Computer Science 513

Modelling Computation of Languages

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #8

Goals for Today

Goals for Today:

- Explain three (related) attempts to model computability by defining computations of ***languages:*** “String Rewriting Systems”, “Phrase Structure Grammars”, and “Markov Algorithms”
- Present and justify results that these can be used to define the same sets of Turing-computable functions, Turing-recognizable languages and Turing-decidable languages that we have seen before.
- *Brief* description of related attempts, and bit more information about the “Church-Turing thesis”.

Rewrite Rules

Let

$$\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$$

be an alphabet.

Definition: Given a pair of words $g, \widehat{g} \in \Sigma^\star$, the expression

$$g \to \widehat{g}$$

is called a ***rewrite rule***. This is sometimes called a ***semi-Thue production***, or simply ***production***, as well.

- These are named after the Norwegian mathematician Axel Thue.
- Thue is pronounced “too-ay.”

Applications of Rewrite Rules

- If P is the rewrite rule $g \to \widehat{g}$ and $u, v \in \Sigma^\star$ then (we write)

$$u \Rightarrow_P v$$

if and only if there exist strings $\alpha, \beta \in \Sigma^\star$ such that

$$u = \alpha g \beta \qquad \text{and} \qquad v = \alpha \widehat{g} \beta.$$

String-Rewriting Systems

Definition: A ***String Rewriting System*** is a finite set of rewrite rules (all involving the same alphabet Σ). These are also called ***semi-Thue processes***.

- If $u, v \in \Sigma^\star$ and Π is a string rewriting system then we write

$$u \Rightarrow_\Pi v$$

if there exists some rewrite rule $P \in \Pi$ such that

$$u \Rightarrow_P v.$$

Derivations

- Finally, if $u, v \in \Sigma^\star$ and Π is a string rewriting system then we write

$$u \Rightarrow_\Pi^\star v$$

if there is an integer $k \geq 0$ and a sequence $u_0, u_1, u_2, \ldots, u_k$ of words in $\Sigma^\star$ such that

(a) $u = u_0$,

(b) $u_i \Rightarrow_\Pi u_{i+1}$ for every integer i such that $0 \leq i \leq k-1$, and

(c) $u_k = v$.

Note that (taking $k = 0$) $u \Rightarrow_\Pi^\star u$ for every string $u \in \Sigma^\star$.

- A sequence $u_0, u_1, u_2, \ldots, u_k$ satisfying the above properties is called a ***derivation*** of v from u. If the value k is as above then k is the ***length*** of this derivation.

Relaxation of Notation

- If there is no ambiguity — that is, only one string rewriting system is being considered — then we can write

$$u \Rightarrow v$$

instead of

$$u \Rightarrow_\Pi v$$

and we can write

$$u \Rightarrow^\star v$$

instead of

$$u \Rightarrow^\star_\Pi v.$$

Nondeterministic Turing Machines

Nondeterministic Turing machines are variants of "standard" Turing machines that were, ideally, introduced in at least one course that you have taken before. They are reviewed in the supplemental document for Lecture #7 — which also briefly mentions "nondeterministic multi-tape Turing machines".

- As noted in that supplemental document, a language $L \subseteq \Sigma^\star$ is ***Turing-recognizable*** (that is, the language of a "standard" Turing machine) if and only if there exists a ***nondeterministic*** (single-tape, or multi-tape) Turing machine M such that L is the language of M.

Simulation of a String Rewriting System by a Nondeterministic Turing Machine

Let Σ be an alphabet that does not include # — replacing # in Σ with another symbol, otherwise. Let $\widehat{\Sigma} = \Sigma \cup \{\#\}$.

Exercise: Let Π be a String Rewriting System. Informally describe a nondeterministic Turing machine (possibly with several tapes — whose input alphabet is Σ and whose language is

$$\{\mu\#\nu \mid \mu, \nu \in \Sigma^\star \text{ and } \mu \Rightarrow_\Pi^\star \nu.\}.$$

One way to do this is to say how tapes will be used and how the finite control might be used to "guess" a derivation of ν from μ, if one exists.

