مقدمه‌ای تورینگی بر محاسبه‌پذیری•

کاوه لاجوردی*

مسئلهٔ مشهور به *Entscheidungsproblem* [مسئلهٔ تصمیم] را هیلبرت در دههٔ ۱۹۲۰ مطرح کرد. خواستهٔ مسئله به‌دست دادنِ دستورالعملِ محاسباتی‌ای بود که در موردِ هر فرمولِ منطقِ محمولاتِ مرتبهٔ اول به ما اطلاع بدهد که آن فرمول اثبات‌پذیر هست یا نه. در ۱۹۳۶چرچ و تورینگ مستقل از هم استدلال کردند که چنین دستورالعملی وجود ندارد.
علی‌الظاهر برای اثباتِ وجود نداشتنِ دستورالعملی محاسباتی لازم است با تعریفِ دقیقی از مفهومِ محاسبه‌پذیری کار کنیم (تورینگ در ضمیمهٔ مقاله‌اش اثبات می‌کند که تعریف‌های او و چرچ مصداقاً معادل‌اند)؛ برجستگیِ مقالهٔ تورینگ در تحلیلِ او از مفهومِ محاسبه‌پذیری است ــ اهمیتِ کارِ تورینگ فراتر از حلِ مسئلهٔ هیلبرت است.
در این مقالهٔ توصیفیْ بعضی ایده‌های مقالهٔ کلاسیکِ تورینگ ([1]) را مرور می‌کنیم.

بعضی مسائل را می‌شود بدونِ نیاز به خلاقیت حل کرد ــ یعنی برای حلِ بعضی مسائل الگوریتم داریم. مثلاً الگوریتمِ خیلی قدیمی‌ای هست (**الگوریتمِ اقلیدسی**) برای پیداکردنِ بزرگ‌ترین مقسوم‌علیهِ مشترک. یک نمونهٔ اجرای این الگوریتم:

$$(۴, ۱۰) = (۴, ۶) = (۴, ۲) = (۲, ۲) = (۲, ۰) = ۲.$$

در هر مرحله معلوم است که باید چه کنیم (بیشینهٔ دو عدد را با تفاضل‌شان جایگزین می‌کنیم و وقتی یکی از دو عدد صفر شد کار را متوقف می‌کنیم و عددِ دیگر را به عنوانِ ب‌م‌مِ دو عدد اعلام می‌کنیم)، و این کار به طرزی کاملاً «مکانیکی» انجام‌شدنی است.

مثالی دیگر از مسئله‌ای که برایش الگوریتم داریم مسئلهٔ اثبات‌پذیری در منطقِ گزاره‌ها است: یک الگوریتمِ مشهور در اینجا چیزی است که به **جدولِ صدق** مشهور است.

ممکن است برای حلِ نوعِ خاصی از مسئله الگوریتم داشته باشیم یا نداشته باشیم؛ اما می‌شود در حالتِ کلّی پرسید که آیا برای همهٔ انواعِ مسائلْ الگوریتم داریم یا نه. توجه کنید که در این بحثِ خاص ــ حل‌پذیریِ الگوریتمی ــ به پیچیدگیِ محاسبه کاری نداریم. چیزی که برایمان مهم است صرفِ وجودِ الگوریتم است: می‌خواهیم بدانیم که آیا «روالی مکانیکی» هست که مسئله را حل کند یا نه؛ فرض می‌کنیم که در موردِ زمانِ محاسبه یا حافظهٔ موردِ نیاز برای اجرای الگوریتم محدودیتی نداریم. مثلاً الگوریتمِ جدولِ صدق متضمنِ صرفِ زمانی است که تابعی نمایی از تعدادِ گزاره‌های اتمی است؛ با این حال، اگر زمان و حافظه برایمان مهم نباشد، می‌دانیم که برای تعیینِ قضیه‌بودن یا نبودنِ فرمول‌های منطقِ گزاره‌ها راه‌حلی مکانیکی داریم. مثالی دیگر که شاید تعجب‌برانگیز باشد این است که تارسکی در دههٔ ۱۹۴۰ با صوری‌سازیِ مناسبی ثابت کرد که هندسهٔ مسطحهٔ اقلیدسی تصمیم‌پذیر است، به این معنا که الگوریتمی هست که هر حکمی دربارهٔ هندسهٔ مسطحه را به آن بدهیم نهایتاً به‌درستی به ما می‌گوید که آن حکم قضیه هست یا نه. دانش‌آموزان نوعاً از وجودِ چنین الگوریتمی خبر ندارند و مسئله‌هایشان را بعضاً خلاقانه حل می‌کنند؛ اما قضیهٔ تارسکی می‌گوید که چنین الگوریتمی وجود دارد. (البته زمانِ این الگوریتم هم چندجمله‌ای نیست؛ بنا بر این معقول نیست که دانش‌آموزان به‌جای «فهمیدنِ» درسِ هندسه‌شان سعی کنند در امتحان از این الگوریتم استفاده کنند!)

