فصل ۵ Code as Data, data as code

برنامه ها توالی ای از نمادها(کاراکترها) هستند ← قابل نمایش با رشته ای از ۰و۱
پس هر برنامه NAND-CIRC (یا به صورت معادل مدارهای بولی) را میشود با رشته ای از ۰و۱ نشان داد.
بنابراین میشه هم به عنوان دستورالعملی برای پردازش به آنها نگاه کرد، هم به عنوان ورودی به یک برنامه دیگر.

Big Idea 6 : هر برنامه، یک متن است و میشه آنرا به عنوان ورودی به یک برنامه دیگر داد

کاربرد این نگاه در مفهوم کامپیوتر های general purpose تداعی میشود که فقط برای یک تسک خاص استفاده نمیشوند. همینطور در یادگیری ماشین و زبان های اسکریپتی.
همانطور که در بحث امنیت، اگر این نگاه را خوب به کار نگیریم، buffer overflow برای تزریق کد توسط مهاجم به کار میرود، درحالی که ما فقط انتظار "متن عادی گذرا" داریم.
در رشته DNA نیز هم اطلاعات وجود دارد و هم نحوه پردازش.

مطالب این فصل:

- نمایش برنامه/مدار هایی در سایز مشخص به شکل رشته ۰و۱ ← جهت شمارش تعداد برنامه های ممکن.
- برای برخی توابع، حداقل سایز ممکن برنامه/مدار جهت پیاده سازی، از اردر نمایی است.
- استفاده از نمایش رشته ای برنامه/مدار ها برای ساخت "مدار universal" که قابلیت evaluate کردن مدارهای دیگر را دارد. همان مفهوم (Meta Circular Evaluators) در برنامه نویسی. نتایج نشان خواهند داد که سایز این مدار باید از سایز مدارهایی که evaluate میکند باید بزرگتر باشد.

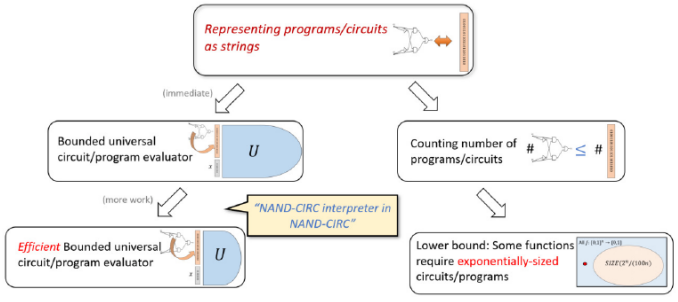

Figure 5.2: Overview of the results in this chapter. We use the representation of programs/circuits as strings to derive two main results. First we show the existence of a universal program/circuit, and in fact (with more work) the existence of such a program/circuit whose size is at most polynomial in the size of the program/circuit it evaluates. We then use the string representation to *count* the number of programs/circuits of a given size, and use that to establish that *some* functions require an *exponential* number of lines/gates to compute.

شکل ۱

۵.۱ نمایش برنامه ها به صورت رشته

مدارهای بولی مانند گرافهای DAG هستند ← امکان نمایش آنها با ماتریس همسایگی یا لیست همسایگی
اما در نهایت برنامه به صورت یک توالی از کاراکترها معرفی میشود:

```
temp_0 = NAND(X[0],X[1])
temp_1 = NAND(X[0],temp_0)
temp_2 = NAND(X[1],temp_0)
Y[0] = NAND(temp_1,temp_2)
```

شکل ۲

برنامه شکل ۲ دارای ۱۰۷ کاراکتر است، در ASCII به هر کدام ۷ بیت اختصاص پیدا میکند ← ۷۴۹ بیت
اگر فرض کنیم برنامه ما s خط است:

- هر خط ۳ متغیر داریم. پس 3s تا متغیر در کل برنامه داریم(تکراری یا غیر تکراری). (اندیس)
- برای شناسایی هر متغیر از اندیس آن استفاده میکنیم ← تعداد بیت لازم برای هر متغیر: log(3s)
- تعداد بیت لازم برای کل برنامه ← s.log(s)

> **Theorem 5.1 — Representing programs as strings.** There is a constant c such that for $f \in SIZE(s)$, there exists a program P computing f whose string representation has length at most $cs \log s$.

شکل ۳ : قضیه ۵.۱

قضیه ۵.۱ نیز به این اشاره میکند و مفهومی به نام عضویت یک تابع f در مجموعه SIZE(s) را معرفی میکند. یعنی تابع f تابعی است که برنامه آن را میتوانیم با حداکثر s خط بنویسیم. مجموعه SIZE(s) نیز مجموعه تمام توابعی است که با حداکثر s خط نوشته میشوند. با محاسبه بالا، برنامه این توابع با حداکثر cs log s میتوانند به نمایش رشته ای تبدیل شوند.

