CSE 311 Section 8 - Languages

Alysa Meng

1. Languages

A formal language L is a set of strings (these strings are often called words), and is a subset of all strings (of finite length) Σ^* that can be made from an alphabet Σ. For instance, we can define $\Sigma = \{0, 1\}$ (for binary strings). Σ^* or $\{0, 1\}^*$ denotes the set of all binary strings.

We use recursive sets, regular expressions, and context-free grammars to recognize a formal language. (Later we'll see finite state machines as another representation, and there are more representations we don't cover in 311... see CSE 431 for those)

2. Brief and informal recap of representations

There's a lot of notation and it's important to not mix these.

Recursive sets of strings

We define a basis and recursive step.

e.g., Basis: $\varepsilon \in S$ and recursive step: if $x \in S$ then $x0 \in S, x1 \in S$ (where $x0 \in S$ says appending a 0 to the end of anything in the set gives another element in the set). If we want to prepend, we use the concatenation notation ($0 \bullet x \in S$, here 0 is treated as a string rather than a character).

Regular expressions

We defined regular expressions recursively, but in practice, these are the expressions with $*, \cup, ()$.

e.g., $(0^*1^*)^*$ and $(0 \cup 1)^*$ both correspond to the language of ALL binary strings.

Context-free grammars

These are the ones with a start symbol and production rules (the thing on the left side of the rule is a non-terminal symbol). These production rules inherently define a recursive process to generate strings in a language. Sometimes, you'll need more than one non-terminal symbol (especially to keep track of counts). Our alphabet Σ is seen when defining terminal symbols (can't be replaced).

e.g., $S \to 1S \mid 0S \mid \varepsilon$, also ALL binary strings. If we want to check that a string like 0010 can be generated by a language, we can do a pick-and-replace process to generate it with the production rules: $S \implies 0S \implies 00S \implies 001S \implies 0010S \implies 0010$.

3. The relationship between representations

All representations above can describe any finite language. Context-free grammars and regular expressions both can be defined recursively. Any language that can be recognized with a regular expression can be recognized by a context-free grammar. Context-free grammars are pretty powerful, but there are languages that no CFG can recognize.

Regular expressions vs. context-free grammars

Given any language recognized by a regular expression, we have a context-free grammar that recognizes it.

Consider $(0 \cup 1)^*00$, regular expression for binary strings ending in 00. We can break this up into two parts: $A = (0 \cup 1)^*$ and $B = 00$. Concatenating A, B together gives us the original regular expression.

A context-free grammar nicely follows (with S as the start symbol):

$S \rightarrow AB$ (pattern that matches the concatenation)

$A \rightarrow 1A \mid 0A \mid \varepsilon$ (matches all binary strings)

$B \rightarrow 00$ (matches the ending 00)

However, not every context-free grammar can be expressed as a regular expression (like the set of all palindromes).

Context-free grammar vs. recursively defined sets

If a context-free grammar with start symbol S as the only non-terminal recursively defines the set of strings that S can generate (a typical recursive set we've worked with).

If a context-free grammar has more than one non-terminal symbol, then it's a simultaneous recursive definition of the sets of strings generated by each non-terminal/production rule.

Thanks for reading!