کمی تعمیم بدهیم. می‌گوییم مجموعهٔ A **تصمیم‌پذیر** است اگر الگوریتمی وجود داشته باشد که به ازای هر x به‌درستی به ما بگوید که x در A هست یا نه. برای بحثِ ما، می‌شود فرض کرد که همیشه با Aهایی سروکار داریم که زیرمجموعهٔ مجموعهٔ اعدادِ طبیعی‌اند: در مسائلِ موردِ نظرِ

• این مقاله مبتنی است بر سخنرانی نویسنده در همایش «ذهن، منطق، و محاسبه: یکصدمین سالروز تولد الن تورینگ» در سوم تیرماه ۱۳۹۱ در مؤسسهٔ پژوهشی حکمت و فلسفهٔ ایران.

* پژوهشکدهٔ فلسفهٔ تحلیلی.

ما، نوعاً با کدگذاری‌های سرراستی می‌شود هر فرمول را به طرزِ منحصر به‌فرد و کارآمدی با عددی طبیعی متناظر کرد.

حالتی که بیشتر موردِ توجهِ ما است این است که مجموعه‌ای از اصولِ یک نظریهٔ ریاضی داریم و A مجموعهٔ قضیه‌های آن جمله‌ها است (یعنی مجموعهٔ جملاتی که بر اساسِ آن اصول قابلِ اثبات‌اند). این مسئله‌ای بود که هیلبرت در دههٔ ۱۹۲۰ به آن علاقه‌مند بود، به‌ویژه برای اصولِ منطقِ مرتبهٔ اول:

حالتِ خاص: مسئلهٔ هیلبرت [*Entscheidungsproblem*]. الگوریتمی به‌دست دهید که به ازای هر فرمولِ داده‌شدهٔ منطقِ محمولاتِ مرتبهٔ اول، مشخص کند که آن فرمولْ قضیه‌ای از منطق هست یا نه.

(استطراداً: مسئلهٔ دهمِ هیلبرت هم ــ مطرح‌شده در پایانِ قرنِ نوزدهم ــ یک مسئلهٔ تصمیم است: آیا الگوریتمی هست که در موردِ هر معادلهٔ دیوفانتی اطلاع بدهد که جواب دارد یا نه؟)

نظریه‌های ریاضی‌ای (یعنی مجموعه‌هایی از اصول) می‌شناسیم که تصمیم‌پذیرند. غیر از هندسهٔ مسطحهٔ اقلیدسی، به اینها هم می‌شود اشاره کرد: نظریهٔ ترتیب‌های خطیِ چگالِ بدونِ ابتدا و بدونِ انتها، جبرهای بولیِ بدونِ اتم، گروه‌های آبلی‌ای که همهٔ اعضایشان از مرتبهٔ اولِ مفروضی باشند. اما آیا همهٔ نظریه‌های ریاضی تصمیم‌پذیرند؟ آیا مسئلهٔ خاصِ هیلبرت حل‌شدنی است؟ آیا الگوریتمی هست که هر فرمولِ منطقِ مرتبهٔ اول را که به آن بدهیم به ما اطلاع بدهد که آن فرمول قضیه هست یا نه؟ (احتمالاً از درسِ مقدماتیِ منطق یادمان هست که یک آزمونِ مثبت برای اعتبارِ منطقی وجود دارد ــ یعنی روالی مکانیکی هست که اگر به نتیجه برسد نشان می‌دهد که فرمولِ داده‌شده منطقاً معتبر است. پس سؤالی معادلْ این خواهد بود که آیا آزمونِ منفی‌ای هم وجود دارد یا نه.)

نوعاً راحت‌تریم که به‌جای تصمیم‌پذیریِ مجموعه‌ها از محاسبه‌پذیریِ تابع‌ها صحبت کنیم، و اینها به نحوی طبیعی به هم مربوط هستند: مجموعهٔ A تصمیم‌پذیر است اگر و فقط اگر الگوریتمی برای محاسبهٔ **تابع مشخصهٔ** A وجود داشته باشد ــ یعنی اگر و فقط اگر این تابعْ محاسبه‌پذیر باشد: تابعی که در x مقدارش ۱ است اگر x در A باشد و ۰ است اگر x در A نباشد. بحث‌مان را به تابع‌هایی محدود می‌کنیم که ورودی‌هایشان از اعدادِ طبیعی می‌آید.

