نظریه‌ی زبان‌ها و ماشین‌ها

درس‌نامه‌ی ۱

کاظم فولادی
http://kazim.fouladi.ir

ویراست اول: ۱۳۸۵
ویراست دوم: ۱۳۸۷
ویراست سوم: ۱۳۹۳

فهرست مطالب

۱ مقدمه‌ای بر نظریه‌ی محاسبه

An Introduction to the Theory of Computation

۱-۱ مقدمه

۱-۱-۱ محاسبه چیست؟

محاسبه: **پردازش اطلاعات** بر اساس مجموعه‌ای متناهی از عملیات یا قواعد.

مثال‌هایی از محاسبه:

- حساب با قلم و کاغذ
- چرتکه
- خط‌کش و بکارگیری ساختارهای هندسی
- کامپیوترهای دیجیتال

دیگر دستگاه‌های محاسبه

- برنامه‌های C، JAVA، ...
- اینترنت و دیگر سیستم‌های توزیع‌شده
- واکنش‌های شیمیایی؟
- سلول‌ها / DNA ؟
- مغز انسان؟
- کامپیوترهای کوانتومی؟

۱-۱-۲ از نظریه چه می‌خواهیم؟

- عمومیت

- مستقل از تکنولوژی
- انتزاع: نادیده گرفتن جزئیات غیرضروری

• **دقت**

- ریاضی بودن، رسمی بودن
- امکان اثبات قضایا در مورد محاسبات (مثبت و منفی):
  چه چیزی می‌تواند محاسبه شود؟
  چه چیزی نمی‌تواند محاسبه شود؟

۱-۱-۳ مشخصه‌های مسائل محاسباتی

- **گسسته بودن (Discreteness)**: امکان بیان حالت سیستم با تعداد محدودی از اطلاعات
- **انتزاع (Abstraction)**: امکان نادیده گرفتن جزئیات
- **عمومیت (Generality)**: امکان اعمال یک مدل ریاضی واحد به تعداد زیادی از تکنولوژی‌ها

۱-۱-۴ حوزه‌های نظریه‌ی محاسبه

پرسش اساسی نظریه‌ی محاسبه، تعیین **حد محاسبه** است:
قابلیت‌های پایه و محدودیت‌های کامپیوتر چیست؟

نظریه‌ی محاسبه *Theory of Computation*	
پرسش پایه:	**حد محاسبه** **قابلیت**‌های پایه و **محدودیت**‌های پایه‌ی محاسبه چیست؟

نظریه‌ی محاسبه، سه حوزه‌ی نظری اصلی دارد:

آتوماتا، محاسبه‌پذیری و پیچیدگی

نظریه‌ی محاسبه *Theory of Computation*		
نظریه‌ی آتوماتا *Automata Theory*	**نظریه‌ی محاسبه‌پذیری** *Computability Theory*	**نظریه‌ی پیچیدگی** *Complexity Theory*

- نظریه‌ی پیچیدگی

  پرسش اساسی: **آیا مسائلی وجود دارند که به طور کارآمد قابل حل نباشند؟**

  چه چیزی باعث می‌شود که برخی از مسائل از نظر محاسباتی دشوار و برخی دیگر آسان باشند؟

نظریه‌ی پیچیدگی ⇐ دسته‌بندی مسایل محاسباتی به آسان و دشوار

چگونه می‌توانیم مسائل دشوار را حل کنیم؟

- **نظریه‌ی محاسبه‌پذیری**

  پرسش اساسی: **آیا مسائلی وجود دارند که با کامپیوتر قابل حل نباشند؟**

  نظریه‌ی محاسبه‌پذیری ⇐ دسته‌بندی مسائل محاسباتی به قابل حل و غیر قابل حل

- **نظریه‌ی آتوماتا**

  پرسش اساسی: **کامپیوتر چیست؟**

  نظریه‌ی آتوماتا ⇐ مدل‌های مختلف برای محاسبه

  نقطه‌ی شروع ما برای مطالعه‌ی نظریه‌ی محاسبه: نظریه‌ی آتوماتا و زبان‌ها

نظـــریه‌ی محاســـبـــه
Theory of Computation

نظریه‌ی آتوماتا *Automata Theory*	**نظریه‌ی محاسبه‌پذیری** *Computability Theory*	**نظریه‌ی پیچیدگی** *Complexity Theory*

پرسش پایه:

کامپیوتر چیست؟
مدل محاسبه

۱-۲ مقدمات ریاضی

۱-۲-۱ مجموعه‌ها

- **عضویت** در مجموعه‌ها ($x \in S$، $x \notin S$)
- روش‌های **نمایش مجموعه‌ها** (نمایش با اعضا، نمایش با نماد ریاضی)
- **مجموعه‌ی جهانی** (Universal set) و **مجموعه‌ی تهی** (Empty set) و خواص آن‌ها
- **عملگرهای مجموعه‌ای**: اجتماع، اشتراک، تفاضل، متمم
- **زیرمجموعه** بودن و **تساوی** مجموعه‌ها، زیرمجموعه‌ی محض، مجموعه‌های مجزا (disjoint)
- **مجموعه‌ی توانی** یک مجموعه (۲^S)
- مجموعه‌های **متناهی** (finite) و **نامتناهی** (infinite)

۱-۲-۲ رابطه و تابع

- **ضرب دکارتی** دو مجموعه
- **تابع** ($f : A \to B$): توابع تام (total) ($D_f = A$) و توابع جزیی (partial) ($D_f \subset A$)
- توابع یک به یک، توابع پوشا
- **رابطه**: انواع روابط (هم‌ارزی، ترتیب)

