Discussion of Selected Exercises: Section 4.2.8 of Compilers: Principles, Techniques and Tools

Topic: Context-Free Grammars

2013-05-15

January 3, 2019

Abstract

This paper contains the author's answers and proofs for selected exercises from Section 4.2.8 of the "dragon book"—Compiler: Principles, Techniques and Tools (hereinafter "CPTT"). This book, while an excellent resource, can be challenging for self-study as it does not provide a means of verifying one's answers outside of a classroom setting (unless the reader has confidence in his/her proofs). This paper is intended for two audiences: (a) those reading the book and looking for clarification and discussion on the exercises and (b) those who are curious on the topic of context-free grammars that do not possess the text. The selected exercises are those that the author felt would be most useful for discussion and, as such, are expected to be challenging to the reader. Less challenging portions of exercises may be discussed to segue into the more challenging portions.

1 Context-Free Grammars

The focus of this discussion (and of Section 4.2 in CPTT) is on context-free grammars (or simply "grammars").

2 Convention and Notation

The following notational conventions are used throughout this paper. In most cases, they have been borrowed from the text.

For grammars, capital symbols are used to represent non-terminals. The S symbol is used to denote the starting non-terminal. The symbol $\rightarrow$ is used to separate the non-terminal from its production body, whereas $\Rightarrow$ indicates a single step in a derivation. Leftmost and rightmost derivations are denoted

$\underset{lm}{\Rightarrow}$ and $\underset{rm}{\Rightarrow}$ respectively. $\overset{*}{\Rightarrow}$ means "derives in zero or more steps", whereas $\overset{+}{\Rightarrow}$ means "derives in one or more steps". The symbol | separates multiple productions for a single non-terminal. Any time punctuation is placed at the end of a grammar or derivation, it should be read as part of the surrounding paragraph, *not* as part of the production or derivation. For example, in the grammar

$$S \to 0\ S\ 1 \mid \epsilon,$$

the trailing comma is not part of the construction. Furthermore, whitespace is not significant and may be discarded. ϵ is the empty string.

"The text" refers to CPTT, whereas "this paper" refers to the paper you are currently reading.

3 Exercise 4.2.3—Grammar Design

This exercises requests that the reader design grammars for a series of language descriptions a–f; we will discuss each of them. Although the text does not request it, proofs will be provided for each, as they are useful to demonstrate correctness and an excellent practice in discipline.

Exercise 4.2.3a *The set of all strings of 0's and 1's such that every 0 is immediately followed by at least one 1.*

The grammar for this exercise is fairly trivial, but will serve as a useful introduction to the formalities of this paper. First, let us consider a grammar that demonstrates such a property. Our alphabet is $\Sigma = \{0, 1\}$. The only restriction on the sentences of our grammar is that each 0 must be followed by a 1—this therefore means that we can have any number of adjacent 1's, but it is not possible to have adjacent 0's. Considering that our alphabet Σ has only two characters, this grammar is fairly simple:

$$S \to 1S \mid 01S \mid \epsilon. \tag{1}$$

As an example, let us consider some of the sentences that we may wish to be derived by this grammar. In particular, consider derivation of the string 01011:

$$S \Rightarrow 01\ S \Rightarrow 01\ 01\ S \Rightarrow 01\ 01\ 1S \Rightarrow 01\ 01\ 1\ \epsilon \overset{*}{\Rightarrow} 01\ 01\ 1. \tag{2}$$

Notice also that a string of 1's—such as 1111—is also derivable given our grammar:

$$S \Rightarrow 1\ S \Rightarrow 1\ 1\ S \Rightarrow 1\ 1\ 1\ S \Rightarrow 1\ 1\ 1\ 1\ S \Rightarrow 1\ 1\ 1\ 1\ \epsilon \overset{*}{\Rightarrow} 1\ 1\ 1\ 1, \tag{3}$$

as is the empty string ϵ in one step:

$$S \Rightarrow \epsilon. \tag{4}$$

To prove that grammar (1) is correct, we must prove two independent statements:

1. The *only* strings derivable from grammar (1) are those of 0's and 1's such that every 0 is immediately followed by at least one 1;

2. The grammar accepts all such strings.

We will prove these statements in order. For the first statement, we must show that, at any given step n of grammar (1), the only derivable strings contain a 1 after each and every 0 (or that the string contains no 0's). For the second statement, we must show that any string containing 0's and 1's such that every 0 is followed by at least one 1 is derivable from our grammar. Grammar proofs are discussed in Section 4.2.6 of the text.

