مدل مداری برای محاسبات کلاسیک

وحیدکریمی پور_ دانشکده فیزیک _ دانشگاه صنعتی شریف

۲۴ فروردین ۱۳۹۸

۱ مقدمه

درقسمت اول این درس سعی کردیم که اصول اساسی مکانیک کوانتومی را معرفی کنیم. در این بررسی کمی فراتر از آن چیزی رفتیم که در درس های متعارف مکانیک کوانتومی معمول است و توانستیم تعاریف تعمیم یافته‌ای از اندازه گیری و هم چنین دینامیک سیستم های کوانتومی معرفی کنیم. البته بحث ما از مبانی مکانیک کوانتومی هنوز پایان نیافته‌است و هنوز به مفاهیم مهمی مثل درهم تنیدگی و اندازه‌های آن، نپرداخته‌ایم. امادر پایان این درس دوباره به این مفاهیم برخواهیم گشت و آن وقتی است که نظریه اطلاعات کوانتومی را معرفی می کنیم. برای مدتی می بایست روند فعلی را متوقف کرده و به بررسی اصول اساسی کامپیوترهای کوانتومی بپردازیم. روش ما در چند درس آینده آن خواهد بود که نخست مدل مداری برای محاسبات کلاسیک را معرفی می کنیم و سپس به معرفی مدل مداری برای محاسبات کوانتومی می پردازیم. سپس سعی می کنیم چندین آلگوریتم کوانتومی ساده را که توانایی های کامپیوترهای کوانتومی را در مقایسه با کامپیوترهای کلاسیک نشان می دهند معرفی کنیم. این آلگوریتم ها به ترتیب پیچیدگی معرفی خواهند شد. سپس به اختصار با اصول اساسی نظریه محاسبه آشنا می شویم.

۲ پردازش اطلاعات کلاسیک

می دانیم که در کامپیوترهای کلاسیک تمام اطلاعات به شکل رشته‌ای از متغیرهای $0,1$ ذخیره می شوند. پردازش داده‌ها از هرنوع که باشد، چیزی نیست جز انجام اعمال منطقی روی این رشته‌ها. هر متغیر دو حالته که می تواند دو مقدار 0 یا 1 را اختیار کند یک بیت نامیده می شود. یک بیت را با متغیر x نشان می دهیم. یک رشته دو بیتی با نمادِ x_1x_0 و یک رشته‌ی n بیتی با نمادِ $x_{n-1}x_{n-2}\cdots x_1x_0$ نشان داده می شود. می توانیم یک عدد را در پایه ۲ به صورت $x = x_{n-1}x_{n-2}\cdots x_1x_0$ نشان دهیم که دراین صورت مقدار عددی آن عبارت خواهد بود از

$$x = x_{n-1}\times 2^{n-1} + x_{n-2}\times 2^{n-2} + \cdots x_1\times 2^1 + x_0\times 2^0. \tag{۱}$$

مجموعه تمام متغیرهای n بیتی را با B_n نمایش می دهیم. چنین مجموعه‌ای 2^n عضو دارد که ارزش عددی آنها بین مقدارِ 0 تا 2^n-1 تغییر می کند. گاهی اوقات از نمادِ $\{0,1\}^n$ نیز برای B_n استفاده می شود:

$$B_n := \{(x_0, x_1, x_2\cdots x_{n-1}) \ \mid\ x_i\in\{0,1\}\}. \tag{۲}$$

هر نوع پردازش اطلاعات چیزی نیست جز یک سلسله توابعِ پشت سر هم که روی یک رشته‌ی ورودی با طول معین انجام می شود. تمام این توابع را می توان با ترکیب توابع مقدماتی ای که تنها روی یک بیت و یا دو بیت اثر می کنند، ساخت. ساده ترین توابع مقدماتی عبارتند از توابع NOT، OR ، AND و XOR که به صورت زیر تعریف می شوند:

$$\begin{aligned}
NOT:\quad x &\longrightarrow \quad \overline{x} := x+1 \qquad mod\ \ 2\\
OR:(x,y) &\longrightarrow \quad x\vee y := x+y-xy \quad mod\ \ 2\\
AND:(x,y) &\longrightarrow \quad x\wedge y : xy\\
XOR:(x,y) &\longrightarrow \quad x\oplus y : x+y-2xy.
\end{aligned} \tag{۳}$$

هرکدام از این توابع مقدماتی را اصطلاحاً یک دروازه یک گیت می نامند. هم چنین ازاین خاصیت مهم نیز استفاده می شود که از یک بیت کلاسیک مثل x همواره می توان نسخه‌های متعدد بدست آورد. توابع مقدماتیِ فوق خواص ساده و درعین حال مهمی دارند. مهمترین این خواص را در زیر می نویسیم. خواننده می تواند براحتی درستی این خواص را بیازماید.

