثبات کرد، این در اولین شرط، یعنی شرط تمامیت [۱۳] در نظر گرفته شده است. همچنین ما از یک دستگاه اثبات انتظار داریم که اگر یک قضیه غلط باشد نتوان برای آن یک اثبات آورد. این در شرط دوم، یعنی شرط درستی [۱۴] لحاظ شده است.

حال کلاس *dIP* را این‌گونه تعریف می‌کنیم که شامل همه‌ی زبان‌هایی باشد که برای آن یک Deterministic interactive proof system با $k(n)$ راند وجود داشته باشد. $k(n)$ چندجمله‌ای نسبت به $n = |x|$ است.

براساس مشاهده‌ی پیش هر Deterministic interactive proof system با $k(n)$ قابل تبدیل به یک Deterministic interactive proof system معادل با یک راند است.

[13] Completeness
[14] Soundness

توجه شود که برای prover یا همان $P$ هیچ محدودیت از نظر توانایی محاسباتی گذاشته نشده است. این با شهود ما نیز سازگار است، چراکه یک certificate تقلبی به هیچ روی نباید قابل اثبات باشد و مهم نیست که prover چقدر در این زمینه باهوش بوده باشد.

### لم ۲.۲. ***dIP=NP***

اگر $L \in NP$ آن‌گاه برای آن یک TM چندجمله‌ای نسبت به $|x|$ با نام $M$ وجود دارد به شکلی که:

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{poly(|x|)} \text{ s.t } M(x,u) = 1$$

که $u$ در حقیقت همان certificate برای $x \in L$ است. حال می‌خواهیم نشان دهیم که $L \in dIP$ برقرار است. در اولین راند prover همان $u$ را برای verifier ارسال می‌کند. verifier نیز همان $M(x,u)$ را محاسبه می‌کند. از آن‌جا که prover قدرت بی‌انتها دارد می‌تواند در زمان نمایی با Exhastive search تمام certificate های ممکن را بررسی کند و certificate مناسب را پیدا کند. در نتیجه $NP \subseteq dIP$ برقرار است.
■

حال فرض کنیم که $L \in dIP$، پس برای آن یک Proof system با دو تابع قطعی $P$ و $V$ داریم به شکلی که $out_v\langle V, P\rangle(x) = 1$ می‌باشد. حال اگر $x \in L$ برقرار باشد آن‌گاه یک Message history به شکل $(y_1, y_2, y_3, \ldots, y_{k(n)})$ برای آن وجود دارد که $V(x,\epsilon) = y_1$ و $P(x,y_1) = y_2$ و $V(x, y_1y_2) = y_3$ و ... پس certificate برای $x \in L$ را همان تاریخچه‌ی پیام یا Message history می‌گیریم که به شکل $(y_1, y_2, y_3, \ldots, y_{k(n)})$ تعریف می‌شود. از آن‌جا که $V$ یک $TM$ چندجمله‌ای است، پس می‌توانیم از آن به عنوان یک subroutine در $V'$ که verifier با تعریف $NP$ است استفاده کنیم. برای آن‌که درستی این certificate را بررسی کنیم چک می‌کنیم که $V(x,\epsilon) = y_1$ و $V(x, y_1y_2) = y_3$ و $V(x, y_1y_2y_3y_4) = y_5$ و ... و $V(x, y_1y_2y_3y_4\ldots y_k) = 1$ باشد. اگر این شرطها برقرار باشد $V'$، Accept می‌کند. در واقع verifier با تعریف $NP$ بررسی می‌کند که certificate یک Message history معتبر نسبت به $P$ و $V$ باشد به شکلی که $out_v\langle P, V\rangle(x) = 1$ باشد. برای این‌کار او سوال‌های V را بررسی می‌کند. برای مثال با چک کردن $V(x, y_1y_2y_3y_4) = y_5$ بررسی می‌کند که اگر $P$ برای $V$ رشته‌ی $y_4$ را به عنوان جواب $y_3$ ارسال کند آیا واقعا $V$ از او سوال $y_5$ را می‌پرسد؟ به این ترتیب از آن‌جا که می‌دانیم اگر $x \in L$ باشد وجود دارد $P$ به شکلی چنین محاوره‌ای با $V$ داشته باشد، پس اگر $x \in L$ وجود دارد رشته‌های $y_2, y_4, \ldots$ به شکلی که جواب‌هایی باشند برای سوال‌های $y_1, y_3, \ldots$ و همچنین مشاهده شود که تعداد پیام‌ها و اندازه هر پیام نیز چندجمله‌ای است، پس certificate نیز چندجمله‌ای است. به این ترتیب اثبات کردیم که

$$x \in L \Rightarrow \exists u \in \{0,1\}^{poly(|x|)}, V'(x,u) = 1$$

توجه شود که یک طرف تعریف $NP$ را اثبات کردیم، حال باید اثبات کنیم اگر چنین certificate وجود داشته باشد هم $x \in L$ است. فرض می‌کنیم که چنین Message history وجود دارد. حال $P$ را این‌گونه تعریف می‌کنیم که $P(x, y_1) = y_2$ و $P(x, y_1y_2y_3) = y_4$ و $P(x, y_1y_2y_3y_4y_5) = y_6$ و ... پس وجود دارد $P$ به شکلی که $out_v\langle V, P\rangle(x) = 1$ باشد. حال از عکس‌نقیض شرط soundness نتیجه می‌گیریم که $x \in L$ است و در نهایت داریم:

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{poly(|x|)}, V'(x,u) = 1$$

در نتیجه $L \in NP$ و چون $L$ دلخواه بود $dIP \subseteq NP$ و در آخر $dIP = NP$ اثبات می‌شود.
■

# ۳ کلاس IP

## ۱.۳ ترکیب تصادفی‌سازی و برهم‌کنش

پیش از این دیدیم که اضافه شده Interaction به روند اثبات به شکلی که همه چیز قطعی باشد کمک چندانی به ما نمی‌کند و کلاسی که به ما می‌دهد همان کلاس $NP$ خواهد بود و همچنین در هیچ‌کدام از موارد در واقعیت به چیزی بیش از یک راند Interaction نیازی نیست. حال می‌خواهیم بررسی کنیم که تصمیمات تصادفی چه تاثیری روی قدرت محاسباتی ما می‌گذارد و چه میزان به آن توانایی می‌دهد.

برای روشن شدن قدرت تصادفی سازی تصمیمات در روند Interaction بگذارید یک مساله ذهنی را مطرح کنیم. آلیس دارای دو جوراب همسان با رنگ‌های متمایز سبز و قرمز است. باب که کوررنگ است ادعای آلیس را مبنی بر متمایز بودن رنگ‌های آن دو نمی‌پذیرد. به نظر او این دو جوراب یکرنگند. آلیس می‌خواهد ادعای خود را به باب ثابت کند.

آلیس هر دو جوراب را به باب می‌دهد و به او می‌گوید کدام سبز و کدام قرمز است. سپس او جوراب اول را در یک دست و جوراب دوم را در دست دیگر نگه می‌دارد. آن‌گاه از آلیس می‌خواهد که رویش را برگرداند. باب سکه‌ای پرت می‌کند و اگر سکه رو بود کاری نمی‌کند، در غیر این صورت جای آن دو را تعویض می‌کند. حال از آلیس می‌خواهد که برگردد و سپس از او می‌پرسد که کدام جوراب سبز و کدام قرمز است. اگر ادعای آلیس درست باشد او به راحتی جواب درست را می‌دهد. در غیر این صورت او مجبور است با احتمال $\frac{1}{2}$ حدس بزند. پس از $10$ با تکرار، احتمال این که او متقلب باشد و هر $10$ بار را شانسی درست حدس زده باشد $\frac{1}{2^{10}}$ و بسیار ناچیز است. پس باب پس از $10$ بار تکرار با احتمال $1 - \frac{1}{2^{10}}$ یعنی حدود $99\%$ متقاعد شده است که آلیس راست می‌گوید.

یکی از نکات مهم این مساله‌ی ذهنی مخفی بودن جواب سکه از نگاه آلیس است. در این‌جا آلیس که prover و همان ادعا کننده است با هوش فراوان خود هم نمی‌تواند جواب تصادفی سکه را زمانی که پشتش به باب است حدس بزند. حال این سوال ممکن است پیش بیاید که اگر آلیس به غیر از هوش بالا توانایی جادویی هم داشت و می‌توانست با جادو از جواب سکه آگاه شود، آیا باز هم روشی بود که باب توسط آن بتواند متقاعد شود که آلیس یک متقلب نیست؟ به روشنی همین پروتوکول جواب نخواهد داد ولی آیا پروتوکول دیگری برای این هدف وجود دارد؟

## ۲.۳ Probabilistic verifier

برای آن‌که بتوانیم از قابلیت Interaction به خوبی استفاده کنیم، نیاز داریم که تصمیمات verifier را با استفاده از اعداد تصادفی *probabilistic* کنیم. این به آن معناست که سوال‌هایی که verifier از prover می‌پرسد برمبنای سه پارامتر Message history و $x$ که همان ورودی است و $r$ که از یک *Random generator* دریافت می‌شود می‌باشد. این به ما اجازه می‌دهد که همان گونه که برای $P$ یک نسخه‌ی احتمالاتی تعریف کردیم برای $NP$ نیز یک نسخه‌ی احتمالاتی تعریف کنیم. [۵] همچنین به verifier اجازه می‌دهیم که با احتمال بسیار کم نتیجه‌گیری اشتباه کند. به عبارتی اجازه می‌دهیم که تعداد کمی از prover های متقلب با موفقیت تقلب کرده و باعث شوند که verifier یک proof تقلبی را بپذیرد و این احتمال مبتنی بر اعداد تصادفی ساخته شده از Random generator باشد. می‌بایست توجه کنیم که verifier جوری باشد که با احتمال بالا هر proof داده شده برای یک قضیه‌ی غلط را بی‌توجه به این‌که prover چه استراتژی داشته باشد نپذیرد. چراکه با توجه به قدرت محاسباتی بالای prover او می‌تواند از هوش خود استفاده کند و بین همه‌ی استراتژی‌ها، بهترین استراتژی را انتخاب کند. اضافه کردن این قابلیت ها باعث می‌شود که زبان‌های توصیف شده توسط این مدل کلاس بزرگی از مساله‌های سخت، یعنی **PSPACE** را در بر بگیرد.

تفاوت این مدل با مدل deterministic به روشنی احتمالاتی بودن verifier است. پس می‌بایست ابتدا *Probabilistic verifier* را به شکل دقیق تعریف کنیم. تفاوت آن با تعریف قبلی در تعریف تابع $f$ است. تابع $f$ در این مدل باید تصمیمات احتمالاتی بگیرد. برای این کار تابع را به یک مقدار تصادفی وابسته می‌کنیم. یعنی $f$ به شکل $f : \{0,1\}^* \times \{0,1\}^* \times \{0,1,\#\}^* \to \{0,1\}^*$ تعریف می‌شود که در این‌جا پارامتر اول همان ورودی $x$ و پارامتر دوم مقدار تصادفی ساخته شده‌ی $r$ و پارامتر سوم Partial message history است. توجه شود که برای جداسازی $y_i$ ها از هم بهتر است یک جداکننده *(seperator)* داشته باشیم به همین علت کارکتر # را نیز در آن لحاظ کرده‌ایم. روشن است که اگر خروجی تابع $f$ را وابسته به پارامتر $r$ کنیم و $r$ را تصادفی انتخاب کنیم، $f$ یک تابع احتمالی روی متغیر $r$ خواهد بود. در این مدل prover به اعداد تصادفی تولید شده دسترسی ندارد. با احتمالی کردن $f$ در واقعیت Interaction بین $f$ و $g$ یا همان $\langle f,g\rangle(x)$ را تصادفی بر حسب مقدار $r$ کردیم و در نتیجه خروجی $out_f\langle f,g\rangle(x)$ نیز تصادفی خواهد بود. پس آن را مانند منبع [۵] با $out_f\langle f,g\rangle(x,r)$ نشان می‌دهیم تا وابستگی آن به عدد تصادفی $r$ روشن باشد.

