Chapter 5

Language Theory

One way to study the region algebra is to examine where it fits into the language hierarchy: finite languages, regular languages, context-free languages, etc. This question is complicated by two issues. First, the region algebra describes substrings within a string, not the string itself, so it is necessary to define what is meant by the language recognized by a region algebra expression. Second, the pure region algebra includes only two ground terms, Ω and $\emptyset$. As a result, the class of languages generated by a pure region algebra expression is relatively trivial. In order to make the question relevant and interesting, we have to augment the region algebra with additional ground terms, such as literal strings, regular expressions, and context-free grammars. This is not an artifice, however; it corresponds to the way the region set algebra is designed to be used in practice, to combine text structure detected by other parsers, such as regular expressions and context free grammars.

This chapter establishes the following results:

- The region algebra with only Ω and $\emptyset$ as ground terms (and omitting *forall*) can only recognize the set of all possible regions or the empty set.
- The region algebra with literal strings as ground terms (and omitting *forall*) can recognize a strict subset of regular languages called *noncounting languages* [MP71].
- The region algebra with regular expressions as ground terms can recognize the regular languages.
- The region algebra with context-free grammar nonterminals as ground terms (and omitting *forall*) can recognize a proper superset of the context-free languages, and a subset of the context-sensitive languages. Whether *all* context-sensitive languages can be recognized by the region algebra is still an open question.

5.1 Preliminary Definitions

We begin with some definitions, following the standard conventions used by Hopcroft and Ullman [HU79]. Let Σ be a finite set of symbols, or *alphabet*. Σ^* is the set of all strings over the alphabet Σ, including the empty string ϵ. A *language* L is a subset of Σ^*. A *language class* is a collection of languages.

We now define some familiar language classes. $\mathcal{F}$ is the class of finite languages, i.e. languages L such that the set L is finite. $\mathcal{R}$ is the class of *regular* languages, where a regular language has the following recursive definition:

- $\emptyset$ (the empty language) and $\{\epsilon\}$ (the language containing the empty string) are regular;
- $\{a\}$ for any symbol $a \in \Sigma$ is regular;
- if L and L' are regular, then $L \cup L'$ is regular;
- if L and L' are regular, then $LL' = \{xx' | x \in \text{L and } x' \in L'\}$ is regular;
- if L is regular, then $L^* = \{x_1 \cdots x_n | n \geq 0 \text{ and } \text{x}_i \in L\}$ is regular.
- there are no other regular languages.

$\mathcal{CFL}$ is the class of *context-free* languages, which are defined as follows. A *context-free grammar* is a four-tuple $G = (V, \Sigma, S, P)$ where V is a finite set of nonterminals, Σ is a finite set of terminals ($V \cap \Sigma = \emptyset$) , $S \in V$ is the start symbol, and P is a finite set of productions of the form $A \rightarrow \alpha$, where A is a nonterminal and α is a string of terminals and nonterminals. We write $\alpha \Rightarrow_G \beta$ if the string β can be obtained from the string α by replacing some nonterminal that appears on the left side of a production in G by its right side. We write $\alpha \Rightarrow_G^* \beta$ if β can be obtained from α by zero or more applications of productions. Then the language generated by a context-free grammar is $L(G) = \{x \in \Sigma^* | S \Rightarrow_G^* x\}$. A language is *context-free* if it is generated by some context-free grammar.

Finally, $\mathcal{CSL}$ is the class of *context-sensitive* languages, defined as follows. A *context-sensitive grammar* is a grammar $G = (V, \Sigma, S, P)$ in which productions take the form $\alpha \rightarrow \beta$, where both α and β are strings of terminals and nonterminals and $|\alpha| \leq |\beta|$. We write $\gamma \Rightarrow_G \delta$ if the string δ can be obtained from the string γ by replacing some string α that appears on the left side of a production in G by its right side. Extending $\Rightarrow_G$ to $\Rightarrow_G^*$ in the usual fashion, we say that the language generated by a context-sensitive grammar is $L(G) = \{x \in \Sigma^* | S \Rightarrow_G^* x\}$, and call a language *context-sensitive* if it is generated by some context-sensitive grammar.

It is well-known that $\mathcal{F} \subset \mathcal{R} \subset \mathcal{CFL} \subset \mathcal{CSL}$, where all the inclusions are strict. These relationships are part of the *Chomsky language hierarchy*.

A class of languages is *closed* under some operator if applying the operator to languages in the class always produces another language in the class. Regular languages are closed under union, intersection, and complement. Context-free languages are closed under union, but *not* under intersection or complement. Context-sensitive languages are closed under all three operations [Imm88].

5.2 Finite State Transducers

Several proofs about the power of the region algebra will depend on representing algebra operators as *finite state transducers* (FST), also called generalized state machines. A finite state transducer is a a nondeterministic finite state machine in which every transition is labeled not only by an input symbol, but also by an output string drawn from another, possibly different alphabet. Formally, a finite state transducer is a tuple $M = (Q, \Sigma, \Delta, \delta, q_0, F)$ where Q is a set of states; Σ and Δ are the

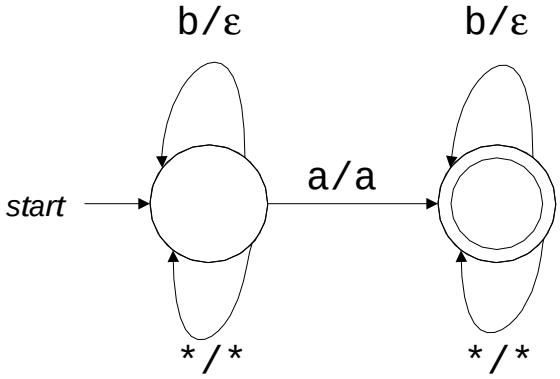

Figure 5.1: A finite state transducer that tests for the letter a and deletes the letter b.

input and output alphabets, respectively; δ is a mapping from $Q \times \Sigma$ to finite subsets of $Q \times \Delta^*$; q_0 is the starting state; and F is the set of accepting states. Note that a transition is labeled with an output *string*, which may be ϵ (the empty string).

Figure 5.1 shows a simple transducer as a state diagram. Transition labels take the form a/w, indicating that the transition occurs (nondeterministically) on input symbol a and emits the output string w, which may be the empty string ϵ. For convenience, state diagrams use $*$ to represent all input symbols that do not already have an explicit transition, and $*/*$ to represent a transition that copies the input symbol to the output. Accepting states are indicated by a double circle. The transducer shown in Figure 5.1 accepts any input string that contains at least one a and emits an output string with all occurrences of b deleted.

A finite state transducer maps an input string x to an output string y if there is some sequence of transitions from q_0 to a final state in F such that the concatenation of the input symbols is x and the concatenation of the output strings is y. Formally, define the extended transition function δ^* mapping $Q \times \Sigma^*$ to $Q \times \Delta^*$ as follows:

$$\begin{aligned} \delta^*(q\,\epsilon) &= \{(q, \epsilon)\} \\ \delta^*(q, xa) &= \{(p, w_1 w_2) | \exists p'.(p', w_1) \in \delta^*(q, x) \text{ and } (p, w_2) \in \delta(p', a)\} \end{aligned}$$

If L is a language over Σ, the mapping $M(L)$ is the language over Δ defined by

$$M(L) = \{y | (p, y) \in \delta^*(q_0, x) \text{ for some } x \in L \text{ and } p \in F\}$$

For our purposes, an important property of finite state transducers is that R and $\mathcal{CFL}$ are closed under mapping through an FST:

Theorem 3. *If L is a regular (or context-free) language and M is any finite state transducer, then $M(L)$ is also regular (or context-free).*

A proof of Theorem 3 is given by Hopcroft and Ullman [HU79, Theorem 11.1].

Context-sensitive languages are closed under a more constrained form of finite state transducers. An FST is *ϵ-free* if no transition has an output string ϵ. Then Hopcroft and Ullman also prove [HU79, Theorem 11.1]:

Theorem 4. *If L is context-sensitive and M is an ϵ-free finite state transducer, then $M(L)$ is context-sensitive.*

5.3 Languages Recognized by a Region Expression

Recall that a region is identified by its start and end offsets, so the region y in the string xyz is described by the pair $[|x|, |x| + |y|]$. For any region expression E and any string w, let E/w be the set of regions produced by applying the expression E to w. For example, "i" *just-before* "s" $/mississippi = \{[1, 2], [4, 5]\}$. Note that the set of regions is a set of locations in w, not a set of strings, so E/w is not itself a language. The question now is how to translate from regions to strings, so that the region algebra can be related to the Chomsky language hierarchy.