PROOF The only strings derivable from S are those of 0's and 1's such that every 0 is immediately followed by at least one 1. We shall perform this proof inductively on the number of steps n in a given derivation.

BASIS The basis is $n = 1$. In one step, our grammar may produce one of three strings: A string beginning with a 1 (the first production of S), a string beginning with a 0 followed by a 1 (the second production of S) and the empty string ϵ (the final production of S).

The empty string ϵ has no 0's and so follows the rules of the language. The same is true for any string beginning with a 1. The third and final string that can be generated when $n = 1$ is 01. This string does contain a 0 and therefore also satisfies our requirement.

INDUCTION We shall now assume that all derivations of fewer than n steps result in either a sentence containing no 0's or a sentence that contains 0's followed by one or more 1's. Such a derivation must have the form

$$S \Rightarrow xS \stackrel{*}{\Rightarrow} xy. \tag{5}$$

Since x is derived in fewer than n steps then, by our inductive hypothesis, x must contain 0's only if followed a 1; the same is true of y.

Additionally, according to grammar (1), y must be of one of the productions

$$S \to 1S$$
$$S \to 01S$$
$$S \to \epsilon.$$

Each of these productions have already been discussed in our basis; therefore, y cannot contain a 0 followed by another 0. Additionally, it is required that adjacent 1's be permitted after a 0, which is possible by the first production (as demonstrated in (3)). As such, xy must contain only 0's followed by one or more 1's and our hypothesis has been proved. □

To ensure a thorough understanding of the above proof, it is worth mentioning why (5) used both the $\Rightarrow$ and $\stackrel{*}{\Rightarrow}$ derivation symbols. Our basis applies when $n = 1$; the inductive hypothesis applies otherwise (when $n > 1$). As such, we must have *at least* one production in (5).

Now that we have proved that we may only derive sentences from grammar (1) that contain 0's followed by one or more 1's, we must now show that the grammar may be used to derive all such possible strings.

PROOF Any string s of length l consisting of 1's and 0's such that any 0 is followed by at least one 1 is derivable from S.

BASIS A string of length 0 ($l = 0$) must be ϵ, which is derivable from S in one step.

INDUCTION Assume that any string s of a length less than l is derivable from S. Such a string must have the form $xy, y \in \{1, 01, \epsilon\}$—that is, we can consider s to be the concatenation of y with a previously derived string. Since the length of x is clearly less than l, it must by derivable from S by our inductive hypothesis. Furthermore, xy must have a derivation of the form

$$S \stackrel{+}{\Rightarrow} x\ S \Rightarrow x\ y, \tag{6}$$

thereby proving that s is derivable from S. □

The derivation (6) may seem to be too abstract to be useful; since this is our first proof, it is worth clarifying why it does in fact complete the proof. We first showed that any string of the language of 0's and 1's that we have been studying can be described as the concatenation of a smaller such string with 0, 01 or ϵ (which completes the string). This string, as we stated, has the form xy. Therefore, we must show that S supports concatenation—(6) demonstrates this with x fairly abstractly, since it does not matter what exactly x is. From the productions of S in grammar (1), it is understood that x can be any string of terminals (that is—any derivation) leading up to that point in the derivation (6).

We must now show that the remaining part of xy—that is, y—is derivable. The only non-terminal remaining after x is S. We have defined y to be any string of terminals in the set $\{0, 01, \epsilon\}$. Clearly, each of these strings are derivable from S. Therefore, we can replace S in (6) with y, indicating that this is a valid derivation given our definition of y; it is up to the reader of the proof to make this connection. Note that, while the domain of y happens to be every production of S, this is not necessary for the proof—that is the subject of the first proof.