اگر کسی بیاید و ادعا کند الگوریتمی برای حلِ مسئله‌ای دارد، الگوریتم‌اش را بررسی می‌کنیم. برای این کار لازم نیست دقیقاً بتوانیم مفهومِ کلّـیِ الگوریتم را تعریف کنیم. اما چه باید بکنیم اگر ــ مثلاً بعد از ناکامی‌های بسیار در یافتنِ الگوریتمِ مطلوب ــ بخواهیم این حدس را بررسی کنیم که الگوریتمِ مطلوبِ هیلبرت وجود ندارد؟ در اینجا به نظر می‌رسد که باید بتوانیم تعریفی از الگوریتم، یا شاید تعریفی از محاسبه‌پذیری، به‌دست بدهیم. تورینگ در مقاله‌اش چنین کاری می‌کند.

در محاسبه چه می‌کنیم؟ به نظر می‌رسد که این کارها را (و فقط اینها را): یادداشت می‌کنیم، به یادداشت‌هایمان نگاه می‌کنیم، بعضی نوشته‌هایمان را دستکاری می‌کنیم، و در همهٔ این کارها از دستورهای بی‌ابهامی پیروی می‌کنیم. اینها را اگر مجرد کنیم می‌رسیم به تعریفِ ماشینِ تورینگ.

تورینگ فرض می‌کند که چیزی که رویش یادداشت می‌کنیم نواری یک‌بُعدی است که به خانه‌هایی تقسیم شده، و تقریباً روشن است که این محدودیتْ جدی نیست [دست‌کم روشن است برای هر کس که دیده باشد که نقاطِ با مختصاتِ صحیح در صفحه در تناظرِ ۱ـ۱ با مجموعهٔ عددهای صحیح‌اند]. و فرض می‌کند که مجموعهٔ نمادهایی که می‌نویسیم متناهی است ــ توضیحِ تورینگ این است که اگر این مجموعه متناهی نمی‌بود آنگاه نمادهایی می‌بودند که هر اندازه بخواهیم به هم شبیه‌اند (در پاورقی‌ای محکی برای سنجشِ شباهتِ دو نماد به دست می‌دهد)، که علی‌القاعده اتفاقِ نامطلوبی است. اما تذکر می‌دهد که این فرض محدودیتِ جدی‌ای ایجاد نمی‌کند چرا که می‌شود دنباله‌هایی از نمادها را چونان یک تک‌نماد لحاظ کرد ــ مثلاً «۹۹۹۹۹۹۹۹۹۹۹۹» را یک نماد می‌انگاریم. و می‌گوید که نمادهای مرکب را، اگر خیلی طولانی باشند، نمی‌توان در یک نگاه تشخیص داد ــ مثلاً فرقِ اینها را می‌توانید ببینید؟: «۹۹۹۹۹۹۹۹۹۹۹» و «۹۹۹۹۹۹۹۹۹۹۹۹».

به علاوه، فرض می‌کند که کرانِ بالایی هست برای تعدادِ نمادهایی که محاسبه‌کننده می‌تواند در هر زمان مشاهده کند ــ اگر بیشتر بخواهد ببیند باید مشاهداتِ مکرر انجام بدهد. نیز، فرض می‌کند که تعدادِ وضعیت‌های محاسبه‌گر متناهی است، و دلیل‌اش همان است که در موردِ نمادها می‌گوید: اگر تعدادی نامتناهی وضعیتِ ذهنی باشد، بعضی از آنها آن‌قدر به هم شبیه‌اند که نمی‌شود بین‌شان فرق گذاشت.

در هر عملی که محاسبه‌گر انجام می‌دهد حداکثر یک نماد را تغییر می‌دهد. بدونِ کم‌شدن از کلّـیت می‌شود فرض کرد که نمادی که تغییر می‌کند همانی است که دارد مشاهده می‌شود.

تورینگ برای توجیهِ تعریف‌اش سه کار می‌کند: تحلیلِ مفهومِ شهودیِ محاسبه‌پذیری (که به‌اجمال گفتم)، اثباتِ یکی‌بودنِ تعریفِ خودش با تعریفِ دیگری که چرچ در همان سال مطرح کرده بود (در این مورد و برای بحثِ تاریخ [4] را ببینید)، و اثباتِ اینکه ردهٔ بزرگی از توابعِ شهوداً محاسبه‌پذیر محاسبه‌پذیرِ تورینگی هم هستند.