برای درک بهتر این مدل، ماشین تورینگ گفت‌وگو کننده‌ی مربوط به آن را طبق منبع [۶] می‌آوریم:

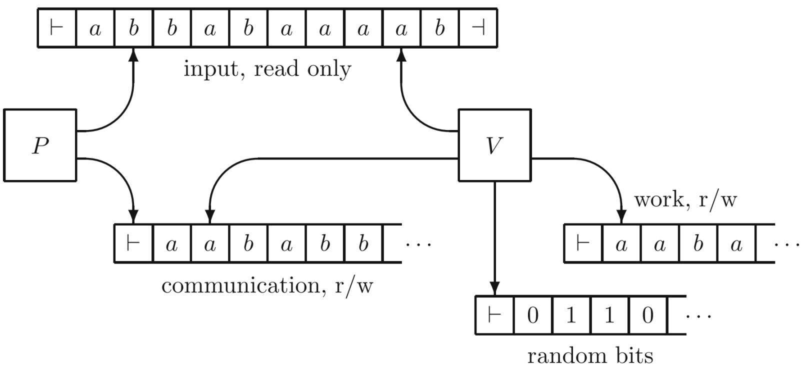


Figure 2: Interactive TMs with probabilistic verifier [6]

توجه به چند نکته در مورد مدل ماشین تورینگی ضروری است. در این مدل نوار بیت‌های تصادفیِ $V$ ، خارج از دسترس $P$ مطابق با تعریف اصلی است. ماشین $P$ در این مدل به زمان و مکان خاصی محدود نیست ولی باید روی هر ورودی حتما بعد از مدتی متوقف شود. از آن‌جا که $P$ توانایی بی‌نهایت دارد برای آن نوار کار [۱۵] نمی‌کشیم. پس از نوشتن پیام روی نوار مبادله‌ی پیام [۱۶] هر بازیکن با تغییر state کنترل را به بازیکن دیگر پاس می‌دهد. [۶] همچنین اندازه‌ی عدد تصادفی تولید شده را نیز چندجمله‌ای نسبت به $|x|$ می‌گیریم.

**تعریف ۳.** **(کلاس *IP*)** *برای هر عدد صحیح $k \geq 1$ (که ممکن است تابعی از $n$ باشد) می‌گوییم زبان $L$ درون $\boldsymbol{IP[k]}$ است، اگر وجود داشته باشد یک ماشین تورینگ احتمالاتیِ [۱۷] چندجمله‌ای با نام $V$ به شکلی که بتواند یک interaction ، $k$ رانده با یک تابع $P : \{0,1\}^* \to \{0,1\}^*$ داشته باشد به شکلی که شرطهای زیر برقرار باشد:*

$$\begin{array}{ll} (Completeness) & x \in L \Rightarrow \exists P : Pr[out_v\langle V,P\rangle(x,r)=1] \geq \frac{3}{4} \\ (Soundness) & x \notin L \Rightarrow \forall P : Pr[out_v\langle V,P\rangle(x,r)=1] \leq \frac{1}{4} \end{array}$$

*به شکلی که $r$ یک عدد تصادفی با اندازه‌ی $poly(|x|)$ است. حال کلاس **IP** را به شکل زیر تعریف می‌کنیم:*

$$\boldsymbol{IP} = \bigcup_{c \geq 1} \boldsymbol{IP}[n^c]$$

*که $n = |x|$ می‌باشد.*

به بیانی، اگر $x \in L$ برقرار باشد آن‌گاه یک prover با نام $P$ پیدا می‌شود که گفت‌وگویِ آن با verifier که آن را با $V$ معرفی کردیم، با احتمال بالا باعث accept این verifier بشود (تمامیت). اما اگر $x \notin L$، آن‌گاه هیچ prover نباید بتواند باعث accept این verifier با احتمال بالا بشود (درستی).

## ۳.۳ $\overline{GISO} \in \mathbf{IP}$

ابتدا مساله‌ی $GISO$ یا همان *graph isomorphism* را بررسی می‌کنیم. می‌گوییم دو گراف $G$ و $H$ باهم ایزومورف هستند اگر نودهایِ $G$ که آن را با $V(G)$ نشان می‌دهیم بتوانند به شکلی برچسب‌گذاری (شماره‌گذاری) شوند که با $H$ برابر شوند. حال داریم:

$$GISO = \{\langle G,H\rangle | G \text{ and } H \text{ are isomorphic graphs}\}$$

به عبارتی اگر یک جایگشت از برچسب‌های $G$ مانند $\pi$ وجود داشته باشد که $\pi(G) = H$ می‌گوییم دو گراف $G$ و $H$ باهم ایزومورف هستند. همچنین منظور از نماد $\pi(G)$ اعمال جایگشت $\pi$ روی برچسب‌های گراف $G$ است. اگر $G$ و $H$ ایزمورف باشند از نوشتار $H \cong G$ برای نشان دادن آن استفاده می‌کنیم. به روشنی $GISO \in NP$ برقرار است. یک certificate برای $H \cong G$ به آسانی همان توصیف $\pi$ می‌باشد

---
[15] Work tape
[16] Communication tape
[17] Probabilistic Turing machine

به شکلی که $H = \pi(G)$ برقرار باشد. $\pi$ می‌تواند در $n \log n$ نمایش داده شود. چراکه هر اندیس در $\log n$ بیت قابل نمایش است و همچنین $n$ اندیس داریم.

اگرچه $GISO$ یک مساله طبیعی به نظر می‌رسد، یکی از مهم‌ترین و سرسخت‌ترین سوال‌های باز در نظریه پیچیدگی محاسبه محسوب می‌شود. با آن‌که به راحتی نشان دادیم که $GISO \in NP$ این‌که آیا این زبان عضو $coNP$ است یا نه، سوالی باز است. به بیانی ما هنوز نمی‌دانیم که برای زبان جفت گراف‌های ناایزومورف آیا می‌توان یک certificate کوتاه (چندجمله‌ای) داد یا نه؟ همچنین در آینده نشان می‌دهیم که اگر $GISO \in NPC$ باشد، *سلسله مراتب چندجمله‌ای فرو می‌ریزد* . [۱۸] این برای محققان نظریه پیچیدگی محاسبه یک مدرک [۱۹] محکم برای $NP$-complete نبودن این زبان محسوب می‌شود. [۷] این مساله و مساله‌ی FACTORING جز مساله‌های خیلی عجیبی در $NP$ به حساب می‌آیند که هنوز عضو $P$ بودن آن‌ها مشخص نشده و همچنین باور دانشمندان علوم کامپیوتر بر $NP$-hard نبودن هر دوی آن‌هاست. [۱]. در حقیقت این مساله به قدری برای دانشمندان علوم کامپیوتر جالب بوده است که کلاس پیچیدگی‌ای به نام آن یعنی کلاس $GI$ تعریف کرده‌اند که شامل تمام مساله‌هایی می‌شود که به $GISO$ قابل کاهش هستند.

حال ما مکمل این مساله، یعنی $\overline{GISO}$ را در نظر می‌گیریم، یعنی:

$$\overline{GISO} = \{\langle G, H\rangle | G \ncong H\}$$

این که آیا $\overline{GISO}$ در $NP$ هست یا نه یک مساله باز است، چرا که هنوز نمی‌دانیم چگونه می‌توان برای این‌که دو گراف ایزومورف نیستند یک certificate چندجمله‌ای داد. ولی می‌توانیم برای آن پروتوکولی بدهیم که prover بتواند توسط آن با احتمال بالایی verifier را متقاعد کند که دو گراف ایزومورف نیستند.

این پروتوکول برای بار اول در منبع [۳] معرفی شد. فرض کنیم دو گراف $G_0$ و $G_1$ داریم. اگر آن‌ها ایزومورف باشند، prover می‌تواند با دادن $\pi$، verifier را متقاعد کند. اما اگر این دو ایزومورف نباشند، چگونه می‌تواند verifier را متقاعد سازد؟ در این پروتوکول verifier به صورت تصادفی $G_0$ یا $G_1$ را انتخاب می‌کند. سپس یک جایگشت $\pi$ تصادفی می‌سازد و آن را روی گراف انتخاب شده اعمال می‌کند. اگر نام گرافی که این جایگشت روی آن اعمال شده است را $H$ بگذاریم، داریم:

$$H := \pi\left(random\left(G_0, G_1\right)\right)$$

حال $H$ را برای prover می‌فرستند. prover باید تشخیص دهد که این جایگشت برای کدام یک از $G_0$ یا $G_1$ است. اگر $G_0 \ncong G_1$ آن‌گاه prover در هر مرحله می‌تواند تشخیص دهد که آیا وجود دارد $\pi$ به شکلی که $H = \pi(G_0)$ یا آیا وجود دارد $\pi$ به شکلی که $H = \pi(G_1)$ باشد؟ چراکه prover دارای توانایی محاسباتی نامتناهی است پس می‌تواند همه‌ی جایگشت‌های ممکن را برای رسیدن به $H$ بررسی کند. اما اگر گراف‌ها ایزومورف باشند، prover نمی‌تواند تشخیص دهد که این جایگشت برای کدام گراف است، چراکه اگر هر دو ایزومورف باشند، این جایگشت به همان احتمال برای $G_0$ است که برای $G_1$، پس حتا با وجود توان محاسباتی بالا، prover مجبور است در این مورد حدس بزند و احتمال درست بودن حدس آن برابر $\frac{1}{2}$ است. پس از تکرار این فرایند به اندازه‌ی 100 بار، احتمال این‌که prover متقلب باشد و در هر صد بار حدس درست زده باشد برابر $\frac{1}{2^{100}}$ و احتمال آن‌که راستگو باشد و به درستی $G_0 \ncong G_1$ باشد بالای 99% است. پس verifier با احتمال بالا متقاعد خواهد شد. در نتیجه $\overline{GISO} \in IP$ می‌باشد.
■

[۱۸] این جمله را در بخشی جداگانه به طور کامل توضیح می‌دهیم.