الف: تمام توابع دوتایی جابجایی و شرکت پذیرهستند، یعنی

$$x \bigcirc y \quad = \quad y\bigcirc x$$

$$(x \bigcirc y) \bigcirc z \quad = \quad x \bigcirc (y \bigcirc z), \qquad \bigcirc \in \{\vee, \quad \wedge, \quad \oplus\} \tag{۴}$$

ب: اثر این توابع روی مقادیر خاص به صورت زیر است:

$$x \vee 0 = x, \quad x \vee 1 = 1, \quad x \vee x = x, \quad x \vee \overline{x} = 1. \tag{۵}$$

$$x \wedge 0 = 0, \quad x \wedge 1 = x, \quad x \wedge x = x, \quad x \wedge \overline{x} = 0. \tag{۶}$$

$$x \oplus 0 = x, \quad x \oplus 1 = \overline{x}, \quad x \oplus x = 0, \quad x \oplus \overline{x} = 1. \tag{۷}$$

ج: ترکیب NOT و توابع AND و NOT. روابط زیر اصطلاحاً به قوانین دمورگان مشهورهستند:

$$\overline{x \vee y} = \overline{x} \wedge \overline{y} \qquad \overline{x \wedge y} = \overline{x} \vee \overline{y} \tag{۸}$$

د: ترکیب NOT و تابع XOR که کاربردهای زیادی در اثبات قضایا و خواص دیگر دارد.

$$\begin{aligned} \overline{x \oplus y} &= \overline{x} \oplus y = x \oplus \overline{y} \\ \overline{x} \oplus \overline{y} &= x \oplus y. \end{aligned} \tag{۹}$$

حال سوال این است که آیا دروازه های مقدماتی فوق برای ساختن هرتابع دلخواه کافی هستند یا خیر. قضیه زیر در واقع پاسخ مثبتی به این سوال است.

قضیه: هرتابع توابع دلخواهِ $f : B_m \longrightarrow B_n$ را می توان با ترکیب تعدادی متناهی تابعِ NOT، AND و OR ساخت. به عبارت دیگر این دروازه های مقدماتی یک مجموعه دروازه های عام یا جهان شمول $Universal\ set\ of\ gates$ هستند.

اثبات: می دانیم که هر تابع $f: B_m \longrightarrow B_n$ چیزی نیست جز n تابع متفاوت از B_m به B_1. بنابراین قضیه را برای این توابع ثابت می کنیم. برای اثبات از استقراء استفاده می کنیم. می دانیم که برای $n=1$ قضیه برقرار است زیر تنها توابع ممکن عبارتند از تابع همانی و تابع NOT. حال فرض کنید که قضیه برای n برقرار است. عبارت $f(x_1, x_2, \cdots x_{n+1})$ را در نظر بگیرید.

می دانیم که

$$f(x_1, x_2, \cdots x_n, x_{n+1}) = \begin{cases} f(x_1, x_2, \cdots x_n, 0) & if \quad x_{n+1}=0, \\ \\ f(x_1, x_2, \cdots x_n, 1) & if \quad x_{n+1}=1, \end{cases} \tag{۱۰}$$

رابطه فوق را می توان به صورت زیر نوشت:

$$f(x_1, x_2, \cdots x_n, x_{n+1}) = [\overline{x_{n+1}} \wedge f(x_1, x_2, \cdots x_n, 0)] \vee [x_{n+1} \wedge f(x_1, x_2, \cdots x_n, 1)]. \tag{۱۱}$$

اما توابع درون کروشه توابع n متغیره هستند که فرض استقراء می گوید می توان آنها را برحسب گیت های مقدماتی نوشت. بنابراین قضیه ثابت می شود.

