Lecture 11

Pushdown automata

This is the last lecture of the course devoted to context-free languages. We will, however, refer to context-free languages from time to time throughout the remainder of the course, just as for regular languages.

The first part of the lecture focuses on the pushdown automata model of computation, which provides an alternative characterization of context-free languages to the definition based on CFGs. The second part of the lecture is devoted to some further properties of context-free languages that we have not discussed thus far, and that happen to be useful for understanding pushdown automata.

11.1 Pushdown automata

The *pushdown automaton* (or PDA) model of computation is essentially what you get if you equip an NFA with a stack. As we shall see, the class of languages recognized by PDAs is precisely the class of context-free languages, which provides a useful tool for reasoning about this class of languages.

A few simple examples

Let us begin with an example of a PDA, expressed in the form of a state diagram in Figure 11.1. The state diagram naturally looks a bit different from the state diagram of an NFA or DFA, because it includes instructions for operating with the stack, but the basic idea is the same. A transition labeled by an input symbol or ε means that we read a symbol or take an ε-transition, just like an NFA; a transition labeled $(\downarrow, a)$ means that we *push* the symbol a onto the stack; and a transition labeled $(\uparrow, a)$ means that we *pop* the symbol a off of the stack.

Thus, the way the PDA P illustrated in Figure 11.1 works is that it first pushes the stack symbol $\diamond$ onto the stack (which we assume is initially empty) and enters

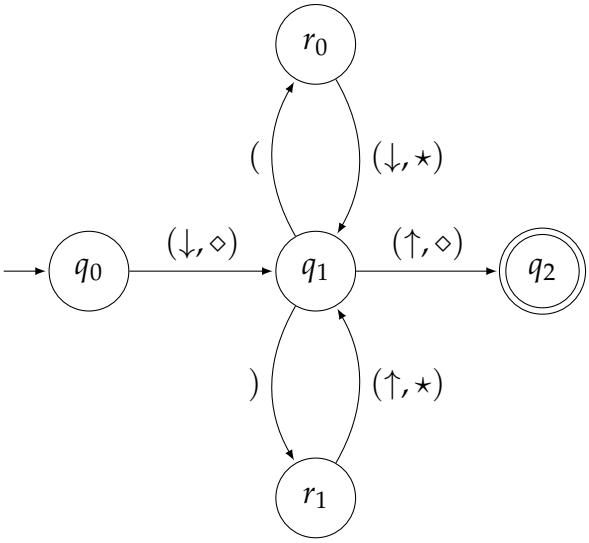

Figure 11.1: The state diagram of a PDA recognizing BAL.

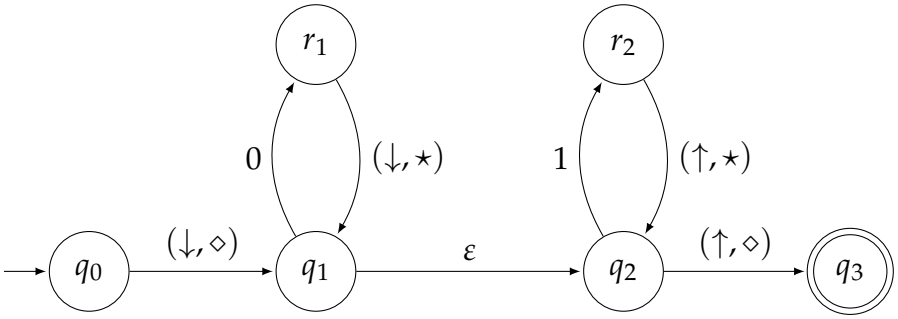

Figure 11.2: The state diagram of a PDA recognizing SAME.

state q_1 (without reading anything from the input). From state q_1 it is possible to either read the left-parenthesis symbol "(" and move to r_0 or read the right-parenthesis symbol ")" and move to r_1. To get back to q_1 we must either push the symbol $\star$ onto the stack (in the case that we just read a left-parenthesis) or pop the symbol $\star$ off of the stack (in the case that we just read a right-parenthesis). Finally, to get to the accept state q_2 from q_1, we must pop the symbol $\diamond$ off of the stack. Note that a transition requiring a pop operation can only be followed if that symbol is on the top of the stack.