[19] Evidence

# ۴ Zero-Knowledge Proof (ZKP)

در بخش معرفی، با مثالی دیدِ نسبی به این مبحث ایجاد کردیم. زمانی که دانشجویی می‌خواهد به استادش ثابت کند که قضیه‌ای درست است بدون آنکه هیچ دانشی نسبت به اثبات خود لو بدهد. به عبارتی پروسه‌ای که پس از آن verifier تنها دانشی که به دست آورده درستیِ ادعا است و دیگر هیچ. *(البته در مثال ما، میزان کمی دانش اضافی، یعنی فاکتورهای اعداد بزرگ منتقل شده بودند، پس آن مثال اثبات با دانش صفر محسوب نمی‌شود ولی شهود مناسبی نسبت به این تعریف می‌دهد.)* بین دو مفهوم verification و knowledge تفاوت وجود دارد. به عبارتی می‌توانیم به وسیله‌ی برهم‌کنش و تصادفی‌سازی تشخیص دهیم که ادعای prover درست است بدون این‌که هیچ ایده‌ای نسبت به چگونگی اثبات درستیِ آن داشته باشیم. نشان می‌دهیم که می‌توان دیگران را متقاعد کرد که قضیه‌ای درست است بدون آن که هیچ گونه سرنخی در مورد آن بدهیم. [۸]

چه گفت‌وگویی باعث انتقال دانش اضافی می‌شود؟ می‌توان گفت گفت‌و گویی که خروجی تابعی را به ما می‌دهد که محاسبه‌ی آن برای ما ممکن نیست. در مثال گفته شده، ما خود نمی‌توانستیم فاکتورهای عدد بزرگ RSA را محاسبه کنیم، به همین علت اثباتی که مثال زدیم به معنای واقعی ZK نبود، ولی چون این اطلاعات به طور مستقیم به خود اثبات قضیه‌ی دانشجو مربوط نبود، به ما شهود می‌داد که این امر چگونه ممکن است. برای مثال اگر $A$ برای $B$ عدد تصادفی $n$ بیتی بفرستد، برای او $n$ بیت اطلاعات [20] ارسال کرده است. ولی ما می‌گوییم این $n$ بیت اطلاعات دارای هیچ دانشی نیستند. چراکه $B$ می‌تواند خودش عدد تصادفی تولید کند. [۲]

چنین اثباتی پس از دیدن مثال $\overline{GISO}$ به نظر ممکن می‌رسد. interactive proof برای این که دو گراف ایزومورف نیستند، هیچ دانش اضافه‌ای به verifier منتقل نمی‌کند. توجه کنید که در هر راند verifier جواب درست را خود می‌داند، پس جواب prover چیزی به دانش او اضافه نمی‌کند. به عبارتی verifier می‌تواند چنین گفت‌وگویی را خودش بسازد بی‌آنکه نیازی به برهم‌کنش با prover داشته باشد. چراکه verifier در هر راند هم سوال را می‌داند و هم جواب درست را پس هیچ دانش‌اضافی پس از پروسه اثبات به دست نمی‌آورد ولی توجه شود که با تکرار و درست بودن جوابِ prover در هر راند verifier متقاعد می‌شود که prover راست می‌گوید و گراف‌ها ناایزومورف هستند. پس این اثبات یک ZKP محسوب می‌شود. چراکه اثبات هم با احتمال بالا متقاعد کننده است و هم هیچ دانشی به verifier منتقل نمی‌کند که خود نتواند آن را بسازد.

می‌خواهیم نشان دهیم که همه‌ی اثبات‌ها قابل تبدیل به یک اثبات ZK هستند. برای این‌کار ابتدا نشان می‌دهیم مساله زیر دارای ZKP است:

$$3COL = \{\langle G\rangle | G \text{ is an undirected graph with a legal 3-coloring. }\}$$

می‌دانیم که $3COL \in NP$-complete [۹] می‌باشد (اثبات از طریق کاهش $3SAT$ به این مساله (لینک اثبات) ). پس اگر برای این مساله یک ZKP بدهیم، می‌توانیم از خاصیت complete بودن این مساله استفاده کنیم و هر proof را به یک proof برای مساله $3COL$ ببریم و آن را به صورت ZK اثبات کنیم.

می‌خواهیم برای $3COL$ یک پروتوکول بدهیم که توسط آن بشود این مساله را به صورت ZK اثبات کرد. اثبات معمولی برای این مساله، به راحتیِ همان رنگبندی آن است. حال prover می‌خواهد به verifier نشان دهد که این نمونه از مساله یک ۳ـرنگ بندی دارد بدون آن‌که هیچ دانشی در مورد این رنگ بندی و یا در مورد هرچیز دیگر لو بدهد. پروتوکول باید تضمین کند که اگر برای گراف دلخواه $G$ داشتیم $\langle G\rangle \notin 3COL$ و prover متقلب در پی گول زدن verifier بود، verifier بتواند تقلب او را با احتمال

[20] Information

زیاد رو کند و مچش را بگیرد. ابتدا برای آن‌که پروتوکول را توصیف کنیم، بخشی فرمال از اثبات را با تشبیه به یک فرایند فیزیکی از دید خواننده برای راحتی مخفی می‌کنیم. سپس اشاره می‌کنیم که چگونه مشابه این فرایند فیزیکی توسط رمزنگاری [۲۱] ممکن است.

ابتدا prover ادعا می‌کند که گراف سه رنگ پذیر است. برای این‌که گفته‌ی خود را اثبات کند. برای هر راس یک پاکت نامه در نظر می‌گیرد، رنگ‌ها را درون پاکت قرار داده و روی میز جلوی verifier قرار می‌دهد. فرایند فیزیکی در پاکت کردن و روی میز گذاشته به معنی تعهد prover نسبت به رنگ بندی فعلی روی میز گذاشته است. حال verifier در هر مرحله درخواست می‌کند که دو پاکت که برای دو راس مجاور هستند باز شوند. او با این کار می‌خواهد مچ prover را در صورت تقلب بگیرد. اگر این دو راس مجاور رنگ‌های یکسان در پاکت‌های نامه داشتند، prover متقلب است و verifier به سرعت reject می‌کند. در غیر این صورت باز هم ممکن است prover متقلب و خوش‌شانس بوده باشد. پس با تکرار این فرایند شانس کشف تقلب prover توسط verifier بالا می‌رود. ولی یک مشکل بسیار جدی برای این اثبات وجود دارد. با تکرار این فرایند دانش verifier نسبت به رنگ‌بندی افزایش پیدا می‌کند. به بیانی پس از هر راند به آرامی نحوه‌ی رنگ آمیزی مجاز این گراف برای verifier روشن‌تر می‌شود و این چیزی نیست که verifier با توانایی محاسباتیِ چندجمله‌ای خود بتواند آن را انجام دهد. در نتیجه برای رفع این مشکل prover از این واقعیت استفاده می‌کند که اگر گراف $G$ یک سه رنگ بندی مجاز داشت به شکل خودکار 6 سه‌رنگ بندی متمایز خواهد داشت. به این شکل که برای سه رنگ، شش جایگشت وجود دارد. به این ترتیب پس از هر راند، prover می‌تواند یکی از این 6 جایگشت را به تصادف انتخاب کند و گراف را توسط آن رنگ‌بندی کرده و رنگ‌ها را درون پاکت روی میز بگذارد. به این ترتیب ادعا می‌کنیم که اثبات گفته شده هم یک interactive proof است و هم ZK یا بدون درز دانش می‌باشد. برای این‌که این ادعا را اثبات کنیم، بیایید روی هر راند تمرکز کنیم. از نگاه verifier دو رنگ انتخاب شده همواره دو رنگ تصادفی از سه رنگ به نظر می‌رسد. فرض کنیم دو راس را verifier انتخاب کرده و دو رنگ به دست آورده است. احتمال این‌که این دو رنگ به هر یک از این 6 جایگشت تعلق داشته باشند یکسان است. به این ترتیب از نگاه verifier همواره دو رنگ تصادفی به نظر می‌رسند و این چیزی است که خود verifier بدون کمک prover می‌توانست آن را انجام دهد. او می‌توانست همواره دو عدد تصادفی را از بین سه عدد انتخاب کند و این‌ها از نظر محاسباتی تمایز ناپذیر [۲۲] محسوب می‌شوند. [۸]

بخشی که از توصیف آن به شکل فرمال پرهیز کردیم استفاده از رویه‌ی "تعهد روی یک بیت مخفی" یا همان "committing on a secret bit" می‌باشد. این رویه می‌تواند به شکل فیزیکی به معنای مخفی کردن اطلاعات مانند مثال پاکت نامه باشد یا در رمزنگاری به شکل پیاده‌سازی یک تابع یکسویه [۲۳] صورت می‌گیرد. در واقع توسط این تابع اطمینان میابیم که از داشتن پاکت‌ها نمی‌توان به محتوای درون آن رسید، ولی می‌توان اطمینان داشت که اگر درخواست کردیم که فلان پاکت را برایمان باز کنند، درون آن را با محتوایِ دیگری جابه‌جا نکرده باشند، به عبارتی روی آن متعهد می‌شویم. [۳]

حال می‌توانیم از $NP$-complete بودن مساله $3COL$ برای تبدیل هر اثبات به یک اثبات ZK استفاده کنیم. شاید به نظر خواننده برسد که منظور ما از هر اثبات محدود به دنیای پیچیدگی محاسبه است ولی این طور نیست. اثبات یک دنباله محدود از جملات درست‌ـساخت [۲۴] در زبانی فرمال است که یا اصل [۲۵] بوده یا فرض [۲۶] یا آنکه با استفاده از قواعد استنتاج از قبلی‌ها به دست آمده‌اند به شکلی که verify کردن آن سریع انجام می‌شود. در نتیجه اگر اثباتی برای یک قضیه وجود داشته باشد. برای مثال اگر اثباتی برای حدس گلدباخ داشته باشیم. برای آن یک verifier چندجمله‌ای داریم که می‌تواند درست بودن این

[21] Cryptography
[22] Computationally indistinguishable
[23] One-way function
[24] Well-formed
[25] Axiom
[26] Assumption

اثبات را در زمان چندجمله‌ای تشخیص دهد. پس هر جمله در ریاضیات [27] مانند قضیه‌ی چهار رنگ یا حتا جمله‌های اشتباه مانند این که: *مجموعه اعداد اول متناهی‌اند.* قابل بیان به شکل یک certificate با تعریف $NP$ است. در حقیقت با این گفته‌ها تاکید می‌کنیم $NP$ توانسته به خوبی مفهوم اثبات را برای ما فرمال کند. از قضیه‌ی کوک_لوین یک نتیجه‌ی مهم این است که می‌توانیم از الگوریتم داده شده برای تبدیل یک certificate به certificate یک زبان $NP$-complete استفاده کنیم و این تبدیل در زمان چندجمله‌ای انجام می‌شود. [۱۰] حال از این نتیجه برای اثبات استفاده می‌کنیم. برای هر certificate ابتدا آن را به یک certificate معادل برای زبان $3COL$ تبدیل می‌کنیم. منظور از معادل مطابق با تعریف *Mapping-reduction* می‌باشد. این تبدیل در زمان چندجمله‌ای انجام می‌شود. حال پروتوکول گفته شده برای اثبات ZK را که توضیح داده شد اجرا می‌کنیم. به این ترتیب یک ZKP برای هر اثبات داده‌ایم. پس نتیجه می‌گیریم برای هر اثبات ، یک اثبات ZK یا بدون درز دانش وجود دارد. [۸] البته توجه شود که اثبات‌های $ZK$ مبتنی بر تعریف interactive proof بوده و در نتیجه احتمالاتی‌اند.

البته این ممکن است با شهود خواننده سازگار نباشد. دلیل آن این است که اثبات‌هایی که ما می‌کنیم به طور تمام در زبان فرمال بیان نمی‌شوند و ما از زبان طبیعی برای سادگی و کوتاه کردن اثبات‌های خود بهره می‌بریم. برای مثال جملاتی مانند "*همانطور که می‌دانیم ...*" یا "*به همین ترتیب برای اعداد زوج ...*" یا "*با توجه به قضیه‌ی فلان ...*" این‌ها همه جملاتی هستند که از فرمال نبودن زبان طبیعی برای بیان ساده‌تر یک اثبات از آن‌ها استفاده می‌کنیم. از طرفی، اثبات‌های فرمال [28] اثبات‌هایی هستند که به وسیله‌ی سمبول‌های یک منطق خاص، برای مثال منطق مرتبه‌ی اول و یکسری اصل و فرض و قوانین استنتاج ساخته می‌شوند. پس اگر قرار بود، برای مثال اثبات کوک لوین را، به شکل فرمال بنویسیم، احتمالا تعداد صفحات آن به صورت نمایی بیشتر می‌شد. به بیان دقیق‌تر آن‌چه نشان دادیم تنها برای جملات فرمال برقرار است.