Before we put this exercise to rest (indeed, we completed the exercise requirement in the first paragraph following the exercise definition), it is also worth noting that this grammar may also be accepted by a finite automata (and consequently, a regular expression); this is demonstrated by Figure 1. It should be noted that this is not the case with all of the exercises that follow. ■

The above example was fairly simple, yet resulted in a realitively lengthy discourse far past what was required by the text; the reader can expect such a discussion to continue for all examples that follow.

Exercise 4.2.3b *The set of all strings of 0's and 1's that are palindromes; that is, the string reads the same backward as forward.*

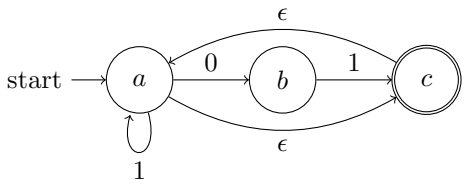

Figure 1: An NFA corresponding to the extended regular expression $\left(0^?1^+\right)^*$ describing grammar (1).

As the exercise stated, a *palindrome* is a string that reads the same in both directions; let us consider some examples before attempting to construct a grammar. The following list of strings are all palindromes, one per line:[1]

$$\begin{array}{ccc} 1 & 00 & 1 \\ 1100 & 11 & 0011 \\ 010 & 1 & 010 \\ & 0 & \end{array} \tag{7}$$

The above palindromes have been laid out so that their symmetry is apparent. At first glance, one can imagine constructing a palindrome out of pairs of characters, like the second row of (7):

$$\begin{array}{ccc} & 11 & \\ 1 & 11 & 1 \\ 11 & 00 & 11 \\ 110 & 00 & 011 \\ 1100 & 11 & 0011 \end{array} \tag{8}$$

In this case, each palindrome would always have an even number of characters. However, it is important to note the bottom two palindromes of (7), which have an *odd* number of characters:

$$\begin{array}{ccc} & 00 & \\ 0 & 11 & 0 \\ 01 & 00 & 10 \\ 010 & 1 & 010 \end{array} \tag{9}$$

Given this evaluation and the understanding that $2n$ is always even for some positive integer n, it would be accurate to recursively construct a palindrome from the edges inward in pairs. Once we reach the center, we may end with ϵ if we wish to have an even ($2n$) number of characters, or otherwise may add a single character to create a palindrome containing an odd ($2n + 1$) number of characters.

[1] An example of an English palindrome is "Mr. Owl ate my metal worm" (discarding punctuation and capitalization.)

$$\begin{aligned} S &\to 0\ S\ 0 \mid 1\ S\ 1 \mid M \\ M &\to 0 \mid 1 \mid \epsilon \end{aligned} \tag{10}$$

In grammar (10) above, we define out start non-terminal S with productions for the outer pairs. The non-terminal M represents the acceptable inner ("middle") characters, which determines if the length of the palindrome is even (if ϵ is used) or odd (0 or 1). We will leave demonstrations of such derivations to the proof.

To prove that grammar S is the proper grammar for all palindromes, we must again prove two things: That language $L(S)$ can produce only palindromes of 0's and 1's and that all such palindromes can be derived from S. The difference between these two descriptions may be subtle for such a simple grammar, but the distinction is important to ensure that $L(S)$ represents *nothing more and nothing less* than a language that may be used for such palindromes.

As before, the proofs will be inductive—the first proof on the number of steps n of a derivation of S and the second on the length l of the palindrome s. Our alphabet Σ is once again $\{0, 1\}$.

PROOF The only strings derivable from grammar S are palindromes consisting of 0's and 1's.

BASIS The basis is $n = 2$, which is the fewest number of steps from which a string may be derived from S.[2] Such a derivation must be of the form

$$S \Rightarrow M \Rightarrow x,$$

where x is 0, 1, or ϵ. In the latter case, the derived string is clearly a palindrome of length zero. In the case of 0 or 1, the length of the string is one, which must be a palindrome.