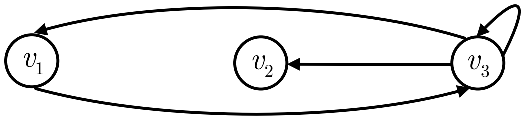
گراف‌ها و درخت‌ها

- **گراف** (رأس‌ها و یال‌ها)
- گراف ساده، گراف جهت‌دار، برچسب یال، برچسب رأس
- **گشت (walk)**: دنباله‌ای از یال‌های متصل به هم
- طول یک گشت: تعداد یال‌های گشت از مبدأ تا مقصد

- **مسیر (path)**: گشت بدون یال تکراری
- **مسیر ساده (simple path)**: مسیر بدون رأس تکراری
- **دور (چرخه) (cycle)**: یک گشت از رأس v به خودش بدون یال تکراری (دور با پایه‌ی v)
- **دور ساده (simple cycle)**: دور بدون رأس تکراری (بجز رأس پایه)
- **حلقه (loop)**: یک یال از یک رأس به خودش
- **درخت جهت‌دار (درخت)**: گراف جهت‌دار بدون دور و دارای یک رأس خاص به نام ریشه

مثال

گراف زیر را در نظر می‌گیریم:



یک گشت: $(v_۱, v_۳), (v_۳, v_۳), (v_۳, v_۱), (v_۱, v_۳), (v_۳, v_۲)$
یک مسیر: $(v_۱, v_۳), (v_۳, v_۲)$
یک دور: $(v_۱, v_۳), (v_۳, v_۳), (v_۳, v_۱)$

۱-۲-۳ روش‌های اثبات

روش‌های متداول برای اثبات قضایای ریاضی:

- **اثبات با استقرای ریاضی** (استقرای ضعیف، استقرای قوی)
- **اثبات با برهان خلف**
  برای اثبات $\alpha \Rightarrow \beta$ نشان می‌دهیم که $\alpha \wedge \neg\beta$ نادرست است.
- **اثبات با ساختن (proof by construction)**
  برای اثبات مواردی که هدف نشان دادن **وجود** یک شیئ است.

مثال

می‌خواهیم ثابت کنیم برای هر $n \geq ۳$، یک گراف ساده با n رأس وجود دارد که درجه‌ی تمام رئوس آن برابر با ۲ است.
اثبات: گراف G را به صورت زیر می‌سازیم و حکم ثابت است:

$$\begin{aligned} G &= (V, E) \\ V &= \{۱, ۲, ۳, \ldots, n\} \\ E &= \{(i, i+۱) : i = ۱, ۲, \ldots, n-۱\} \cup \{(n, ۱)\} \end{aligned}$$

۱-۳ مفاهیم بنیادی نظریه‌ی محاسبه

۱-۳-۱ مفهوم اول: زبان

الفبا

تعریف **الفبا (Alphabet).** یک مجموعه‌ی متناهی و ناتهی از نمادهای تجزیه‌ناپذیر

الفبا را معمولاً با Σ نمایش می‌دهیم. به هر عضو الفبا $a \in \Sigma$ یک نماد می‌گوییم.

مثال

هر یک از مجموعه‌های زیر یک الفبا را نشان می‌دهد:

$$\Sigma_1 = \{1, 2, 3, 4, \ldots, 9\}$$
$$\Sigma_2 = \{a, b, c, \ldots, z, A, B, C, \ldots, Z\}$$
$$\Sigma_3 = \{*, +, /, =\}$$

رشته‌ها

تعریف **رشته (String).** دنباله‌ای متناهی از نمادهای الفبا

مثال

هر یک از موارد زیر به ترتیب یک رشته روی الفبای Σ_1، Σ_2 و Σ_3 است:

$$w_1 = 12345$$
$$w_2 = aabbaaab$$
$$w_3 = +/*+**$$

- **الحاق دو رشته (concatenation):** اگر u و v دو رشته باشند، حاصل الحاق u و v رشته‌ای است که از قرار دادن v به دنبال u به صورت uv به دست می‌آید.

- **توان‌های یک رشته:** (بیانگر تکرار یک رشته)

$$w^{۱} = w, \qquad w^{n+۱} = w^n w$$

- **طول یک رشته:** طول رشته‌ی w که با $|w|$ نشان داده می‌شود، برابر با تعداد نمادهای آن تعریف می‌شود.

$$w = a_۱ a_۲ \cdots a_n \qquad \Rightarrow \qquad |w| = n$$

- **رشته‌ی تهی:** رشته‌ای که طول آن صفر است. رشته‌ی تهی با λ (یا ϵ) نشان داده می‌شود.

$$|\lambda| = ۰$$

رشته‌ی تهی عضو خنثای عمل الحاق است:

$$\forall w \in \Sigma^* \qquad \Rightarrow \qquad \lambda w = w\lambda = w$$

بنا بر تعریف برای هر رشته‌ی w داریم:

$$w^{۰} = \lambda$$

رشته‌ی تهی عضو هیچ الفبایی نیست:

$$\forall \Sigma \ \lambda \notin \Sigma$$

◄ **تذکر** رشته‌ی تهی با فاصله‌ی خالی ($\square$) متفاوت است ($|\square| = ۱$).