[27] Mathematical statement

[28] Formal proofs

## ۵ سلسله‌مراتب چندجمله‌ای (The polynomial hierrarchy)

### ۱.۵ کلاس PH

اگر $A \in NP$ آن‌گاه می‌توانیم $A$ را به شکل زیر نمایش بدهیم:

$$A = \{x|(\exists y), |y| \leq p(|x|) \text{ and } (x, y) \in B\}$$

برای یک $B \in P$ و یک چندجمله‌ای $p$.

تعریف بالا $NP$ را به وسیله‌ی سور وجودی ($\exists$) و کلاس $P$ تعریف کرده است. به عبارتی بیان دارد که برای $A \in NP$ یک ماشین تورینگ قطعی چندجمله‌ای $M_B$ داریم که $L(M_B) = B$ و ورودی آن یک جفت $x, y$ است که $x$ همان ورودی اصلی و $y$ همان certificate با اندازه‌ی چندجمله‌ای می‌باشد. برای آن‌که شهود بیشتری نسبت به $M_B$ داشته باشیم. می‌دانیم که اگر $A \in NP$ آن‌گاه برای آن یک $NTM$ در زمان چندجمله‌ای با نام $N$ وجود دارد. اجرای یک $NTM$ یک درخت است. حال اگر بخواهیم این ماشین را قطعی و چند جمله‌ای کنیم. یک راه این است که یک پارامتر دیگر با نام $y$ به ماشین شبیه‌ساز آن که $M_B$ است بدهیم و $y$ در واقع دنباله‌ی حدس‌هایی است که $N$ روی ورودی $x$ باید بزند تا به Accept برسد. به عبارتی $y$ شاخه‌ای از محاسبه را نشان می‌دهد که به Accept ختم می‌شود. به این ترتیب ماشین $M_B$ قطعی است، چون در هر شاخه‌ی غیر قطعی به او گفته شده که کدام مسیر را برود و در نتیجه محاسبه‌ی آن زنجیر است، نه درخت. همچنین روشن است که $y$ یا دنباله‌ی حدس‌های موفق، همان certificate است.

حال بیایید $coNP$ را نیز به همین روش تعریف کنیم، اگر $A \in coNP$ آن‌گاه $\overline{A} \in NP$ پس داریم:

$$A = \{x|(\forall y), |y| \leq p(|x|) \text{ and } (x, y) \notin B\}$$

برای یک $B \in P$ و یک چندجمله‌ای $p$ ؛ از آن‌جا که $B \in P$ پس $\overline{B} \in P$ اگر نام $\overline{B}$ را $B'$ بگذاریم داریم:

$$A = \{x|(\forall y), |y| \leq p(|x|) \text{ and } (x, y) \in B'\}$$

به این ترتیب کلاس $coNP$ را بوسیله‌ی سور عمومی ($\forall$) و کلاس $P$ تعریف کردیم.

البته هنوز به تعریف سلسله‌مراتب چندجمله‌ای نرسیدیم، ولی در این نقطه در حد اشاره می‌گوییم که به همین دلیل است که در سلسله مراتب چندجمله‌ای گاهی $NP$ را با اختصار $\exists P$ نشان می‌دهیم و $coNP$ را با اختصار $\forall P$ نشان می‌دهیم.

حال مساله بهینه‌سازی زیر را در نظر بگیریم:

$$MAXCLIQUE := \{\langle G, k\rangle| \text{ The largest clique in G has size exactly } k\}$$

هنوز وجود این مساله در $NP$ روشن نشده است و اگر روشن بود هیچ‌کدام از این سطور و بخشی به نام سلسله مراتب چندجمله‌ای وجود نداشت. ما می‌توانیم برای یک گراف $G$ وجود یک خوشه‌ی [۲۹] با اندازه‌ی $k$ را با certificate چندجمله‌ای verify کنیم ولی چگونه می‌توان برای بزرگترین خوشه بودن در $G$ یک certificate داد؟ همچنین این زبان در $coNP$ نیز ظاهر نشده است. البته می‌توان برای $\langle G, k\rangle \notin MAXCLIQUE$ یک certificate داد تنها در صورتی که یک خوشه با اندازه‌ی بزرگتر از $k$ وجود داشته باشد ولی راه ساده‌ای برای دادن certificate چندجمله‌ای زمانی که $\langle G, k\rangle \notin MAXCLIQUE$ و بزرگترین خوشه‌اندازه‌ی اندازه‌ای کمتر از $k$ داشته باشد به ذهن نمی‌رسد. حال اگر بخواهیم این زبان را به همان فرمی که پیشتر با سور وجودی و سور عمومی توصیف کنیم به چنین فرمی نیاز داریم:

$$A = \{x|(\exists y_1)(\forall y_2)|y_1|, |y_2| \leq p(|x|) \text{ and } (x, y_1, y_2) \in B\}$$

[29] Clique

برای یک زبان $B \in P$ و یک چندجمله‌ای $p$.
برای این زبان به خصوص به این معنی که وجود دارد یک خوشه با اندازه‌ی $k$ به شکلی که برای هر زیرمجموعه از راس‌های $G$ که اندازه‌ی $k+1$ دارد، این زیرمجموعه تشکیل یک خوشه نمی‌دهد. با وجود این‌که هنوز سلسله مراتب چندجمله‌ای تعریف نشده است، با نگاهی که تا به حال داشتیم می‌توان حدس زد که این کلاس را به اختصار با $\exists\forall P$ می‌شناسیم و جالب است اشاره کنیم که به آن $\Sigma_2$ می‌گوییم و $MAXCLIQUE \in \Sigma_2$ می‌باشد. خیلی از مساله‌های بهینه سازی در $\Sigma_2$ می‌باشند.

ایده‌ی استفاده سورها [۳۰] برای توصیف کلاس‌های دیگر پیچیدگی می‌تواند بیش از این بسط پیدا کند. در ریاضیات و علوم کامپیوتر سورها ابزار بسیار مهمی برای توصیف به حساب می‌آیند. سورهایی که یکی در میان جابه‌جا می‌شوند [۳۱] ، مانند آن‌چه در توصیف مساله‌ی $MAXCLIQUE$ داشتیم می‌توانند زبان‌های بسیار پیچیده‌ای را توصیف کنند. یک دنباله‌ی جابه‌جا شونده از سورها مانند زیر:

$$\exists y_1 \forall y_2 \exists y_3 \forall y_4 \ldots$$

می‌توانند به عنوان یک بازی دونفره بیان شود. [۱۱] بازیکن‌ها در نوبت خود حرکت انجام می‌دهند. اگر فرض کنیم بازیکن اول شروع کننده است داریم:

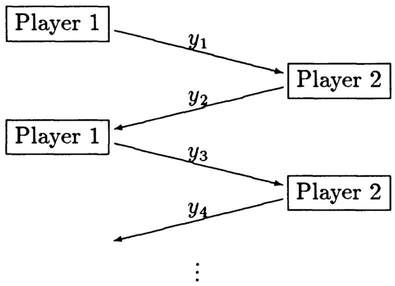


Figure 3: Quantified formula as a Game [11]

بازی به این شکل است که بازیکن اول یک مقداردهی به $y_1$ نسبت می‌دهد و باید آن را به شکلی انجام دهد که برای هر مقداردهی به $y_2$ توسط بازیکن دوم برنده شود، پس از آن‌که بازیکن دو مقداردهی به $y_2$ را انجام داد، بازیکن اول بازهم باید یک مقدار دهی به $y_3$ نسبت دهد به شکلی که به ازای هر مقداردهی ممکن به $y_4$ توسط بازیکن دوم، برنده شود و به همین ترتیب ... . حال می‌گوییم بازیکن اول استراتژی برد دارد اگر که وجود داشته باشد مقدار دهی به $y_1$ به شکلی که برای هر مقداردهی به $y_2$ وجود داشته باشد یک مقداردهی به $y_3$ به شکلی که برای هر مقداردهی به $y_4$ و الی آخر که در نهایت بازیکن اول بتواند موفق شود که مقداردهی پیدا کند که در زبان باشد یا به بیان منطقی مقداردهی باشد که $\phi$ را پس از جایگزینی True کند یا به بیان ماشین تورینگی مقداردهی پیدا کند که با خوراندن آن به ماشین به همراه ورودی $x$ خروجی آن را accept باشد. [۱۱]
.

به همین شکل اگر بخواهیم کلاس $NP$ را با تعریفی که در همین بخش کردیم توصیف کنیم، یعنی ما یک بازیِ تک رانده پارمتری [۳۲] شده با $x$ داریم به شکلی که :

$$x \in A \Leftrightarrow \text{ Player 1 has a winning strategy}$$

[30] Quatifiers
[31] Alternating quantifiers
[32] Prametrized

در این مبحث، بازیکن اول معمولا prover خوانده شده و بازیکن دوم verifier و این دو نام معنا و مفهومی مشابه با آن‌چه در تعریف interactive proof داشتیم دارند. به این معنا که بازیکن اول تلاش می‌کند که به بازیکن دوم ثابت کند که $x \in A$ برقرار است. (توجه شود که $y$ یک وکتور از متغیرهاست.)

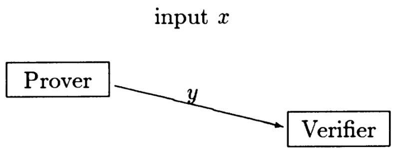


Figure 4: $NP$ as a Game [11]

روی ورودی $x$ اگر $x \in A$ باشد آن‌گاه برای prover یک مقداردهی وجود دارد که با اعمال آن روی وکتور $y$ در اولین حرکت و پس از اعلام این مقداردهی به بازیکن دوم، بازیکن دوم که verifier نام دارد را مجبور به accept می‌کند و در نتیجه بازیکن اول استراتژی برد دارد. در مقابل اگر $x \notin A$ آن‌گاه بی توجه به این‌که بازیکن اول چه مقداردهی برای $y$ انتخاب کند، بازیکن دوم این مقداردهی را قبول نخواهد کرد. در این مورد به خصوص verifier نقش موثری در بازی ندارد و برد و باخت را با چک کردن $(x, y) \in B$ مشخص می‌کند. همین رویه را می‌توان برای بازی $k$ رانده گسترش داد و در نهایت به تعریف بعدی می‌رسیم. [۱۱]

**تعریف ۴.** [۱۱] *زبان $A$ در کلاس $\Sigma_k$ برای $k \geq 0$ است اگر و تنها اگر که وجود داشته باشد زبان $B \in P$ و یک چندجمله‌ای $p$ به شکلی که :*

$$x \in A \Leftrightarrow (\exists y_1)(\forall y_2)\dots(Qy_k)|y_1|,|y_2|,\dots,|y_k| \leq p(|x|) \ and \ (x, y_1, \dots, y_k) \in B$$

*به شکلی که $Q = \forall$ اگر که $k$ زوج باشد و $Q = \exists$ اگر که فرد باشد.*

*به همین ترتیب کلاس $\Pi_k$ تعریف می‌شود با این تفاوت که شروع سورها در آن کلاس با سور عمومی $(\forall)$ بوده و روشن است که جای زوج و فرد در گزاره‌ی آخر نسبت به $Q$ نیز تغییر می‌کند.*

توجه شود که برای حالت خاص $K = 0$ دیگر هیچ $y_i$ و سوری در تعریف وجود ندارد و همچنین چون $B \in P$ برقرار است طبق تعریف داریم:

$$A = \{x|(x) \in B\}$$

و در نتیجه به عبارت زیر می‌رسیم:

$$\Sigma_0 = \Pi_0 = P$$

همچنین روشن است که طبق تعریف‌های اول این بخش $\Sigma_1 = NP$ و $\Pi_1 = coNP$ می‌باشد.