Simulation of a Nondeterministic Turing Machine by a String Rewriting System

Now let

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

be a nondeterministic Turing machine, and let

$$\widehat{\Sigma} = Q \cup \Gamma \cup \{h\}$$

where "h" is a symbol that is not in either Q or Γ (so that it should be renamed, otherwise).

Simulation of a Nondeterministic Turing Machine by a String Rewriting System

Definition: A ***Post word*** is a string

$$h\,\omega_1\,q\,\omega_2 h \in \widehat{\Sigma}^\star$$

where $\omega_1, \omega_2 \in \Gamma^\star$ and $q \in Q$ — so that this word is a ***configuration*** of $\mathcal{M}$, surrounded by zero or more leading and trailing $\sqcup$'s, and beginning and ending with the "marker" h.

Consider a ***string rewriting system***,

$$\Sigma(M) = \Pi_1 \cup \Pi_2,$$

over the alphabet $\widehat{\Sigma}$, that is defined as follows.

Simulation of a Nondeterministic Turing Machine by a String Rewriting System

Rewrite rules included in Π_1 can be used to (in some way) "simulate" an execution of M:

(a) Suppose that $(r, \tau, \mathrm{L}) \in \delta(q, \sigma)$ for a state $r \in Q$ and symbol $\tau \in \Gamma$. Then rewrite rule

$$\nu\, q\, \sigma \to r\, \nu\tau$$

should be added to Π_1 for every symbol $\nu \in \Gamma$. The rewrite rule

$$h\, q\, \sigma \to h\, r\, \sqcup\, \tau$$

should be added to Π_1 as well.

Simulation of a Nondeterministic Turing Machine by a String Rewriting System

If $\sigma = \sqcup$ (for this case) then the rewrite rule

$$\nu q h \rightarrow r \nu \tau h$$

must be also added to Π_1 for every symbol $\nu \in \Gamma$, and the rewrite rule

$$h q h \rightarrow h r \tau h$$

must be added to Π_1 too.

Simulation of a Nondeterministic Turing Machine by a String Rewriting System

(b) Next suppose that $(r, \tau, \mathrm{R}) \in \delta(q, \sigma)$ for a state $r \in Q$ and symbol $\tau \in \Gamma$. The rewrite rule

$$q\sigma \to \tau r$$

should be added to Π_1.

If $\sigma = \sqcup$ then the rewrite rule

$$qh \to \tau rh$$

should be added to Π_1 as well.

Simulation of a Nondeterministic Turing Machine by a String Rewriting System

(c) Finally, suppose that $(r, \tau, \mathrm{S}) \in \delta(q, \sigma)$ for a state $r \in Q$ and a symbol $\tau \in \Gamma$. The rewrite rule

$$q\sigma \to r\tau$$

should be added to Π_1. If $\sigma = \sqcup$ then the rewrite rule

$$qh \to r\tau h$$

should be added to Π_1 too.

Π_1 does not include any other rewrite rules.

Simulation of a Nondeterministic Turing Machine by a String Rewriting System

- It is not difficult to prove the following by induction on k: If $\omega \in \Sigma^\star$, $\zeta \in \widehat{\Sigma}^\star$, and

$$hq_0 \sqcup \omega h \Rightarrow_{\Pi_1}^\star \zeta$$

then ζ is a Post word. Furthermore, ζ represents a configuration of M that can be reached from the initial configuration for M on input ω using exactly k moves of M, where k is the length of the derivation in Π_1 used to obtain ζ from $hq_0 \sqcup \omega h$.

Simulation of a Nondeterministic Turing Machine by a String Rewriting System

- One can also use induction on k to establish the following: Let $\omega \in \Sigma^\star$, let $\mu_1, \mu_2 \in \Gamma^\star$ and let $q \in Q$ such that the configuration

$$\mu_1 q \mu_2$$

can be reached from the initial configuration of M for ω using k moves of M. Then there exists a Post word ζ representing this configuration such that

$$hq_0 \sqcup \omega h \Rightarrow_{\Pi_1}^\star \zeta.$$

Furthermore, ζ can be obtained from $hq_0 \sqcup \omega h$ using a derivation with length k.