Several languages might reasonably be associated with a region expression E:

- The *inner language* $L_0(E)$ is the set of substrings that correspond to the regions matching E:

  $$L_0 = \{y | \exists y, z.[|x|, |x| + |y|] \in E/xyz\}$$

  The inner language captures the matches themselves, but completely omits the context. For example, the inner language of "i" *just-before* "s" consists of just one string, i, which is the same as the inner language of the (semantically quite different) expression "i".

- The *outer language* $L_1(E)$ is the set of strings w that have at least one match to the expression E:

  $$L_1(E) \equiv \{w | E/w \neq \emptyset\}$$

  The outer language represents the context of the matches, but not the locations of the matches themselves. For example, the outer language of "i" *just-before* "s" is the regular language $\Sigma^*\{is\}\Sigma^*$, which is the same as the outer language of the expression "s" *after* "i".

Since neither language fully captures the semantics of the region algebra, it is also useful to define a language that explicitly indicates the location of a region in context. This language consists of strings over the extended alphabet $\Sigma \cup \{[,]\}$, where the square brackets represent unique delimiters that do not occur in the underlying alphabet Σ. The square brackets are used to delimit the substring that matches the region expression.

The *region language* for a region expression $L(E)$ is therefore defined as

$$L(E) \equiv \{x[y]z | [|x|, |x| + |y|] \in E/xyz\}$$

For example, $L($ "i" *just-before* "s" $) = \Sigma^*\{[i]s\}\Sigma^*$. Strings in this language include $m[i]ssissippi$, $miss[i]ssippi$, $[i]s$, and $xyz[i]spdq$.

The region language is stronger than the inner and outer languages, in the sense that both the inner language and the outer language can be obtained from the region language. The inner language is the set of strings inside the square brackets: $L_0(E) = \{y | x[y]z \in L(E)\}$. The outer language is the region language with the square brackets removed: $L_1(E) = \{xyz | x[y]z \in L(E)\}$.

The region language is also stronger in the sense that its membership in a language class implies that the inner and outer languages belong to the same class:

Theorem 5. *If $L(E)$ is regular (or context-free), then $L_0(E)$ and $L_1(E)$ are also regular (or context-free).*

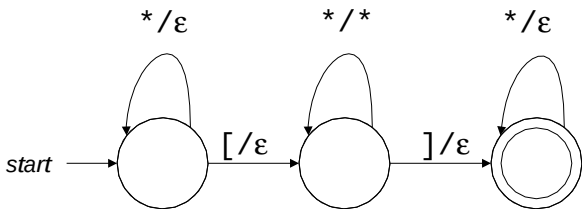

Figure 5.2: Finite state transducer M_0 maps a region language $L(E)$ to the corresponding inner language $L_0(E)$ by erasing symbols outside the square brackets and the square brackets themselves.

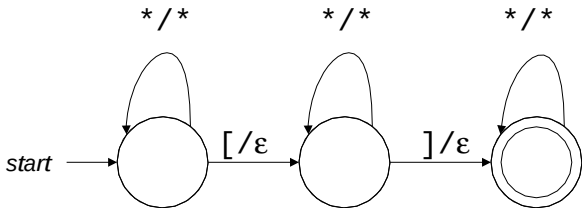

Figure 5.3: Finite state transducer M_1 maps a region language $L(E)$ to the corresponding outer language $L_1(E)$ by erasing just the square brackets.

Proof. We will exhibit a finite state transducer M_0 that maps $L(E)$ into $L_0(E)$, and another transducer M_1 that maps $L(E)$ into $L_1(E)$. Thus, by Theorem 3, if $L(E)$ is regular (or context-free), then $L_0(E)$ and $L_1(E)$ are also regular (or context-free). The M_0 transducer is shown in Figure 5.2. It accepts any input string with a square-bracketed region, such as $x[y]z$, and emits only the part in brackets, y. This has the effect of mapping a region language $L(E)$ to the corresponding inner language $L_0(E)$. The M_1 transducer, shown in Figure 5.3, simply erases the square brackets, transforming a string like $x[y]z$ into xyz. It should be clear that this maps the region language $L(E)$ to its outer language $L_1(E)$. □

The converse of Theorem 5 is not necessarily true, however, because the inner and outer languages do not contain enough information. Consider the following grammar:

$$S \;\rightarrow\; aSb \,|\, \epsilon$$

This grammar matches a context-free language of the form $a^n b^n$. Let E be the expression a^*b^* *in* S. Then the inner language $L_0(E)$ is simply the regular language a^*b^*, but the region language $L(E)$ is $a^i[a^{n-i}b^j]b^{n-j}$, which is not regular but context-free.

A similar example shows regularity of the outer language does not imply regularity of the region language. Consider the grammar

$$\begin{aligned} S &\rightarrow aS \,|\, Sb \,|\, B \\ B &\rightarrow aBb \,|\, \epsilon \end{aligned}$$

Let E be the expression B *in* S. The outer language $L_1(E)$ is the set of all strings recognized by the entire grammar, which is actually the regular language a^*b^*. The region language $L(E)$, however, corresponds to the language $a^*[a^nb^n]b^*$, which is not regular but context-free.

5.4 Restricted Algebras

In order to simplify the presentation, it will be helpful to restrict discussion to certain subsets of the algebra operators. such as relational or set operators. Each subset is denoted by a letter:

- R: the relational operators *before, after, in, contains, overlaps-start,* and *overlaps-end*.
- S: the set operators $\cup, \cap$, $-$, Ω, and $\emptyset$.
- A: the *forall* operator.

The most difficult operator to deal with semantically is *forall*, because it binds variables. However, the region algebra presentation in Chapter 3 depended on *forall* to define *then* (string concatenation), which is a fundamental operator in other language formalisms like regular expressions and context-free grammars. When we want to avoid the difficulties of *forall* but still have access to a concatenation operator, we will treat *then* as if it were a primitive operator:

- T: the operator *then.*

Region algebra expressions may also include *ground terms* (nullary operators). We consider four kinds of ground terms in this chapter:

- $\emptyset$: no ground terms (other than the set operators Ω and $\emptyset$).
- $\mathcal{F}$: strings in a finite language (i.e., literal strings) may be used as ground terms.
- $\mathcal{R}$: regular expressions may be used as ground terms.
- $\mathcal{CFL}$: nonterminals of a context-free grammar may be used as ground terms.

Note that each class of ground terms is a superset of the previous class: regular languages include finite languages as a special case, and context-free languages include regular languages.

We use these letters to name a subset of the region algebra. The permitted operators are listed, followed by a subscript indicating the permitted ground terms. For example:

- $RS_\emptyset$ is the class of region algebra expressions using only relational operators (R), set operators (S), and no ground terms other than Ω and $\emptyset$.
- $RST_\mathcal{F}$ is the class of expressions using relational operators, set operators, concatenation (*then*), and literal strings.
- $RSTA_\mathcal{R}$ is the class of expressions using any algebra operator and regular expressions as ground terms. Since *then* can be defined in terms of *forall*, this class can also be written $RSA_\mathcal{R}$. I prefer to write $RSTA$, since it makes it clearer that $RSTA$ is a superset of RST.

Obviously, $RS_G \subset RST_G \subset RSTA_G$ for any choice of ground terms G, and $P_\emptyset \subset P_\mathcal{F} \subset P_\mathcal{R} \subset P_{\mathcal{CFL}}$ for any choice of operators P.

Not all restrictions will be interesting. This chapter will focus on RST (the region algebra without *forall*) and $RSTA$ (with *forall*), for all the choices of ground terms.

5.5 Algebra Semantics

We can now define the region algebra operators in terms of the region languages they recognize.

Set Operators (S)

The set operators are straightforward. For example, $x[y]z$ is in the language $L(E_1 \cap E_2)$ if and only if it is in both $L(E_1)$ and $L(E_2)$.

$$\begin{aligned} L(E_1 \cap E_2) &= L(E_1) \cap L(E_2) \\ L(E_1 \cup E_2) &= L(E_1) \cup L(E_2) \\ L(E_1 - E_2) &= L(E_1) - L(E_2) \end{aligned}$$

The nullary set operators are defined as follows:

$$\begin{aligned} L(\Omega) &= \{x[y]z | x, y, z \in \Sigma^*\} \\ L(\emptyset) &= \emptyset \end{aligned}$$

Relational Operators (R)

The relational operators are similarly straightforward. For example, if $x[y]z$ is in $L(E)$, then for all u, v, w such that $x = uvw$, we have $u[v]wyz$ in $L(\textit{ before } E)$.

$$\begin{aligned} L(\textit{ before } E) &= \{v[w]xyz | vwx[y]z \in L(E)\} \\ L(\textit{ after } E) &= \{vwx[y]z | v[w]xyz \in L(E)\} \\ L(\textit{ in } E) &= \{vw[x]yz | v[wxy]z \in L(E)\} \\ L(\textit{ contains } E) &= \{v[wxy]z | vw[x]yz \in L(E)\} \\ L(\textit{ overlaps-start } E) &= \{v[wx]yz | vw[xy]z \in L(E)\} \\ L(\textit{ overlaps-end } E) &= \{vw[xy]z | v[wx]yz \in L(E)\} \end{aligned}$$