INDUCTION Now assume that every string derived in less than n steps is a palindrome. Such a derivation must be of the form

$$S \Rightarrow x\ S\ x \overset{*}{\Rightarrow} x\ y\ x.$$

That is, the string x appears on both the left and right of y. Since the derivation of y from S takes fewer than n steps—specifically, $n - 1$ steps—y must be a palindrome by our inductive hypothesis. Because x is added to both the beginning and end of y, then any string derived in n steps must be a palindrome. □

Let us further demonstrate the above proof by deriving (8) from S:[3]

$$S \Rightarrow 1\ S\ 1 \Rightarrow 1\ 1\ S\ 1\ 1 \Rightarrow \cdots \overset{+}{\Rightarrow} 1\ 1\ 0\ 0\ 1\ \epsilon\ 1\ 0\ 0\ 1\ 1 \tag{11}$$

[2] $n = 1$ steps cannot result in a string consisting only of nonterminals, as it would result in $0S0$, $1S1$ or M.

[3] The dots were added so as not to confuse the reader as to what was going on; the symbol $\overset{+}{\Rightarrow}$ is sufficient and therefore the dots will be omitted in the future.

and additionally (9):

$$S \Rightarrow 0\ S\ 0 \Rightarrow 0\ 1\ S\ 1\ 0 \Rightarrow 0\ 1\ 0\ S\ 0\ 1\ 0 \Rightarrow 0\ 1\ 0\ 1\ 0\ 1\ 0. \tag{12}$$

The induction step works by recognizing the basis as the middle of the string (nonterminal M in grammar (10))—ϵ for palindromes of an even length and the $\lceil n/2 \rceil^{th}$ character for those of an odd length (1 in the case of the latter derivation). Call this string b. We know that b is a palindrome, as explained in the proof above. For our inductive step, we recognize that, for each step n, we add two characters—one to the beginning and one to the end—to the result of step $n-1$. As such, since the derivation of $n-1$ steps must be a palindrome, the derivation in n steps must also be—it is not possible to derive anything but a palindrome from M and S maintains this designation.

For completeness, we must now show that all possible palindromes of the alphabet Σ can be derived from S.

PROOF Every palindrome consisting of 0's and 1's is derivable from S.

BASIS If the string s is of length $l \leq 1$, then it must be ϵ, 0 or 1, all of which are palindromes derivable by M.

INDUCTION Observe that any palindrome of length $l > 1$ must contain the same character at positions 1 and l.[4] Assume that each string with a length less than l is derivable from S. Since s is a palindrome, then it must have the form $xyx, x \in \Sigma$, where y is also a palindrome. Since y has a length $l - 2 < l$, then it must be derivable from S by the inductive hypothesis. The palindrome s must therefore have a derivation of the form

$$S \Rightarrow x\ S\ x \overset{*}{\Rightarrow} x\ y\ x,$$

which thus proves that s is derivable from S. □

It is also worth noting that, unlike the first exercise, we cannot represent a palindrome as a finite automaton (and therefore cannot represent it as a regular expression). Let us prove this assertion.

PROOF S cannot be represented by any finite automata. Specifically, a finite automaton representing S may accept all strings that are palindromes of the alphabet Σ, but such an automaton must also accept strings that are not palindromes. We shall prove this statement by contradiction.

CONTRADICTION Given the alphabet Σ, a palindrome may contain any character from Σ at any arbitrary position n and may be of length $l \geq 0$. As such, we must be able to represent this automaton by the regular expression $(0|1)^*$, whose corresponding minimum-state DFA is shown in Figure 2. However, it is also necessary that characters c_n and c_{l-n+1} be the same symbol in Σ—a requirement that minimum-state DFA of Figure 2 cannot guarantee.

Consider that the only way for a finite automata to maintain a history of states is to have a state to represent each unique history. However, to accept a

[4] 1-indexed for notational convenience.

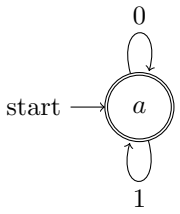

Figure 2: The minimum-state DFA for the regular expression $(0|1)^*$.