**مشاهده ۱.۵.** *از آن‌جا که $P$ تحت مکمل بسته است و در تعریف‌های داده شده $B \in P$ است، پس داریم:*

$$\Pi_k = co\text{-}\Sigma_k \text{ and } \Sigma_k = co\text{-}\Pi_k$$

تعاریف بالا به ما اجازه می‌دهد که یک عملگر بسیار مفید معرفی کنیم که پیش‌تر اشاره‌ی کوچکی نیز به آن کردیم. دیدیم که کلاس $coNP$ را با $\forall P$ و کلاس $NP$ را با $\Sigma P$ و کلاس $\Sigma_2$ را با $\exists\forall P$ و کلاس $\Pi_2$ را با $\forall\exists P$ نشان دادیم. حال طبق منبع [۱۱] این عملگر را به شکل فرمال تعریف می‌کنیم.

**تعریف ۵.** *برای کلاسی از زبان‌ها مثل $\mathcal{C}$ کلاس زبان $\exists\mathcal{C}$ این‌گونه تعریف می‌شود:*

$$\exists\mathcal{C} = \{x|\exists y, |y| \leq p(|x|) \; and \; (x,y) \in A\}$$

*برای یک $A \in \mathcal{C}$ و یک چندجمله‌ای $p$.*

*به همین ترتیب نیز $\forall\mathcal{C}$ تعریف می‌شود.*

**گزاره ۲.۵.**

$$\begin{array}{clcc}
1 & \Sigma_k & = & \underbrace{\exists\forall\ldots Q}_{k\ times} P \\
2 & \Pi_k & = & \underbrace{\forall\exists\ldots Q}_{k\ times} P \\
3 & \exists\Sigma_k & = & \Sigma_k \\
4 & \forall\Pi_k & = & \Pi_k \\
5 & \exists\Pi_k & = & \Sigma_{k+1} \\
6 & \forall\Sigma_k & = & \Pi_{k+1}
\end{array}$$

گزاره‌های 3 و 4 نیاز به توضیح دارند. توجه شود که در این گزاره‌ها دو سور یکسان پشت سر هم ظاهر می‌شوند. برای مثال در 3 دو سور یکسان به شکل $(\exists y_1)(\exists y_2)$ داریم که $y_1$ و $y_2$ هر دو وکتور هستند. به راحتی می‌توان با ادغام دو وکتور یک وکتور به شکل $y_1' = y_1 y_2$ ساخت و دو سور را تبدیل به یک سور کرد یعنی $\underbrace{(\exists y_1)(\exists y_2)}_{(\exists y_1')}$ خواهد شد. به این ترتیب دو سور پشت سر هم را همواره می‌توان در هم ادغام کرد، به همین ترتیب برای 4 نیز برقرار است.

**مشاهده ۳.۵.**

$$\begin{array}{clcl}
1 & \Sigma_k & \subseteq & \Sigma_{k+1} \\
2 & \Sigma_k & \subseteq & \Pi_{k+1} \\
3 & \Pi_k & \subseteq & \Pi_{k+1} \\
4 & \Pi_k & \subseteq & \Sigma_{k+1}
\end{array}$$

توجه کنید که با اضافه شدن پارامتر به چندتایی‌ها [۳۳] قدرت توصیفی آن‌ها بیشتر می‌شود و نه کمتر. به عبارتی اگر زبانی با دو certificate قابل توصیف است، می‌توانیم certificate سوم را یک dummy certificate بگیریم و روی آن چه سور عمومی $(\forall)$ و چه سور وجودی $(\exists)$ ببندیم فرقی نمی‌کند. به این ترتیب می‌توانیم هر زبان در سلسله مراتب‌های پایین‌تر را شبیه‌سازی کنیم.

[33] Tuples

**مشاهده ۴.۵.**

$$\Sigma_k \cup \Pi_k \subseteq \Sigma_{k+1} \cap \Pi_{k+1}$$

این مشاهده از تعاریف تا اینجا گفته شده به آسانی به دست می‌آید.

برای هر $0 \leq k$ نمودار پایین شمول [34] بین این کلاس‌ها را نشان می‌دهد:

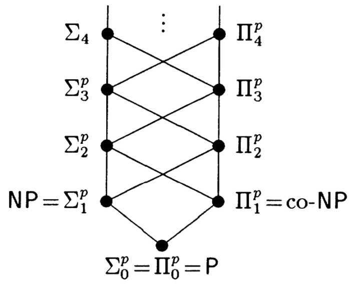


Figure 5: Polynomial hierarchy [11]

**تعریف ۶.**

$$PH = \bigcup_{0 \leq k} \Sigma_k$$

**قضیه ۵.۵.**

$$PH \subseteq PSPACE$$

اثبات آن ساده است. مشاهده کنید که هر زبان عضو $PH$ قابل بیان به شکل یک نمونه مساله‌ی $TQBF$ است. همچنین توجه کنید که تعداد سورها در $TQBF$ وابسته به طول ورودی یعنی $n$ می‌باشد در صورتی که در $PH$ این تعداد برابر یک ثابت $k$ است. در نتیجه $TQBF$ فرم کلی‌تر این زبان‌هاست. در نهایت می‌توان این گونه گفت که چون $TQBF \in PSPACE$ پس هر نمونه با تعداد ثابت از سورهای آن نیز در $PSPACE$ است.
■

**قضیه ۶.۵.** *اگر* $P = NP$ *آن‌گاه* $PH = P$

اثبات استقرایی ولی ساده است. اگر $P = NP$ پس می‌توانیم از سورهای وجودی ($\exists$) صرف نظر کنیم. همچنین از آن‌جا که $P$ نسبت به مکمل بسته است، با این فرض $NP = coNP$ خواهد بود و در نتیجه می‌توانیم از سورهای عمومی ($\forall$) نیز صرف نظر کنیم. البته این کار با تعریف استقرایی گزاره شماره‌ی دو انجام می‌شود. همچنین در این اثبات از گزاره‌ی شماره‌ی دو، برای ادغام سورهای شبیه به هم استفاده خواهد شد. به این ترتیب هر زبان در $PH$ طبق تعریف در $P$ می‌افتند.
■

هنوز نمی‌دانیم که آیا هیچ‌کدام از این شمول‌ها، به صورت شمول محض [35] هستند یا خیر، ولی اگر $\Sigma_k \neq \Sigma_{k+1}$ برای یک $k$ برقرار باشد، آن‌گاه داریم:

$$\Sigma_0 \neq \Sigma_1 \neq \ldots \neq \Sigma_k \neq \Sigma_{k+1}$$

[34]Inclusion
[35]Proper

همچنین با در نظر گرفتن عکس نقیض [36] آن اگر $\Sigma_i = \Sigma_{i+1}$ با استقرا داریم، داریم:

$$\Sigma_i = \Sigma_{i+1} = \Sigma_{i+2} = \dots$$

**گزاره ۷.۵.** *برای هر $i \leq 1$ جمله‌های زیر هم‌ارزند:*

$$\begin{array}{llcl}
1 & \Sigma_i & = & \Sigma_{i+1} \\
2 & \Pi_i & = & \Pi_{i+1} \\
3 & \Sigma_i & = & \Pi_i \\
4 & \Sigma_i & = & \Pi_{i+1} \\
5 & \Pi_i & = & \Sigma_{i+1} \\
6 & PH & = & \Sigma_i
\end{array}$$

اثبات: روشن است که گزاره‌ی 6 گزاره‌های 1 و 2 و 4 و 5 را نتیجه می‌دهد. البته از 6 به 3 رفتن هم ساده است ولی برای نمونه آن را بیشتر توضیح می‌دهیم. برابری گزاره‌های 1 و 2 نیز مانند 4 و 5 با استفاده از مکمل‌گیری به دست می‌آید. این‌که 1 نتیجه می‌دهد 6 را نیز از روی متن بالای این گزاره نتیجه می‌شود.

می‌خواهیم از 6 به 3 برویم. مشاهده کنید که $PH = co\text{-}PH$ می‌باشد یا به بیانی $PH$ تحت مکمل بسته است. توجه شود که منظورمان یک طبقه‌ی خاص نیست بل‌که تمام $PH$ است. همچنین برای هر $i$ عبارت $\Sigma_i \subseteq \Pi_{i+1}$ و $\Pi_i \subseteq \Sigma_{i+1}$ برقرار است.

می‌خواهیم از 1 به 3 برسیم. می‌دانیم که $\Sigma_i \subseteq \Pi_{i+1} = co\text{-}\Sigma i + 1 = co\text{-}\Sigma_i = \Pi_i$. شمول عکس $\Sigma_i \subseteq \Pi_{i+1}$ نیز با استفاده از مکمل گیری به دست می‌آید.

می خواهیم از 3 به 4 برویم. این در نتیجه‌ی $\Sigma_i = \Pi_i = \forall\Pi_i = \forall\Sigma_i = \Pi_{i+1}$ می‌باشد.
می‌خواهیم از 4 به 1 برسیم. این در نتیجه‌ی $\Sigma_{i+1} = co\text{-}\Pi_{i+1} = co\text{-}\Sigma_i = \Pi_i \subseteq \Pi_{i+1} = \Sigma_i$ می‌باشد. [۱۱]
■

اگر هر یک از جملات بالا به نحوی نشان داده شود، به اسطلاح می‌گوییم که سلسله مراتب چند جمله‌ای فرو ریخته است [37] ، در نظریه پیچیدگی از این جمله استفاده‌ی فروان می‌شود. برای مثال می‌گویند اگر $FACTORING \in NP$-complete آن‌گاه سلسله مراتب چند جمله‌ای فرو می‌ریزد. فرو ریختن سلسله مراتب چندجمله‌ای از نگاه متخصصان پیچیدگی محاسبه و منطق بسیار نامحتمل به نظر می‌رسد و اگر فرضی به چنین چیزی ختم شود معمولا درست بودن آن فرض را بسیار نامحتمل می‌پندارند. از آن‌جا که باور عمومی بر آن است که که همه‌ی طبقات سلسله مراتب رابطه‌ی شمول محض دارند.

