String Rewriting Systems

The first step in exploring the formal aspects of programming languages is the definition of their structure or *syntax*.

In order to accomplish this we will use a formal system known as String Rewriting System (SRS).

We begin with the definition of strings over an alphabet.

String Rewriting Systems

Definition: [Strings over an Alphabet][1]

- An *alphabet* is a finite, nonempty set – we refer to the elements of an alphabet as *symbols*.
- A finite sequence of symbols from a given alphabet is called a *string over the alphabet*.
- A string that consists of a sequence $a_1, a_2, \ldots, a_n$ of symbols is denoted by the juxtaposition $a_1 a_2 \ldots a_n$.
- The length of some string s is denoted by $|s|$ and assumed to equal the number of symbols in the string.
- Strings that have zero symbols, called *empty strings*, are denoted by ϵ with $|\epsilon| = 0$.

[1] Based on material from the book "An Introduction to the Theory of Computation," Eitan Gurari, Ohio State University,Computer Science Press, 1989.

String Rewriting Systems

Example: $\Gamma_1 = \{a, \ldots, z\}$ and $\Gamma_2 = \{0, \ldots, 9\}$ are alphabets. *abb* is a string over Γ_1, and 123 is a string over Γ_2. *ba*12 is neither a string over Γ_1 nor a string over Γ_2 but it is a string over $\Gamma_1 \cup \Gamma_2$. The string $314\ldots$ is not a string over Γ_2, because it is not a finite sequence.

Some other observations:

- The empty string ϵ is a string over any alphabet.
- The empty set $\emptyset$ is not an alphabet because it contains no element.
- The set of natural numbers is not an alphabet, because it is not finite.

String Rewriting Systems

Definition: [Kleene Closure] Given some alphabet Γ then the set of all possible strings over Γ including the empty string ϵ is denoted by Γ^* and is called the *Kleene Closure of* Γ. (Similar to the power set construction with the exception that the constructed set is always infinite.)

Example: Let $\Gamma = \{a\}$, then $\Gamma^* = \{\epsilon, a, aa, aaa, aaaa, \ldots\}$.

Example: Let $\Gamma = \{a, b\}$, then

$$\Gamma^* = \{\epsilon, a, b, aa, bb, ab, ba, aaa, aab, \ldots\}.$$

String Rewriting Systems

Definition: [String Concatenation] Given some alphabet Γ with $s_1 \in \Gamma^*$ and $s_2 \in \Gamma^*$, then the *concatenation of the strings* written as $s_1 s_2$ also belongs to Γ^*, that is the string $s_1 s_2 \in \Gamma^*$.

String Rewriting Systems

Definition: [String Rewriting System] A *string rewriting system* is a tuple $(\Gamma, \longrightarrow)$ where,

- Γ is an *alphabet*.
- $\longrightarrow$ is a finite binary relation in Γ^*, i.e., $\longrightarrow \subseteq \Gamma^* \times \Gamma^*$. Each element $(u, v) \in \longrightarrow$ is called a *(rewriting) rule* and is usually written as $u \longrightarrow v$.

An *inference step* in this formal system is: given a string $u \in \Gamma^*$ and a rule $u \longrightarrow v$ then the string u can be *rewritten* as the string $v \in \Gamma^*$. We write,

$$u \Rightarrow v.$$

Note: Rule definitions, $u \longrightarrow v$, and rule applications or inference steps, $u \Rightarrow v$, are two separate things and we use different symbols.

String Rewriting Systems

In order for an SRS $(\Gamma, \rightarrow)$ to be useful we allow rules to be applied to *substrings* of given strings; let $s = xuy, t = xvy$ with $x, y, u, v \in \Gamma^*$, and a rule $u \rightarrow v$, then we say that s *rewrites to* t and as before we write,

$$s \Rightarrow t.$$

More formally,

Definition: [one-step rewriting relation] Let $(\Gamma, \rightarrow)$ be a string rewriting system, then the *one-step rewriting relation* $\Rightarrow \subseteq \Gamma^* \times \Gamma^*$ is the set with $(s, t) \in \Rightarrow$ for strings $s, t \in \Gamma^*$ if and only if there exist $x, y, u, v \in \Gamma^*$ with $s = xuy, t = xvy$, and $u \rightarrow v$.