- تمرین: یک مدار ساده طراحی کنید که دو عدد یک بیتی را باهم جمع کند.

- تمرین: یک مدار ساده برای نیم جمع کننده [۱] طراحی کنید . کار این نیم جمع کننده این است که سه عدد یک بیتی را گرفته و حاصل جمع آنها را به عنوان یک عدد دو بیتی تولید می کند.

- تمرین: با ترکیب کردن نیم جمع کننده ها یک مدار کامل بسازید که اعداد n بیتی را باهم جمع کرده و یک عدد $n+1$ تولید می کند

[۱] Half Adder

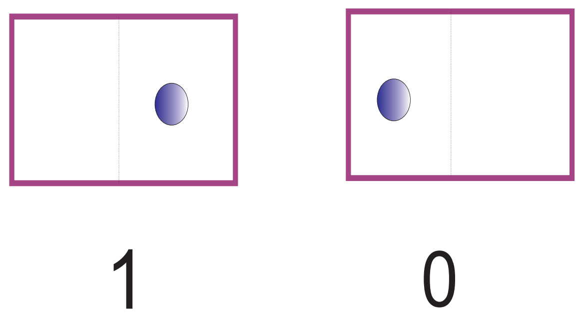
■ تمرین: تابع $f : B^3 \longrightarrow B^1$ به این ترتیب تعریف می شود که هرگاه یکی یا بیشتر از بیت های ورودی برابر با یک باشند، مقدار تابع برابر با صفر و در غیر این صورت برابر با صفر است. این تابع را برحسب گیت های مقدماتی بنویسید.

۳ مدارهای کلاسیک برگشت پذیر

می دانیم که با کوچکتر شدن تراشه ها، دائماً مصرف انرژی کامپیوتر هانیز کاهش می یابد. سوال مهمی که دراین جا پیش روی ماست آن است که آیا می توان عمل محاسبه را لااقل به صورت نظری بدون اتلاف انرژی انجام داد یا خیر، حتماً یک حد نظری از مصرف انرژی وجود دارد که از آن نمی توان عبور کرد.

توابع مقدماتی ای که به آنها اشاره کردیم وارون پذیر نیستند. یک گیت وارون ناپذیر به لحاظ نظری حتماً توانی مصرف می کند که نمی توان با هیچ پیشرفتی در فناوری از آن جلوگیری کرد، یعنی هر چقدر هم که بکوشیم از هدر رفتن انرژی در مدارها و قطعات جلوگیری کنیم، یک حدِّ تئوریک از اتلاف انرژی وجود دارد که از آن نمی توانیم حذر کنیم. این حد را نخستین بار رالف لاندائر ($Ralph\ \ Landauer$) در ۱۹۶۹ نشان داده است. مبنای استدلال او این است که در گیت وارون ناپذیر مقداری اطلاعات گم می شود (زیرا نمی توان از خروجی تابع به ورودی آن پی برد) و هر نوع پاک کردن اطلاعات نیز با کاهش آنتروپی سیستم و افزایش انتروپی محیط همراه است. کافی است برای درک این نکته به ساده ترین تعبیه یک بیت نگاه کنیم. فرض کنید که یک اتاقک میکروسکوپی ساخته ایم که در درون آن یک مولکول وجود دارد، شکل 1. وقتی که مولکول در سمت راست اتاقک است مقدار بیت برابر با ۱ و وقتی که درسمت چپ اتاقک است مقدار بیت برابر با صفراست. تعداد N تا این بیت ها را در نظر بگیرید که مجموعاً یک عدد را نشان می دهند.