It is not too hard to see that the language recognized by this PDA is the language BAL of balanced parentheses; these are precisely the input strings for which it will be possible to perform the required pushes and pops to land on the accept

state q_2 after the entire input string is read.

A second example is given in Figure 11.2. In this case the PDA recognizes the language

$$\text{SAME} = \{0^n 1^n \,:\, n \in \mathbb{N}\}. \tag{11.1}$$

In this case the stack is essentially used as a counter: we push a star for every 0, pop a star for every 1, and by using the "bottom of the stack marker" $\diamond$ we check that an equal number of the two symbols have been read.

Definition of pushdown automata

The formal definition of the pushdown automata model is similar to that of nondeterministic finite automata, except that one must also specify the alphabet of stack symbols and alter the form of the transition function so that it specifies how the stack operates.

Before we get to the definition, let us introduce some notation that will be useful for discussing stack operations. For any alphabet Γ, which we will refer to as the *stack alphabet*, the *stack operation alphabet* $\updownarrow\Gamma$ is defined as

$$\updownarrow\Gamma = \{\downarrow, \uparrow\} \times \Gamma. \tag{11.2}$$

The alphabet $\updownarrow\Gamma$ represents the possible stack operations for a stack that uses the alphabet Γ; for each $a \in \Gamma$ we imagine that the symbol $(\downarrow, a)$ represents pushing a onto the stack, and that the symbol $(\uparrow, a)$ represents popping a off of the stack.

Definition 11.1. A *pushdown automaton* (or PDA for short) is 6-tuple

$$P = (Q, \Sigma, \Gamma, \delta, q_0, F) \tag{11.3}$$

where Q is a finite and nonempty *set of states*, Σ is an alphabet (called the *input alphabet*), Γ is an alphabet (called the *stack alphabet*), which must satisfy $\Sigma \cap \updownarrow\Gamma = \varnothing$, δ is a function of the form

$$\delta : Q \times \big(\Sigma \cup \updownarrow\Gamma \cup \{\varepsilon\}\big) \to \mathcal{P}(Q), \tag{11.4}$$

$q_0 \in Q$ is the *start state*, and $F \subseteq Q$ is a *set of accept states*.

The way to interpret a transition function having the above form is that the set of possible labels on transitions is $\Sigma \cup \updownarrow\Gamma \cup \{\varepsilon\}$; we can either read a symbol a, push a symbol from Γ onto the stack, pop a symbol from Γ off of the stack, or take an ε-transition.

Strings of valid stack operations

Before we discuss the formal definition of acceptance for PDAs, it will be helpful to think about stacks and valid sequences of stack operations. Consider any stack alphabet Γ, and let the stack operation alphabet $\updownarrow\Gamma$ be as defined previously.

We can view a string $v \in (\updownarrow\Gamma)^*$ as either representing or failing to represent a valid sequence of stack operations, assuming we read it from left to right and imagine starting with an empty stack. If a string does represent a valid sequence of stack operations, we will say that it is a *valid stack string*; and if a string fails to represent a valid sequence of stack operations, we will say that it is an *invalid stack string*.

For example, if $\Gamma = \{0, 1\}$, then these strings are valid stack strings:

$$\begin{gathered}(\downarrow, 0)(\downarrow, 1)(\uparrow, 1)(\downarrow, 0)(\uparrow, 0)(\uparrow, 0), \\ (\downarrow, 0)(\downarrow, 1)(\uparrow, 1)(\downarrow, 0)(\uparrow, 0).\end{gathered} \tag{11.5}$$