۵.۲ شمارش برنامه ها، و حد کمینه برای سایز برنامه های NAND-CIRC

از نتایج قضیه ۵.۱ میشود به این اشاره کرد که تعداد برنامه های به طول مشخص، محدود به طول رشته هایی است که این برنامه ها را میتوان با آن نمایش داد. پیامد این موضوع گریبان SIZEn,m(s) را به شرح زیر میگیرد:

قضیه ۵.۲

Theorem 5.2 — Counting programs. For every $s, n, m \in \mathbb{N}$,

$$|SIZE_{n,m}(s)| \leq 2^{O(s \log s)}.$$

That is, there are at most $2^{O(s \log s)}$ functions computed by NAND-CIRC programs of at most s lines. [1]

شکل ۴ : قضیه ۵.۲

یعنی کاردینالیتی مجموعه توابعی با n ورودی و m خروجی که با s خط برنامه پیاده سازی میشوند، از اردر نمایی به توان s log s است.

اثبات:

تابع E را به صورت یک به یک از مجموعه SIZEn,m(s) به رشته هایی به طول حداکثر c.s log s تعریف میکنیم. اگر بشود این کار را انجام داد، یعنی |SIZEn,m(s)| کمتر است از تعداد رشته های به طول حداکثر c.s log s . یعنی کمتر از:

$$l = cs\,log\,s \;\dashrightarrow\; 1 + 2 + 4 + \ldots + 2^{l} = 2^{l+1} - 1 \sim O(s\,log\,s)$$

از آنجا که هر تابع عضو SIZEn,m(s) توسط برنامه ای به طول حداکثر s خط نوشته میشود، با توجه به قضیه ۵.۱، نمایش رشته ای آن برنامه نیز حداکثر c.s log s بیت خواهد بود.

از طرفی نگاشت f به E(f) یک به یک است. چرا که برای دو تابع متفاوت f و f ' ، قطعا یک ورودی x وجود خواهد داشت که حاصل f(x) و f '(x) متفاوت خواهند بود. بنابراین برنامه مربوط به f و f ' نمیتوانند یکسان باشند.

پس تعداد آنها طبق سامیشن بالا، بدست می آید و اثبات تمام میشود. ■

طبق این اثبات (رجوع به سامیشن)، تعداد توابعی که با برنامه ای با تعداد خط کم نوشته میشوند، بسیار کمتر است از تعداد توابعی که با برنامه هایی با تعداد خط بسیار زیاد نوشته میشوند.

هر تابع را میتوان با گفتن خروجی اش به ازای تمامی حالات ممکن روی n ورودی (2^n حالت) بیان کرد.

(مثلا :

- ۰۰ به ۱ مپ شود
- ۰۱ به ۱ مپ شود
- ۱۰ به ۱ مپ شود
- ۱۱ به ۰ مپ شود)

این برای تابعی با ۲ ورودی است که $2^2 = 4$ مقدار را برای تعریف این تابع باید مشخص کرد. (یک تابع)

اکنون اگر بخواهیم تمامی توابع ممکن را ردیف کنیم و ببینیم چند تابع وجود دارند که ۲ ورودی را به ۱ خروجی مپ میکنند، انگار باید دید این 2^n ورودی را به چند حالت میتوان مقداردهی کرد که رشته ای به طول 2^n بدست آید، که نمایی از اردر 2^n خواهد بود.

بنابراین، تعداد توابع ممکن که n ورودی را به ۱ خروجی مپ میکنند، از اردر 2^{2^n} خواهد بود.

از این ازدیاد تعداد توابع، میشه به قضیه ۵.۳ رسید که میگه:

قضیه ۵.۳

Theorem 5.3 — Counting argument lower bound. There is a constant $\delta > 0$, such that for every sufficiently large n, there is a function $f : \{0,1\}^n \to \{0,1\}$ such that $f \notin SIZE_n\left(\frac{\delta 2^n}{n}\right)$. That is, the shortest NAND-CIRC program to compute f requires more than $\delta \cdot 2^n/n$ lines. [3]

شکل ۵ : قضیه ۵.۳

اثبات:

از قضیه ۵.۲ میدانیم که : $|SIZE_n(s)| \leq 2^{cs\ log\ s}$

حالا اگر دلتا را برابر با معکوس c تعریف کنیم ($\delta = 1/c$)، بدیهی است که تعداد توابعی که طولشان برابر است با $s = \delta 2^n/n$ است، در نامساوی زیر صدق میکنند.

$$|SIZE_n(\frac{\delta 2^n}{n})| \leq 2^{c\frac{\delta 2^n}{n}\log s} < 2^{c\delta 2^n} = 2^{2^n}$$

دقت کردید چی شد؟ الان داریم "تعداد توابعی از n ورودی به ۱ خروجی" رو در نظر میگیریم (یعنی 2^{2^n}).