Simulation of a Nondeterministic Turing Machine by a String Rewriting System

- These results imply that if $\omega \in \Sigma^\star$ then M accepts ω if and only if

$$hq_0 \sqcup \omega h \Rightarrow_{\Pi_1}^\star \zeta$$

  for at least one Post word $\zeta \in \widehat{\Sigma}^\star$ that represents an accepting configuration of M (that is, a configuration such that M is in state q_{accept}).
- Since $\delta(q_{\text{accept}}, \sigma) = \emptyset$, no rewrite rules in Π_1 can be applied to any Post word ζ that represents an accepting configuration.

Simulation of a Nondeterministic Turing Machine by a String Rewriting System

Π_2 includes (only) the following rewrite rules.

(d) Π_2 includes a rewrite rule

$$q_{\text{accept}}\sigma \rightarrow q_{\text{accept}}$$

for every symbol $\sigma \in Q \cup \Gamma = \widehat{\Sigma} \setminus \{h\}$, so that extra symbols to the ***right*** of q_{accept} in ζ can initially be deleted.

Simulation of a Nondeterministic Turing Machine by a String Rewriting System

(e) Π_2 also includes a rewrite rule

$$\sigma q_{\text{accept}} h \to q_{\text{accept}} h$$

for every symbol $\sigma \in Q \cup \Gamma = \widehat{\Sigma} \setminus \{h\}$, so that extra symbols to the ***left*** of q_{accept} can be deleted, after all the symbols to the right have.

Simulation of a Nondeterministic Turing Machine by a String Rewriting System

- The rewrite rules in Π_2 of type(d) can be applied to use any Post word $\zeta \in \widehat{\Sigma}^\star$ that represents an accepting configuration of M to produce a Post word μ with the form

$$\mu = h\nu q_{\text{accept}} h$$

  for $\nu \in (Q \cup \Gamma)^\star$. Rewrite rules of type (e), above, can then applied to obtain the Post word $hq_{\text{accept}}h$.
- These rewrite rules cannot be applied to Post words representing rejecting configurations of M, or any other configurations of M that are not accepting configurations, at all.

Simulation of a Nondeterministic Turing Machine by a String Rewriting System

- It follows that M accepts a string $\omega \in \Sigma^\star$ if and only if

$$hq_0 \sqcup \omega h \Rightarrow_\Pi^\star hq_{\text{accept}} h.$$

- The following result has now been established.

Simulation of a Nondeterministic Turing Machine by a String Rewriting System

Theorem #1: Let

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

be a nondeterministic Turing machine. Let h be a new symbol — so that $h \notin Q \cup \Gamma$ — and let $\widehat{\Sigma}$ be the alphabet

$$\widehat{\Sigma} = Q \cup \Gamma \cup \{h\}.$$

Then there exists a string rewriting system $\Sigma(M)$ — with alphabet $\widehat{\Sigma}$ — such that, for every string $\omega \in \Sigma^\star$, M accepts ω if and only if

$$hq_0 \sqcup \omega h \Rightarrow^\star_{\Sigma(M)} hq_{\text{accept}}h.$$

More about String Rewriting Systems

- The previous exercise and theorem suggest that string rewriting systems are “equivalent” to Turing machines — they can be used to model “Turing-recognizable languages” in a very loose sense. This is (arguably) imprecise since “languages” of string rewriting systems have not been defined.
- The connection is (possibly) a bit clearer once ***phrase structure grammars*** are considered. The next results, about string rewriting systems, will be useful, shortly, when phrase structure grammars.

Moving to Deterministic Turing Machines

- While the string rewriting systems $\Sigma(M)$ have been defined for *nondeterministic* Turing machines, one can also define these for a *deterministic* Turing machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

as follows.

Moving to Deterministic Turing Machines

1. Set
$$\widehat{M} = (Q, \Sigma, \Gamma, \widehat{\delta}, q_0, q_{\text{accept}}, q_{\text{reject}})$$
to be the *nondeterministic* Turing machine such that
$$\widehat{\delta}(q, \sigma) = \{\delta(q, \sigma)\}$$
for every state $q \in Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}$, and
$$\widehat{\delta}(q_{\text{accept}}, \sigma) = \widehat{\delta}(q_{\text{reject}}, \sigma) = \emptyset.$$
for all $\sigma \in \Gamma$.
2. Define $\Sigma(M)$ to be $\Sigma(\widehat{M})$.