In the first case the stack is transformed like this (where the left-most symbol represents the top of the stack):

$$\varepsilon \to 0 \to 10 \to 0 \to 00 \to 0 \to \varepsilon. \tag{11.6}$$

The second case is similar, except that we do not leave the stack empty at the end:

$$\varepsilon \to 0 \to 10 \to 0 \to 00 \to 0. \tag{11.7}$$

On the other hand, these strings are invalid stack strings:

$$\begin{gathered}(\downarrow, 0)(\downarrow, 1)(\uparrow, 0)(\downarrow, 0)(\uparrow, 1)(\uparrow, 0), \\ (\downarrow, 0)(\downarrow, 1)(\uparrow, 1)(\downarrow, 0)(\uparrow, 0)(\uparrow, 0)(\uparrow, 1).\end{gathered} \tag{11.8}$$

For the first case we start by pushing 0 and then 1, which is fine, but then we try to pop 0 even though 1 is on the top of the stack. In the second case the very last symbol is the problem: we try to pop 1 even through the stack is empty.

It is the case that the language over the alphabet $\updownarrow\Gamma$ consisting of all valid stack strings is a context-free language. To see that this is so, let us first consider the language of all valid stack strings that also leave the stack empty after the last operation. For instance, the first sequence in (11.5) has this property while the second does not. We can obtain a CFG for this language by mimicking the CFG for the balanced parentheses language, but imagining a different parenthesis type for each symbol.

To be more precise, let us define a CFG G so that it includes the rule

$$S \to (\downarrow, a)\, S\, (\uparrow, a)\, S \tag{11.9}$$

for every symbol $a \in \Gamma$, as well as the rule $S \to \varepsilon$. This CFG generates the language of valid stack strings for the stack alphabet Γ that leave the stack empty at the end.

If we drop the requirement that the stack be left empty after the last operation, then we still have a context-free language. This is because this is the language of all *prefixes* of the language generated by the CFG in the previous paragraph, and the context-free languages are closed under taking prefixes.

Definition of acceptance for PDAs

Next let us consider a formal definition of what it means for a PDA P to accept or reject a string w.

Definition 11.2. Let $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA and let $w \in \Sigma^*$ be a string. The PDA P *accepts* the string w if there exists a natural number $m \in \mathbb{N}$, a sequence of states $r_0, \ldots, r_m$, and a sequence

$$a_1, \ldots, a_m \in \Sigma \cup \updownarrow\Gamma \cup \{\varepsilon\} \tag{11.10}$$

for which these properties hold:

1. $r_0 = q_0$ and $r_m \in F$.
2. $r_{k+1} \in \delta(r_k, a_{k+1})$ for every $k \in \{0, \ldots, m-1\}$.
3. By removing every symbol from the alphabet $\updownarrow\Gamma$ from $a_1 \cdots a_m$, the input string w is obtained.
4. By removing every symbol from the alphabet Σ from $a_1 \cdots a_m$, a valid stack string is obtained.

If P does not accept w, then P *rejects* w.

For the most part the definition is straightforward. In order for P to accept w, there must exist a sequence of states, along with moves between these states, that agree with the input string and the transition function. In addition, the usage of the stack must be consistent with our understanding of how stacks work, and this is represented by the fourth property.

As you would expect, for a given PDA P, we let $\mathrm{L}(P)$ denote the *language recognized* by P, which is the language of all strings accepted by P.

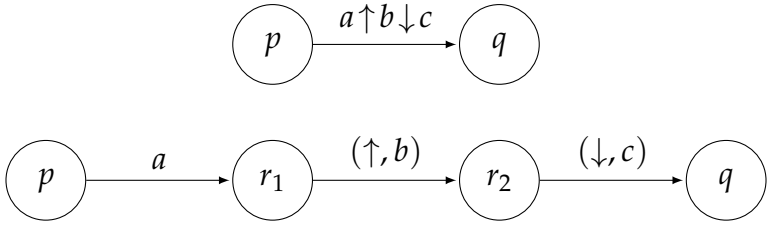

Figure 11.3: The shorthand notation for PDAs appears on the top, and the actual PDA states and transitions represented by this shorthand notation appears on the bottom.