In plain English: any two string s, t belong to the relation $\Rightarrow$ if and only if they can be related by a rewrite rule.

String Rewriting Systems

Exercise: Given an SRS with $(\Gamma, \rightarrow)$, show that $\rightarrow \subseteq \Rightarrow$.

(Spoiler alert, next page holds the solution)

Proof: Let $(\Gamma, \rightarrow)$ be an SRS. We use the definition of a subset,

$$\rightarrow \subseteq \Rightarrow \text{ iff } \forall e \in \rightarrow . e \in \Rightarrow,$$

for our proof. There is nothing to prove for the 'only if' direction. More interesting is the 'if' direction, if we can show that all elements of $\rightarrow$ are also elements of $\Rightarrow$ then it follows from the definition that $\rightarrow \subseteq \Rightarrow$.

Observe that an element of $\Rightarrow$ is the pair (xuy, xvy) with $u, v, x, y \in \Gamma^*$ if $(u, v) \in \rightarrow$. Thus, $\Rightarrow$ contains pairs of strings where the first string contains a substring that is the left side of a rule in the rewriting system. Also observe that $(u, v) \in \Rightarrow$ with x and y the empty strings. It follows that all elements of $\rightarrow$ are members of $\Rightarrow$.$\square$

String Rewriting Systems

Given a string rewriting system $(\Gamma, \rightarrow)$, we can obviously apply the rewriting rules to the results of a rewriting step. This gives rise to *derivations*

$$s_n \Rightarrow s_{n-1} \Rightarrow \ldots \Rightarrow s_1 \Rightarrow s_0,$$

with $s_k \in \Gamma^*$.

We say that s_0 is a *normal form* if s_0 cannot be rewritten any further.

The *transitive closure* $\Rightarrow^*$ of the one-step rewriting relation is the set all pairs of strings that are related to each other via zero or more rewriting steps, e.g.,

$$s_n \Rightarrow^* s_0,$$

and

$$s_i \Rightarrow^* s_i.$$

It is easy to see that the following holds,

$$\rightarrow \subseteq \Rightarrow \subseteq \Rightarrow^*$$

String Rewriting Systems

Example: The urn game. An urn contains black and white beads. The game has the following rules:

- if you remove two black beads you have to replace them with a black bead.
- if you remove two white beads you have to replace them with a black bead.
- if you remove a white and a black bead you have to replace them with a white bead.

Given the contents of an urn, what is the outcome of the game?

The game can be set up as a string rewriting system $(\Gamma, \rightarrow)$. Let $\Gamma = \{b, w\}$ and let $\rightarrow$ be defined by the following pairs,

$$\begin{aligned} bb &\rightarrow b \\ ww &\rightarrow b \\ bw &\rightarrow w \\ wb &\rightarrow w \end{aligned}$$

$$bwbw \Rightarrow bww \Rightarrow ww \Rightarrow b$$

$$bbww \Rightarrow bww \Rightarrow ww \Rightarrow b$$

$$bbw \Rightarrow bw \Rightarrow w$$

$$bwb \Rightarrow bw \Rightarrow w$$

String Rewriting Systems

Observations:

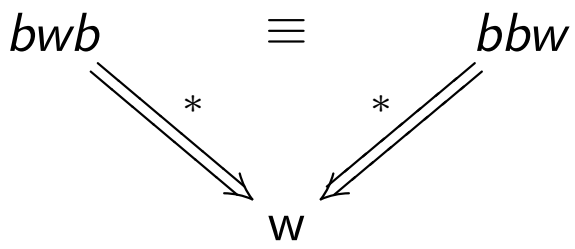
- It can be shown that for each urn there exists a unique normal form, the order of rule application does not matter.
- If we interpret $\Rightarrow^*$ as an equivalence relation (it is actually a preorder - symmetry is typically not preserved in SRSs) then we can interpret the normal form as a *representative value* for an urn since it is equivalent to the original string according to our view of rewriting as an equivalence. Consider,

$$bwb \Rightarrow bw \Rightarrow w,$$

the normal form 'w' can be considered the *representative value* for the urn .