- **معکوس (reverse) یک رشته:** رشته‌ای است که اگر از انتها خوانده شود برابر با آن رشته شود.

$$w = a_۱ a_۲ \cdots a_n \qquad \Rightarrow \qquad w^R = a_n \cdots a_۲ a_۱$$

تعریف بازگشتی:

$$w = av, \qquad a \in \Sigma, v \in \Sigma^* \qquad \Rightarrow \qquad w^R = v^R a$$

- **زیررشته (substring):** بخشی از نمادهای پی در پی یک رشته، یک زیررشته‌ی آن نام دارد. می‌گوییم u زیررشته‌ی w است و می‌نویسیم $u \sqsubseteq w$ اگر و فقط اگر رشته‌های α و β وجود داشته باشند که برای آن‌ها داشته باشیم:

$$w = \alpha u \beta$$

حداکثر تعداد زیررشته‌های یک رشته w: $|\text{SUBSTRINGS}(w)| \leq \frac{۱}{۲}(|w|(|w| + ۱)) + ۱$

  - **پیشوند (prefix):** زیررشته‌ای که از ابتدای رشته شروع شود ($\alpha = \lambda$)
    تعداد پیشوندهای یک رشته w: $|\text{PREFIXES}(w)| = |w| + ۱$
  - **پسوند (suffix):** زیررشته‌ای که به انتهای رشته ختم شود. ($\beta = \lambda$)
    تعداد پسوندهای یک رشته w: $|\text{SUFFIXES}(w)| = |w| + ۱$

- **زیردنباله (subsequence):** رشته‌ی حاصل از حذف صفر نماد یا بیشتر از رشته.
  حداکثر تعداد زیردنباله‌های یک رشته w: $|\text{SUBSEQUENCES}(w)| \leq ۲^{|w|}$
  رشته‌ی تهی، زیررشته، پیشوند، پسوند و زیردنباله‌ی هر رشته‌ای است.
  هر رشته‌ای زیررشته، پیشوند، پسوند و زیردنباله‌ی خودش است.

چند رابطه در مورد رشته‌ها. برای $u, v, w \in \Sigma^*$ و $n \geq 0$ داریم:

- $|\lambda| = 0$
- $|a| = 1, \qquad a \in \Sigma$
- $|uv| = |u| + |v|$
- $|w^n| = n|w|$
- $(uv)^R = v^R u^R$
- $(w^n)^R = (w^R)^n$

بستار یک الفبا. مجموعه‌ی کلیه‌ی رشته‌های قابل تولید از الفبای Σ را می‌توان از اتصال صفر یا تعدادی از نمادهای Σ به دست آورد. این مجموعه را با Σ^* نشان می‌دهیم:

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots = \bigcup_{i=0}^{\infty} \Sigma^i$$

که در آن توان‌های Σ به صورت زیر تعریف می‌شود:

$$\Sigma^0 = \{\lambda\}, \qquad \Sigma^{i+1} = \{aw : a \in \Sigma, w \in \Sigma^i\}$$

اگر رشته‌ی تهی را از Σ^* کنار بگذاریم، Σ^+ را خواهیم داشت:

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots = \bigcup_{i=1}^{\infty} \Sigma^i$$

بدیهی است که

$$\Sigma^+ = \Sigma^* - \{\lambda\}$$
$$\Sigma^* = \Sigma^+ \cup \{\lambda\}$$

- اگرچه Σ متناهی است، اما Σ^* و Σ^+ همیشه نامتناهی هستند.
- مشخص است که Σ^i حاوی همه‌ی رشته‌های به طول i بر روی الفبای Σ است و برای هر i داریم:

$$|\Sigma^i| = |\Sigma|^i$$

مثال

برای یک الفبای دو حرفی Σ داریم:

$$\begin{aligned}
\Sigma &= \{a, b\} \\
\Sigma^* &= \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\} \\
\Sigma^+ &= \{a, b, aa, ab, ba, bb, aaa, aab, \ldots\}
\end{aligned}$$

زبان

تعریف **زبان.** یک زبان مجموعه‌ای از رشته‌های یک الفباست، یعنی $L \subseteq \Sigma^*$

اگر مجموعه‌ی L متناهی باشد، L یک زبان متناهی است و در غیر این صورت نامتناهی است.

تعریف **جمله.** به هر عضو مجموعه‌ی L یک رشته یا **جمله** گفته می‌شود.

مثال

L_1 و L_2 هر دو زبان‌هایی بر روی الفبای $\Sigma = \{a, b\}$ هستند:

$$\begin{aligned} L_1 &= \{a, aa, aab\} \\ L_2 &= \{a^n b^n : n \geq 0\} \end{aligned}$$

عملیات روی زبان‌ها: فرض می‌کنیم $L, L_1, L_2 \in \Sigma^*$

- اجتماع دو زبان:

$$L_1 \cup L_2 = \{x : x \in L_1 \vee x \in L_2\}$$

- اشتراک دو زبان:

$$L_1 \cap L_2 = \{x : x \in L_1 \wedge x \in L_2\}$$

- تفاضل دو زبان:

$$L_1 - L_2 = \{x : x \in L_1 \wedge x \notin L_2\}$$

- متمم یک زبان:

$$\bar{L} = \Sigma^* - L$$

- عکس یک زبان:

$$L^R = \{x^R : x \in L\}$$

- الحاق دو زبان:

$$L_1 L_2 = \{x_1 x_2 : x_1 \in L_1 \wedge x_2 \in L_2\}$$

- توان یک زبان:

$$L^0 = \{\lambda\}, \qquad L^{n+1} = L^n L = L L^n$$

- بستار ستاره‌ای:

$$L^* = L^0 \cup L^1 \cup L^2 \cup \cdots = \bigcup_{i=0}^{\infty} L^i$$

- بستار مثبت:

$$L^+ = L^1 \cup L^2 \cup L^3 \cup \cdots = \bigcup_{i=1}^{\infty} L^i$$

مطابق این دو تعریف داریم:

$$L^* = L^+ \cup \{\lambda\}$$
$$L^+ = LL^*$$

◄ **تذکر** رابطه‌ی $L^+ = L^* - \{\lambda\}$ در حالت کلی درست نیست (چرا؟).