در قضیه ۵.۱ "طول یک برنامه که یک تابع را پیاده سازی کرده" بدست آوردیم.

در فصل ۴ گفتیم که "تعداد خط لازم برای برنامه ای که توابع n تا ورودی را به ۱ ورودی مپ میکنند"، از اردر زیر است (قضیه ۴.۱۵):

$c.2^n/n$

حالا داریم میگیم اگر "تعداد برنامه هایی که با s خط پیاده سازی میشوند" را در نظر بگیریم، از اردر زیر خواهد بود (قضیه ۵.۲):

$2^{cs\ log\ s}$

در نتیجه اگر تعداد خط برنامه را عددی کمتر از $c.2^n/n$ بگیریم (در اینجا $\delta.2^n/n$ با تعریف $\delta = 1/c$ که چون c بزرگتر از ۱ است، δ کوچکتر از ۱ میشود) ، در نتیجه تعداد توابعی که میشه با این تعداد خط برنامه پیاده سازی کرد، قطعا کمتر از تعداد برنامه هایی است که با $c.2^n/n$ خط برنامه میتوان پیاده سازی کرد.

در نتیجه حداقل یک برنامه وجود دارد که نیاز به حداقل این تعداد بسیار زیاد خط برنامه دارد. ■

Big Idea 6 : داره به همین مفهومی که اثبات کردیم اشاره میکنه که برخی برنامه ها نمیتوانند با کمتر از تعداد گیت نمایی از اردر "اِن" پیاده سازی شوند

به طور کلی در اینجا داره اشاره میکنه که اکثر توابع اتفاقا از همین الگو پیروی میکنند و نیاز به این تعداد نجومی گیت/خط برنامه دارند و مابقی توابعی که با تعداد اندکی گیت/خط برنامه پیاده سازی میشوند استثنا هستند.

Remark 5.4

داره به این اشاره میکنه که یک نمایش بهتر هم (جایگزین ASCII) از جهت بهینه بودن وجود داره و اون هم نمایش DAG است که با ماتریس مجاورت میتوان ضریب ثابت c را مقداری کاهش داد.

۵.۲.۱ نظریه سلسله مراتب سایز

با استفاده از قضیه ۴.۱۵ و ۵.۳ که در صفحه پیش دیدیم، میتوان به نتیجه زیر رسید:

$$SIZE_n\left(0.1\frac{2^n}{n}\right) \subsetneq SIZE_n\left(10\frac{2^n}{n}\right)$$

که در آن، زیرمجموعه "ناسره" بودن مد نظر است.

درواقع میگوید اگر بخواهیم تعداد گیت/خط برنامه را افزایش دهیم، قطعا توابع بیشتری را میتوانیم پیاده سازی کنیم.

قضیه ۵.۵

Theorem 5.5 — Size Hierarchy Theorem. For every sufficiently large n and $10n < s < 0.1 \cdot 2^n/n$,

$$SIZE_n(s) \subsetneq SIZE_n(s + 10n) \ .$$

شکل ۶ : قضیه ۵.۵

اثبات:

در این قضیه، یک تابع f* را فرض کنید که عضو SIZE(0.1*2^n/n) نیست.

یک تابع دیگر هم فرض کنید lex که یک ورودی 2^n بیتی میگیرد و به صورت یکتا به هر ورودی عددی از ۰ تا $2^n - 1$ میدهد.

زیر توابعی از f* را فرض کنید مانند f_0 تا f_{2^n} که هر کدام از آنها فقط به ازای زمانی که lex(x) کمتر مساوی اندیس i در f_i باشد، خروجی f*(x) را برمیگرداند وگرنه ۰ برمیگرداند:

$$f_i(x) = \begin{cases} f^*(x) & lex(x) < i \\ 0 & \text{otherwise} \end{cases}.$$

پس فرق میان f_{i-1} و f_i فقط یک ورودی x است که به ازای آن lex(x) = i و خروجی تابع به ازای آن ورودی مطابق با f* خواهد بود.

اکنون فرض کنیم که تابعی وجود دارد با مشخصات s ای که در قضیه داده شده (که کاملا مشخص است که چنین s ای وجود دارد).

اکنون اگر یک f_i پیدا کنیم که اولین تابعی باشد که عضو SIZE(s) نیست، میدانیم که f_{i-1} عضو این مجموعه میباشد.

اکنون میخواهیم ثابت کنیم این دو تابع که حداکثر به ازای ۱ ورودی با هم تفاوت دارند، تابع f_i را میتوان صرفا با 10n گیت/خط برنامه بیشتر از f_{i-1} پیاده سازی کرد.