String Rewriting Systems

Now, we say that two urns are equivalent if they have the same normal form,



String Rewriting Systems

Example: Palindrome generator. We construct a string rewriting system $(\Gamma, \rightarrow)$ with $\Gamma = \{a, b, \ldots, z, \alpha\}$ and $\rightarrow$ defined by the pairs,

$$
\begin{array}{rcl}
\alpha & \rightarrow & a\alpha a \\
\alpha & \rightarrow & b\alpha b \\
 & \vdots & \\
\alpha & \rightarrow & z\alpha z \\
a\alpha a & \rightarrow & a \\
b\alpha b & \rightarrow & b \\
 & \vdots & \\
z\alpha z & \rightarrow & z \\
\alpha & \rightarrow & \text{“”}
\end{array}
$$

We often refer to α as the *start symbol* because all derivations start with this symbol.

Derivation: $\alpha \Rightarrow r\alpha r \Rightarrow ra\alpha ar \Rightarrow rad\alpha dar \Rightarrow radar$

Exercise: Derive the normal form: *racecar*
Exercise: Derive the normal form: *redder*

String Rewriting Systems

Observations:

- Observe how SRSs can generate structure in the normal forms (this is what we will rely on when we use SRSs to describe languages).
- Unrestricted SRSs are equivalent to Turing Machines (see proof sketch on the following pages).
- Our restricted SRSs generate precisely the languages that TMs recognize – that is languages generated by restricted SRSs are *Turing Recognizable* (see proof sketch on the following pages).
- The fact that languages generated by restricted SRSs are Turing recognizable is both a blessing and a curse: we know that we can build machines that can process these languages but the machines can sometimes get stuck in an infinite loop. What we need is Turing decidable languages – we will see that we can reach this goal by restricting the shape of the SRSs to *context-free grammars*.

String Rewriting Systems

Proposition: SRSs are equivalent to Turning machines.

Proof: We show equivalence by showing that each can simulate the other.

We first show that TMs can simulate SRSs. Assume that we have a three-tape machine with one tape holding the input string, the other holding the list of rules, and the third is used as a scratchpad. It is easy to see that computation works by pattern matching of left-sides of rules to the string on the first tape. Computation stops once no further left-sides of rules can be identified in the string on the first tape.

We now show that SRSs can simulate turing machines. First we introduce special symbols such as beginning and end of input, current position of RW-head, and special state information if necessary. We then partition the rules into sections: one for moving the virtual RW-head forward, one for moving it backward, and one for doing the actual computations all using the special symbols. When no further rules can be applied to the input string then computation will halt.

String Rewriting Systems

Proposition: Strings generated by restricted SRSs are Turing recognizable.

Proof: Given a restricted SRS we can simulate this SRS with a TM (see previous proof). We can turn this TM into a recognizer by emitting an *accept* if the simulated SRS generates a normal form that matches the input string, otherwise we reject. It is clear that this machine is a recognizer since every normal form generated by the original SRS is accepted by the TM. However, there is the possibility that the machine will loop forever trying to derive a particular normal form, therefore the TM is a recognizer and not a decider (deciders always halt).

String Rewriting Systems

Shameless Plug: If these kind of computational issues interest you take my CSC544 class this coming spring, Spring '17. We will investigate decidability and Turing recognizability as well as the Turing completeness of a number of interesting computational models including the lambda calculus which is a special kind of rewriting system.