مثال

اگر $L = \{a^n b^n : n \geq 0\}$ باشد در این صورت داریم:

$$\begin{aligned} L^R &= \{b^n a^n : n \geq 0\} \\ L^2 &= LL = \{a^n b^n : n \geq 0\}\{a^m b^m : m \geq 0\} = \{a^n b^n a^m b^m : n, m \geq 0\} \end{aligned}$$

زبان تهی: زبان تهی، زبانی است که هیچ رشته‌ای ندارد و با $\varnothing$ یا $\{\}$ نشان داده می‌شود.
بدیهی است که $\{\} \neq \{\lambda\}$.
زبان تهی عضو صفر عمل الحاق زبان‌هاست:

$$L\varnothing = \varnothing L = \varnothing$$

در حالی که $\{\lambda\}$ عضو خنثای عمل الحاق زبان‌هاست:

$$L\{\lambda\} = \{\lambda\}L = L$$

به علاوه داریم:

$$\varnothing^* = \{\lambda\}, \qquad \varnothing^+ = \varnothing$$

چند رابطه در مورد عملیات روی زبان‌ها. اگر L، L_1، L_2 و L_3 زبان‌هایی روی Σ^* باشند، داریم:

- $L_1L_2 \neq L_2L_1$ (در حالت کلی)
- $|L_1L_2| \neq |L_2L_1|$ (در حالت کلی)
- $|L_1L_2| \leq |L_1||L_2|$
- $(L_1L_2)L_3 = L_1(L_2L_3)$
- $L_1(L_2 \cup L_3) = L_1L_2 \cup L_1L_3$
- $(L_2 \cup L_3)L_1 = L_2L_1 \cup L_3L_1$
- $L_1(L_2 \cap L_3) \subseteq L_1L_2 \cap L_1L_3$
- $(L_2 \cap L_3)L_1 \subseteq L_2L_1 \cap L_3L_1$
- $L_1 \subseteq L_2 \Rightarrow L_1^n \subseteq L_2^n, \quad n \geq 0$
- $L_1 \subseteq L_1L_2^*$
- $L_1 \subseteq L_2^*L_1$
- $L_1 \subseteq L_2 \Rightarrow L_1^* \subseteq L_2^*$
- $L_1 \subseteq L_2 \Rightarrow L_1^+ \subseteq L_2^+$
- $L^*L = LL^* = L^+$
- $L^*L^* = L^* = (L^*)^* = (L^*)^+ = (L^+)^*$
- $(L_1 \cup L_2)^* = (L_1^* \cup L_2^*)^* = (L_1^*L_2^*)^*$
- $L_1(L_2L_1)^* = (L_1L_2)^*L_1$

الفبای Σ را در نظر بگیرید، داریم:

مجموعه‌ی همه‌ی رشته‌ها با طول زوج	$(\Sigma^2)^*$
مجموعه‌ی همه‌ی رشته‌ها با طول فرد	$(\Sigma^2)^*\Sigma$
مجموعه‌ی همه‌ی رشته‌ها به طول n	Σ^n
مجموعه‌ی همه‌ی رشته‌ها با طول مضرب k	$(\Sigma^k)^*$

۱-۳-۲ مفهوم دوم: گرامر

تعریف **گرامر (Grammar).** گرامر G یک چهارتایی مرتب به صورت

$$G = (V, T, S, P)$$

است که در آن

V مجموعه‌ای متناهی از ناپایانه‌ها *(non-terminal)* یا متغیرها *(variable)*

T مجموعه‌ای از پایانه‌ها *(terminal)* (الفبای گرامر)

S یک عنصر خاص از V با نام نماد شروع *(start symbol)*

P مجموعه‌ای متناهی از قواعد تولید *(production rule)*

با فرض اینکه V و T ناتهی و مجزا هستند $(V \cap T = \varnothing)$.

قواعد تولید قواعد تولید اساس تعریف یک گرامر است. هر قاعده‌ی تولید (قاعده) یک زوج مرتب به صورت $(x, y) \in P$ است که به شکل

$$x \rightarrow y$$

نمایش داده می‌شود که در آن

$x \in (V \cup T)^+$ (هر ترکیبی از پایانه‌ها و ناپایانه‌های گرامر بجز رشته‌ی تهی)

$y \in (V \cup T)^*$ (هر ترکیبی از پایانه‌ها و ناپایانه‌های گرامر)

از قواعد تولید در عمل اشتقاق (derivation) برای «تولید رشته» استفاده می‌شود.

اشتقاق منظور از اشتقاق، اعمال یک قاعده‌ی تولید بر روی یک رشته است.
اگر رشته‌ی w به صورت $w = uxv$ باشد و قاعده‌ی تولید $x \rightarrow y$ را داشته باشیم، می‌توانیم آن را بر روی این رشته اعمال کنیم و رشته‌ی z به صورت $z = uyv$ را به دست آوریم. به این عمل اشتقاق می‌گوییم و آن را به صورت زیر نشان می‌دهیم

$$w \Rightarrow z$$

و می‌گوییم w، z را مشتق می‌کند یا z از w مشتق می‌شود.
می‌توان قواعد تولید را به طور پی در پی و به تعداد دلخواه استفاده کرد تا رشته‌ی مورد نظر به دست آید:

$$w_۱ \Rightarrow w_۲ \Rightarrow \ldots \Rightarrow w_n$$

و گفته می‌شود که w_n از $w_۱$ مشتق می‌شود. می‌نویسیم

$$w_۱ \Rightarrow^* w_n$$

که به معنی آن است که w_n طی صفر یا چند مرحله تولید می‌شود.