چرا چنین باوری بین آن‌ها مشترک است؟ این امکان وجود دارد که این باور را گسترش باور دانشمندان پیچیدگی محاسبه به $P \neq NP$ دانست. می‌دانیم که $SAT \in NP$-complete می‌باشد و می‌دانیم که $TQBF \in PSPACE$-complete می‌باشد. اثبات $complete$ بودن $TQBF$ از اثبات $complete$ بودن $SAT$ در درون خود بهره می‌برد که نتیجه‌ی کار کوک_لوین است. حال برای هر طبقه از $PH$ نیز ما به طور مشابه نوع خاصی از $TQBF$ را که با $TQBF_k$ معرفی‌اش می‌کنیم در نظر می‌گیریم. در این نسخه تعداد سورها محدود و برابر $k$ است. به بیانی $TQBF_k = (\exists y_1)(\forall y_2)\dots(Qy_k), \phi(y_1, \dots, y_k)$ زبان complete برای هر طبقه است. اثبات کامل بودن آن نیز همانطور که قابل حدس است به سادگی

[36] Contrapositive

[37] The polynomial hierrarchy collapses

نتیجه‌ای از قضیه کوک_لوین است. فقط توجه شود که مانند پیش هر $y_i$ یک وکتور از متغیرهاست و با $TQBF$ اصلی که روی هر تک متغیر سورها جابه‌جا می‌شوند کمی تفاوت دارد. فرض کنیم که $PH$ فرو بریزد [۳۸]، آن‌گاه برای هر $n \leq n_0$ داریم $\Pi_n, \Sigma_n \subset \Sigma_{n_0}$ به این ترتیب $TQBF_{n_0}$ برای تمام $PH$، complete است. برای جهت مخالف اگر $PH$ یک مساله‌ی complete داشته باشد آن‌گاه تعداد طبقات آن محدود و برابر $n_0$ می‌شود. به عبارتی تمام طبقات بالاتر در طبقه‌ی $n_0$ فرو می‌ریزند. به این ترتیب $PH$ فرو می‌ریزد اگر و تنها اگر که یک مساله complete برای آن تعریف شود (*حتا $TQBF$ عمومی نمی‌تواند برای کل $PH$ کامل باشد.*) این قضیه برای دانشمندان ناباورانه است همانطور که $NP \subset P$ برای آن‌ها ناباورانه است. برای نمونه در مثال $MAXCLIQUE$ دیدیم که هنوز نمی‌دانیم آیا می‌توان برای یک فرمول $\Sigma_2$ به شکل $(\exists y_1)(\forall y_2), \phi(y_1, y_2)$ یک مقداردهی True کننده به شکل کارا پیدا کنیم که نیازی به گشتن در بین همه‌ی مقداردهی‌های ممکن نباشد. این استدلال برای رفتن از هر طبقه‌ای به طبقه‌ی بالاتر پیش می‌آید. منطق‌دانان نیز بر این عقیده اند که نمی‌توان هر فرمول سور دار را به فرمولی با تعداد محدود سور تبدیل کرد و با افزایش تعداد سورها واقعا توانایی توصیف بیشتری به ما داده می‌شود. همچنین در نظر داشته باشیم که اگر به هر نحوی نشان داده شود که $PSPACE \subset PH$ آن‌گاه با نتیجه‌های قبل داریم $PH = PSPACE$ آن‌گاه $TQBF$ در یک طبقه‌ی محدود در $PH$ قرار می‌گیرد و از آن‌جا که هر مساله تمام در هر طبقه قابل تبدیل به $TQBF$ است $PH$ فرو می‌ریزد و اتفاق عجیبی می‌افتند. به این شکل که از آن‌جا که $TQBF$ در طبقه‌ی محدود $n_0$ است پس $TQBF = TQBF_{n_0}$ یعنی دیگر نیازی نیست که تعداد سورها وابسته به اندازه‌ی ورودی باشد. از طرف دیگر اگر بدون این فرض $PH$ فرو بریزد، می‌تواند مدرکی بر این باشد که $PH = PSPACE$ چراکه اگر این‌طور نباشد این اتفاق عجیب می‌افتد که $TQBF = TQBF_{n_0}$ خواهد بود. هرچند توضیح این‌که چرا باور عمومی بر این است و این‌که باورها مبتنی بر مدرکند نه اثبات سخت است، ولی به نظر می‌رسد که این دلیل‌ها دلایل قوی برای حدس فرو نریختن $PH$ باشد.

برای مثال دیاگرام زیر نشان می‌دهد که اگر فرض ناباورانه‌ی فروریختن سلسله مراتب در یک طبقه‌ی خاص، مانند طبقه‌ی سوم اتفاق بیفتد، نمودار شمول کلاس‌های $PH$ به چه شکل می‌شود:

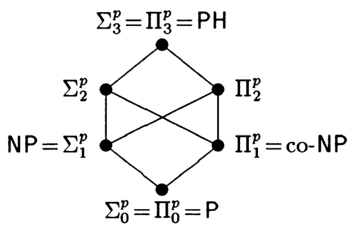


Figure 6: PH collapses to $\Sigma_3$ (*inclusion diagram*) [11]

**قضیه ۸.۵.** *اگر $NP = coNP$ آن‌گاه* $PH = \Sigma_1 = NP = coNP$

اثبات: می‌خواهیم نشان دهیم که $\Sigma_i = \Sigma_1$ برای هر $i$ برقرار است. اثبات با استقراست. پایه همان $i = 1$ است که بدیهی است. فرض کنیم برای یک $i$ بزرگتر از 1 برقرار است. یک زبان $L \in \Sigma_{i+1}$ را در نظر بگیریم. بر اساس تعریف یک ماشین تورینگ چندجمله‌ای با ورودی یک چندتایی اندازه‌ی $i + 2$ وجود دارد (یاداوری: ورودی اول همان $x$ بود) که نام آن $M_B$ است (نام‌گذاری براساس زبان $B$) می‌دانیم که $x \in L$ اگر و تنها اگر که $(\exists y_1)(\forall y_2) \ldots M_B(x, y_1, \ldots, y_{i+1}) = 1$ باشد. حال زبان $L' = \{(x, y)|(\forall y_2)(\exists y_3) \ldots M_B(x, y, y_2, \ldots, y_{i+1}) = 1\}$ را در نظر بگیریم. روشن است که

[38] Collapses

$L' \in \Pi_i$ برقرار است. براساس فرض استقرا داریم $\Pi_i = co\text{-}\Sigma_i = co\text{-}\Sigma_1 = \Pi_1$. از طرفی براساس فرض داریم $NP = coNP$ یعنی $\Sigma_1 = \Pi_1$ و در نتیجه $L' \in \Sigma_1$ خواهد بود. حال مشاهده کنید که $x \in L$ اگر و تنها اگر که $\exists y(x, y) \in L'$ و از آن‌جا که $L' \in \Sigma_1$ پس یک ماشین تورینگ چندجمله‌ای با نام $M_{B'}$ برای آن وجود دارد به شکلی که $w \in L'$ برقرار است اگر و تنها اگر که $(\exists z) M_{B'}(w, z) = 1$ برقرار باشد. پس $x \in L$ اگر و تنها اگر که $(\exists (y, z)) M_{B'}((x, y), z) = 1$ برقرار باشد. توجه شود که دو وکتور $x$ و $y$ را ادغام کردیم. در نتیجه $L \in \Sigma_1$ برقرار است.

■

توجه شود که براساس روش قضیه‌ی بالا می‌توان در گزاره‌ی قبلی از 3 به 6 رفت.

البته $P \neq NP$ هنوز اثبات نشده است ولی پذیرش این حدس که به حدس کوک هم معروف است، در بین دانشندان علوم کامپیوتر رایج است. حال می‌خواهیم این حدس را به ساختاری که تعریف کردیم گسترش دهیم. برای مثال در مرحله‌ی بعدی فرضی قوی‌تر داریم که $NP \neq coNP$ می‌باشد. همچنین کلاس‌های $PH$ به امکان می‌دهند که فرض‌های قوی و قوی‌تر را برای شناخت بهتر $NP$ وارد فرضیات خود بکنیم. این کار برای آنالیز ساختار درونی $NP$ توصیه می‌شود. [۱۱]

$$\begin{array}{cl}
\boxed{\mathsf{PH} \neq \Sigma_k^p \text{ for all } k} & \\
\Downarrow & \\
\vdots & \\
\Downarrow & \\
\boxed{\mathsf{PH} \neq \Sigma_3^p} & \\
\Downarrow & \\
\boxed{\mathsf{PH} \neq \Sigma_2^p} & \\
\Downarrow & \\
\boxed{\mathsf{PH} \neq \Sigma_1^p} & (\mathsf{NP} \neq \text{co-}\mathsf{NP}) \\
\Downarrow & \\
\boxed{\mathsf{PH} \neq \Sigma_0^p} & (\mathsf{P} \neq \mathsf{NP})
\end{array}$$

Figure 7: Cook’s hypothesis and beyond [11]

توجه شود که $PH \neq \Sigma_0$ هم ارز است با فرض $P \neq NP$ و $PH \neq \Sigma_1$ هم ارزست با $NP \neq co\text{-}NP$. برخی از نتایج نیازمند حدس‌های قوی‌تری نسبت به $P \neq NP$ می‌باشند. برای نمونه نتیجه‌ای که در بخش بعدی به آن خواهیم پرداخت و مرتبط با interactive proofs است. این نتیجه که اگر $PH$ در $\Sigma_2$ فرو نریزد $(PH \neq \Sigma_2)$ آن‌گاه $GISO \notin NP$-complete برقرار است. [۱۱]

## ۲.۵ Alternating TM

یک نمونه‌ی جالب از ماشین‌های تورینگ، *Alternating TM* ها هستند. این ماشین نوعی تغییر یافته از $NTM$ است. به یاد داریم که بخش پیش $NP$ را با سور وجودی $(\exists)$ تعریف کردیم. تکرار عملگر منطقیِ $\vee$ درون سور وجودی $(\exists)$ وجود دارد. تعریف پذیرش رشته در $NTM$ هم مشابه بود. اینکه حداقل یک شاخه‌یِ محاسبه accept پیدا شود. به بیانی در هر گره از درخت محاسبه، آن گره یک گیت منطقی $\vee$ را محاسبه می‌کند. اما این ماشین‌ها که به اختصار به آن‌ها ATM می‌گوییم، به ما کمک می‌کنند

تا علاوه بر سور وجودی ($\exists$) بتوانیم سور عمومی ($\forall$) را نیز توسط ماشین تورینگ تعریف کنیم. در واقع $ATM$ به ما کمک می‌کند که بتوانیم زبان‌هایی را توصیف کنیم که حتا برای آن‌ها یک certificate کوتاه هم وجود ندارد و در نتیجه نمی‌توانند به تنهایی توسط $NTM$ تعریف شوند. [۱۰]

این ماشین‌ها مانند $NTM$ ها هستند با این تفاوت هر حالت داخلی ماشین در آن، به غیر از دو حالت accept و reject دارای یکی از برچسب‌های $\exists$ یا $\forall$ هستند. مشاهده شود که در $NTM$ هر حالت یک حالت وجودی ($\exists$) بود. به این معنا که برای هر نود، می‌گوییم آن نود یک نود رونده به accept است اگر یکی از شاخه‌های آن به accept برسد. به همین ترتیب می‌توان گفت یک نود، نود رونده به accept است اگر فرزند اول به accept برسد **یا** فرزند دوم به accept برسد **یا** ... به همین ترتیب برای همه‌ی فرزاندن عملگر $\vee$ محاسبه می‌شود. همین تعریف برای نودهایی هست که در کانفیگوریشن معادل آن‌ها حالت روی حالت $\exists$ می‌باشد. برای نودهای با کانفیگوریشن $\forall$ هم به همین ترتیب فقط گیت $\wedge$ را محاسبه می‌کنیم. به این شکل که یک نود در درخت محاسبه یک نود رونده به accept است اگر همه‌ی فرزندان آن نود رونده به accept باشند. به همین شکل در درخت محاسبه بین نودهای وجودی ($\exists$) و نودهای عمومی ($\forall$) جابه‌جا می‌شویم. [۱۰]

**تعریف ۷.** *(Alternating time) برای هر $T : \mathbb{N} \to \mathbb{N}$ می‌گوییم یک $ATM$ با نام $M$ در زمان $T(n)$ روی هر ورودی $x \in \{0,1\}^*$ اجرا می‌شود، اگرکه اندازه‌ی بزرگترین شاخه‌ی محاسبه‌ی آن با تعریف محاسبه در $ATM$ کمتر از $T(|x|)$ باشد.*

*می‌گوییم $A \in ATIME(T(n))$ اگر که ثابت $c$ و یک $ATM$ با نام $M$ در زمان $cT(n)$ وجود داشته باشد که برای هر ورودی $x \in \{0,1\}^*$ ماشین $M$ روی ورودی $x$ به حالت $accept$ برود اگر و تنها اگر که $x \in L$ برقرار باشد.*

همچنین می‌خواهیم بتوانیم نوعی از $ATM$ ها را تعریف کنیم که توسط آن بشود $PH$ را به شکل خاص توصیف کرد. برای این‌کار باید تعداد جابه‌جایی بین سورهای عمومی ($\forall$) و سورهای وجودی ($\exists$) به مقدار یک ثابت $k$ باشد.