Then Operator (T)

Since we will need *then* as a primitive operator, we define it:

$$L(E_1 \textit{ then } E_2) = \{w[xy]z | w[x]yz \in L(E_1) \wedge wx[y]z \in L(E_2)\}$$

Following the convention in the language theory literature, the expressions in this chapter will omit the explicit *then*, representing the concatenation of two region expressions by simple juxtaposition:

$$E_1 E_2 \equiv E_1 \textit{ then } E_2$$

Literal Strings ($\mathcal{F}$)

The region language corresponding to a literal string expression w is the set of all strings that contain w in square brackets:

$$L(x) = \Sigma^* \{ [w] \} \Sigma^*$$

Regular Expressions ($\mathcal{R}$)

The region language corresponding to a regular expression R is the set of all strings that contain a string recognized by R in square brackets:

$$L(R) = \Sigma^* \{ [\} L(R) \{] \} \Sigma^*$$

Context-Free Grammar Nonterminals ($\mathcal{CFL}$)

Suppose a context-free grammar G has start symbol S and some nonterminal A. Then

$$L(A) = \{ x[y]z | S \Rightarrow_G^* xAz \wedge xAz \Rightarrow_G^* xyz \}$$

These definitions are sufficient to prove results about the region algebra without the *forall* operator: $RST_\emptyset$, $RST_\mathcal{F}$, $RST_\mathcal{R}$, and $RST_\mathcal{CFL}$. Discussion of *forall* is postponed to a later section, since it requires some extra machinery to define its semantics.

5.6 Trivial Ground Terms ($RST_\emptyset$)

We first dispense with a trivial class, $RST_\emptyset$. Algebra expressions of this class can only use Ω and $\emptyset$ as ground terms. Since such an expression has no way to test the symbols of the string, it must either accept *all* possible region languages ($L(\Omega) = \{ x[y]z | x, y, z \in \Sigma^* \}$) or none at all ($L(\emptyset) = \emptyset$). Thus the region language class recognized by $RST_\emptyset$ contains precisely two languages, $L(\Omega)$ and $L(\emptyset)$.

Theorem 6. *The region languages recognized by $RST_\emptyset$ are $L(\Omega)$ and $L(\emptyset)$.*

Proof. Proof by structural induction on an algebra expression E.

- Ground terms: only Ω and $\emptyset$ are allowed, so an expression consisting of just one term recognizes either $L(\Omega)$ or $L(\emptyset)$.
- Relational operators: If $L(E) = \emptyset$, then $L(\textit{ before } E) = \emptyset$, trivially. So suppose $L(E) = L(\Omega)$. For any string xyz, we must have $xyz[] \in L(E)$, so $x[y]z \in L(\textit{ before } E)$. Since xyz was arbitrary, $L(\textit{ before } E) = L(\Omega)$. Similar arguments work for *after, in, contains, overlaps-start*, and *overlaps-end*.

- Set operators: the intersection, union, or difference of any combination of $\{L(\Omega), L(\emptyset)\}$ can only be $L(\Omega)$ or $L(\emptyset)$.

- Concatenation: if either $L(E_1)$ or $L(E_2)$ is empty, then $L(E_1E_2)$ is empty. So suppose $L(E_1) = L(E_2) = L(\Omega)$. For any string xyz, we must have $x[y]z \in L(E_1)$ and $xy[]z \in L(E_2)$, so $x[y]z \in L(E_1E_2)$. Since xyz was arbitrary, $L(E_1E_2) = L(\Omega)$.

□

Corollary 2. *The only inner languages and outer languages recognized by $RST_\emptyset$ are Σ^* and $\emptyset$.*

This theorem shows that the region algebra is of little use by itself. It depends on other ground terms, such as literal strings, regular expressions, or context-free grammars, to recognize useful languages.

5.7 Finite Ground Terms ($RST_{\mathcal{F}}$)

The next interesting specialization of the region algebra is $RST_{\mathcal{F}}$, which adds literal strings (including the empty string ϵ) to $RST_\emptyset$. This algebra is more powerful than it looks at first blush. Suppose the alphabet Σ is $\{a, b\}$. The inner language of a simple expression like a *in* aba is the finite language $\{a\}$, but the inner language of the expression $\neg$ *contains* "b" is the infinite language a^*. (Recall that the unary complement $\neg E$ is equivalent to the set difference $\Omega - E$.) Another interesting example is

$$(a\Omega) \cap (\Omega b) \cap (\neg \textit{ contains } aa) \cap (\neg \textit{ contains } bb)$$

whose inner language is $a(ba)^*b$.

The use of *contains* in these examples is not strictly necessary, because *contains* E is equivalent to $\Omega E \Omega$. For simplicity, then, we will begin by ignoring the relational operators and considering the class $ST_{\mathcal{F}}$ (set operators, concatenation, and literal strings). $ST_{\mathcal{F}}$ is the region algebra equivalent of a class of languages that have been well studied, namely *star-free regular expressions* [MP71]. As the name suggests, a star-free regular expression does not use the Kleene star operator *, but it may use intersection and complement as well as concatenation and union. Formally:

Definition 1. *A* star-free regular expression *has the following recursive definition:*

- *$\emptyset$ (the empty set), ϵ (the empty string), and a for any $a \in \Sigma$ are star-free regular expressions;*

- *if R and R' are star-free regular expressions, then $R \cup R'$, $R \cap R'$, $\neg R$, and RR' are star-free regular expressions;*

- *there are no other star-free regular expressions.*

For convenience, and to parallel the region algebra, we will use Ω in star-free regular expressions to represent the expression $\neg\emptyset$. Note that Ω matches the set of all possible strings Σ^* without having to resort to a star operator.

5.7.1 Noncounting Languages (NC)

McNaughton and Papert [MP71] show that the star-free regular expressions recognize a class of languages they call NC, or "noncounting languages". NC is a strict subset of the regular languages. The intuition behind noncounting languages is that a noncounting language must be recognized without needing to count modulo an integer ≥ 2. For example, $(aaa)^*$ is not in NC, because it requires testing whether the length of a string of a's is a multiple of 3. The language $a^*ba^*ba^*$ is in *NC*, however. Although it requires counting the number of b's and testing that the count is exactly 2, this test does not require counting modulo an integer. We can readily find a star-free regular expression for this language: $(\neg(\Omega b\Omega))b(\neg(\Omega b\Omega))b(\neg(\Omega b\Omega))$.

Formally, the language class NC is defined as follows:

Definition 2. *A language L is* noncounting *if and only if for some $n > 0$ and all strings $u, v, w \in \Sigma^*$, $uv^n w \in L$ if and only if $uv^{n+x}w \in L$ for all positive integers x.* NC *is the class of noncounting languages.*

Using this definition, we can show why $(aaa)^*$ is not in NC. Suppose there is some n that satisfies the definition for $(aaa)^*$, and take $u = \epsilon$, $v = a$, and $w = \epsilon$. If $uv^n w = a^n$ is in the language, then $uv^{n+1}w = a^{n+1}$ is *not* in the language. Conversely, if $uv^n w$ is not in the language, then either $uv^{n+1}w$ or $uv^{n+2}w$ *is* in the language. By contradiction, then, no n exists that can make $(aaa)^*$ a noncounting language. Thus NC is a strict subset of the regular languages R.

To illustrate how the definition of NC will be used in proofs, let us prove the analog of Theorem 5 for noncounting languages.

Theorem 7. *For any algebra expression E, if the region language $L(E)$ is noncounting, then the inner language $L_0(E)$ and the outer language $L_1(E)$ are also noncounting.*

Proof. Since $L(E)$ is noncounting, there exists an integer n such that $uv^n w \in L$ if and only if $uv^{n+x}w \in L$ for all strings u, v, w and positive integers x. The proof will first show that the inner language $L_0(E)$ satisfies the same property. If $uv^n w \in L_0(E)$, then $p[uv^n w]q \in L(E)$ for some $p, q \in \Sigma^*$. We can pump this string to get $p[uv^{n+x}w]q \in L(E)$ for any integer x, which implies that $uv^{n+x}w \in L_0(E)$. A similar argument shows that $uv^n w \notin L_0(E)$ implies $uv^{n+x}w \notin L_0(E)$. Since u, v, w, and x were arbitrary, we have shown that $L_0(E)$ is noncounting.