در فرآیند اشتقاق، هرگاه سمت چپ یک قاعده با یک زیررشته از یک رشته تطابق پیدا کند، می‌توانیم سمت راست آن قاعده را با آن زیررشته جایگزین کنیم.

$$w \Rightarrow z \quad \textbf{iff} \quad w = w_1 u w_2,\ z = w_1 v w_2,\ w_1,\ w_2 \in (V \cup T)^*,\ u \to v \in P$$

برای بیان جزئیات عمل اشتقاق از نمادگذاری‌های زیر استفاده می‌شود:

$\Rightarrow_G$	اشتقاق توسط گرامر G	
$\Rightarrow$	اشتقاق در یک مرحله G	(دارای خاصیت تراگذری)
$\Rightarrow^*$	اشتقاق در صفر مرحله یا بیشتر G	(دارای خاصیت بازتابی و تراگذری)
$\Rightarrow^+$	اشتقاق در یک مرحله یا بیشتر G	(دارای خاصیت تراگذری)
$\Rightarrow^n$	اشتقاق در n مرحله	
$\overset{(r)}{\Rightarrow}$	اشتقاق با قاعده‌ی r	

مثال

گرامر $G = (\{S\}, \{a, b\}, S, P)$ با مجموعه قواعد $P = \{S \to aSb, S \to \lambda\}$ را در نظر می‌گیریم. برای این گرامر می‌توان اشتقاق زیر را به دست آورد:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

که می‌توان به جای آن نوشت:

$$S \Rightarrow^* aabb$$

◄ **نکته** (قرارداد) پایانه‌ها را با حروف کوچک و ناپایانه‌ها را با حروف بزرگ می‌نویسیم.

زبان تولید شده توسط یک گرامر

تعریف **زبان تولید شده توسط یک گرامر.** اگر $G = (V, T, S, P)$ یک گرامر باشد، آن‌گاه مجموعه‌ی

$$L(G) = \{w \in T^* : S \Rightarrow^* w\}$$

زبان تولید شده توسط G نام دارد.

مثال

زبان تولید شده توسط گرامر $G = (\{S\}, \{a, b\}, S, P)$ با مجموعه قواعد $P = \{S \to aSb, S \to \lambda\}$ عبارت است از:

$$L = \{a^n b^n : n \geq ۰\}$$

اگر $L \in w$ باشد، آن‌گاه

$$S \Rightarrow w_۱ \Rightarrow w_۲ \Rightarrow \ldots \Rightarrow w_n \Rightarrow w$$

را یک اشتقاق از جمله‌ی w می‌گویند. رشته‌های S، $w_۱$، $w_۲$، ...، w_n که حاوی پایانه‌ها و ناپایانه‌ها هستند به **فرم جمله‌ای** (sentential form) موسوم هستند.
مجموعه‌ی همه‌ی فرم‌های جمله‌ای قابل اشتقاق از یک گرامر را با $SF(G)$ نمایش می‌دهیم و به صورت

$$SF(G) = \{W \in (T \cup V)^* : S \Rightarrow^* W\}$$

تعریف می‌کنیم. بدیهی است که

$$L(G) \subset SF(G)$$

◄ **نکته** برای این که ثابت کنیم یک زبان توسط یک گرامر تولید می‌شود، باید ثابت کنیم که
۱) گرامر تمام رشته‌های زبان را می‌سازد (هر رشته‌ی $w \in L$ می‌تواند توسط گرامر تولید شود).
۲) گرامر هیچ جمله‌ای را خارج از آن زبان تولید نمی‌کند (هر جمله‌ی تولید شده توسط گرامر در L است).

برای اثبات اینکه $L = L(G)$ است، باید ثابت کنیم که $L \subseteq L(G)$ و $L(G) \subseteq L$.

تعریف **هم‌ارزی گرامرها.** دو گرامر $G_۱$ و $G_۲$ هم‌ارز هستند، اگر و فقط اگر هر دو یک زبان واحد را تولید کنند.

$$G_۱ \equiv G_۲ \quad \textbf{iff} \quad L(G_۱) = L(G_۲)$$

◄ **نکته** **یک شرط کافی برای هم‌ارزی (معادل بودن) دو گرامر** $G_۱$ **و** $G_۲$:
اگر قاعده‌ی $x \to y$ را در $G_۱$ داشته باشیم، باید این قاعده در $G_۲$ نیز موجود باشد و یا اشتقاق $x \Rightarrow^*_{G_۲} y$ را بتوان تولید نمود.

◄ **تذکر** برای بررسی هم‌ارزی گرامرها، ابتدا بررسی کنید که آیا هر دو گرامر رشته‌ی λ را تولید می‌کنند یا خیر.

چند زبان معروف و گرامر متناظر با آنها. برای $\Sigma = \{a, b\}$ داریم:

$L(G)$	$P(G)$
$\{a^n : n \geq ۰\}$	$S \to aS \mid \lambda$
$\{a^n : n \geq ۱\}$	$S \to aS \mid a$
$\{w : n_a(w) = n_b(w)\}$	$S \to aSb \mid bSa \mid SS \mid \lambda$
$\{w : w \in \{a,b\}^*\}$	$S \to aS \mid bS \mid \lambda$
$\{ww^R : w \in \{a,b\}^*\}$	$S \to aSa \mid bSb \mid \lambda$
$\{a^nb^n : n \geq ۰\}$	$S \to aSb \mid \lambda$
$\{a^nb^m : n \geq ۰, m > n\}$	$S \to aSb \mid B,\ \ B \to bB \mid b$

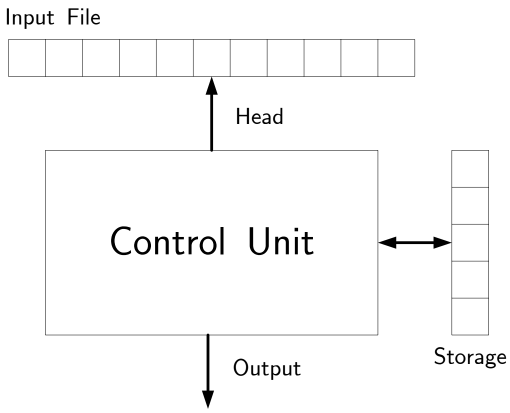
گرامر برای ترکیب زبان‌ها

$$L_۱ = L(G_۱), \qquad G_۱ = (V_۱, T, S_۱, P_۱)$$
$$L_۲ = L(G_۲), \qquad G_۲ = (V_۲, T, S_۲, P_۲)$$
$$V_۱ \cap V_۲ = \varnothing, \qquad S \notin V_۱ \cup V_۲$$