به‌دست‌دادنِ تعریفی صوری آسان است؛ بیایید در سطحی غیرصوری کار کنیم. هر **ماشینِ تورینگ** تشکیل شده است از این چیزها: یک نوار که به خانه‌هایی تقسیم‌بندی شده است، و طول‌اش بالقوه نامتناهی است ــ مثلاً فرض کنیم نوارمان کاغذی است، و می‌توانیم هر قدر لازم شد به هر طرف‌اش که خواستیم کاغذِ اضافه بچسبانیم. ماشین در هر زمان/مرحله فقط یکی از این خانه‌ها را می‌بیند. روی هر خانه در هر زمان یا نوشته شده «۰» یا نوشته شده «۱» (و نه هر دو). بخشی از ماشین که نوار را می‌خوانَد در هر زمان ــ مطابقِ دستوراتی که از قبل تعیین شده ــ یک خانه به طرفِ راست یا چپ می‌رود، یا نوشتهٔ خانه‌ای که دارد می‌بیند را تغییر می‌دهد. نیز، مطابقِ دستورهای از قبل تعیین‌شده، ماشین وضعیت‌اش را تغییر می‌دهد یا نمی‌دهد. کاری که ماشین در هر زمان/مرحله انجام می‌دهد با اینها

مشخص می‌شود: وضعیتِ فعلیِ ماشین، خانه‌ای از نوار که دارد خوانده می‌شود، و دستوراتِ (یا برنامهٔ) ماشین. هر دستور چیزی به این شکل است:

$$s_c\text{-}Symbol\text{-}s_n\text{-}A$$

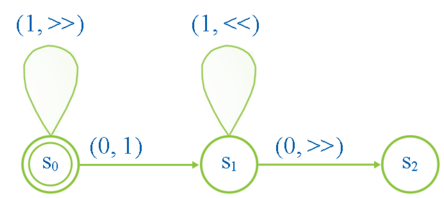
که در اینجا s_c وضعیتِ فعلیِ ماشین است. یک وضعیتِ مشخصِ s_0 هست که کارِ ماشین همیشه با آن شروع می‌شود. اگر ماشین به وضعیتی برسد که هیچ دستوری با آن وضعیت شروع نشود ماشین متوقف می‌شود؛ هم‌چنین است اگر ماشین به وضعیتی برسد که در آن لازم باشد چند دستور را اجرا کند که با هم در تعارض‌اند. (شهوداً، وضعیت‌ها همان شمارهٔ سطرهای الگوریتمی است که برنامه برای اجرایش نوشته شده.) $Symbol$ نمادی است که ماشین دارد می‌خواند (که یا «۰» است یا «۱»). s_n وضعیتِ بعدیِ ماشین است. عملِ A یکی از اینها است: تبدیلِ «۱» به «۰»، تبدیلِ «۰» به «۱»، حرکت به خانهٔ راستِ خانهٔ فعلی، حرکت به خانهٔ چپِ خانهٔ فعلی. هر **برنامه** متشکل است از تعدادی متناهی از دستورات.

مثلاً این یک برنامه است:

$$s_0\text{-}1\text{-}s_0\ {>>},\ s_0\text{-}0\text{-}s_1\text{-}1,$$
$$s_1\text{-}1\text{-}s_1\text{-}\ {<<},\ s_1\text{-}0\text{-}s_2\text{-}\ {>>}\ .$$

این برنامه چه می‌کند؟ با وضعیتِ s_0 شروع می‌کند. اگر خانه‌ای که دارد می‌خواند حاویِ ۱ بود، در همان وضعیتِ s_0 می‌مانَد و می‌رود سراغِ خانهٔ سمتِ راستِ آن. در وضعیتِ s_0 اگر خانه‌ای که دارد می‌خواند حاویِ ۰ باشد در آن خانه می‌نویسد ۱، و وضعیت‌اش را به s_1 تغییر می‌دهد. در وضعیتِ s_1 اگر خانه‌ای که می‌خواند حاویِ ۱ بود وضعیت را عوض نمی‌کند و به چپ می‌رود. در وضعیتِ s_1 اگر چیزی که می‌خواند ۰ باشد یکی می‌رود به راست و واردِ s_2 می‌شود. می‌بینید که ابهامی در کار نیست، و دنبال‌کردنِ برنامه هم خلاقیتی نمی‌طلبد.

روشِ گویاتری برای نمایشِ این ماشین این است (شکل از [2]):



***

هر عددِ طبیعیِ n را با $(n+1)$ نمادِ «۱» که پشتِ سرِ هم (یعنی در خانه‌های متوالیِ نوار) نوشته شده نشان می‌دهیم.

می‌گوییم ماشین تورینگِ داده‌شده‌ای تابعِ nـ‌متغیرهٔ f را **محاسبه می‌کند** اگر:

اولاً به ازای هر x_1، ...، x_n که f برایش تعریف شده این اتفاق بیفتد: روی نوارْ نمایشِ x_1 را طبقِ قراردادِ بالا می‌نویسیم. بعد (یعنی در سمتِ راستِ آخرین «۱») یک «۰» می‌گذاریم، بعد نمایشِ x_2 را می‌نویسیم و در سمتِ راست‌اش یک «۰» می‌گذاریم، و به همین ترتیب تا x_n. روی بقیهٔ خانه‌های نوار فقط «۰» است. ماشین در وضعیتِ ابتدایی روی اولین «۱» از سمتِ چپ است. ماشین سرانجام متوقف می‌شود در حالی که روی نوار تعدادی «۱» است که بین‌شان هیچ «۰»ای نیست. بقیهٔ نوار «۰» است. ماشین دارد اولین «۱» از سمتِ چپ را می‌خواند. و این دنباله از «۱»ها نمایشِ مقدارِ تابع به ازای x_1، ...، x_n است.