**تعریف ۸.** *برای هر $i \in \mathbb{N}$ ما $\Sigma_i TIME(T(n))$ را این‌گونه تعریف می‌کنیم که زبان‌هایی را در بر بگیرد که برای آن‌ها یک $ATM$ در زمان $T(n)$ وجود داشته باشد به شکلی که حالت اولیه [۳۹] آن یک حالت وجودی ($\exists$) باشد و روی هر ورودی در گراف کانفیگوریشن با تعریف $ATM$ هر مسیر شروع شونده از initial configuration بیشترین تعداد جابه‌جایی سورها کمتر از $i - 1$ باشد.*

*به همین ترتیب $\Pi_i TIME(T(n))$ تعریف می‌شود با این تفاوت که حالت اولیه می‌بایست از نوع $\forall$ باشد.*

**مشاهده ۹.۵.** *برای هر $i \in \mathbb{N}$ داریم:*

$$\Sigma_i = \bigcup_{0 \leq c} \Sigma_i Time(n^c)$$
$$\Pi_i = \bigcup_{0 \leq c} \Pi_i Time(n^c)$$

[39] Initial state

این مشاهده به راحتی از تعریفهایی که برای $\Sigma_i$ و $\Pi_i$ داشتیم نتیجه می‌شود. در واقع هم از تعریف جدید می‌توان به تعریف قبلی رسید و هم از تعریف قبلی به تعریف جدید. بخش مهم آن این است که نشان دهیم برای یک $TQBF_i$ یک $ATM$ محدود شده به $i$ جابه‌جایی سور داریم.

جالب است که بررسی کنیم اگر چنین محدودیتی نگذاریم و بتوانیم به تعداد دلخواه بین سورها جابه‌جا شویم چه اتفاقی خواهد افتاد. در ادامه کلاس $AP$ را بدون چنین محدودیتی تعریف می‌کنیم.

**تعریف ۹.**

$$AP = \bigcup_{0 \leq c} ATIME(T(n^c))$$

**قضیه ۱۰.۵.** $AP = PSPACE$

ایده ثبات: به سادگی $TQBF \in AP$ کافیست $ATM$ آن را این‌گونه در نظر بگیریم که برای سورهای وجودی ($\exists$) از حالت $\exists$ استفاده کنیم و برای سورهای عمومی ($\forall$) از حالت $\forall$ استفاده کنیم. پس از آن جا که $TQBF \in PSPACE$-complete نتیجه می‌گیریم که $PSPACE \subseteq AP$ برقرار است. برای $AP \subseteq PSPACE$ نیز از همان روش بازگشتی استفاده می‌کنیم که برای نشان دادن $TQBF \in PSPACE$ استفاده کردیم. [۱۰]
■

## ۶ بازی آرتور– مرلین

برای تعاریف این بخش از منبع [۴] استفاده می‌کنیم. یک proof system به فرم بازی آرتور مرلین مانند یک interactive proof system هست با این تفاوت که مرلین که در این‌جا همان prover است بسیار قدرتمندتر از prover های آن تعریف است. به این شکل که او می‌تواند تمام تاریخچه‌ی محاسبه [۴۰] آرتور و همچنین بیت‌های رندوم ساخته شده توسط او را ببیند. گویی مرلین که از نام آن بر می‌آید توانایی‌های جادویی دارد.

در مثال $\overline{GISO}$ برای نمونه، مخفی کردن بیت‌های تصادفی نقشی کلیدی را در پروتکول بازی می‌کرد و رو بازی کردن را به نظر ناممکن می‌ساخت. با توانایی بالای مرلین و دسترسی او به تاریخچه‌ی محاسبه آرتور او تنها با داشتن بیت‌های تصادفی می‌تواند همه‌ی رفتار آرتور را شبیه‌سازی کند. پس رفتار آرتور اینجا به نظر کم تاثیر و تنها در اندازه‌ی ساخت بیت‌های تصادفی و قبول یا رد رشته در انتهای بازی می‌باشد.

به این ترتیب می‌توانیم ماشین آرتور را که $M_A$ می‌شناسیم این گونه توصیف کنیم که هر سوال آن تنها بیت‌های تصادفی بوده و هر عدد تصادفی تولید شده را یک راند از بازی حساب می‌کنیم. برای تعریف دقیق باید نقش مرلین را در قالب یک اوراکل [۴۱] در نظر داشته باشیم.

**تعریف ۱۰.** *می‌گوییم که زبان $L$ یک proof system آرتور– مرلین با احتمال خطای $\epsilon$ دارد اگر وجود داشته باشد یک ماشین آرتور با نام $M_A$ به شکلی که شرایط زیر برقرار باشد:*

*وجود داشته باشد اوراکل با نام $F_m$ به شکلی که* $\quad (\forall x) x \in L \Rightarrow Pr[M_A^{F_M}(x) = 1] \geq 1 - \epsilon$

*برای هر اوراکل با نام $F_m$ داشته باشیم* $\quad (\forall x) x \notin L \Rightarrow Pr[M_A^{F_M}(x) = 1] \leq \epsilon$

این که اولین پیام را آرتور بدهد یا مرلین دو کلاس متفاوت $AM$ و $MA$ را ایجاد می‌کند.

**تعریف ۱۱.** *کلاس زبان‌های $AM$ شامل زبان‌هایی می‌شود که یک proof system آرتور– مرلین با خطای کمتر از $\frac{1}{4}$ برای آن وجود دارد و اولین پیام را آرتور ارسال می‌کند.*

*به همین ترتیب $MA$ تعریف می‌شود با این تفاوت که مرلین گفت‌وگو را شروع می‌کند.*

برای هر تابع $r(n)$ کلاس $AM_{r(n)}$ کلاس از زبان‌هاست که برای آن‌ها یک proof system آرتور– مرلین با احتمال خطای کمتر از $\frac{1}{4}$ وجود دارد به شکلی که تعداد راندهای بازی برابر $r(n)$ است و آرتور اولین پیام را می‌دهد. به همین ترتیب تعریف $MA_{r(n)}$ را داریم که در آن مرلین شروع می‌کند.

از نگاه آرتور که خود یک ماشین تورینگ احتمالاتی [۴۲] چندجمله‌ای است او به یک اوراکل غیرقابل اعتماد دسترسی دارد که می‌خواهد آرتور را متقاعد کند که $x \in L$ می‌باشد. همچنین توجه شود که از آن‌جا که مرلین بسیار قوی است و به همه چیز آرتور دسترسی دارد می‌تواند به راحتی رفتار او را شبیه‌سازی کرده و به شکل بهینه بازی کند.

[40] Computation history
[41] Oracle
[42] Probabilistic TM

از نگاه مرلین مدل جالب‌تری قابل توصیف است. مرلین می‌خواهد آرتور را که یک verifier احتمالاتیِ چندجمله‌ای است متقاعد کند که $x \in L$ می‌باشد. برای این‌کار او به آرتور یک proof احتمالاتی بهینه تحویل می‌دهد که با استفاده از random generator مشترکشان ساخته شده است. به عبارتی مرلین proof را با بازی جای دو نفرشان می‌سازد. او به شکل غیرقطعی پیام خودش را ساخته و پیام آرتور را با توجه به عدد تصادفی فعلی می‌سازد. به این ترتیب کل گفت‌وگو را تولید می‌کند. در واقع ماشین او نوعی تغییر یافته از $NTM$ است که در برخی از نودها احتمالاتی است و برخی از نودها غیرقطعی. این ماشین ساختار مشابه با $ATM$ دارد با این تفاوت جای حالتهای $\wedge$ این‌جا حالت‌ها احتمالاتی و وابسته به عدد تصادفی داریم. برای تعریف دقیق باید این ماشین را که probabilistic NTM نام دارد تعریف دقیق کنیم. [۴]

برای تعریف دقیق آن از منبع [۱۲] استفاده می‌کنیم.

**تعریف ۱۲.** *یک $probabilistic\ NTM$ یا به اختصار $PNTM$ یک ماشین تغییر یافته از نوع $NTM$ است که دو نوع حالت غیر قطعی دارد، حالت‌های $random$ و حالت‌های $guess$. حالت‌های $guess$ مانند حالت‌های $\exists$ در $ATM$ تعریف می‌شوند و در حالت‌های $random$ انتخاب کانفیگوریشن بعدی از بین کانفیگوریشن‌های ممکن بعدی بر پایه‌ی عدد تصادفی تولید شده انجام می‌شود.*

برای کانفیگوریشن دلخواه $c$ احتمال accept را با نماد $p(c)$ نشان می‌دهیم و به این شکل است که اگر $c = \text{Accepting configuration}$ آن‌گاه $p(c) = 1$ و اگر $c = \text{Rejecting configuration}$ آن‌گاه $p(c) = 0$ و اگر $c$ یک حالت deterministic بود، یعنی تنها از آن می‌شد به یک کانفیگوریشن بعدی رفت. اگر نام configuration بعدی آن را $c'$ بگذاریم آن‌گاه $p(c) = p(c')$ و اگر $c$ یک کانفیگوریشن با حالت random بود آن‌گاه می‌دانیم که کانفیگوریشن بعدی آن وابسته به عدد تصادفی تولید شده است. در این صورت به ازای هر مقدار تصادفی ممکن $x$ استنتاج $c \vdash c_x$ یک استنتاج قطعی است. به عبارتی گویی ما مقدار تولید شده‌ی تصادفی را تعیین شده گرفتیم. حال $p(c) = \text{Average}(p(c_x))$ برای هر مقدار ممکن برای $x$ خواهد بود. اگر $c$ یک کانفیگوریشن با حالت guess بود، مرلین که توانایی بی‌انتها دارد، بهترین حدس را انتخاب می‌کند، پس اگر حدس $x$ را تعیین شده در نظر بگیریم حال $c \vdash c_x$ یک استنتاج قطعی است و $p(c) = Max\{p(c_x)\}$ برای هر مقدار ممکن برای $x$ خواهد بود. روشن است که $Pr[N(w) = 1] = p(c_{start})$ به شکلی که $c_{start}$ کانفیگوریشن اولیه است.