Moving to Deterministic Turing Machines

- It easily proved that $\widehat{M}$ accepts ω if and only if M accept ω, for every string $\omega \in \Sigma^\star$. Thus $L(\widehat{M}) = L(M)$.
- Furthermore, $\widehat{M}$ *rejects* ω if and only if M rejects ω, and $\widehat{M}$ *loops* on ω if and only if M loops on ω, for all $\omega \in \Sigma^\star$ as well.
- Thus $\widehat{M}$ *decides* a language $L \subseteq \Sigma^\star$ if and only if M decides L.
- Theorem #1 can now be proved for *deterministic* Turing machines, instead of nondeterministic Turing machines, without changing the proof.

Simulation of a Deterministic Turing Machine

Lemma #2: Suppose that M is a ***deterministic*** Turing machine and that $\Sigma(M)$ is the string rewriting system obtained from M as described above. Let $\zeta \in \widehat{\Sigma}^\star$ be a Post word.

(a) There is *at most* one rewrite rule in $\Sigma^\star$ that can be applied to Σ, and this can be applied in only one way.

(b) There is *at most* one string $\widehat{\zeta} \in \widehat{\Sigma}^\star$ such that

$$\zeta \Rightarrow_{\Sigma(M)} \widehat{\zeta}.$$

(c) If $\zeta \Rightarrow_{\Sigma(M)} \widehat{\zeta}$ then $\widehat{\zeta}$ is also a Post word.

Simulation of a Deterministic Turing Machine

(d) If M ***decides*** a language $L \subseteq \Sigma^\star$ then for every string $\omega \in \Sigma^\star$ such that $\omega \notin L$,

$$hq_0 \sqcup \omega h \Rightarrow^\star_{\Sigma(M)} \zeta$$

for some Post word, ζ, that includes the symbol for M's reject state q_{accept}.

Simulation of a Deterministic Turing Machine

How This is Proved:

- This can be established by an examination of the rewrite rules included in $\Sigma(M)$ — noting that if M is a deterministic Turing machine — so that there is always exactly one transition that can be applied to any non-halting configuration of M.
- One can see, by inspection of the rewrite rules, that each is of the form "$\alpha \rightarrow \beta$" where α includes a symbol $q \in Q$, as well as another symbol immediately.

Simulation of a Deterministic Turing Machine

- Since a Post word include exactly one symbol in Q this can be used to argue that there is never more than one rewrite rule in $\Sigma(M)$ that is applicable and, furthermore, that it can be applied in only one way — establishing part (a).
- Part (b) is a consequence of part (b).
- Part (c) is also (reasonably easily) established by examining each of the rewrite rules included in $\Sigma(M)$.
- Part (d) can also be established by an examination of the rules in $\Sigma(M)$: If M rejects ω after t steps then rewrite rules in Σ_1 can be applied, to "$hq_0 \sqcup \omega h$", to obtain a Post word containing "q_{reject}", with a derivation that also has length t. No rewrite rules in $\Sigma(M)$ be applied after that. □

$\Sigma(M)$

- Once again, consider the string rewriting system $\Sigma(M)$, for either a deterministic Turing machine or a nondeterministic Turing machine M.
- As noted above, if $\omega \in \Sigma^\star$ (where Σ is the input alphabet for M), then M accepts ω if and only if

$$hq_0 \sqcup \omega h \Rightarrow^\star_{\Sigma(M)} hq_{\text{accept}} h.$$

$\Omega(M)$

Definition: If $g, \widehat{g} \in \Sigma^\star$ for some alphabet Σ, then the ***inverse*** of the rewrite rule

$$g \to \widehat{g}$$

is the rewrite rule

$$\widehat{g} \to g.$$

Definition: $\Omega(M)$ is the string rewriting system that includes all of the *inverses* of the rewrite rules in $\Sigma(M)$.