ثانیاً اگر f به ازای ورودی‌ای تعریف نشده باشد، ماشین وقتی به شرحِ بالا با آن ورودی شروع کند هرگز متوقف نشود، یا به شکلی غیر از آنچه در بالا گفتیم متوقف بشود.

توضیحِ آدمیزادپسندی با مثال. اگر ماشینی قرار باشد تابع جمع را محاسبه کند باید از جمله این اتفاق بیفتد (برای محاسبهٔ ۱ + ۲): ماشین اگر شروع کند از اولین «۱» در سمتِ چپ از این نوار:

...۰۰۰۰۰۰۰۱۱۱۰۱۱۰۰۰۰۰ ...

ختم کند به اولین «۱» از سمتِ چپ در این نوار:

...۰۰۰۰۰۰۰۱۱۱۱۰۰۰۰۰ ...

تابعی را **محاسبه‌پذیرِ تورینگی** می‌گوییم که ماشینِ تورینگی وجود داشته باشد که محاسبه‌اش کند. ماشینی که مثال زدم تابعِ تالی را محاسبه می‌کند (تابعی تک‌متغیره که به عددِ ورودیْ یکی اضافه می‌کند و به عنوانِ خروجی تحویل می‌دهد).

تقریباً روشن است که شکلِ قرارداد مهم نیست؛ مهم این است که قراری بگذاریم که ورودی‌ها و خروجی‌های ماشین را چطور تعبیر کنیم. چیزهای دیگری هم هست که مهم نیست، اما مهم‌نبودن‌شان بعضاً خیلی آشکار نیست: نوار می‌تواند دوبعدی باشد یا نباشد، می‌شود جلورفتنِ نوار خانه‌به‌خانه نباشد (و از هر خانه‌ای بتوان به هر خانه‌ای پرید)، می‌شود ماشین در آنِ واحد هم حرکت کند و هم نوشته‌ای را تغییر بدهد، تعدادِ نمادهایی که ماشین در اختیار دارد می‌شود هر عددِ متناهی‌ای باشد — اینها ردهٔ توابعِ محاسبه‌پذیرِ تورینگی را تغییر نمی‌دهد (گرچه می‌تواند تعدادِ مراحلِ لازم برای محاسبه را تغییر بدهد). حتی می‌شود در مراحلی تصادفی کارکرد: ماشین به‌تصادف از حالتی به حالتِ بعدی برود — باز هم ردهٔ توابعِ محاسبه‌پذیرِ تورینگی همان خواهد بود.

***

جوری که تا اینجا توضیح دادیم، هر ماشینِ تورینگ حداکثر یک تابع را محاسبه می‌کند (گرچه ماشین‌های تورینگِ متفاوتی می‌توانند تابعِ واحدی را محاسبه کنند). این فرقِ بزرگی است با کامپیوترهای امروزی که می‌شود برنامه‌ریزی‌شان کرد برای محاسبهٔ تابع‌های مختلف. تورینگ در بخشِ ششم

مقاله‌اش صحبت از ماشینِ جامع می‌کند ــ چیزی که امروز به آن می‌گویند **ماشینِ جامعِ تورینگ**: ماشینِ تورینگی که به آن می‌گوییم که به ما بگوید که فلان ماشینِ تورینگ اگر روی بهمان ورودی کار کند نتیجه چه خواهد بود (و ماشین هم به ما جوابِ درست می‌دهد!). اما چطور می‌شود دربارهٔ ماشینِ تورینگی با ماشینِ تورینگِ دیگری صحبت کرد؟

نکته این است که روش‌های کارآمدی برای متناظرکردنِ هر ماشینِ تورینگ با عددی طبیعی وجود دارد. واردِ جزئیاتِ هیچ‌کدام از این روش‌ها نمی‌شوم؛ فقط به عنوانِ مثال بگویم که یکی از این روش‌ها ماشینی که برای تابعِ تالی کار می‌کند را متناظر می‌کند با این عدد (مطابقِ [2]):

$$\underline{۱۰۱۱۰۱۰۱۱۱۱}۰۰\underline{۱۰۱۰۱۱۰۱۱}۰۰\underline{۱۱۰۱۱۰۱۱۰۱۱۱}۰۰\underline{۱۱۰۱۰۱۱۱۰۱۱۱۱}.$$