Proving that the outer language $L_1(E)$ is noncounting is slightly harder, but the techniques used will come in handy for the proofs in the next section. We will show that $L_1(E)$ satisfies Definition 2 for the integer $3n$, rather than n. Suppose we are given a string $uv^{3n}w \in L_1(E)$. Since this string is in the outer language of E, there must be some string in the region language of E that differs only in that it has a left and right bracket inserted around the matching region; i.e., there must be some $p[q]r \in L(E)$ such that $pqr = uv^{3n}w$. Any way of splitting $uv^{3n}w$ into three strings p, q, and r must leave at least v^n as a substring of either p, q, or r. Assume without loss of generality that p contains v^n, so that $p = p_1 v^n p_2$. Then we can pump $p_1 v^n p_2[q]r$ to get $p_1 v^{n+x} p_2[q]r \in L(E)$, so $p_1 v^{n+x} p_2 qr = uv^{3n+x}w \in L_1(E)$. A similar argument shows that $uv^n w \notin L_1(E)$ implies $uv^{n+x}w \notin L_1(E)$, so $L_1(E)$ is noncounting. □

5.7.2 $RST_{\mathcal{F}} = NC$

We are now ready to prove that $RST_{\mathcal{F}}$ has the same power as star-free regular expressions. First we show that the region languages recognized by $RST_{\mathcal{F}}$ are noncounting languages.

Theorem 8. *If L is the region language of an expression in $RST_{\mathcal{F}}$, then L is noncounting.*

Proof. By structural induction on the algebra expressions in $RST_{\mathcal{F}}$.

- Ground terms. The region languages of all ground terms can be described by star-free regular expressions: $L(\Omega)$ is $\Omega[\Omega]\Omega$, $L(\emptyset)$ is $\emptyset$, and $L(w)$ is $\Omega[w]\Omega$.

- Set operators. NC is closed under intersection, union, and complement (since star-free expressions are likewise closed), so if $L(E_1)$ and $L(E_2)$ are noncounting, then so are $L(E_1 \cap E_2)$, $L(E_1 \cup E_2)$, and $L(E_1 - E_2)$.

- Concatenation. Suppose $L(E_1)$ and $L(E_2)$ are noncounting. Then there exist integers $n_1 > 0$ and $n_2 > 0$ satisfying Definition 2 for $L(E_1)$ and $L(E_2)$, respectively. Let $n = \max(n_1, n_2)$. The claim is that $2n$ satisfies Definition 2 for the language $L(E_1E_2)$. We need to show that for any strings u, v, w, $uv^{2n}w \in L(E_1E_2)$ if and only if $uv^{2n+x}w \in L(E_1E_2)$ for any $x > 0$. Since $uv^{2n}w$ is a string in a region language, it must contain exactly one left and one right square bracket, in that order. Neither bracket can fall in v^{2n}, which leaves only three cases:

  1. Both brackets in u. Write $uv^{2n}w = u_1[u_2]u_3v^{2n}w$ for some u_1, u_2, u_3. Then $u_1[u_2]u_3v^{2n}w \in L(E_1E_2)$ if and only if $u_1[r]su_3v^{2n}w \in L(E_1)$ and $u_1r[s]u_3v^{2n}w \in L(E_2)$ for some $rs = u_2$. Since $L(E_1)$ and $L(E_2)$ are in NC, we have $u_1[r]su_3v^{2n+x}w \in L(E_1)$ and $u_1r[s]u_3v^{2n+x}w \in L(E_2)$, which is true if and only if $uv^{2n+x}w \in L(E_1E_2)$.
  2. Both brackets in w. Symmetric to case 1.
  3. Left bracket in u, right bracket in w. Write $uv^{2n}w = u_1[u_2v^{2n}w_1]w_2$. Then $u_1[u_2v^{2n}w_1]w_2 \in L(E_1E_2)$ if and only if for $u_1[r]sw_2 \in L(E_1)$ and $u_1r[s]w_2 \in L(E_2)$ for some $rs = u_2v^{2n}w_1$. Any way of splitting the region must leave at least v^n as a substring of either r or s. Assume without loss of generality that r contains v^n, so $r = u_2v^nt$ for some t. Then we can pump v^n to v^{n+x} because $L(E_1)$ is in NC: $u_1[u_2v^{n+x}t]sw_2 \in L(E_1)$, so $uv^{2n+x}w \in L(E_1E_2)$.

- Relational operators. The proofs for relational operators are similar to the proof for concatenation. In each case, $L(E)$ is assumed to be noncounting by the induction hypothesis, and n is the integer satisfying satisfying Definition 2 for $L(E)$. We show that $3n$ satisfies Definition 2 for $L(\textit{ op } E)$.

  – *before*: For $uv^{3n}w \in L(\textit{ before } E)$, the crucial case is $uv^{3n}w = u_1[u_2]u_3v^{3n}w$. The other two cases are trivial, like cases 1 and 2 for concatenation. $u_1[u_2]u_3v^{3n}w \in L(\textit{ before } E)$ if and only if $u_1u_2r[s]t \in L(E)$ for some $rst = u_3v^{3n}w$. Any way of splitting $u_3v^{3n}w$ into three strings r, s, t must leave at least v^n as a substring of r, s, or t, so we can pump v^n to v^{n+x}, giving $uv^{3n+x}w \in L(\textit{ before } E)$.

  – *after*: symmetric with *before*.

  – *overlaps-start*: There are two nontrivial cases. When $uv^{3n}w = u_1[u_2]u_3v^{3n}w$, the string is in $L(\textit{ overlaps-start } E)$ if and only if $u_1r[s]t \in L(E)$ for some $rst = u_2u_3v^{3n}w$ (where $|r| < |u_2|$ and $|rs| > |u_2|$). Any way of splitting $u_2u_3v^{3n}w$ into three strings

r, s, t must leave at least v^n as a substring of s or t, so we can pump v^n to v^{n+x}, giving $uv^{3n+x}w \in L(\textit{ overlaps-start } E)$. For the other hard case, where $uv^{3n}w = u_1[u_2v^{3n}w_1]w_2$, we must have $u_1r[s]t \in L(E)$ for some $rst = u_2u_3v^{3n}w$. Again, any way of forming r, s, t must leave at least v^n as a substring of s or t, so we can pump v^n to v^{n+x} and get $uv^{3n+x}w \in L(\textit{ overlaps-start } E)$

- *overlaps-end*: symmetric with *overlaps-start*.
- *in*: There are two nontrivial cases. When $uv^{3n}w = u_1[u_2]u_3v^{3n}w$, the string is in $L(\textit{ in } E)$ if and only if $p[qu_2r]s \in L(E)$ for some $pq = u_1$ and $rs = u_3v^{3n}w$. Any way of forming r and s must leave at least v^n in one of them, so we can pump to get $uv^{3n+x}w \in L(\textit{ in } E)$. The other hard case, when $uv^{3n}w = uv^{3n}w_1[w_2]w_3$, is symmetric.
- *contains*: Only one nontrivial case: when $uv^{3n}w = u_1[u_2v^{3n}w_1]w_2$, the string is in $L(\textit{ contains } E)$ if and only if $u_1r[s]tw_2 \in L(E)$ for some $rst = u_2v^{3n}w_1$. Any way of forming r, s, and t must leave at least v^n in one of them, so we can pump to get $uv^{3n+x}w \in L(\textit{ contains } E)$.

□

Corollary 3. *The inner and outer languages of an expression in $RST_{\mathcal{F}}$ are noncounting.*

The previous theorem showed that every language recognized by $RST_{\mathcal{F}}$ is noncounting. It is trivial to show the converse, that every noncounting language is recognized by some expression in $RST_{\mathcal{F}}$.

Theorem 9. *If L is noncounting, then L is the inner language of some expression in $RST_{\mathcal{F}}$.*

Proof. There is some star-free regular expression that recognizes L. Every star-free regular expression corresponds directly to an algebra expression in $ST_{\mathcal{F}}$ whose inner language is the same, and every algebra expression in $ST_{\mathcal{F}}$ is also in $RST_{\mathcal{F}}$. □

5.8 Regular Ground Terms ($RST_{\mathcal{R}}$)

We now move up to a stronger set of ground terms: regular expressions. Note that allowing regular expressions as *ground terms* in the region algebra does not mean that we allow regular expression operators anywhere in the expression. In particular, the Kleene star operator * is not permitted to have any non-regular operators, such as $\cap$, $-$, or *in*, in its scope. The other regular expression operators, union and concatenation, are also algebra operators in $RST_{\mathcal{R}}$, so this restriction does not apply to them. For example, these expressions are in $RST_{\mathcal{R}}$:

$$a^* \textit{ before } b^*$$
$$(a \cup b)^* \textit{ in } (a^*b^*)^*$$

but these are not:

$$(a \textit{ before } ab)^*$$
$$(a\Omega \cap \Omega b)^*$$

This restriction mirrors the LAPIS implementation, which uses regular expressions only as ground terms. It would be interesting to make the Kleene star a full-fledged region algebra operator — and, in fact, the theorems proved in this section imply that the Kleene star would not change the recognition power of $RST_{\mathcal{R}}$, since it is regular — but its precise definition and implementation is left as future work.