$L_۱ \cup L_۲ = L(G)$	$G = (V_۱ \cup V_۲ \cup \{S\}, T, S, P_۱ \cup P_۲ \cup \{S \to S_۱ \mid S_۲\})$
$L_۱L_۲ = L(G)$	$G = (V_۱ \cup V_۲ \cup \{S\}, T, S, P_۱ \cup P_۲ \cup \{S \to S_۱S_۲\})$
$L_۱^* = L(G)$	$G = (V_۱ \cup \{S\}, T, S, P_۱ \cup \{S \to SS_۱ \mid \lambda\}) \ \equiv$ $G = (V_۱ \cup \{S\}, T, S, P_۱ \cup \{S \to S_۱S \mid \lambda\})$

۱-۳-۳ مفهوم سوم: ماشین (آتوماتون)

آتوماتون (ج. آتوماتا) ، یک مدل انتزاعی از کامپیوتر است.

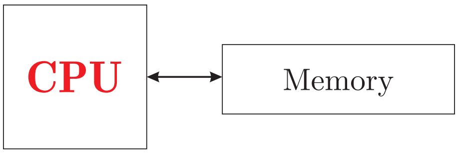


- آتوماتون مکانیزمی برای خواندن **ورودی** دارد.
ورودی آتوماتون یک رشته‌ی الفبایی داده شده است که بر روی نوار ورودی نوشته شده است.
آتوماتون می‌تواند فایل ورودی را بخواند، اما نمی‌تواند آن را تغییر دهد.
نوار ورودی به چند سلول تقسیم شده و هر سلول یک نشانه از الفباست.
نوار ورودی از چپ به راست خوانده می‌شود (هر بار یک نشانه).
مکانیزم خواندن می‌تواند انتهای رشته‌ی ورودی را تشخیص دهد.
- آتوماتون می‌تواند **خروجی** تولید کند.
- آتوماتون می‌تواند دارای **حافظه‌ی موقت** باشد.
حافظه دارای تعداد نامتناهی سلول است که هر سلول آن یک نماد الفبا را در خود جای می‌دهد.
آتوماتون می‌تواند محتوای حافظه‌ی موقت را تغییر دهد.
- آتوماتون دارای یک **واحد کنترل** است که هر زمان می‌تواند در یکی از حالات داخلی خود باشد.
تعداد حالات آتوماتون متناهی است.
  - آتوماتون می‌تواند با ترتیب مشخص تغییر حالت دهد.
  - آتوماتون در هر زمان در یک حالت خاص قرار دارد و نماد مشخصی را از ورودی می‌خواند.
  - حالت بعدی آتوماتون توسط تابع گذر (transition function) مشخص می‌شود.
  - تابع گذر بر اساس **نماد ورودی**، **حالت فعلی** و **محتوای حافظه** حالت بعدی را تعیین می‌کند.
  - به **حالت کنترل**، **باقیمانده‌ی ورودی** و **محتوای حافظه** یک پیکربندی (configuration) می‌گویند.
  - تغییر حالت می‌تواند موجب تولید خروجی یا تغییر در حافظه شود.
  - تغییر وضعیت از یک حالت به حالت دیگر، حرکت (move) نام دارد.

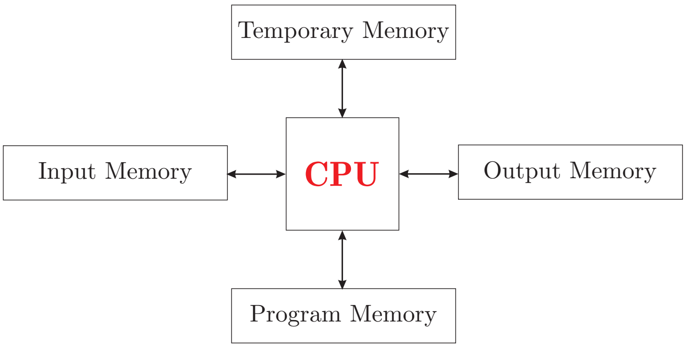
حافظه و **نوع خروجی** بر نوع آتوماتون تاثیر دارد.

محاسبه با آتوماتون

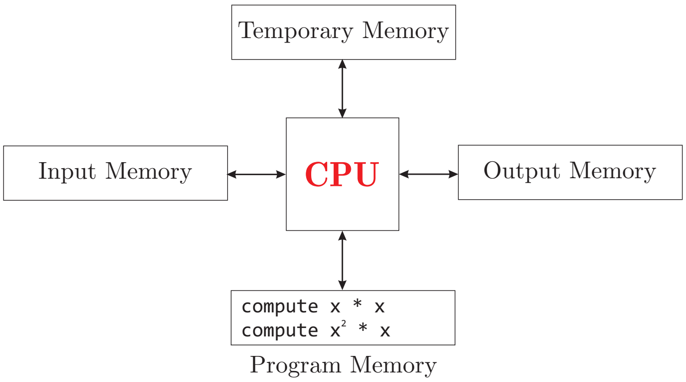
یک کامپیوتر متداول با واحد پردازش مرکزی (CPU) و حافظه (Memory) را در نظر می‌گیریم.