چنین عددی را می‌توان شمارهٔ سریالِ ماشینِ ۱+ دانست. (هرکدام از بخش‌هایی از نمایشِ این عددِ غول‌آسا که زیرش خط کشیده‌ام متناظر است با یکی از دستورهای برنامه.) ماشینِ جامعْ این عدد و عددِ دیگری را می‌گیرد، و عددِ دوم را با یک جمع می‌کند و نتیجه را به شکلِ استاندارد تحویل می‌دهد. در حالتِ کلّـی، ماشینِ جامع از روی شمارهٔ سریالْ ماشینِ مربوط را «بازسازی می‌کند» و بعد برنامهٔ ماشینِ بازسازی‌شده را اجرا می‌کند.

در برخوردی امروزی با موضوع، وجودِ ماشینِ جامع نتیجهٔ ساده‌ای از قضیهٔ مهمی به اسمِ قضیهٔ صورتِ نرمال است که کلینی در ۱۹۳۸ منتشر کرده است. نکتهٔ جالب این است که تورینگ توصیفِ کاملی از برنامهٔ ماشینِ جامع به‌دست می‌دهد ــ سه صفحه است و خواندن‌اش سخت است (و، چنان که امیل پُست توضیح داده ([6])، در توصیف‌اش اشتباهاتی هم هست).

* * *

یادمان نرفته که می‌خواستیم تکلیفِ مسئلهٔ هیلبرت را روشن کنیم! تورینگ توضیح می‌دهد که چیزی که دارد اثبات می‌کند با قضیهٔ مشهورِ گودل متفاوت است. و توضیح می‌دهد که اگر به ازای هر حکمِ φ، یا φ اثبات‌پذیر می‌بود یا $\neg\varphi$، آنگاه به‌راحتی روالی برای حلِ مسئلهٔ هیلبرت به‌دست می‌آمد: ماشینی راه می‌انداختیم که شروع کند به اثباتِ همهٔ قضایا. فرض کنید فرمولی داده شده است که می‌خواهیم ببینیم قضیه هست یا نه. دیر یا زود، یا خودِ فرمول را می‌دیدیم یا نقیض‌اش را. در حالتِ اول می‌گفتیم «بله»، در حالتِ دوم «خیر». (در حالتِ کلّـی، اگر مجموعهٔ اصول‌مان اولاً کامل باشد ــ یعنی هر حکمِ قابلِ بیان در زبانِ اصول را یا اثبات کند یا رد کند ــ و ثانیاً در موردِ هر حکمی بتوانیم به روش مکانیکی بفهمیم که جزوِ اصول‌مان هست یا نه، آنگاه مجموعهٔ قضیه‌ها تصمیم‌پذیر است.)

حالا چه کنیم که این‌طور نیست که هر حکمی یا خودش قضیه باشد یا نقیض‌اش؟ تورینگ توضیح می‌دهد که ایده سرراست است. یک راهِ طبیعیِ رسیدن به این اثبات توجه به چیزی است که شاید در نگاهِ اول به نظر برسد که نشان می‌دهد تعریفِ تورینگ از محاسبه‌پذیری تعریفِ خوبی نیست. کمی تجربهٔ ریاضی با مجموعه‌های شمارا به ما امکان می‌دهد نشان دهیم که مجموعهٔ همهٔ ماشین‌های تورینگْ شمارا است.* به‌علاوه، همان تجربهٔ اندک نشان‌مان می‌دهد که می‌شود این شمارش را به‌طورِکارآمدی انجام داد: نه فقط همهٔ ماشین‌های تورینگ را فهرست می‌کنیم، بلکه این کار را چنان می‌کنیم که اگر به ما nای بدهند می‌توانیم با دنبال‌کردنِ دستوراتِ سرراستی و بدونِ نیاز به خلاقیت بگوییم که nاُمین ماشینِ تورینگ دقیقاً چیست.

با در دست داشتنِ چنین شمارشی، بیایید تابعِ fای را به این صورت تعریف کنیم: اگر x عددی طبیعی باشد، xاُمین ماشینِ تورینگ را پیدا می‌کنیم. این ماشین تابعی را محاسبه می‌کند که معقول است اسم‌اش را «f_x» بگذاریم. اگر این ماشین با ورودیِ x متوقف نشود آنگاه $f(x)$ را ۱۳ تعریف می‌کنیم؛ اگر ماشین با ورودیِ x متوقف بشود آنگاه $f(x)$ را $f_x(x) + ۱۳$ تعریف می‌کنیم. نحوهٔ ساختْ نشان می‌دهد که f را هیچ ماشینِ‌تورینگی محاسبه نمی‌کند؛ پس شاید به نظر برسد که تعریفِ تورینگ تعریفِ خوبی نباشد چراکه شاید به نظر برسد که f شهوداً محاسبه‌پذیر است.

