Semi-Thue Processes
CSC 341, “Automata, Formal Languages, and Computational Complexity”
Department of Computer Science · Grinnell College
August 31, 2020

Basic definitions

A semi-Thue production on an alphabet Γ is a rewriting rule of the form $u \rightarrow v$, where $u$ and $v$ are strings over Γ. Such a rule licenses the replacement of $u$ with $v$ as a substring of some otherwise fixed string; in other words, it licenses the conversion of a string $w$ into a string $w'$ whenever there exist strings $r$ and $s$ such that $w = rus$ and $w' = rvs$. If $P$ is the production $u \rightarrow v$, then in such cases we say that $rus \Rightarrow_P rvs$.

A semi-Thue process is a finite set of semi-Thue productions on the same alphabet. If $w \Rightarrow_P w'$ for some member $P$ of a given semi-Thue process $\Pi$, we say that $w \Rightarrow_\Pi w'$.

As in the theory of context-free grammars, we say that $w$ derives $w'$ in a semi-Thue process $\Pi$, $w \Rightarrow_\Pi w'$, if there is a sequence $(w_1, \ldots, w_n)$ of strings such that $w = w_1$, $w' = w_n$, and, for every positive integer $i < n$, $w_i \Rightarrow_\Pi w_{i+1}$.

Example 1. Consider the semi-Thue process $\Pi_1 = \{ba \rightarrow ab, ca \rightarrow ac, cb \rightarrow bc\}$ on the alphabet $\Gamma = \{a, b, c\}$. Starting from any string over $\Gamma$, this process can alphabetize its characters; for instance, $caccba \Rightarrow_{\Pi_1} aabccc$, thus:

\[
caccba \Rightarrow_{\Pi_1} aaccba \\
\Rightarrow_{\Pi_1} accbca \\
\Rightarrow_{\Pi_1} acbcba \\
\Rightarrow_{\Pi_1} abccca \\
\Rightarrow_{\Pi_1} abccca \\
\Rightarrow_{\Pi_1} abccca \\
\Rightarrow_{\Pi_1} abccca \\
\Rightarrow_{\Pi_1} abccca \\
\Rightarrow_{\Pi_1} abccca \\
\Rightarrow_{\Pi_1} abccca \\
\Rightarrow_{\Pi_1} aabccc.
\]

Example 2. Let $\Pi_2$ be the semi-Thue process $\{# \rightarrow #a, a1 \rightarrow 11a, a# \rightarrow #\}$ on the alphabet $\{#, a, 1\}$. Starting from the “seed” string $#1#$, this process generates the powers of two, in the sense that $#1# \Rightarrow_{\Pi_2} #1^n#$ if, and only if, $n$ is a power of 2. For instance:

\[
#1# \Rightarrow_{\Pi_2} #1a# \\
\Rightarrow_{\Pi_2} #11a# \\
\Rightarrow_{\Pi_2} #11#.
\]
A context-free grammar can be regarded as a special kind of semi-Thue process in which one symbol in the alphabet is distinguished as the start symbol, some other members of the alphabet are distinguished as terminals, every production has exactly one non-terminal on its left-hand side, and the language generated by the grammar is the set of strings of terminals that can be derived from the start symbol. But semi-Thue processes are much more general. As the preceding examples show, they can be used to model computations that are beyond the reach of context-free grammars.

The word problem for a given semi-Thue process $\Pi$ is the problem of determining which strings can be derived from which other strings in $\Pi$ — that is, to determine the membership of the set {$$(w, w') \mid w \Rightarrow_{\Pi} w'$$}.

Semi-Thue Processes That Model Turing Machines

The derivations shown in examples 1 and 2 above are vaguely reminiscent of the sequences of configurations that describe the operation of Turing machines, though there are no symbols for state markers in the semi-Thue processes illustrated there (unless you want to consider the progress marker $a$ in example 2 as a kind of machine state). But we can systematically design semi-Thue processes to emulate Turing machines by adding symbols for their states.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ be any Turing machine. We can model the computation of $M$ on an input string $s$ by setting up a semi-Thue process $\Pi_M$ that will derive the successive configurations of $M$ from its initial configuration $q_0s$.