$RST_{\mathcal{R}}$ trivially recognizes all regular languages, because it includes regular expressions as a subset:

Theorem 10. *If L is regular, then L is the inner language of some expression in $RST_{\mathcal{R}}$.*

The harder proof goes in the other direction: that every language recognized by $RST_{\mathcal{R}}$ is regular.

Theorem 11. *If L is the region language for some expression in $RST_{\mathcal{R}}$, then L is regular.*

Proof. As usual, the proof proceeds by structural induction on algebra expressions in $RST_{\mathcal{R}}$.

- Ground terms. The ground terms all produce regular sets: $L(\Omega)$ is $\Sigma^*[\Sigma^*]\Sigma^*$, $L(\emptyset)$ is $\emptyset$, and $L(r)$ is $\Sigma^*[r]\Sigma^*$.
- Set operators. Regular sets are closed under intersection, union, and complement, so if $L(E_1)$ and $L(E_2)$ are regular, then so are $L(E_1 \cap E_2)$, $L(E_1 \cup E_2)$, and $L(E_1 - E_2)$.
- Relational operators. Represent each algebra operator *op* by a finite state transducer (FST) mapping $L(E)$ to $L(\,\mathit{op}\,E)$. The FSTs are shown in Figure 5.4. For example, the FST for *before* accepts input strings of the form $vwx[y]z$ and produces all possible output strings of the form $v[w]xyz$. Thus, the image of $L(E)$ under this FST is $L(\,\mathit{before}\,E)$. By Theorem 3, if $L(E)$ is regular, then $L(\,\mathit{op}\,E)$ is also regular for all relation operators *op*.

□

Corollary 4. *The inner language and outer language of any expression in $RST_{\mathcal{R}}$ are regular.*

The corollary follows from Theorem 5.

5.9 Context-Free Ground Terms ($RST_{\mathcal{CFL}}$)

We now come to the strongest variant of the region algebra under discussion in this chapter: algebra expressions with context-free grammar nonterminals as ground terms, $RST_{\mathcal{CFL}}$.

It should be noted that, unlike string literals and regular expressions, a grammar nonterminal implicitly constrains the context in which it can match. For example, consider the grammar

$$\begin{aligned} S &\rightarrow aS \mid Sb \mid A \\ A &\rightarrow cde \end{aligned}$$

Although the nonterminal A matches a literal string cde, the algebra expression A is not equivalent to the literal expression cde. A can only match a region when the entire string can be derived from the grammar. Thus $L(A)$ includes $[cde]$ and $aaa[cde]bb$, but not $bb[cde]a$.

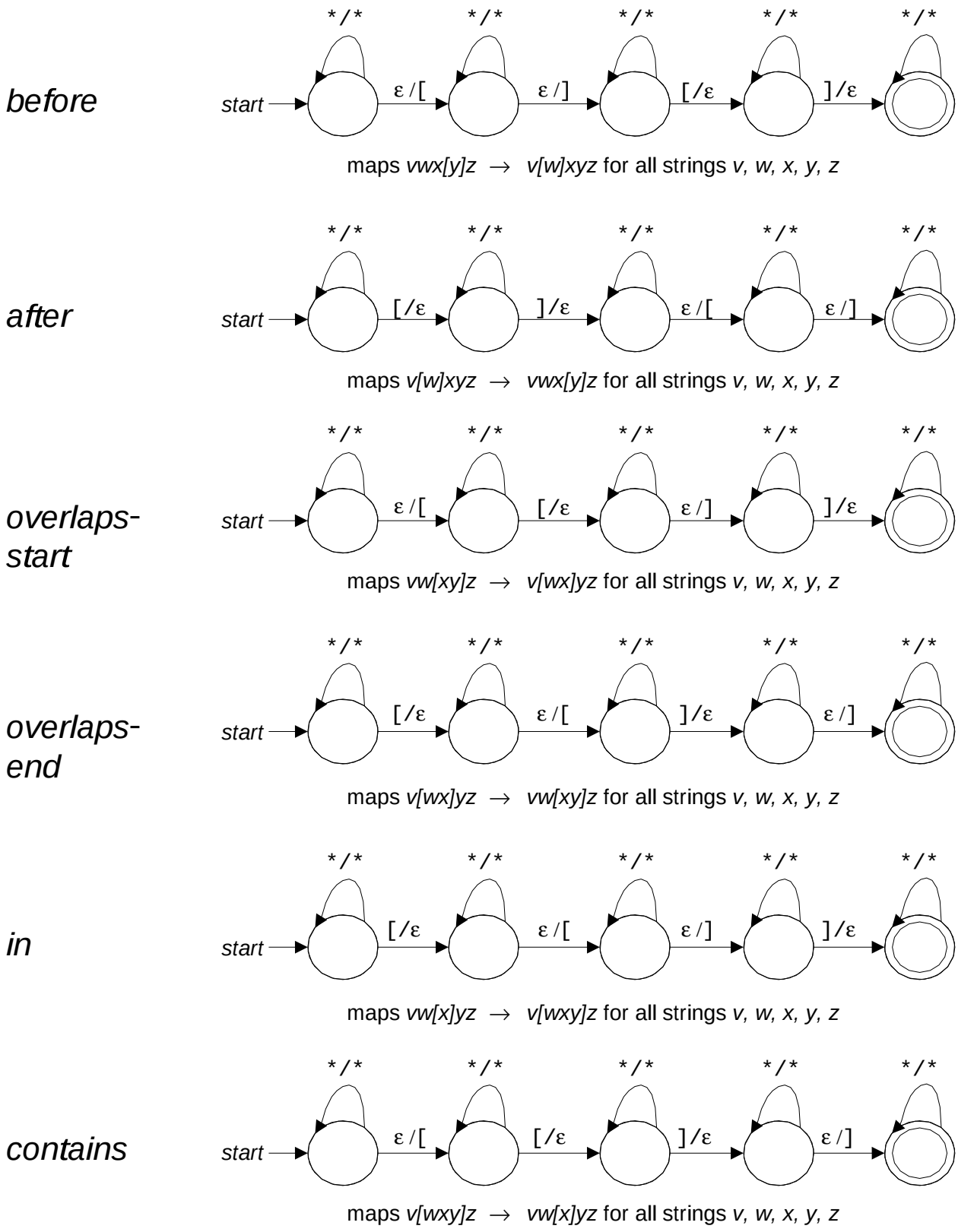

Figure 5.4: Finite state transducers for the relational operators.

Nevertheless, since any literal string or regular expression can be represented by a context-free grammar G, literal and regular ground terms can be represented by a grammar nonterminal ground term (the start symbol of G), so $RST_{\mathcal{CFL}}$ contains all the expressions in $RST_{\mathcal{F}}$ and $RST_{\mathcal{R}}$.

Theorem 12. *If L is context-free, then L is the inner (and outer) language of some expression in $RST_{\mathcal{CFL}}$.*

Proof. Since L is context-free, there is some context-free grammar $G = (V, \Sigma, S, P)$ such that $L = L(G)$. Then the algebra expression S is the desired expression, since $L_0(S) = L(G)$ and $L_1(S) = L(G)$. □

Thus we have $\mathcal{CFL} \subset \mathcal{RST}_{\mathcal{CFL}}$. Unlike $RST_{\mathcal{R}}$, which could recognize no languages outside R, $RST_{\mathcal{CFL}}$ is stronger than $\mathcal{CFL}$.

Theorem 13. *There exists an expression in $RST_{\mathcal{CFL}}$ whose inner language and outer language are not context-free.*

Proof. Consider the two languages $L_1 = a^n b^n c^*$ and $L_2 = a^* b^n c^n$, recognized by the following grammars:

$$\begin{aligned} S_1 &\rightarrow X_1 C_1 \\ X_1 &\rightarrow aX_1 b \mid \epsilon \\ C_1 &\rightarrow cC_1 \mid \epsilon \end{aligned}$$

$$\begin{aligned} S_2 &\rightarrow A_2 Y_2 \\ A_2 &\rightarrow aA_1 \mid \epsilon \\ Y_1 &\rightarrow bY_1 c \mid \epsilon \end{aligned}$$

Although both L_1 and L_2 are context-free, their intersection $L_1 \cap L_2 = \{a^n b^n c^n\}$ is not context-free [HU79, Example 6.1]. Yet $L_1 \cap L_2$ is the inner (and outer) language of the region algebra expression $S_1 \cap S_2$. □

Since $RST_{\mathcal{CFL}}$ is strictly larger than $\mathcal{CFL}$, one might ask whether it is included in the context-sensitive languages, $\mathcal{CSL}$. The answer is in the affirmative:

Theorem 14. *If L is the region language of an expression in $RST_{\mathcal{CFL}}$, then L is context-sensitive.*

Proof. By structural induction on expressions in $RST_{\mathcal{CFL}}$.