اما آیا تابعی که تعریف کردیم واقعاً شهوداً محاسبه‌پذیر است؟ نکته این است که اگرچه برای ورودیِ دلخواهِ f می‌توانیم ماشینِ مربوط را پیدا کنیم و ورودیِ مربوط را به آن بدهیم و اگر متوقف شد مقدارِ f را به‌دست بیاوریم، ولی معلوم نیست که چطور می‌شود فهمید که ماشینی متوقف نمی‌شود. نشان نداده‌ایم که f شهوداً محاسبه‌پذیر است مگر اینکه توانسته باشیم این مسئله را حل کنیم:

مسئلهٔ توقف. آیا این تابع محاسبه‌پذیر است؟: تابعی که به عنوانِ ورودی عددی طبیعی و «شمارهٔ سریال»ِ ماشینِ تورینگِ دلخواهی را می‌گیرد و خروجی‌اش ۱ یا ۰ است بر حسبِ اینکه آن ماشین با آن ورودی متوقف بشود یا نشود.

حالا می‌رسیم به

طرحِ کلّیِ تورینگ برای اثباتِ حل‌ناپذیریِ مسئلهٔ هیلبرت

۱. اگر الگوریتمِ مطلوبِ هیلبرت وجود داشته باشد آنگاه مسئلهٔ توقف حل‌پذیر است.

۲. مسئلهٔ توقف حل‌پذیر نیست.

مستقل از اینکه این تابعی که محاسبه‌پذیربودن‌اش موضوعِ مسئلهٔ توقف است شهوداً محاسبه‌پذیر باشد یا نه، می‌شود نشان داد که ماشینِ تورینگی وجود ندارد که محاسبه‌اش کند. فرض کنید چنین ماشینی وجود داشته باشد، مثلاً ماشینِ `Halt`. ماشینِ `Copy` را در نظر بگیرید که یک ورودی

* هر ماشین متناظر است با برنامه‌اش، که این هم متناظر است با دنباله‌ای متناهی از نمادها. از اینجا فوراً معلوم می‌شود که این‌طور نیست که همهٔ تابع‌ها محاسبه‌پذیرِ تورینگی باشند چراکه مجموعهٔ تابع‌های از اعدادِ طبیعی به اعدادِ طبیعی ناشمارا است. مجموعهٔ زیرمجموعه‌های مجموعهٔ اعدادِ طبیعی هم ناشمارا است؛ لذا این‌طور هم نیست که همهٔ زیرمجموعه‌های اعدادِ طبیعیْ تصمیم‌پذیر باشند. همین ملاحظات نشان می‌دهد که حتی بیشترِ زیرمجموعه‌های اعدادِ طبیعی تصمیم‌ناپذیرند. اما، تا اینجای این بحث، هنوز نمی‌دانیم که آیا این مجموعه‌ی خاصِ موردِ نظرِ ما ــ مجموعه‌ی قضیه‌های منطقِ مرتبه‌ی اول ــ تصمیم‌ناپذیر است یا نه.

می‌گیرد، و وقتی متوقف می‌شود در روی نوار همان ورودی را نوشته و سمتِ راست‌اش یک «۰» گذاشته و بعد، در سمتِ راستِ این «۰»، دوباره همان ورودی را نوشته. و ماشینِ Not را در نظر بگیرید که اگر ورودی‌اش چیزی غیر از «۰» باشد هرگز متوقف نمی‌شود و اگر ورودی‌اش «۰» باشد متوقف می‌شود و جوابِ «۱» می‌دهد. حالا ماشینی بسازید که وقتی ورودی را می‌گیرد (این ورودی فقط یک تک‌عدد است) اول کارهای Copy را رویش انجام می‌دهد، بعد کارهای Halt را روی نتیجهٔ Copy، و نهایتاً کارهای Not را روی خروجیِ Halt. با فرضِ وجودِ Halt می‌شود این ماشینِ ترکیبی را دقیقاً مشخص کرد. به این ماشینِ ترکیبی بگوییم M. [یعنی اسم‌اش را بگذاریم «M»، نه اینکه چنین خطاب‌اش کنیم!] فرض کنیم شمارهٔ سریالِ این ماشین m باشد. اگر m را به عنوانِ ورودی بدهیم به M چه می‌شود؟ طبقِ روشِ ساخت، M با این ورودی متوقف می‌شود اگر و فقط اگر ماشینِ با شمارهٔ m با ورودیِ m متوقف نشود. اما شمارهٔ M همان m است؛ پس M با این ورودی متوقف می‌شود اگر و فقط اگر با این ورودی متوقف نشود. این ــ با شرمندگی ــ تناقض است. پس M وجود ندارد. پس Halt وجود ندارد.