$\Sigma(M)$ and $\Omega(M)$

The following is now easily proved:

Theorem #3: If M is a (nondeterministic *or* deterministic) Turing machine and $\omega \in \Sigma^\star$ (where Σ is the input alphabet for M) then M accepts ω if and only if

$$hq_{\text{accept}}h \Rightarrow^\star_{\Omega(M)} hq_0 \sqcup \omega h.$$

Method of Proof: Prove (by induction on the length of derivations) that if $\zeta, \widehat{\zeta} \in \widehat{\Sigma}^\star$, then

$$\zeta \Rightarrow^\star_{\Sigma(M)} \widehat{\zeta} \quad \text{if and only if} \quad \widehat{\zeta} \Rightarrow^\star_{\Omega(M)} \zeta.$$

The claim then follows by Theorem #1 and the observation (about $\Sigma(M)$) given above. □

Phrase-Structure Grammars

Definition: A ***phrase-structure grammar*** is a 4-tuple

$$G = (V, \Sigma, \Pi, S)$$

where

- V is a finite but nonempty set of ***variables***.
- Σ is another finite but nonempty set of ***terminals***; $V \cap \Sigma = \emptyset$.
- Π is a ***string rewriting system*** over the alphabet $V \cup \Sigma$.
- S is a variable in V, called the ***start variable*** of G.

Languages of Grammars

Definition: If $G = (V, \Sigma, \Pi, S)$ is a phrase-structure grammar then the ***language of G***, denoted, $L(G)$, is the set

$$L(G) = \{\omega \in \Sigma^\star \mid S \Rightarrow_\Pi^\star \omega\}.$$

- ***Note:*** This is, by definition, a subset of $\Sigma^\star$ (and not just of $(\Sigma \cup V)^\star$).

Phrase-Structure Grammars

Theorem #4: Let $L \subseteq \Sigma^\star$ such that L is the language of a nondeterministic Turing machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}).$$

Then there exists a phrase-structure grammar G such that $L = L(G)$.

How This is Proved: The result is trivial if $q_0 = q_{\text{accept}}$ so it is assumed that $q_0 \neq q_{\text{accept}}$. A construction from the String Rewriting System $\Omega(M)$ to produce a phrase-structure grammar G such that $L = L(G)$ can then be presented.

The supplementary document describes this construction and a proof of its correctness.

Phrase-Structure Grammars

- It is also possible to use any phase-structure grammar

$$G = (V, \Sigma, \Pi, S)$$

  to design a nondeterministic Turing machine (which nondeterministically guesses a derivation) such that $L(M) = L(G)$.
- Thus the set of languages of phrase-structure grammars is the same as the languages of nondeterministic Turing machines — that is, the set of Turing-recognizable languages.
- Phrase-structure grammars have applications in computational linguistics and in compiler development. They will be seen again in this course when the ***Chomsky Hierarchy*** is introduced.

Markov Algorithms

- String rewriting systems and phrase-structure grammars are, inherently, nondeterministic: There can be more than one rewrite rule that can be applied at any time so that more than one string can be derived from a given one. It is not, at all clear, how these can be used to compute functions (instead of to recognize languages).
- In 1951, Andrey Andreyevič Markov, Jr. introduced the following (deterministic) model of computations. This is based on string rewriting systems, but one can use it in a more straightforward way to define the computation of functions too.

Markov Algorithms

Definition: Let Γ be an alphabet. A ***Markov algorithm*** is a finite (ordered) ***sequence***, M, of rewrite rules — sometimes called ***substitution formulas*** —

$$\begin{aligned} \alpha_1 &\to \beta_1 \\ \alpha_2 &\to \beta_2 \\ &\vdots \\ \alpha_n &\to \beta_n \end{aligned}$$

where $\alpha_i, \beta_i \in \Gamma^\star$ for $1 \leq i \leq n$. M is also called the ***scheme***.

Each substitution formula is either ***simple*** or ***final***. Simple substitution formulas are denoted "$\alpha \to \beta$" — that is, using the notation used for regular string rewriting systems. Final substitution rules replace the symbol "$\to$" with "$\to \cdot$", so that, for $\alpha, \beta \in \Gamma^\star$, one writes "$\alpha \to \cdot\beta$" instead of "$\alpha \to \beta$" when giving a final substitution formula using these strings.