- Ground terms. $L(\Omega)$ and $L(\emptyset)$ are clearly context-sensitive. So suppose the expression is a nonterminal A in some context-free grammar $G = (V, \Sigma, S, P)$. We will form a new context-free grammar $G' = (V_1 \cup V_2, \Sigma \cup \{[,]\}\, S_1, P')$ which puts square brackets around occurrences of A in its derivation, so that $L(G')$ is the region language of A. For every nonterminal N in the original grammar G, G' contains two nonterminals, N_1 and N_0. Intuitively, every N_1 will expand to a string containing square brackets, and every N_0 will expand to a string containing no square brackets. For every production $N \rightarrow \alpha$ in the original grammar, form all possible productions $N_1 \rightarrow \alpha_1$ by adding the subscript 1 to one nonterminal in α and 0 to all other

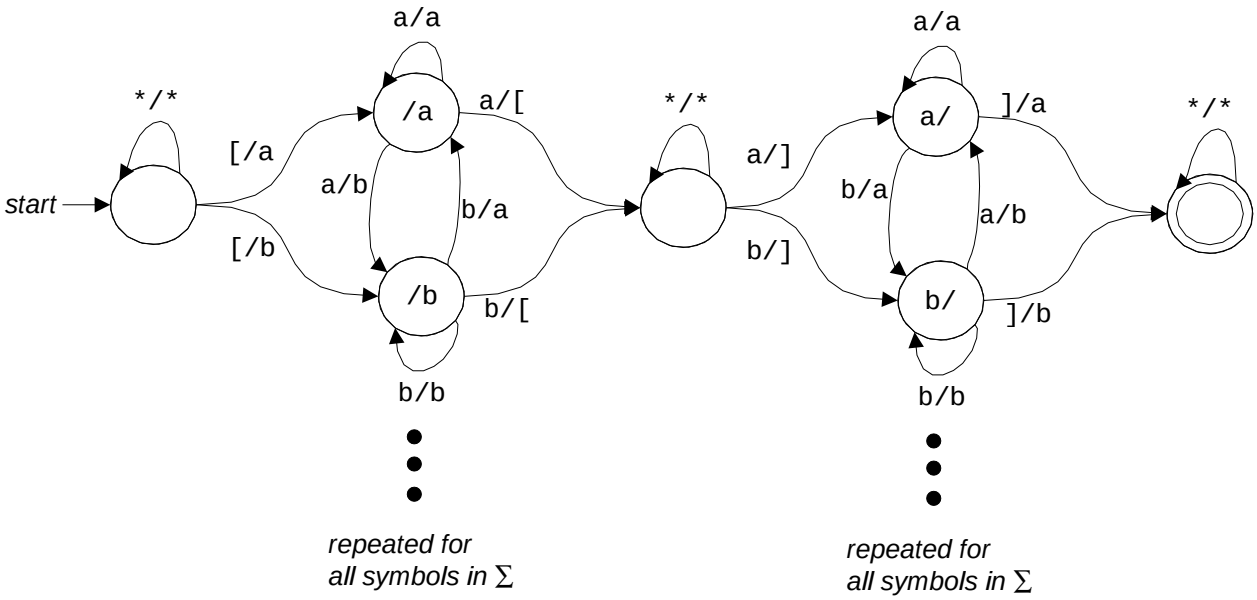

Figure 5.5: An ϵ-free FST for *in*.

nonterminals, and all productions $N_0 \rightarrow \alpha_0$ by adding the subscript 0 to all nonterminals in α. Finally, add the production $A_1 \rightarrow [A_0]$. The resulting grammar derives a string $x[y]z$ if and only if the original grammar derives xAz and thence xyz, so $L(G') = L(A)$. Since $L(G')$ is context-free, it is obviously context-sensitive.

- Set operators. Context-sensitive languages are closed under union, intersection, and complement. (Closure under union is well-known [HU79], but closure under intersection and complement is a relatively recent result [Imm88].) Thus if $L(E_1)$ and $L(E_2)$ are context-sensitive, then so are $L(E_1 \cap E_2)$, $L(E_1 \cup E_2)$, and $L(E_1 - E_2)$.

- Relational operators. Each relational operator can be represented by a finite state transducer, as shown in Figure 5.4. However, the FSTs shown in the figure incorporate ϵ transitions, and context-sensitive languages are only closed under ϵ-free FSTs (Theorem 4). However, each of the FSTs can be transformed into an equivalent ϵ-free FST by replicating states to remember symbols consumed but not yet emitted (or vice versa). At most two symbols must be stored at any time, so the set of states is still finite. For example, Figure 5.5 shows the ϵ-free FST for the *in* operator. As a result, if $L(E)$ is context-sensitive, then $L(\,op\,E)$ is also context-sensitive for any relational operator *op*.

□

It remains an open question whether all context-sensitive languages can be recognized by $RST_{\mathcal{CFL}}$.

5.10 The *forall* Operator ($RSTA_{\mathcal{R}}$ and friends)

The arguments thus far have allowed only RST — algebra expressions containing relational operators, set operators, and concatenation. It remains to deal with the *forall* operator. The principal complication of *forall* is that it binds variables, so that the meaning of an expression E depends not only on E itself but on the bindings assigned to the free variables in E. As a result, the semantics of E will no longer be a set of strings, but instead a function mapping the bindings of its free variables to the set of strings it recognizes. This function takes a particular form, namely an ***n-way regular relation.***

Before defining regular relations, let us illustrate the technique with a simple special case: expressions with only one free variable. To be precise, whenever *forall* $(\alpha : E_1) \, . \, E_2$ occurs in the expression, E_2 contains no occurrence of *forall*. Many of the uses of *forall* in Chapter 3 satisfy this restriction, particularly the adjacency and overlapping relational operators.

In *forall* $(\alpha : E_1) \, . \, E_2$, the expression E_2 contains a single free variable α that is bound to each region matching E_1. In terms of languages, the region language $L($ *forall* $(\alpha : E_1) \, . \, E_2)$ is the set of strings generated by E_2 when α is bound to each string in the region language $L(E_1)$. Thus, the expression E_2 acts as a *transducer* which takes a string bound to α as input and produces a match to E_2 as output. When E_2 is a simple relational expression like *before* α, then it is easy to see that it is actually a *finite state transducer.* (Finite state transducers for the relational operators were given in Figure 5.4.) We will see that the same property extends to all operators in the algebra: for any expression E_2 with only one free variable, E_2 is a finite state transducer. The region language recognized by the *forall* expression, $L($ *forall* $(\alpha : E_1) \, . \, E_2)$, is the language recognized by E_1 mapped through the finite state transducer for E_2. By Theorem 3, therefore, if $L(E_1)$ is regular, context-free, or context-sensitive, then $L($ *forall* $(\alpha : E_1) \, . \, E_2)$ is likewise regular, context-free, or context-sensitive.

The following sections extend this technique to an arbitrary number of free variables by introducing n-way regular relations.

5.10.1 N-way Regular Relations

Regular relations, also called rational relations in the algebra literature, extend the concept of regularity to Cartesian products of languages. The development here follows Kaplan & Kay [KK94].

First we give some preliminary definitions. An *n-way string relation* is a set of n-tuples of strings over some alphabet Σ. Equivalently, an n-way string relation is a subset of the n-way Cartesian product $\Sigma^* \times \cdots \times \Sigma^*$. We write a tuple of strings as $< x_1, \ldots, x_n >$. For the n-tuple of empty strings, we write $\epsilon_n =< \epsilon, \ldots, \epsilon >$. String concatenation is extended over n-tuples as follows: if $x_1, \ldots x_n$ and $y_1, \ldots, y_n$ are strings, then

$$< x_1, \ldots, x_n >< y_1, \ldots, y_n > \equiv < x_1 y_1, \ldots, x_n y_n >$$

The concatenation of two relations is the pairwise concatenation of their elements:

$$R_1 R_2 \equiv \{< x_1 y_1, \ldots, x_n y_n > \mid < x_1, \ldots, x_n > \in R_1 \text{ and } < y_1, \ldots, y_n > \in R_2\}$$

Exponentiation of an n-way relation is defined as expected:

$$\begin{aligned} R^0 &\equiv \{\epsilon_n\} \\ R^i &\equiv R^{i-1} R \end{aligned}$$

The Kleene star R^* is defined as $\cup_{i=0}^{\infty} R^i$.