این اساساً کاری است که تورینگ در بخشِ هشتمِ مقاله‌اش («کاربردهای فرآیندِ قطری‌سازی») انجام می‌دهد ــ البته در آنجا صحبت از وجودِ ماشینی است که به ما بگوید ماشینِ داده‌شده با ورودیِ داده‌شده آیا هرگز «۰» در خروجی‌اش ظاهر می‌شود یا نه؛ اما ایده ــ و مسئله ــ اساساً همان است.

در بخشِ یازدهِ مقاله تورینگ چیزی که نشان می‌دهد عملاً این است (با تقریرِ [3]) که به ازای هر ماشینِ تورینگ و هر عددِ طبیعیِ n روشِ کارآمدی برای ساختنِ یک مجموعهٔ متناهیِ Δ از جملات و یک جملهٔ H هست که H در Δ اثبات‌پذیر است اگر و فقط اگر آن ماشین با ورودیِ n توقف کند. ساختنِ Δ و H اساساً ساده است. با داده‌شده‌بودنِ ماشینِ تورینگِ مشخصی، محمول‌هایی معرفی کنید برای نمایشِ وضعیت‌ها، و با کمکِ تابعِ تالی و با فرضِ شماره‌داشتنِ خانه‌های نوار، محمول‌هایی معرفی کنید که بگوید ماشین در هر مرحله در چه وضعیتی است و چه نمادی را دارد می‌خواند. اینها Δ را می‌سازد. جملهٔ H می‌گوید که ماشین اگر ورودی‌اش n باشد بالاخره متوقف می‌شود. نوشتنِ اینها، و اثباتِ اینکه ماشینِ داده‌شده با ورودیِ n متوقف می‌شود اگر و فقط اگر H در Δ اثبات بشود، تمرینِ دلپذیری است.

حالا اگر الگوریتمِ مطلوبِ هیلبرت وجود می‌داشت، با آن ماشینِ Halt را می‌ساختیم، که می‌دانیم ساختنی نیست. پس الگوریتمِ مطلوبِ هیلبرت وجود ندارد. (یادآوری: Δ جملهٔ H را اثبات می‌کند اگر و فقط اگر این جملهٔ شرطیْ یک قضیهٔ منطق باشد: جمله‌ای شرطی که تالی‌اش H است و مقدم‌اش ترکیبِ عطفیِ همهٔ جمله‌های Δ.)

توجه کنید که چه کرده‌ایم. این قضیه‌ای ریاضی است که ماشینِ Halt وجود ندارد. در مرحلهٔ بعدی نشان دادیم [یا: نشان می‌دادیم، اگر وقت داشتیم] که با فرضِ اینکه روالی برای تصمیم‌گیری در موردِ اعتبارِ منطقی وجود دارد، روالی برای این وجود دارد که مسئلهٔ توقف را حل کنیم. از اینجا نتیجه گرفتیم که در صورتی که الگوریتمِ مطلوبِ هیلبرت وجود می‌داشت آنگاه ماشینِ Halt وجود می‌داشت. این مطلبِ اخیر منوط است به اینکه هرگاه روالی برای پیداکردنِ مقادیرِ تابعی وجود داشته باشد ماشینِ تورینگِ متناظری وجود دارد ــ یعنی منوط است به اینکه همهٔ توابعِ شهوداً محاسبه‌پذیرْ محاسبه‌پذیرِ تورینگی‌اند. این ادعای اخیر چیزی است که به آن **بَرنَهادهٔ چرچ-تورینگ** می‌گویند.

مراجع

1. **A.M. Turing**, *On computable numbers, with an application to the Entscheidungsproblem*, Proceedings of the London Mathematical Society **42** (1936/37): 230-265. Corrections: ibid., **43** (1937), 544-546.
`http://www.turingarchive.org/viewer/?id=466&title=01b`
Reprinted in Martin Davis, ed., *The Undecidable: Basic Papers on Undecidable Problems and Computable Functions* (1965), Dover reprint, 2004.

2. **David Barker-Plummer**, *Turing machines*, Stanford Encyclopedia of Philosophy,
`http://plato.stanford.edu/entries/ turing-machine/`

3 **George S. Boolos and Richard C. Jeffrey**, *Computability and Logic*, 3rd ed., Cambridge University Press, 1989.

4. **Robin Gandy**, *The confluence of ideas in 1936*, in Rolf Herken, ed., The Universal Turing Machine: A Half-century Survey, Oxford University Press, 1988, pp. 55-111.

5. **Piergiorgio Odifreddi**, *Classical Recursion Theory*, North-Holland, 1989.

6. **Emil Post**, *Recursive unsolvability of a problem of Thue*, The Journal of Symbolic Logic **12** (1947), 1-11. Reprinted in Davis, op. cit.