Some useful shorthand notation for PDA state diagrams

There is a shorthand notation for PDA state diagrams that is sometimes useful, which is essentially to represent a sequence of transitions as if it were a single transition. In particular, if a transition is labeled

$$a \uparrow b \downarrow c, \tag{11.11}$$

the meaning is that the symbol a is read, b is popped off of the stack, and then c is pushed onto the stack. Figure 11.3 illustrates how this shorthand is to be interpreted. It is to be understood that the "implicit" states in a PDA represented by this shorthand are unique to each edge. For instance, the states r_1 and r_2 in Figure 11.3 are only used to implement this one transition from p to q, and are not reachable from any other states or used to implement other transitions.

This sort of shorthand notation can also be used in case multiple symbols are to be pushed or popped. For instance, an edge labeled

$$a \uparrow b_1 b_2 b_3 \downarrow c_1 c_2 c_3 c_4 \tag{11.12}$$

means that a is read from the input, $b_1 b_2 b_3$ is popped off the top of the stack, and then $c_1 c_2 c_3 c_4$ is pushed onto the stack. We will always follow the convention that the top of the stack corresponds to the left-hand side of any string of stack symbols, so such a transition requires b_1 on the top of the stack, b_2 next on the stack, and b_3 third on the stack—and when the entire operation is done, c_1 is on top of the stack, c_2 is next, and so on. One can follow a similar pattern to what is shown in Figure 11.3 to implement such a transition using the ordinary types of transitions from the definition of PDAs, along with intermediate states to perform the operations in the right order.

Finally, we can simply omit parts of a transition of the above form if those parts are not used. For instance, the transition label

$$a \uparrow b \tag{11.13}$$

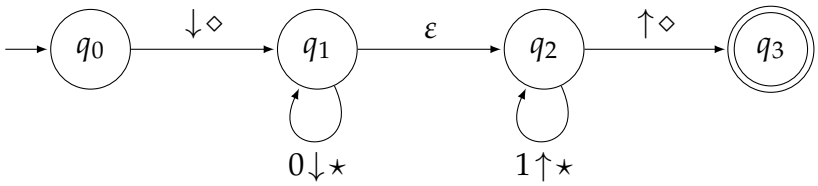

Figure 11.4: The state diagram of a PDA for SAME.

means "read a from the input, pop a off of the stack, and push nothing," the transition label

$$\uparrow b \downarrow c_1 c_2 \tag{11.14}$$

means "read nothing from the input, pop b off of the stack, and push $c_1 c_2$," and so on. Figure 11.4 illustrates the same PDA as in Figure 11.2 using this shorthand.

A remark on deterministic pushdown automata

It must be stressed that pushdown automata are, by default, considered to be nondeterministic.

It is possible to define a deterministic version of the PDA model, but if we do this we end up with a *strictly weaker* computational model. That is, every deterministic PDA will recognize a context-free language, but some context-free languages cannot be recognized by a deterministic PDA. One example is the language PAL of palindromes over the alphabet $\Sigma = \{0, 1\}$; this language is recognized by the PDA in Figure 11.5, but no deterministic PDA can recognize it.

We will not prove this fact, and indeed we have not even discussed a formal definition for deterministic PDAs, but the intuition is clear enough. Deterministic PDAs cannot detect when they have reached the middle of a string, and for this reason the use of a stack is not enough to recognize palindromes; no matter how you do it, the machine will never know when to stop pushing and start popping. A nondeterministic machine, on the other hand, can simply guess when to do this.

11.2 Further examples

Next we will consider a few additional operations under which the context-free languages are closed. These include string reversals, symmetric differences with finite languages, and a couple of operations that involve inserting and deleting certain alphabet symbols from strings.