Regular relations are defined recursively as follows:

- $\emptyset$ and $\{< x_1, \cdots, x_n >\}$ for any $< x_1, \cdots, x_n > \in \Sigma^* \times \cdots \times \Sigma^*$ are regular relations;
- if R and R' are regular relations, then RR', $R \cup R'$, and R^* are regular relations;
- there are no other regular relations.

Every regular n-way relation is recognized by an n-tape finite state transducer, a generalization of the finite state transducer defined in Section 5.2. Every transition in an n-tape finite state transducer is labeled with n symbols, which may be either symbols in Σ or the empty string ϵ. Formally, a finite state transducer is a tuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q is a set of states; Σ is the alphabet; δ is a mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \cdots \times (\Sigma \cup \{\epsilon\})$ to finite subsets of Q; q_0 is the starting state; and F is the set of accepting states.

It is easy to show that regular relations are closed under a number of operations:

- union $R \cup R'$;
- concatenation RR';
- Kleene star R^*;
- product $R \times R' = \{< x_1, \ldots, x_n, y_1, \ldots, y_m > \mid < x_1, \ldots, x_n > \in R \text{ and } < y_1, y_2, \ldots, y_m > \in R'\}$;
- join $R \textit{ join } R' = \{< x_1, \ldots, x_n, y_2, \ldots, y_m > \mid < x_1, \ldots, x_n > \in R \text{ and } < x_n, y_2, \ldots, y_m > \in R'\}$
- composition $R \circ R' = \{< x_1, \ldots, x_{n-1}, y_2, \ldots, y_m > \mid < x_1, \ldots, x_n > \in R \text{ and } < x_n, y_2, \ldots, y_m > \in R'\}$. Note that composition is like a join where the common component x_n is removed.

Regular relations are also closely related to regular languages. The projection $\pi_i(R) = \{x_i | < x_1, \ldots, x_n > \in R\}$. If L is a regular language, then the 1-relation $< L >= \{< x > | x \in L\}$ is a regular relation. The image of L under a regular relation R, written $R/_i L = \{< x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n > \mid < x_1, \ldots, x_n > \in R \text{ and } x_i \in L\}$, is also a regular relation. In particular, $R/_i \Sigma^*$ deletes the ith component of a n-way regular relation R to produce an $n-1$-way regular relation.

Regular relations are not, in general, closed under intersection and difference. For example, consider the two regular relations $R_1 =< \epsilon, b >^* < a, c >^*$ and $R_2 =< a, b >^* < \epsilon, c >^*$. The relation R_1 consists of tuples of the form $< a^n, b^* c^n >$, and relation R_2 is $< a^n, b^n c^* >$, so the intersection $R_1 \cap R_2$ is $< a^n, b^n c^n >$. This relation cannot be regular because its projection $b^n c^n$ is not regular.

Closure under intersection and difference is vital if we want to use regular relations for region algebra semantics. Fortunately, Kaplan and Kay show that a useful subclass of regular relations *is* closed under these operations. A *same-length* relation is a relation R such that for every $< x_1, \ldots, x_n >$ in R, $|x_1| = \cdots = |x_n|$. Same-length regular relations are closed under:

- intersection $R \cap R'$;
- difference $R - R'$;
- union $R \cup R'$;
- concatenation RR';
- Kleene star R^*;
- join $R \mathit{join} R'$;
- composition $R \circ R'$;
- image with a regular language $R/_i L$.

Same-length regular relations are not closed under product $R \times R'$, however, since a tuple in R may be paired with a tuple in R' of a different length.

5.10.2 Algebra Semantics With *forall*

We are now ready to give semantics for region expressions with variables. Assume that all variables in an expression are distinct, so that every *forall* operator binds a unique variable. It is trivial to rewrite the expression with fresh variables to make this the case. Let V be the set of variables that can be used in a region expression, and $v \in V^*$ be a string of variables. For a string of bound variables v and a region expression E whose free variables can all be found in v, let $R_v(E)$ denote the $|v|+1$-way relation defined as follows:

$$\begin{aligned} R_v(E) \equiv \{ \; & < x_1[y_1]z_1, \ldots, x_n[y_n]z_n, x[y]z > | \\ & [|x|, |xy|] \in E/xyz \text{ when each } v_i \text{ is bound to the region } [|x_i|, |x_i y_i|] \\ & \text{ and } x_i y_i z_i = xyz \text{ for all } i\} \end{aligned}$$

For any tuple in $R_v(E)$, the first $|v|$ components represent the regions bound to variables, and the $|v|+1$st component is the region matching the expression E.

Using this definition, we can write the region language $L(E)$ recognized by a region expression that contains variables. Assuming E has no free variables (but possibly bound variables), then

$$L(E) = \pi_1(R_\epsilon(E))$$

i.e., the projection of the 1-relation $R_v(E)$ where no variables are bound.

An important constraint in the definition of $R_v(E)$ is that for every tuple $< x_1[y_1]z_1, \ldots, x_n[y_n]z_n, x[y]z >$ in $R_v(E)$, it must be the case that $x_1 y_1 z_1 = \cdots = x_n y_n z_n = xyz$. In other words, the n bound variables and the output must all refer to regions in the same string, xyz. A tuple that satisfies this constraint will be called *well-formed*. For convenience in later definitions, we let $\Delta \equiv \{x[y]z | xyz \in \Sigma^*\}$ be the set of all well-formed region strings, and

$$\Delta_n \equiv \{< x_1[y_1]z_1, \ldots, x_n[y_n]z_n > | x_i y_i z_i = w \text{ for some } w \text{ and all } i\}$$

be the set of all well-formed region tuples. It is always the case that $R_v(E) \subset \Delta_{|v|+1}$.

Lemma 2. *Δ_n is a same-length regular relation.*

Proof. By definition Δ_n is a same-length relation, because every string in a tuple has length $|w|+2$. The proof that Δ_n is regular goes by induction on n. The 1-way relation $\Delta_1 =< \Delta >$ is regular because the language Δ is regular. The 2-way relation Δ_2 is regular because it is the union of the six 2-way regular relations *before*, *after*, *in*, *contains*, *overlaps-start*, and *overlaps-end* described by the finite state transducers shown in Figure 5.4. (Claim 1 implies that the union of the six binary region relations gives all possible relations between regions.) For $n > 2$, $\Delta_n = \Delta_{n-1}$ *join* Δ_2, so by the induction hypothesis and closure of same-length regular relations under join, Δ_n is regular. □

The next section will show that $R_v(E)$ is also a same-length regular relation for expressions E in $RSTA_{\mathcal{R}}$.

The semantics of all region algebra operators, including *forall*, can be written in terms of $R_v(E)$. We omit concatenation, since it was shown in Chapter 3 that concatenation can be defined in terms of *forall*.

Ground Terms

The ground terms include literal strings w, regular expressions r, and context-free grammar non-terminals A. A ground term ignores the bound variables and returns the region language $L(E)$. However, the entire relation $R_v(E)$ must still satisfy the constraint that all strings in a tuple are identical (modulo the positions of the square brackets). Since Δ_n expresses this constraint, we can write $R_v(E)$ for ground terms E as follows:

$$\begin{aligned} R_v(w) &= \Delta_{|v|+1} \textit{ join } < L(w) > \\ R_v(r) &= \Delta_{|v|+1} \textit{ join } < L(r) > \\ R_v(A) &= \Delta_{|v|+1} \textit{ join } < L(A) > \end{aligned}$$

Set Operators

The set operators are straightforward:

$$\begin{aligned} R_v(E_1 \cap E_2) &= R_v(E_1) \cap R_v(E_2) \\ R_v(E_1 \cup E_2) &= R_v(E_1) \cup R_v(E_2) \\ R_v(E_1 - E_2) &= R_v(E_1) - R_v(E_2) \\ R_v(\Omega) &= \Delta_{|v|+1} \\ R_v(\emptyset) &= \emptyset \end{aligned}$$

Relational Operators (R)

The relational operators are represented as 2-way finite state transducers in Figure 5.4, which are equivalent to 2-way string relations. Let $R(op)$ be the 2-way relation for *op*. Then we can write each $R_v(\textit{ op } E)$ as a composition as follows:

$$\begin{aligned}
R_v(\textit{ before } E) &= R_v(E) \circ R(\textit{ before }) \\
R_v(\textit{ after } E) &= R_v(E) \circ R(\textit{ after }) \\
R_v(\textit{ in } E) &= R_v(E) \circ R(\textit{ in }) \\
R_v(\textit{ contains } E) &= R_v(E) \circ R(\textit{ contains }) \\
R_v(\textit{ overlaps-start } E) &= R_v(E) \circ R(\textit{ overlaps-start }) \\
R_v(\textit{ overlaps-end } E) &= R_v(E) \circ R(\textit{ overlaps-end })
\end{aligned}$$