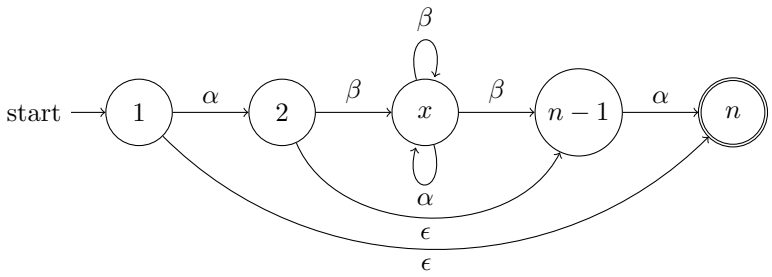

Figure 3: An NFA with a finite set of states must at some point transition to a previous or identical state in order to accept input of any length. $\Sigma = \{\alpha, \beta\}$.

string of any length, we would need an automaton containing a potentially infinite number of states, which is not finite (and therefore not a finite automaton). Therefore, it is not possible to represent the history of every possible palindrome using a finite set of states.

Given this, it must stand that a finite automaton must at some point contain a state that transitions to a previous or current state, such as the NFA in Figure 3. Since the history of the string is "stored" purely in the possible states leading up to the current state, this transition t equates to a loss of "memory", without which the right-hand portion of the palindrome cannot be properly matched. Furthermore, since each position n may contain any character in Σ, and since the transition t can only yield a set of future states with a limited (finite) precision, each of these future states must be redundant. Since each NFA can be represented by an equivalent DFA and each DFA for some grammar has a single common minimum-state DFA, any portion of a finite automaton that can accept a palindrome of any length must be equivalent to Figure 2 (such as state x in Figure 3). We are therefore left to conclude that no finite automata can accept a palindrome of arbitrary length without accepting every string that is a combination of each character in Σ. $\square$

To provide further clarification—any finite automata that transitions to a *previous* state, since it looses a portion of its history, can no longer accurately determine the states leading up to the final state. That is, consider the string

10101 and consider that the first three characters of this string can be represented by the states $\{a, b, a\}$. At this point, we can no longer be certain of what the string may end with, because we have lost any sense of nesting/recursion. Therefore, the states leading to the final state are forced to accept any character in Σ and therefore must be equivalent to the minimum-state DFA of Figure 2. As was mentioned by the text, "finite automata cannot count".

Figure 3 gets around such an issue by transitioning only to current or future states, which permits a *finite* amount of nesting (placing the aforementioned minimum-state DFA x in the middle). However, note a glaring issue—this automaton does not accept β in the first character position. If it did, then we would need a second set of states in order to maintain such a history and know that we should also *end* with β instead of α. The number of states would therefore grow very quickly with the level of nesting and the size of Σ (such a consideration is left to the reader).

We have exhaustively proved that grammar (10) is the correct answer for this exercise. ■

Exercise 4.2.3c *The set of all strings of 0's and 1's with an equal number of 0's and 1's.*

To understand how to approach this problem, we shall consider a number of strings that are derivable from this language. An obvious case is ϵ, which contains zero 0's and zero 1's. Some additional examples are shown in Figure 4 along with their lengths (denoted by l).

s	ϵ	10	01	1010	1001	011100
l	0	2	2	4	4	6

Figure 4: Examples of strings with an equal number of 0's and 1's.

These examples demonstrate a number of important properties. In particular, the length l of the string s is always even, with the number of 0's and 1's $n = l/2$. Additionally, the characters of the alphabet Σ may appear in any order in the string. Therefore, we do not have the luxury of a simple, nested, recursive implementation as we did with the palindrome exercise (at least not exclusively).

Let us construct the grammar iteratively, beginning with the simplest case of ϵ.

$$S \to \epsilon \tag{13}$$

The second case—10—is also fairly easy to fit into S:

$$S \to 10 \mid \epsilon \tag{14}$$

The third case demonstrates an important case regarding our strings: They may begin with either a 0 or a 1 and they may also *end* with either character (more generally, they may begin or end with any character in Σ). However, we

cannot simply adjust our grammar to accept either character in both positions—S must assure that, any time we include a 0 in a production, we also include a 1 (and vice versa). So far, this is guaranteed by S in grammar (14); to keep on this path, we must add 01 as yet another special case.

$$S \to 01 \mid 10 \mid \epsilon \tag{15}$$

The fourth case—1010—introduces the need to handle strings of an arbitrary length. To do this, we must determine at what point we should recurse on S. Looking at the example, we could derive 1010 as two nested applications of S if we recurse between the two terminals.