پاک کردن اطلاعات به معنای آن است که این عدد را صرف نظر از مقداری که دارد به یک عدد مرجع مثلاً $000 \cdots 0$ برگردانیم. این کار را می توانیم به کمک یک پیستون و دیواره‌ی بدون اصطکاک که در وسط اتاقک تعبیه کرده‌ایم انجام دهیم. اما این کار نهایتاً باعث کاهش آنتروپی سیستم به اندازه‌ی $kN\ ln2$ و افزایش آنتروپی محیط به همین مقدار خواهد شد. بنابراین به ازای پاک کردن هر بیت اطلاعات می بایست کاری معادل $kT\ \ \ln 2$ ژول انجام دهیم که در آن T دمای محیط است. بنابراین اگر با توابع وارون ناپذیر کار کنیم با این حد نظری از مصرف انرژی روبرو هستیم. کار مهمِ چارلز بنت (Charles Bennett) آن بوده که نشان داده است می توان با انجام محاسبات به صورت وارون پذیر



شکل ۱: یک مولکول در یک اتاقک می تواند برای ذخیره کردن اطلاعات به کار رود. ولی پاک کردن اطلاعاتی که ذخیره می کند باعث کاهش آنتروپی سیستم و ایجاد گرما در محیط می شود.

مصرف انرژی را در کامپیوترها لااقل به لحاظ نظری به صفر رساند.

چگونه می توان محاسبات را به صورت برگشت پذیر انجام داد و حال آنکه اغلب توابع وارون پذیر نیستد بخصوص می دانیم که گیت های جهان شمولی که تمام مدارهای منطقی از آن ها ساخته می شوند، مثل گیت های AND و OR وراون پذیر نیستند.
برای این کار نخست نشان می دهیم که با اضافه کردن مجموعه ای از بیت ها به اسم بیت خدمتکار $Ancilla\ Bit$ ، می توان هر تابع دلخواه را به صورت برگشت پذیر نیز تعبیه کرد. فرض کنید که $f: B_m \longrightarrow B_n$ یک تابع دلخواه است. تابع $f_r: B_{m+n} \longrightarrow B_{m+n}$ را به شکل زیر تعریف می کنیم:

$$f_r(x,y) := (x, f(x) \oplus y) \tag{۱۲}$$

دراین رابطه $x \in B_m$ ، $y,\ f(x) \in B_n$ و تابع $f(x) \oplus y$ به صورت بیت به بیت تعریف شده است، به عبارت دیگر:

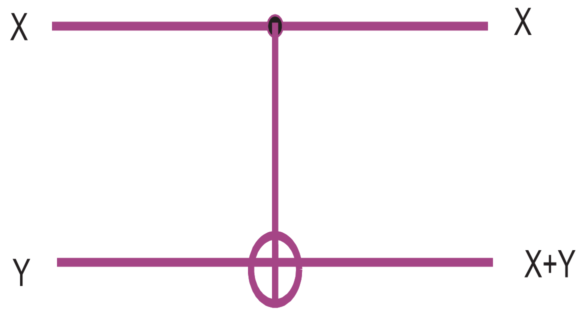
$$f_r(x_1, x_2, \cdots x_m, y_1, y_2, \cdots y_n) := (x_1, x_2, \cdots x_m, f_1(x_1, \cdots x_m) \oplus y_1, \cdots f_n(x_1, \cdots x_m) \oplus y_n). \tag{۱۳}$$

حال دقت می کنیم که

$$f_r(x,0) = (x, f(x) \oplus 0) = (x, f(x)), \tag{۱۴}$$

بنابراین تابع اولیه f به ازای مقدارِ 0 روی بیت های خدمتکار در خروجی ظاهر می شود. هم چنین تابع f_r وارون پذیراست زیرا

$$f_r(x,y) = f_r(x',y') \longrightarrow (x, f(x) \oplus y) = (x', f(x') \oplus y') \longrightarrow x = x' \quad , \quad f(x) \oplus y = f(x) \oplus y' \tag{۱۵}$$



شکل ۲: گیت کنترلی $CNOT$

اما از تساوی $f(x) \oplus y = f(x) \oplus y'$ براحتی و با استفاده از خواص تابع XOR ثابت می شود که $y = y'$. بنابراین تابع f_r یک تابع برگشت پذیر است که تابع f را به ازای مقادیر معینی از بیت های خدمتکار در خود نهفته دارد.