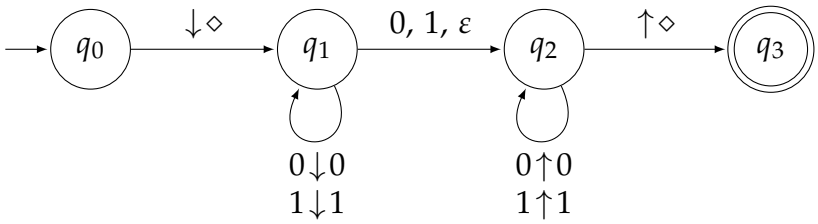

Figure 11.5: A PDA recognizing the language PAL.

Reverse

We already discussed string reversals in Lecture 6, where we observed that the reverse of a regular language is always regular. The same thing is true of context-free languages, as the following proposition establishes.

Proposition 11.3. *Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a context-free language. The language A^{R} is context free.*

Proof. Because A is context free, there must exists a CFG G such that $A = \mathrm{L}(G)$. Define a new CFG H as follows: H contains exactly the same variables as G, and for each rule $X \to w$ of G we include the rule $X \to w^{\mathrm{R}}$ in H. In words, H is the CFG obtained by reversing the right-hand side of every rule in G. It is evident that $\mathrm{L}(H) = \mathrm{L}(G)^{\mathrm{R}} = A^{\mathrm{R}}$, and therefore A^{R} is context free. □

Symmetric difference with a finite language

Next we will consider symmetric differences, which were also defined in Lecture 6. It is certainly not the case that the symmetric difference between two context-free languages is always context free, or even that the symmetric difference between a context-free language and a regular language is context free.

For example, if $A \subseteq \Sigma^*$ is context free but $\overline{A}$ is not, then the symmetric difference between A and the regular language Σ^* is not context free, because

$$A \triangle \Sigma^* = \overline{A}. \tag{11.15}$$

On the other hand, the symmetric difference between a context-free language and a *finite* language must always be context free, as the following proposition shows. This is interesting because the symmetric difference between a given language and a finite language carries an intuitive meaning: it means we modify that language on a finite number of strings, by either including or excluding these

strings. The proposition therefore shows that the property of being context free does not change when a language is modified on a finite number of strings.

Proposition 11.4. *Let Σ be an alphabet, let $A \subseteq \Sigma^*$ be a context-free language, and let and $B \subseteq \Sigma^*$ be a finite language. The language $A \triangle B$ is context free.*

Proof. First, given that B is finite, we have that B is regular, and therefore $\overline{B}$ is regular as well, because the regular languages are closed under complementation. This implies that $A \cap \overline{B}$ is context free, because the intersection of a context-free language and a regular language is context free.

Next, we observe that $\overline{A} \cap B$ is contained in B, and is therefore finite. Every finite language is context free, and therefore $\overline{A} \cap B$ is context free.

Finally, given that we have proved that both $A \cap \overline{B}$ and $\overline{A} \cap B$ are context free, it follows that $A \triangle B = \big(A \cap \overline{B}\big) \cup \big(\overline{A} \cap B\big)$ is context free because the union of two context-free languages is necessarily context free. □

Closure under string projections

Suppose that Σ and Γ are disjoint alphabets, and we have a string $w \in (\Sigma \cup \Gamma)^*$ that may contain symbols from either or both of these alphabets. We can imagine *deleting* all of the symbols in w that are contained in the alphabet Γ, which leaves us with a string over Σ. We call this operation the *projection* of a string over the alphabet $\Sigma \cup \Gamma$ onto the alphabet Σ.

We will prove two simple closure properties of the context-free languages that concern this notion. The first one says that if you have a context-free language over the alphabet $\Sigma \cup \Gamma$, and you project all of the strings in A onto the alphabet Σ, you are left with a context-free language.

Proposition 11.5. *Let Σ and Γ be disjoint alphabets, let $A \subseteq (\Sigma \cup \Gamma)^*$ be a context-free language, and define*

$$B = \left\{ w \in \Sigma^* \;:\; \begin{array}{l} \textit{there exists a string } x \in A \textit{ such that } w \textit{ is} \\ \textit{obtained from } x \textit{ by deleting all symbols in } \Gamma \end{array} \right\}. \tag{11.16}$$

The language B is context free.