Markov Algorithms: Making Them Deterministic

Let $\omega \in \Gamma^\star$.

- A substitution formula "$\alpha \to \beta$" or "$\alpha \to \cdot\beta$" can be applied to ω *almost* as in other string rewriting systems: There must exist strings $\mu, \nu \in \Gamma^\star$ such that $\omega = \mu\alpha\nu$.
- In general, though, it is possible that $\omega = \mu_1\alpha\nu_1 = \mu_2\alpha\nu_2$ for two *different* strings $\mu_1, \mu_2 \in \Sigma^\star$ (in which case, one of these strings is a prefix of the other).
- In order to make this deterministic, we will require that ***shortest*** string $\mu \in \Gamma^\star$, such that $\omega = \mu\alpha\nu$ for a string $\nu \in \Gamma^\star$, when we apply a substitution formula "$\alpha \to \beta$" or "$\alpha \to \cdot\beta$" in order to replace $\omega = \mu\alpha\nu$ with $\mu\beta\nu$.

Markov Algorithms: Making Them Deterministic

One more requirement used to ensure that this is deterministic and to show how "final" substitution formulas are different from "simple" substitution formulas:

- When choosing a substitution formula to be applied to a string $\omega \in \Gamma^\star$ we must always choose the ***first*** substitution formula that can be used — that is, we use the substitution formula "$\alpha_i \to \beta_i$" or "$\alpha_i \to \cdot\beta_i$" for the ***smallest*** integer i such that $1 \leq i \leq n$ and there exists strings $\mu, \nu \in \Gamma^\star$ such that $\omega = \mu\alpha_i\nu$.

Markov Algorithms: Ending a Computation

Another requirement — which explains why "final" substitution rules are different — explains how computations can end:

- No substitution formula can be used, when a Markov algorithm algorithm is applied to a string, after any *final* substitution formula — so the application of a Markov algorithm algorithm to a string $\omega \in \Gamma^\star$ (only) ends for one of two reasons:
  1. A ***final*** substitution formula, "$\alpha_i \rightarrow \cdot\beta_i$" (for some integer i such that $1 \leq i \leq n$) has been applied.
  2. None of the substitution formulas can be applied — because there are not strings $\mu, \nu \in \Gamma^\star$ such that $\zeta = \mu\alpha_i\nu$, for any integer i such that $1 \leq i \leq n$ — where $\zeta \in \Gamma^\star$ is the string that has currently been obtained.

  The final string $\zeta \in \Gamma^\star$ that has been obtained (when the algorithm's execution ends, is considered to be the "output" produced by this application of the algorithm.

Markov-Computable Functions

Confession/Clarification:

- Descriptions of Markov algorithms are, arguably, unclear about how this model can be used to compute functions or recognize languages.
- While it is rarely clearly stated, this model seems to be most useful if one imagines a *second* alphabet, Σ, such that $\Sigma \subseteq \Gamma$. One can think of Σ as being the "input alphabet" and one can imagine this algorithm being used to compute partial or total functions $f : \Sigma^\star \to \Gamma^\star$ — or, sometimes, $f : \Sigma^\star \to \Sigma^\star$.

Markov-Computable Functions

Note: The following definition is not, quite, what seems to be in the literature (when anything can be found in the literature, at all) but is, at least, consistent "in spirit" with what one can find described.

Definition: A partial or total function $f : \Sigma^\star \to \Sigma^\star$ is ***Markov-computable*** if there exists a Markov algorithm, with an alphabet Γ (where $\Sigma \subseteq \Gamma$) and scheme M, such that the following properties are satisfied, for all $\omega \in \Sigma^\star$:

- If $f(\omega)$ is defined then the application of the Markov algorithm to input ω halts, with output $f(\omega)$.
- If $f(\omega)$ is undefined then the application of the Markov algorithm to input ω does not halt.

Markov-Computable Functions

- "Markov-computable" partial functions $f : \mathbb{N} \to \mathbb{N}$ and $f : \mathbb{N}^k \to \mathbb{N}$ (for a positive integer k) can be defined, using encodings of natural numbers as strings, and proceeding as we did when we defined "Turing-computable" functions $f : \mathbb{N} \to \mathbb{N}$ and $f : \mathbb{N}^k \to \mathbb{N}$.