حال سوال این است که چگونه می توان از مجموعه از گیت های ساده تمام توابع وارون پذیر را ساخت؟ این سوال از آن جهت اهمیت دارد که می دانیم گیت های جهان شمول ِ AND و OR و NOT وارون پذیر نیستند و بنابراین نمی توان از آنها برای ساختن توابع وارون پذیر دلخواه استفاده کرد. از توابع وارون پذیر دوبیتی شروع می کنیم. هرتابع وارون پذیرِ $f : B_2 \longrightarrow B_2$ چیزی نیست جز یک جایگشت بین چهار شی، زیرا تعداد ورودی های ممکن برای چنین تابعی عبارت است از $\{(0,0),\ (0,1),\ (1,0),\ (1,1)\}$. گیت تک بیتی NOT وارون پذیراست. دراین جا یک گیت ِ وارون پذیر موسوم به $CNOT = Controlled\ NOT$ به شکل زیر معرفی می کنیم:

$$CNOT : (x,y) \longrightarrow (x, x+y), \tag{۱۶}$$

که درآن جمع به سنجه ۲ انجام می شود. دقت کنید که اگر $x = 0$ باشد آنگاه y به y نگاشته می شود، یعنی روی بیت دوم عمل همانی انجام می شود و اگر $x = 1$ باشد آنگاه، y به $\overline{y}$ نگاشته خواهد شد، یعنی روی بیت دوم گیت ِ NOT اعمال خواهد شد، به همین دلیل این گیت یک گیت ِ کنترلی نامیده می شود. x را بیت کنترل $Control\ Bit$ و y را بیت ِ هدف یا $Target\ Bit$ می نامیم. شکل 2 نمادِ مربوط به این گیت را نشان می دهد.

حال نشان می دهیم که گیت های NOT و $CNOT$ یک مجموعه جهان شمول برای تمام توابع برگشت پذیر دوبیتی تشکیل می دهند، یعنی هر تابع وارون پذیر دو بیتی را می توان با ترکیب دو گیت ساخت. می دانیم که هر تابع وارون پذیر چیزی جز یک جایگشت بین چهار شیء نیست و گروه جایگشت های بین چهارشیء با سه مولد تولید می شود، یعنی سه تابع σ_1، σ_2 و σ_3 وجود دارند که از ترکیب آنها همه توابع دیگر ساخته

σ_3	
0 0,	0, 0
1 0,	0, 1
0 1,	1, 1
1 1,	1, 0

σ_2	
0 0,	0, 0
1 1,	0, 1
1 0,	1, 1
0 1,	1, 0

σ_1	
1 0,	0, 0
0 0,	0, 1
1 1,	1, 1
0 1,	1, 0

جدول ۱: توابعی که تمام توابع وارون پذیر دو بیتی را تولید می کنند.

می شوند. حال توابع σ_1، σ_2 و σ_3 که مولد های گروه جایگشت روی چهارشیء (دراین جا چهار سطر جدول فوق) هستند عبارتند از:

براحتی معلوم می شود که این سه تابع عبارت تحلیلی زیر را دارند:

$$\sigma_1(x,y)=(x,\overline{x}\oplus y) \qquad \sigma_2(x,y)=(x+y,y), \qquad \sigma_3(x,y)=(x,x+y), \tag{۱۷}$$

که ثابت می کند هر سه تابع بر حسب $CNOT$ و NOT قابل بیان هستند. از آنجا که این سه تابع مولد گروه جایگشت های چهارتایی هستند، قضیه برای تمام توابع وارون پذیر دو بیتی ثابت می شود. دقت کنید که در نوشتن سطرهای جدول های توابع σ_i قاعده خاصی را به کار برده ایم به این صورت که هر دو سطر متوالی تنها در یک بیت با یکدیگر متفاوت اند. این گونه مرتب کردن سطر ها را اصطلاحاً کد گری [۲] می گویند. همواره می توان اعضای B_n را چنان مرتب کرد که هردو عنصر متوالی آن تنها در یک بیت باهم اختلاف داشته باشند. دراین صورت می گوییم که برای نوشتن عناصرِ B_n کد گری را به کار برده‌ایم.

■ تمرین: نشان دهید که تمام توابع وارون پذیر دو بیتی

$$f:(x_1,x_0)\longrightarrow(y_1,y_0) \tag{۱۸}$$

[۲] Grey Code

را می توان به صورت زیر نوشت:

$$\begin{pmatrix} y_0 \\ y_1 \end{pmatrix} = M \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} + \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} \tag{۱۹}$$

که در آن M یک ماتریس وارون پذیر و $\begin{pmatrix} a_0 \\ a_1 \end{pmatrix}$ یک بردار است. درایه های این ماتریس و بردار از صفر و یک تشکیل شده اند.