Forall Operator

Finally we consider the *forall* operator. The expression $\textit{forall}\,(\alpha : E_1)\,.\,E_2$ evaluates E_1 and binds a new variable α which is added to the variable subscript v when evaluating E_2. In detail, R_v for this expression is

$$\begin{aligned}
R_v(\textit{ forall }(\alpha : E_1)\,.\,E_2) = \{ \quad & < x_1[y_1]z_1, \ldots, x_n[y_n]z_n, x[y]z > \mid \\
& < x_1[y_1]z_1, \ldots, x_n[y_n]z_n, x_\alpha[y_\alpha]z_\alpha > \in R_v(E_1) \\
& \text{and } < x_1[y_1]z_1, \ldots, x_n[y_n]z_n, x_\alpha[y_\alpha]z_\alpha, x[y]z > \in R_{v\alpha}(E_2)\}
\end{aligned}$$

We will want to show that this is a same-length regular relation, so let us rewrite it as a combination of $R_v(E_1)$ and $R_{v\alpha}(E_2)$ using operators under which same-length regular relations are closed. Starting with the well-formed $n+1$-way relation $R_v(E_1)$, we extend it to a same-length $n+2$-way relation by joining it with Δ_2:

$$\begin{aligned}
R_v(E_1)\,\textit{join}\,\Delta_2 = \{ \quad & < x_1[y_1]z_1, \ldots, x_n[y_n]z_n, x_\alpha[y_\alpha]z_\alpha, x[y]z > \\
& < x_1[y_1]z_1, \ldots, x_n[y_n]z_n, x_\alpha[y_\alpha]z_\alpha > \in R_v(E_1) \\
& \text{and } x_i y_i z_i = xyz\}
\end{aligned}$$

This expression represents the bound variables $v\alpha$ allowed by E_1. To find the result of E_2 given those bindings, we intersect with $R_{v\alpha}(E_2)$ to get

$$\begin{aligned}
(R_v(E_1)\,\textit{join}\,\Delta_2) \cap R_{v\alpha}(E_2) = \{ \quad & < x_1[y_1]z_1, \ldots, x_n[y_n]z_n, x_\alpha[y_\alpha]z_\alpha, x[y]z > \mid \\
& < x_1[y_1]z_1, \ldots, x_n[y_n]z_n, x_\alpha[y_\alpha]z_\alpha > \in R_v(E_1) \\
& \text{and } < x_1[y_1]z_1, \ldots, x_n[y_n]z_n, x_\alpha[y_\alpha]z_\alpha, x[y]z > \in R_{v\alpha}(E_2)\}
\end{aligned}$$

Finally, we apply $/_{|v|+1}\Delta$ to delete the α component of the relation so that the result is a $n+1$-way relation omitting the bound variable α, as required. Thus

$$R_v(\textit{ forall }(\alpha : E_1)\,.\,E_2) = ((R_v(E_1)\,\textit{join}\,\Delta_2) \cap R_{v\alpha}(E_2))/_{|v|+1}\Delta$$

5.10.3 $RSTA_{\mathcal{R}} = R$

Now we are ready to show that $R_v(E)$ is a same-length regular relation, which will lead immediately to the result that $RSTA_{\mathcal{R}}$ recognizes exactly the class of regular languages.

Theorem 15. *$R_v(E)$ is a same-length regular relation for all expressions E in $RSTA_{\mathcal{R}}$.*

Proof. By structural induction on E.

- Ground terms. For a regular expression r, $R_v(r) = \Delta_{|v|+1}$ *join* $< L(r) >$ is a same-length regular relation because Δ_n is same-length regular, $< L(r) >$ is regular and trivially same-length, and same-length regular relations are closed under join. Likewise, $R_v(\Omega) = \Delta_{|v|+1}$ and $R_v(\emptyset) = \emptyset$ are same-length regular.
- Set operators. Same-length regular relations are closed under intersection, union, and difference.
- Relational operators. For each operator *op*, $R($ *op* $)$ is a same-length regular relation, and same-length regular relations are closed under composition, so $R_v($ *op* $E) = R_v(E) \circ R($ *op* $)$ is same-length regular.
- Forall operator. Since $R_v($ *forall* $(\alpha : E_1)\,.\,E_2) = ((R_v(E_1)$ *join* $\Delta_2) \cap R_{v\alpha}(E_2))/_{|v|+1}\Delta$, and same-length regular relations are closed under join, intersection, and deletion of a component, $R_v($ *forall* $(\alpha : E_1)\,.\,E_2)$ is same-length regular.

□

Corollary 5. *If L is the region language for some expression E in $RSTA_{\mathcal{R}}$, then L is regular.*

Proof. $L = L(E)$ is a projection of the regular relation $R_\epsilon(E)$, so L is regular. □

5.10.4 Open Questions: $RSTA_\emptyset$, $RSTA_{\mathcal{F}}$, $RSTA_{\mathcal{CFL}}$

We have seen that adding the *forall* operator to the region algebra over regular expressions does not affect its power. Like $RST_{\mathcal{R}}$, $RSTA_{\mathcal{R}}$ recognizes the class of regular languages, no more, no less. What about the other algebra specializations: no ground terms, finite ground terms, or context-free ground terms? Does *forall* affect their power?

Adding *forall* to $RST_\emptyset$ clearly expands the set of region languages that can be recognized — not a hard job considering that $RST_\emptyset$ can only recognize $L(\Omega)$ and $\emptyset$. For example, *forall* enables the definition of operators like *zero-length*, which matches only the zero-length regions in a string, and *nth* $_n A$, which matches the nth region matching A. Combining these operators produces the expression

$$\textit{ends nth}_{n+1} zero - length$$

which matches only regions containing exactly $n + 1$ zero-length regions, i.e., regions of length n. Since $RSTA_\emptyset$ offers no way to test the characters of the string, however, it seems plausible that $RSTA_\emptyset$ recognizes a class of region languages that depends only on the location of the square brackets and not on the symbols in the string. As a conjecture, $RSTA_\emptyset$ might recognize the set of region languages L such that for any substitution function $h : \Sigma \to \Sigma$ that preserves [and], $h(L) = L$. Characterizing this language class precisely and proving that $RSTA_\emptyset$ recognizes it is left as future work.

Whether $RSTA_{\mathcal{F}}$ recognizes only noncounting languages, like $RST_{\mathcal{F}}$ does, is also an open question. The reader may have noticed that the proof techniques for $RST_{\mathcal{F}}$ were substantially

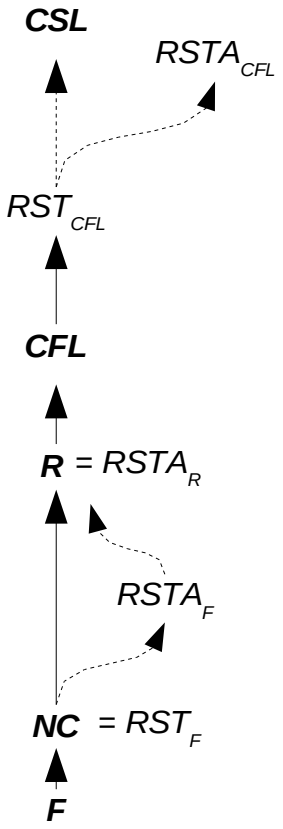

Figure 5.6: Summary of relationships among language classes.

different from $RST_{\mathcal{R}}$ and $RST_{\mathcal{CFL}}$, primarily because noncounting languages are not closed under mapping through an arbitrary finite state transducer. The regular languages are the smallest language class that is closed under FST mapping [HU79, Lemma 11.1]. Since $NC \subset R$, we could not use finite state transducers to show that $RST_{\mathcal{F}}$ recognizes NC. As a result, the regular relation machinery developed for showing that $RSTA_{\mathcal{R}}$ recognizes R will not help to decide whether $RSTA_{\mathcal{F}}$ recognizes NC.

Finally, we consider $RSTA_{\mathcal{CFL}}$. We know that $\mathcal{CFL}$ is a strict subset of $RSTA_{\mathcal{CFL}}$, but is $RSTA_{\mathcal{CFL}}$ a subset of $\mathcal{CSL}$? I believe it is, but resolving the question would require developing some additional theoretical machinery which is left as future work.

5.11 Summary

The important relationships between the region algebra variants discussed in this chapter and the the language classes of the Chomsky language hierarchy are summarized in Figure 5.6. In the figure, solid arrows denote proper inclusion between language classes, e.g., $\mathcal{R} \subset \mathcal{CFL}$. Dashed arrows denote inclusion which is not known to be proper. For example, it is trivial that $RST_{\mathcal{F}} \subseteq RSTA_{\mathcal{F}}$, but it is not known whether $RST_{\mathcal{F}} = RSTA_{\mathcal{F}}$. The dashed arrows represent open problems, as do missing lines, such as the relationship between $\mathcal{CSL}$ and $RSTA_{\mathcal{CFL}}$.