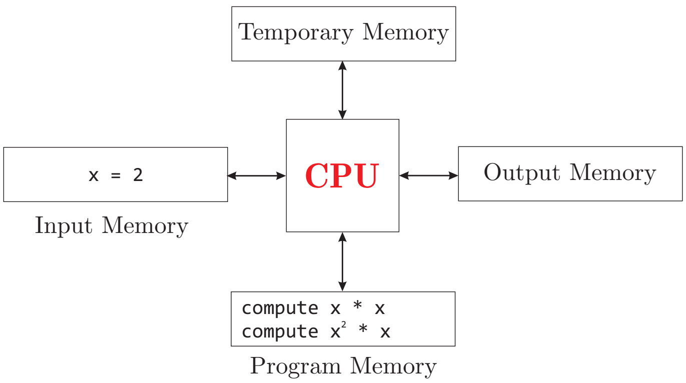
حافظه به بخش‌های متفاوتی تقسیم می‌شود: ورودی، خروجی، برنامه و موقتی.



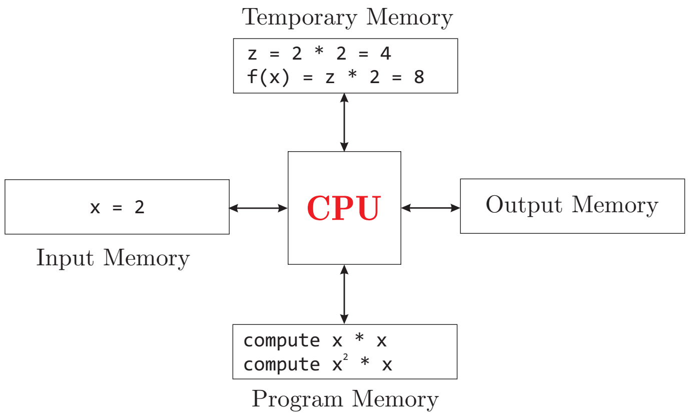
فرض کنید می‌خواهیم تابع $f(x) = x^۳$ را محاسبه کنیم. برنامه‌ی مربوطه در حافظه‌ی برنامه قرار می‌گیرد.



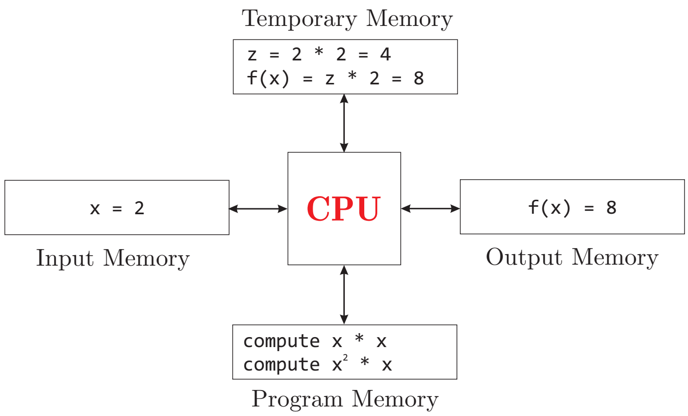
برای مثال ورودی $x = ۲$ در حافظه‌ی ورودی قرار می‌گیرد.



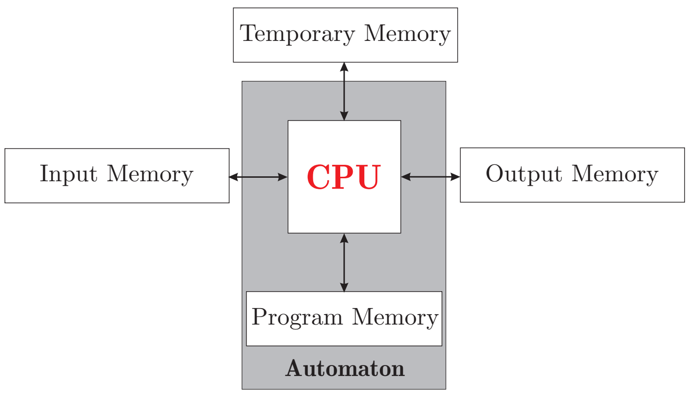
محاسبه دستور به دستور انجام می‌شود و نتایج میانی در حافظه‌ی موقتی قرار می‌گیرد.



سرانجام، خروجی $f(x) = ۸$ در حافظه‌ی خروجی قرار می‌گیرد.



◄ **نکته** ترکیب CPU و حافظه‌ی برنامه به عنوان «آتوماتون» در نظر گرفته می‌شود.



قدرت آتوماتا، به نوع حافظه‌ی موقتی آنها بستگی دارد و با پیچیده‌تر شدن حافظه، قدرت بیشتر می‌شود.

- **آتوماتون متناهی:** حافظه‌ی موقت ندارد (قدرت محاسبه پایین: مانند Vending Machine).
- **آتوماتون پشته‌ای:** حافظه‌ی موقت آن از نوع پشته است (قدرت محاسبه متوسط: مانند زبان‌های برنامه‌سازی).
- **ماشین تورینگ:** حافظه‌ی موقت آن از نوع دسترسی تصادفی است (بالاترین قدرت محاسبه: معادل با الگوریتم).

آتوماتای پذیرنده / تراگذر

- **پذیرنده (accepter)**: آتوماتونی که خروجی آن بلی / خیر باشد (ورودی خود را قبول یا رد کند).
- **تراگذر (transducer)**: آتوماتونی که خروجی آن در قالب یک رشته است.