  When defining Markov-computable functions $f : \mathbb{N}^k \to \mathbb{N}$ for $k \geq 2$ one must require that $\sqcup \in \Gamma \setminus \{\Sigma\}$, since "$\sqcup$" must separate strings in $\Sigma^\star$ that are included in the input.

Markov Recognizable and Markov-Decidable Languages

- One way (which you will probably not find in the literature) to say how Markov algorithms recognize, and decide ***languages*** $L \subseteq \Sigma^\star$ would be to require, as well, a symbol "$\mathbb{T}$" belongs to Γ, and to use the following (nonstandard, but plausible) definitions:
  - The Markov algorithm ***accepts*** a string $\omega \in \Sigma^\star$ if its execution on input ω ends, with output "$\mathbb{T}$".
  - The Markov algorithm ***rejects*** a string $\omega \in \Sigma^\star$ if its execution on input ω ends, with an output string that is not equal to "$\mathbb{T}$".
  - The Markov algorithm ***loops on*** a string $\omega \in \Sigma^\star$ if its execution on input ω does not end, at all.

Markov-Recognizable and Markov-Decidable Languages

- ***Definition:*** The ***language*** of a Markov algorithm, with input alphabet $\Sigma \subseteq \Gamma$, is the set of strings $\omega \in \Sigma^\star$ that are accepted by this Markov algorithm.
- ***Definition:*** A language $L \subseteq \Sigma^\star$ is ***Markov-recognizable*** if there is a Markov algorithm whose language is L.
- ***Definition:*** A Markov algorithm ***decides*** a language $L \subseteq \sigma^\star$ if Σ is the input alphabet for this algorithm, the algorithm accepts every string $\omega \in L$, and the algorithm rejects every string $\omega \in \Sigma^\star$ such that $\omega \notin L$.

  Definition: A language $L \subseteq \Sigma^\star$ is ***Markov-decidable*** if there is a Markov algorithm that decides L.

Equivalence of Models

Theorem #5: Let Σ be an alphabet.

(a) A partial or total function $f : \Sigma^\star \to \Sigma^\star$ is Markov-computable if and only if f is Turing-computable.

(b) A language $L \subseteq \Sigma^\star$ is Markov-recognizable if and only if L is Turing-recognizable.

(c) A language $L \subseteq \Sigma^\star$ is Markov-decidable if and only if L is Turing-decidable.

Equivalence of Models

How This is Proved:

- It is sufficient to consider a ***deterministic*** Turing machine M — that computes a function, or recognizes or decides a language, in every case.
- A small number of substitution rules is added to the set of rewrite rules in the string rewriting system $\Sigma(M)$, use an input string ω to produce the initial Post word "$hq_0 \sqcup \omega h$" at the beginning, and to carry out any "cleanup" needed at the end.
- A careful ordering of the rewrite rules in the Markov algorithm's scheme, and choice of final productions, is sufficient to ensure that the rewrite rules are applied in the order needed to establish the result.
- The supplemental document includes additional details about the proof of this claim. □

Related Models

At least two similar, or related, models of computation have also been proposed and studies.

- ***Post canonical systems*** — also called “Post production systems” are generalizations of string rewriting systems (that have a single “initial word”), allowing a finite set of “initial words” and somewhat more general rewrite rules.
- ***Post machines*** can be thought of as simplified Turing machines (they have finite controls, and replace a Turing machine’s storage tape with a queue) whose instructions manipulate strings. While they do not really resemble the models from this lecture very much, at all, they can still be seen as an attempt to “model computation of languages”.

These will not be considered in this course.

Church-Turing Thesis

- The ***Church-Turing Thesis*** — which at least one author now calls the "Computability Thesis" — is a widely-held belief that the various researchers, who contributed to the work that has now been described, "got it right". That is, the Turing-computable partial and total functions really *are*, in a meaningful sense, the functions that are "computable" — and the "Turing-recognizable languages" and the "Turing-decidable languages" also correspond to the set of languages that are "computable" in two meaningful senses.
- A supplemental document, providing more of the history of this thesis and its development, is also available.