Specifically, we’ll let the alphabet of the semi-Thue process be $\Gamma \cup Q \cup \{\#\}$, where $\#$ is a symbol that is not already in $\Gamma$ and will be used as an endmarker. For every transition $\delta(q, a) = (q', a', d)$, where $q, q' \in Q$, $a, a' \in \Gamma$, and $d \in \{L, R\}$, we’ll add several semi-Thue productions to our process:

- If $d = L$, then, for every symbol $b \in \Gamma$, we’ll have the production $bqa \rightarrow q'ba'$, to simulate the operation of overwriting $a$ with $a'$ and moving the read-write head to the left. In addition, to manage the “stay put” case in which the Turing machine’s
read-write head is already at the left end of the tape, we’ll add the production 
\[ \text{#}qa \rightarrow \text{#}q'a'. \]

- If \( d = R \), then, for every symbol \( b \in \Gamma \), we’ll have the production \( qab \rightarrow a'q'b \), to simulate the operation of overwriting \( a \) with \( a' \) and moving the read-write head to the right. In addition, to manage the case in which the read-write head moves to a previously unaccessed tape cell at the right end of the configuration, we’ll add the production \( qa\# \rightarrow a'q'\#. \)

With this setup, the question of whether \( M \) accepts a string \( s \) is equivalent to the question of whether, in \( \Pi_M \), we can derive at least one string containing \( q_{accept} \) from the initial string \( \text{#}q_0s\# \). If \( M \) accepts \( s \), then we can trace out the steps of the computation and reproduce each one as the application of the corresponding semi-Thue production in our model, ending up with a string containing \( q_{accept} \); conversely, if we have a derivation of such a string from \( \text{#}q_0s\# \), \( M \) reproduces each step in the derivation by making the corresponding state transition and hence ends up in the \( q_{accept} \) state.

We can even reduce this to one specific instance of the word problem by adding the productions \( q_{accept}a \rightarrow q_{accept} \) and \( aq_{accept} \rightarrow q_{accept} \), for every symbol \( a \) in the tape alphabet of \( M \). These productions allow the derivation to continue after the simulated Turing machine has halted, but only to allow the \( q_{accept} \) symbol to “swallow up” the other symbols in the derived string, one by one, until only \( q_{accept} \) remains. Then \( M \) accepts \( s \) if, and only if, \( \text{#}q_0s\# \Rightarrow_{\Pi_M} \text{#}q_{accept}\# \).

Considered as a model of computation, then, semi-Thue processes are at least as powerful as Turing machines, since any Turing machine \( M \) can be modelled by the semi-Thue process \( \Pi_M \). If anything, the semi-Thue process model seems more general.

### Turing Machines That Model Semi-Thue Processes

However, it is also possible to build a nondeterministic Turing machine \( M_\Pi \) that models the operation of any given semi-Thue process \( \Pi \). For each semi-Thue production \( u \rightarrow v \) in \( \Pi \), the nondeterministic Turing machine will operate as follows:

1. The machine makes a nondeterministic choice between proceeding to step 2 or step 3.
2. If the read-write head is positioned over a blank, indicating that it has reached the end of the input, the machine proceeds to step 5; otherwise, it moves right, leaving the tape unchanged, and returns to step 1.
3. The machine now makes a nondeterministic choice among the productions of \( \Pi \) (if the set of productions is empty, it proceeds at once to step 5). It then tries to match the left side of the chosen production to the tape contents, beginning with the cell currently under the read-write head. If it fails, it proceeds to step 5; otherwise, it proceeds to step 4.
4. The machine now backs up to the beginning of the matched substring and overwrites it with the right-hand side of the chosen production. If the right-hand side is longer or shorter than the left-hand side it is replacing, the machine shifts everything following the matched substring rightwards or leftwards on the tape to accommodate the change.
5. The machine moves the read-write head back to the left end of the tape and returns to step 1.

In this mode of operation, \( M_\Pi \) can transform a string \( w \) into a string \( w' \) if, and only if, \( w \Rightarrow_{\Pi} w' \). Clearly, if we give \( M_\Pi \) an initial input that encodes \( \langle w, w' \rangle \rightarrow w\#w' \),
perhaps—we can check after each pass through the steps given above (that is, in the middle of step 5) to see whether the transformed version of $w$ matches $w'$ yet, entering $q_{\text{accept}}$ if it does and returning for another pass if it does not. In that case, the language that $M_\Pi$ recognizes is \{\langle w, w' \rangle \mid w \overset{\Pi}{\Rightarrow} w'\}.

Although $M_\Pi$ is nondeterministic, Theorem 3.16 of the textbook tells us that there is an equivalent deterministic Turing machine. Thus the word problem for any semi-Thue process is Turing-recognizable.

The fact that it is possible to simulate either of these models of computation within the other shows that they are actually equivalent in power and can solve exactly the same problems. Languages that are not Turing-decidable correspond to unsolvable word problems in the semi-Thue process model, so that there is simply no way to determine, for arbitrary strings $w$ and $w'$, whether $w \overset{\Pi}{\Rightarrow} w'$. 


This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit

http://creativecommons.org/licenses/by-sa/4.0/deed.en_US

or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.