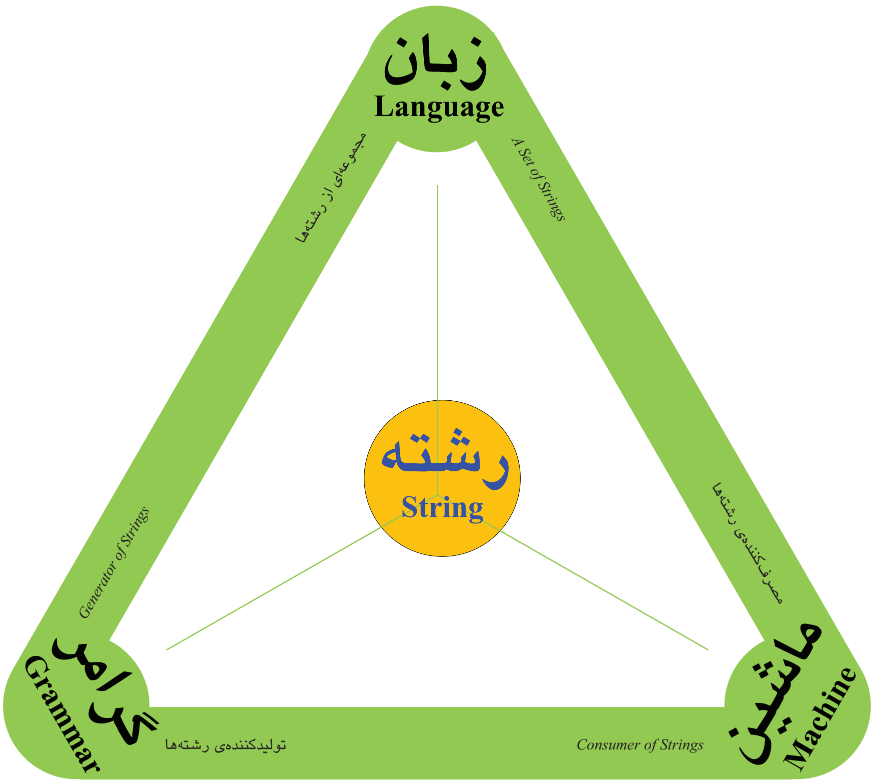
آتوماتای قطعی / غیرقطعی

- **آتوماتون قطعی (deterministic)**: آتوماتونی که در آن با دانستن **حالت فعلی**، **ورودی** و **محتوای حافظه** می‌توان رفتار بعدی آتوماتون را تعیین کرد.
- **آتوماتون غیرقطعی (non-deterministic)**: آتوماتونی که قطعی نباشد: در هر پیکربندی می‌توان چند حرکت مختلف انجام داد.

۱-۴ رابطه‌ی میان زبان، گرامر و ماشین

زبان، گرامر و ماشین، هر سه با مفهوم رشته سر و کار دارند:

- **زبان** مجموعه‌ای از رشته‌هاست،
- **گرامر** رشته‌های زبان را تولید می‌کند، و
- **ماشین** رشته‌های زبان را مصرف می‌کند.

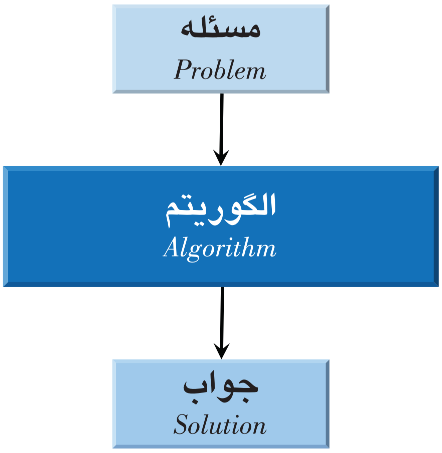


یک زبان می‌تواند صفر یا چند ماشین یا گرامر داشته باشد.

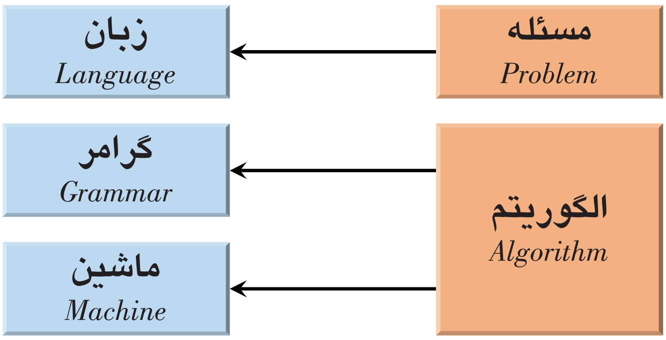
در مقابل، زبان متناظر با یک گرامر یا زبان متناظر با یک ماشین، همواره یکتاست.

این درس به مطالعه‌ی خانواده‌های مختلف زبان‌ها، گرامرها و ماشین‌ها و رابطه‌ی آنها می‌پردازد.

الگوریتم، روالی است که می‌تواند یک مسئله را حل کند و جواب آن را به دست دهد.



در نظریه‌ی محاسبه، هر مسئله متناظر با یک زبان دیده می‌شود. به همین ترتیب، الگوریتم متناظر با گرامر یا ماشین در نظر گرفته می‌شود.



مثال

مسئله‌ی زیر را در نظر بگیرید:

آیا رشته‌ی ورودی w تعدادی مساوی a و b دارد؟

این مسئله را در قالب زبان زیر نشان می‌دهیم:

$$L = \{w \in \{a, b\}^* : n_a(w) = n_b(w)\}$$

جواب مسئله، معادل با این است که $w \in L$ یا $w \notin L$.

الگوریتمی که این مسئله را برای هر ورودی دلخواهی حل می‌کند، ماشینی است که می‌گوید $w \in L$ یا خیر.

مراجع

[1] P. Linz, **An Introduction to Formal Languages and Automata**, 5th Ed., Jones and Bartletts, 2012.

[2] M. Sipser, **Introduction to the Theory of Computation**, 3rd Ed., Cengage Learning, 2013.