آیا توابع n بیتی را نیز می توان به کمک گیت های NOT و $CNOT$ ساخت؟ پاسخ این سوال منفی است. برای این کار کافی است که به عنوان مثال به تابعِ وارون پذیرِ

$$\theta_3 : (x, y, z) \longrightarrow (x, y, z + xy)$$

توجه کنیم. این تابع برحسب متغیرهای x و y غیر خطی است و واضح است که نمی توان آن را برحسب گیت های خطیِ NOT و $CNOT$ نوشت. تابعی که معرفی کرده ایم یک تابع معروف است که به آن گیت توفولی می گویند.

۱.۳ گیت توفولی

گیت توفولی که آن را با θ_3 نمایش می دهیم، تعمیمی از گیت $CNOT$ است و به شکل زیر تعریف می شود:

$$\theta_3(x, y, z) := (x, y, z \oplus xy), \tag{۲۰}$$

به عبارت دیگر این گیت وقتی که بیت های کنترلی اش یعنی x و y هردو مقدارِ 1 داشته باشند، مثل NOT عمل می کنند و درغیر این صورت مثل گیت همانی عمل می کند. این گیت در شکل ?? نشان داده شده است.

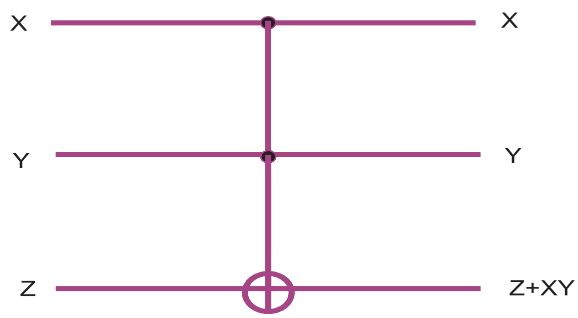
حال نشان می دهیم که گیت های NOT و $CNOT$ واقعاً یک مجموعه گیت جهان شمول برای تمام توابع وارون پذیرِ n بیتی تشکیل می دهند. برای این کار فرض کنید توابعی وارون پذیر با تعریف زیر داشته باشیم:

$$\theta_n(x_1, x_2, \cdots x_n) := (x_1, x_2, \cdots x_{n-1}, x_n \oplus x_1 x_2 \cdots x_{n-1}). \tag{۲۱}$$

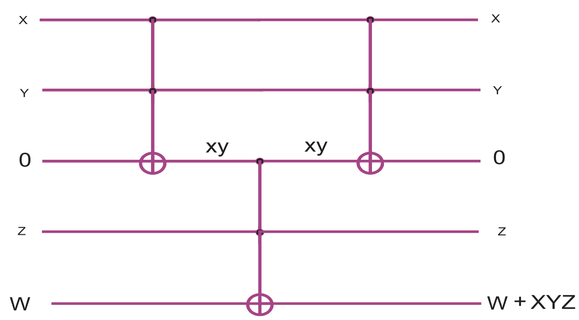
این گیت ها درواقع تعمیم n تایی از گیت های $CNOT$ و یا توفولی هستند. از خواننده دعوت می کنیم که توابع مثلاً 3 بیتی را در نظر بگیرد و ورودی ها را مطابق با کد گری تنظیم کند.

Code Grey
$0, 0, 0$
$0, 0, 1$
$0, 1, 1$
$0, 1, 0$
$1, 1, 0$
$1, 1, 1$
$1, 0, 1$
$1, 0, 0$

جدول ۲: کد گری برای رشته های سه بیتی : برای نوشتن کد کری برای رشته های ۴ بیتی باید جدول بالا را نسبت به خط آخر انعکاس داد و یک ستون متشکل از ۸ صفر و ۸ یک به سمت چپ آن اضافه کرد. این فرایند را می توان همین طور ادامه داد.



شکل ۳: گیت کنترلی $Toffoli$ که آن را با θ_3 نمایش می دهیم.



شکل ۴: چگونگی ساختن یک گیت θ_4 از گیت های θ_3. این روش برای ساخت گیت های θ_{n+1} از گیت θ_n بکار می رود.