حالا فکر کنم راحت متوجه میشید که تفاوت این دو تابع را میشه فقط با مقایسه کردن ورودی داده شده با آن ورودی خاص (که به ازایش lex(x) = i میشد) میتوان خروجی مدنظر را تولید کرد. همانطور که میدانیم، تابع مقایسه را میتوان با O(9n) پیاده سازی کرد و بنابراین قضیه اثبات میشود. ■

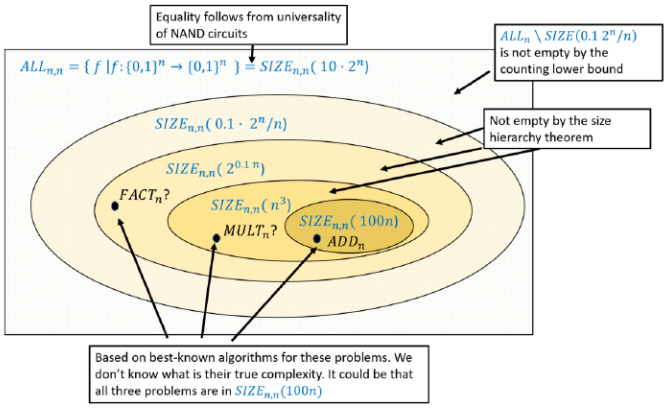

Figure 5.5: An illustration of some of what we know about the size complexity classes (not to scale!). This figure depicts classes of the form $SIZE_{n,n}(s)$ but the state of affairs for other size complexity classes such as $SIZE_{n,1}(s)$ is similar. We know by Theorem 4.12 (with the improvement of Section 4.4.2) that all functions mapping n bits to n bits can be computed by a circuit of size $c \cdot 2^n$ for $c \leq 10$, while on the other hand the counting lower bound (Theorem 5.3, see also Exercise 5.4) shows that *some* such functions will require $0.1 \cdot 2^n$, and the size hierarchy theorem (Theorem 5.5) shows the existence of functions in $SIZE_n(S) \setminus SIZE_n(s)$ whenever $s = o(S)$, see also Exercise 5.5. We also consider some specific examples: addition of two $n/2$ bit numbers can be done in $O(n)$ lines, while we don't know of such a program for *multiplying* two n bit numbers, though we do know it can be done in $O(n^2)$ and in fact even better size. In the above, $FACTOR_n$ corresponds to the inverse problem of multiplying- finding the *prime factorization* of a given number. At the moment we do not know of any circuit a polynomial (or even sub-exponential) number of lines that can compute $FACTOR_n$.

شکل ۷ : سلسله مراتب

۵.۳ نمایش تاپلی

Definition 5.7 — List of tuples representation. Let P be a NAND-CIRC program of n inputs, m outputs, and s lines, and let t be the number of distinct variables used by P. The *list of tuples representation of* P is the triple (n, m, L) where L is a list of triples of the form (i, j, k) for $i, j, k \in [t]$.

We assign a number for each variable of P as follows:

- For every $i \in [n]$, the variable `X[`i`]` is assigned the number i.
- For every $j \in [m]$, the variable `Y[`j`]` is assigned the number $t - m + j$.
- Every other variable is assigned a number in $\{n, n+1, \ldots, t-m-1\}$ in the order in which the variable appears in the program P.

نمایش تاپلی یک برنامه یعنی یک مجموعه t تایی داری که n تای اول ورودی، m تای آخر خروجی و مابقی متغیرهای میانی اند که برای محاسبات به کار میروند. با مثال زیر بیشتر جا می افتد.

■ **Example 5.8 — Representing the XOR program.** Our favorite NAND-CIRC program, the program

```
u = NAND(X[0],X[1])
v = NAND(X[0],u)
w = NAND(X[1],u)
Y[0] = NAND(v,w)
```

computing the XOR function is represented as the tuple $(2, 1, L)$ where $L = ((2, 0, 1), (3, 0, 2), (4, 1, 2), (5, 3, 4))$. That is, the variables `X[0]` and `X[1]` are given the indices 0 and 1 respectively, the variables `u`,`v`,`w` are given the indices $2, 3, 4$ respectively, and the variable `Y[0]` is given the index 5.

۵.۳.۱ از تاپل به رشته

مقدار t همان تعداد متغیر برنامه است، که برای برنامه ای با s خط، حداکثر 3s است

پس هر متغیر در t به یک رشته به طول O(log 3s) تبدیل میشه (با اضافه کردن ۰ های پیشوندی مثلا به عددهای کوچکی مثل ۱.

1 ← 0000001 مثلا...

چون s خط هم داریم، مشابه با قضیه ۵.۱ میشه اندازه برنامه نمایش داده شده با تاپل را به یک نمایش رشته به طول 3s[log 3s] تبدیل کرد.

$$S(s) = 3s\lceil \log(3s) \rceil$$