**تعریف ۱۳.** [۴] *می‌گوییم یک $PNTM$ یک ماشین مرلین است اگر که برای هر ورودی $x$ پس از $Poly(|x|)$ حرکت متوقف شود.*

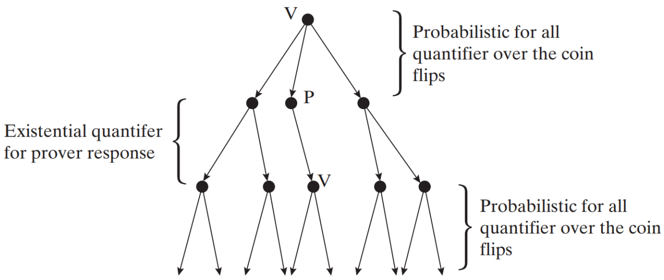


Figure 8: PNTM [10]

ما می‌توانیم مفهوم راند را در ماشین‌های مرلین تعریف کنیم. random round یک محاسبه از ماشین مرلین با نام $M$ ، طولانی ترین دنباله‌ی ممکن از کانفیگوریشن‌هایی است که یا deterministic هستند یا random . به همین ترتیب nondeterministic round یک محاسبه از ماشین $M$ برابر است با طولانی ترین دنباله‌ی ممکن از کانفیگوریشن‌هایی که یا deterministic هستند یا nondeterministic می‌باشند. برای $k \leq 1$ یک ماشین مرلین با $k$ راند ماشینی است که دارای حداکثر $k$، random round و nondeterministic round است.

**تعریف ۱۴.** *می‌گوییم $L \in AM$ اگر ماشین مرلینی با نام $M$ وجود داشته باشد به شکلی که*

$$(\forall x) x \in L \Rightarrow Pr[M(x) = 1] \geq \tfrac{3}{4}$$
$$(\forall x) x \notin L \Rightarrow Pr[M(x) = 1] \leq \tfrac{1}{4}$$

اثبات برابری دو تعریف: یادآوری می‌کنیم که ماشین آرتور یک verifier چندجمله‌ای احتمالاتی است که به یک اوراکل غیرقابل اعتماد دسترسی دارد. این اوراکل می‌خواهد اورا متقاعد کند که $x \in L$ می‌باشد. اگر نام تابع این اوراکل را $F$ بگیریم و نام این ماشین آرتور را $M_A$ بگذاریم، آن‌گاه $F$ می‌خواهد $Pr[M_A^F(x) = 1]$ را Maximize کند.

نشان می‌دهیم که چگونه می‌توان یک ماشین آرتور با نام $M_A$ را به یک ماشین مرلین با نام $M_M$ تبدیل کنیم. هر query آرتور به اوراکل را تبدیل می‌کنیم به دو مرحله پشت سرهم به شکلی که در مرحله‌ی اول یک دنباله‌ی عدد تصادفی تولید می‌کنیم و در مرحله‌ی پس از آن یک دنباله‌ی بیت‌های غیرقطعی تولید می‌کنیم. اگر $F$ تابعی باشد که جواب هر query را به $M_A$ برمی‌گرداند و می‌خواهد احتمال accept را Maximize کند، آن‌گاه داریم

$$(\forall x) Pr[M_A^F(x) = 1] = Pr[M_M(x) = 1]$$

و همچنین $Pr[M_A^F(x) = 1] \geq Pr[M_A^g(x) = 1]$ برای هر اوراکل $g$ برقرار است. براساس تعریف اگر $(M_A, g)$ یک proof system آرتور-مرلین برای $L$ باشد که خطای $\frac{1}{4}$ دارد، آن‌گاه $M_M$ دو شرط تعریف بالا را برآورده می‌کند.

برای جهت خلاف آن فرض کنید، ماشین مرلین با نام $M_M$ وجود دارد که دو شرط درستی و تمامیت را دارد. حال می‌خواهیم $M_M$ را به یک ماشین آرتور با نام $M_A$ تبدیل کنیم. برای این‌کار هر حرکت random در $M_M$ را به یک query از سمت آرتور در $M_A$ تبدیل می‌کنیم و هر حرکت nondeterministic را به جواب این query تبدیل می‌کنیم. مشاهده شود که $Pr[M_M(x) = 1] = Pr[M_A^F(x) = 1]$ اگر $F$ یک اوراکل بهینه [۴۳] باشد برقرار است. به این ترتیب $(M_A, F)$ برای $L$ یک proof system آرتور-مرلین تشکیل می‌دهد. [۴]

■

**قضیه ۱.۶.**

$$\overline{GISO} \in AM[2]$$

ایده اثبات: مجموعه گراف‌های برچسب دار $S = \{H | H \cong G_1 \text{ or } H \cong G_2\}$ را در نظر بگیرید. به سادگی می‌توان دید که $H \in S$ در صورتی که بتواند یک جایگشت ممکن اعمال شده روی $G_1$ یا $G_2$ باشد. یک گراف $n$ راسی دارای حداکثر $n!$ گراف هم‌ارز است. فرض کنیم هر دو دقیقا $n!$ گراف هم‌ارز

[43] Optimum

دارند. بستگی به این‌که $G_1 \cong G_2$ یا $G_1 \ncong G_2$ اندازه‌ی $S$ در یک فاکتور 2 تفاوت دارد:

$$\begin{array}{ll} 1 & G_1 \ncong G_2 \Rightarrow |S| = 2n! \\ 2 & G_1 \cong G_2 \Rightarrow |S| = n! \end{array}$$

حال حالت کلی‌تر را در نظر بگیریم که $G_1$ یا $G_2$ کمتر از $n!$ گراف هم‌ارز داشته باشند. یک گراف $n$ راسی $G$ کمتر از $n!$ گراف هم‌ارز دارد اگر و تنها اگر یک اتومورفیسم نابدیهی داشته باشد که جایگشت $\pi$ است که جایگشت همانی نیست و $\pi(G) = G$ می‌باشد. حال اگر $aut(G)$ را مجموعه‌ی اتومورفیسم‌های گراف $G$ بگیریم می‌توانیم تعریف $S$ را به نفع خودمان تغییر دهیم:

$$S = \{(H, \pi) | H \cong G_1 \text{ or } H \cong G_2 \text{ and } \pi \in aut(H)\}$$

در کتاب [۱۰] از این حقیقت استفاده شده که $aut(G)$ یک زیرگروه است و ما نیز از این حقیقت با توجه به ارجاع به کتاب استفاده می‌کنیم. با توجه به این حقیقت $S$ شرط 1 و 2 در بالا را برآورده می‌کند. پس عضویت در این مجموعه به راحتی قابل verify کردن است. حال برای آن که prover بتواند verifier را به $G_1 \ncong G_2$ متقاعد کند، prover می‌بایست نشان دهد که شرط 1 برقرار است و شرط 2 برقرار نیست. این کار به وسیله‌ی تکنیک‌های گفته شده در [۱۰] ممکن است.
■

**قضیه ۲.۶.** *اگر $GISO \in NP$-complete آن‌گاه $\Sigma_2 = \Pi_2$ و در نتیجه سلسله مراتب چند جمله‌ای به طبقه‌ی دوم فرو می‌ریزد.*

اثبات: $GISO \in NP \cap co\text{-}AM \subseteq AM \cap co\text{-}AM$ می‌باشد. [۱۱] حال اگر $GISO \in NP$-complete در نتیجه $GISO \in coNP$-complete پس وجود دارد تابع $f$ به شکلی که برای هر فرمول $\phi$ با $n$ متغیر داریم

$$(\forall y)\phi(y) \Leftrightarrow f(\phi) \in GISO$$

حال یک $TQBF_2$ دلخواه را در نظر بگیرید:

$$\psi = (\exists x)(\forall y)\phi(x, y)$$

به شکلی که $x$ و $y$ وکتورهایی با اندازه‌های $n$ هستند. حال این فرمول برابر است با

$$(\exists x)g(x) \in GISO$$

به شکلی که $x$ یک وکتور با اندازه‌ی $n$ بوده و $g(x) = f(\phi\restriction_x)$ که $\phi\restriction_x$ فرمولی است که از $\phi(x, y)$ با فیکس کردن یک $x$ به دست می‌آید می‌باشد. می‌دانیم که $GISO$ یک بازی آرتور–مرلین دو رانده دارد که خطای ناچیز $2^{-n}$ دارد. اگر $V$ را verifier این proof system آرتور–مرلین بگیریم و $m$ را اندازه‌ی نوار تصادفی آن بگیریم و $m'$ را طول پیام prover بگیریم، ادعا می‌کنیم که $\psi$ درست است اگر و تنها اگر که

$$(\forall r)(\exists x)(\exists a)V(g(x), r, a) = 1$$

به شکلی که $r$ یک وکتور $m$ تایی باشد و $x$ یک وکتور $n$ تایی باشد و $a$ یک وکتور $m'$ تایی باشد. این به این معناست که

$$(\forall x)g(x) \notin GISO$$

برای وکتور $n$ تایی $x$ برقرار است. حال از این حقیقت استفاده می‌کنیم که ضریب خطای درستی این proof system برابر $2^{-n}$ است. همچنین تعداد $x$ های ممکن برابر $2^n$ است. در نتیجه وجود دارد یک وکتور $r$ با اندازه‌ی $m$ به شکلی که برای هر وکتور $x$ با اندازه‌ی $n$ اگر اولین پیام $r$ باشد، بازیکن prover

در بازی آرتور_ مرلین برای $GISO$ هیچ جواب متقاعد کننده‌ی $a$ ندارد که باعث شود که verifier رشته را بپذیرد. به بیان دیگر داریم:

$$(\exists r)(\forall x)(\forall a)V(g(x), r, a) = 0$$

برای وکتورهای $r$ و $x$ و $a$ با همان اندازه‌های فرمول بالا. از آن‌جا تصمیم صحت آن در $\Pi_2$ است. چراکه جمله به شکل $(\forall x)(\exists y)P(x, y)$ است. بر اساس قضایای اثبات شده در بخش 5 همین نوشتار در مورد سلسله مراتب چندجمله‌ای، خواهیم داشت $\Sigma_2 \subseteq \Pi_2$ و در نتیجه سلسله مراتب چندجمله‌ای به طبقه‌ی دوم فرو می‌ریزد. [۱۰]
■

## References


[1] L. Babai, "Trading group theory for randomness," in *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pp. 421–429, 1985. 1, 11

[2] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on computing*, vol. 18, no. 1, pp. 186–208, 1989. 1, 2, 3, 12

[3] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems," *Journal of the ACM (JACM)*, vol. 38, no. 3, pp. 690–728, 1991. 2, 11, 13

[4] D.-Z. Du and K.-I. Ko, *Theory of computational complexity*, vol. 58. John Wiley & Sons, 2011. 5, 25, 26, 27

[5] M. Sipser, "Introduction to the theory of computation. cengage learning," *International edition*, 2012. 9

[6] D. C. Kozen, *Theory of computation.* Springer Science & Business Media, 2006. 9, 10

[7] L. Babai, "Automorphism groups, isomorphism, reconstruction (chapter 27 of the handbook of combinatorics)," *North-Holland–Elsevier*, pp. 1447–1540, 1995. 11

[8] A. Wigderson, *Mathematics and computation.* Princeton University Press, 2019. 12, 13, 14

[9] L. Stockmeyer, "Planar 3-colorability is polynomial complete," *ACM Sigact News*, vol. 5, no. 3, pp. 19–25, 1973. 12

[10] S. Arora and B. Barak, *Computational complexity: a modern approach.* Cambridge University Press, 2009. 14, 23, 24, 26, 28, 29

[11] J. Kobler, U. Schöning, and J. Torán, *The graph isomorphism problem: its structural complexity.* Springer Science & Business Media, 2012. 16, 17, 18, 19, 20, 21, 22, 28

[12] S. Goldwasser and M. Sipser, "Private coins versus public coins in interactive proof systems," in *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pp. 59–68, 1986. 26