$$S \Rightarrow 1\ S\ 0 \Rightarrow 1\ 0\ S\ 1\ 0 \Rightarrow 1\ 0\ \epsilon\ 1\ 0 \overset{*}{\Rightarrow} 1\ 01\ 0 \tag{16}$$

Of course, one could also adopt an alternate perspective by considering the string to be the production of two adjacent non-terminals.

$$S \Rightarrow S\ S \underset{lm}{\Rightarrow} 10\ S \underset{lm}{\Rightarrow} 10\ 10 \tag{17}$$

Unfortunately, with this information alone, we cannot be certain which of these productions—if such a choice even matters—should be used in our grammar. Perhaps we can gain further insight from the remaining examples.

The next example—1001—can be derived in a manner similar to (17), but not (16); in particular, grammar (15) has no production for the string 00, and so we cannot construct the string from the outside in. Given that, we can be certain that an adjacent non-terminal production is needed and so we will add the production used in (17) to our grammar.

$$S \to 01 \mid 10 \mid S\ S \mid \epsilon \tag{18}$$

However, the aforementioned predicament—the absense of a production that can yield only 00—raises the question of whether or not we can truly derive any string of equal 1's and 0's from the above grammar. Our final example challenges this. 011100 cannot possibly be represented by S in grammar (18) because this grammar constructs the string from left-to-right (or right-to-left) in pairs of 0's and 1's. Therefore, the only way to have adjacent 1's or adjacent 0's is to alternate the productions, which makes it impossible to have more than two adjacent identical characters.

Given this, it seems that both (17) *and* (16) are necessary; the following derivation demonstrates this fact (neither can individually be used to derive the string 011100).

$$S \Rightarrow S\ S \underset{lm}{\Rightarrow} 01\ S \underset{lm}{\Rightarrow} 01\ 1\ S\ 0 \underset{lm}{\Rightarrow} 01\ 1\ 1\ S\ 0\ 0 \underset{lm}{\Rightarrow} 01\ 1\ 1\ \epsilon\ 0\ 0 \overset{*}{\underset{lm}{\Rightarrow}} 01\ 1\ 10\ 0 \tag{19}$$

We thus arrive at grammar (20) below.

$$S \to 0\ S\ 1 \mid 1\ S\ 0 \mid S\ S \mid \epsilon \tag{20}$$

An astute reader may at this point notice that we have created an ambiguity in our grammar: Recall (16) and (17), which had two possible derivations for the same string; both of these derivations are now possible in our grammar. The text defines an ambiguous grammar to be a grammar that contains more than one leftmost or more than one rightmost derivation for the same sentence. This is a particularly interesting example of ambiguity, in particular because we cannot resolve it. Let us consider why.

PROOF Grammar S cannot be disambiguated. We will prove this fact by contradiction.

CONTRADICTION Firstly, recognize that S is ambiguous because there exists some sentence s that has both of the following derivations in $n > 1$ steps, where $a \neq b$:

$$S \Rightarrow a\ S\ b \overset{+}{\Rightarrow} a\ x\ b;$$
$$S \Rightarrow S\ S \Rightarrow a\ S\ b\ S \overset{+}{\Rightarrow} a\ b\ S \overset{+}{\Rightarrow} a\ b\ x.$$

Suppose to the contrary that there is some way to disambiguate x. There must then be some terminal $c \in \Sigma$ in x that may be used to perform the disambiguation and such a disambiguation would imply a difference in the semantics of x between the two derivations. However, $x = x$ and so both derivations hold exactly the same meaning—balanced strings. Furthermore, the productions for producing balanced strings requires each character in Σ; c therefore must not exist. □

Fortunately, this ambiguity is not an issue for our grammar because the multiple derivations are semantically equivalent—we are not arriving at any different result within the context of this exercise. The sentence 1010 of Figure 4 demonstrates this concept: It does not matter whether we consider the sentence to be a single balanced string or the concatenation of two balanced strings; we arrive at the same result regardless with no harm done.[5]

While the discussion thus far is likely to convince the reader that grammar (20) is correct, we shall conclude with a formal proof of this fact. A proof that the grammar cannot be represented by any finite automata shall be omitted, in particular because the productions of S have a structure very similar to the palindrome grammar (10).