حال توابع σ_1 تا σ_7 را به عنوان مولدهای گروه جایگشت های روی 8 شی را باید در نظربگیریم. تمام توابع وارون پذیر سه بیتی از ترکیب این ۷ تابع ساخته می شوند. تابع σ_i تابعی است که سطر i ام و $i+1$ ام را در جدول 3.1 با یک دیگر عوض می کند . خواننده می تواند براحتی بیازماید که این توابع به شکل زیرهستند:

$$\sigma_1(x,y,z)=(x,y,z\oplus\overline{xy}),\qquad \sigma_2(x,y,z)=(x,\overline{x}z\oplus z,z),\qquad \sigma_3(x,y,z)=(x,y,z\oplus\overline{x}y),\quad etc. \qquad (۲۲)$$

منطق استدلال روشن است و نیازی به توضیح اضافه ندارد. بنابراین با داشتن تابع θ_3 و NOT، می توان تمام توابع وارون پذیر سه بیتی را ساخت. به همین ترتیب خواننده می تواند دریابد که با در دست داشتن تابع θ_n و NOT می توان تمام توابع وارون پذیر n بیتی را ساخت. تنها کاری که باقیمانده است آن است که نشان دهیم تمام تابع های θ_n را می توان از تابع θ_3 ساخت. برای فهم این استدلال نیز کافی است به شکل 4 نگاه کنیم.

به این ترتیب نشان داده‌ایم که گیت های θ_3 و NOT برای ساختن تمام توابع وارون پذیر کافی هستند. بحث ما در باره مدارهای برگشت پذیر

کلاسیک دراین جا به پایان می رسد.

۴ تمرین ها:

- تمرین : اگر بخواهیم گیت θ_n را بسازیم حساب کنید که چند تا گیت θ_3 باید مصرف کنیم.

- تمرین: تابع برگشت پذیر $f : B_4 \longrightarrow B_4$ کار زیر را انجام می دهد:

$$f(s_1, s_2, s_3, s_4) = (s_4, s_3, s_2, s_1). \tag{۲۳}$$

با استفاده از گیت های برگشت پذیر این تابع را بسازید.

- تمرین: تابع برگشت پذیر $f : B_5 \longrightarrow B_5$ کار زیر را انجام می دهد:

$$f(s_1, s_2, s_3, s_4, s_5) = (s_5, s_1, s_2, s_3, s_4). \tag{۲۴}$$

با استفاده از گیت های برگشت پذیر این تابع را بسازید.

- تمرین: توابع وارون پذیر چهاربیتی را در نظر بگیرید. این توابع گروه جایگشت S_{16} را تشکیل می دهند. مولدهای این گروه عبارتند از $\sigma_1,\ \sigma_2,\ \cdots\ \sigma_{15}$. با نوشتن کد گری تابع های σ_3 و σ_7 را برحسب گیت های یونیورسال وارون پذیر بنویسید.

■ تمرین : تابع زیر را در نظر بگیرید:

$$f(x,\ y,\ z,\ w) = (x,\ y,\ z \oplus (xy),\ w \oplus (x+y)). \quad (۲۵)$$

دقت کنید که در این رابطه نماد های ضرب و جمع را برای «و» و «یا» ی منطقی بکار برده ایم. نخست ثابت کنید که این تابع وارون پذیر است. سپس با استفاده از گیت های یونیورسال برگشت پذیر یعنی

$$\{NOT, CNOT, Toffoli\}$$

مداری رسم کنید که این تابع را محاسبه کند.

■ تمرین : به شکل (۴) توجه کنید. این که مجبور شده ایم بیت سوم را برابر با 0 بگیریم ممکن است در نظر شما یک نقص به نظر آید. برای اصلاح این امر، این بیت را برابر با مقدار دلخواه t بگیرید. حال خروجی آخرین بیت ، یعنی پایین ترین بیت را بدست آورید. این خروجی مسلما آن چیزی نیست که ما از یک گیت θ_4 انتظار داریم. ولی می توانیم با افزودن یک گیت θ_3 به مدار فوق این اشکال را رفع کنیم و کاری کنیم که خروجی واقعا مطابق انتظار ما باشد. تعیین کنید که چگونه این کار را باید انجام دهیم.