PROOF Only sentences composed of balanced 1's and!0's may be derived from S.

BASIS The basis is $n = 1$. The only sentence that may be derived in 1 step is ϵ, which is clearly balanced (containing zero 0's and zero 1's).

INDUCTION Assume that any sentence derived in fewer than n steps is balanced. Now recognize that any sentence derived in $n > 1$ steps must make use of one of the following productions of S:

[5] Of course, one valid argument is that a more concise and unambiguous grammar will reduce problems during parsing. However, the parser (like Lex, as described by the text) can give precedence to the productions that appear earlier in the grammar to resolve this issue.

$$S \to 0\ S\ 1;$$
$$S \to 1\ S\ 0;$$
$$S \to S\ S.$$

Therefore, the smallest sentence that is not ϵ is either $0x1$ or $1x0$, both of which are balanced (each contains one 0 and one 1). Since x is derivable from S in fewer than n steps, then by our inductive hypothesis, all sentences derivable from S must be balanced. The last remaining production has the form xy, both of which are derivable from S in fewer than n steps and thus must be balanced. Furthermore, since the productions of S produce only 0, 1, or ϵ, S has the alphabet $\Sigma = \{0, 1\}$ and, consequently, may derive no sentence except for those containing balanced 0's and 1's. □

Having proved that only sentences of balanced 0's and 1's are derivable from S, we must now prove that S can derive *all* such strings (that is, all such strings are sentences of S). Such a proof is interesting because our grammar is more sophisticated than the previous examples.

PROOF All strings of balanced 0's and 1's are sentences of S.

BASIS The basis is a string of length $l = 0$, which contains zero 0's and zero 1's. This string must be ϵ, which is derivable from S.

INDUCTION First, recognize that all balanced strings must have a length $l = 2k$—that is, l is always even (as emphasized in Figure 4) and contains k 0's and k 1's. Assume that all strings less than length $2k$ are derivable from S.

Consider any balanced string s of length $2k$. We can consider s to have the form yz—that is, the concatenation of two balanced strings y and z, both of which in turn have the form $axb, a \neq b$ where x itself must be balanced (since $a \neq b$); alternatively, either y or z may be ϵ, which therefore implies that the form yz accepts any balanced string where the first and last characters are not the same.

We must now show that all such strings can be represented by the form yz. First, recognize that $y = axb$ may have either the form $0x1$ or $1x0$; the form yz then permits up to two adjacent identical characters in Σ; any additional adjacent identical characters may be derived by x. Consider $x = \epsilon$; then, clearly axb is balanced and can be concatenated to form a larger balanced string. If $x \neq \epsilon$ but $x_1 = b$,[6] then we can instead consider an alternative interpretation $y' = ax_1$ and $x' = x_2 \cdots x_n b$, and then let $y = y'x'$ (instead of axb).

We are then left with the case where $x_1 = a$. Such a case allows for an arbitrarily deep nesting of adjacent identical characters and therefore axb can be represented by the regular expression a^+b^+. It is therefore clear that the form yz is able to describe any string of balanced characters in the alphabet $\Sigma = \{0, 1\}$. Such a form must have the derivation

[6] x_n denotes the n^{th} character of x.

$$S \Rightarrow S\ S \underset{lm}{\overset{*}{\Rightarrow}} xy.$$

Since this is a leftmost derivation, y is either a balanced string or ϵ. In the former case, it is obvious that both x and y are of a length less than $2k$ and are therefore derivable from S by our inductive hypothesis. Otherwise, $y = \epsilon$ and the length of x is precisely $2k$ and we must consider the form axb; x is clearly of a length of less than $2k$ and is therefore balanced by our inductive hypothesis. Furthermore, it must have a derivation of the form

$$S \Rightarrow a\ S\ b \overset{*}{\Rightarrow} a\ x\ b,$$

thereby proving that axb is derivable from S. □

This proof was considerably more involved than our previous ones and is an excellent segue into proving more sophisticated grammars. Of course, the reader can surely see the challenges that might arise from attempting to prove much more complicated grammars. ■

4 License