Proof. Because A is context free, there exists a CFG G in Chomsky normal form such that $\mathrm{L}(G) = A$. We will create a new CFG H as follows:

1. For every rule of the form $X \to YZ$ appearing in G, include the same rule in H. Also, if the rule $S \to \varepsilon$ appears in G, include this rule in H as well.
2. For every rule of the form $X \to a$ in G, where $a \in \Sigma$, include the same rule $X \to a$ in H.

3. For every rule of the form $X \to b$ in G, where $b \in \Gamma$, include the rule $X \to \varepsilon$ in H.

It is apparent that $\mathrm{L}(H) = B$, and therefore B is context free. □

We can also go the other way, so to speak: if A is a context-free language over the alphabet Σ, and we consider the language consisting of all strings over the alphabet $\Sigma \cup \Gamma$ that result in a string in A when they are projected onto the alphabet Σ, then this new language over $\Sigma \cup \Gamma$ will also be context free. In essence, this is the language you get by picking any string in A, and then inserting any number of symbols from Γ anywhere into the string.

Proposition 11.6. *Let Σ and Γ be disjoint alphabets, let $A \subseteq \Sigma^*$ be a context-free language, and define*

$$B = \left\{ x \in (\Sigma \cup \Gamma)^* \ : \ \begin{array}{l} \textit{the string } w \textit{ obtained from } x \textit{ by deleting} \\ \textit{all symbols in } \Gamma \textit{ satisfies } w \in A \end{array} \right\}. \tag{11.17}$$

The language B is context free.

Proof. Because A is context free, there exists a CFG G in Chomsky normal for such that $\mathrm{L}(G) = A$. Define a new CFG H as follows:

1. Include the rule
$$W \to bW \tag{11.18}$$
in H for each $b \in \Gamma$, as well as the rule $W \to \varepsilon$, for a new variable W not already used in G. The variable W generates any string of symbols from Γ, including the empty string.
2. For each rule of the form $X \to YZ$ in G, include the same rule in H without modifying it.
3. For each rule of the form $X \to a$ in G, include this rule in H:
$$X \to WaW \tag{11.19}$$
4. If the rule $S \to \varepsilon$ is contained in G, then include this rule in H:
$$S \to W \tag{11.20}$$

Intuitively speaking, H operates in much the same way as G, except that any time G generates a symbol or the empty string, H is free to generate the same string with any number of symbols from Γ inserted. We have that $\mathrm{L}(H) = B$, and therefore B is context free. □

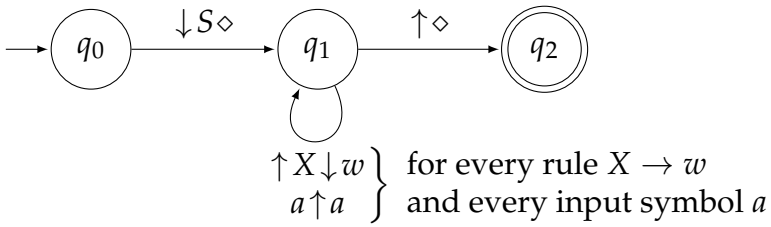

Figure 11.6: A PDA recognizing the language of an arbitrary CFG.

11.3 Equivalence of PDAs and CFGs

As suggested earlier in the lecture, it is the case that a language is context free if and only if it is recognized by a PDA. This section gives a high-level description of one way to prove this equivalence.

Every context-free language is recognized by a PDA

To prove that every context-free language is recognized by some PDA, we can define a PDA that corresponds directly to a given CFG. That is, if

$$G = (V, \Sigma, R, S) \tag{11.21}$$

is a context-free grammar, then we can obtain a PDA P such that $\mathrm{L}(P) = \mathrm{L}(G)$ in the manner suggested by Figure 11.6. The stack symbols of P are taken to be $V \cup \Sigma$, along with a special bottom of the stack marker $\diamond$ (which we assume is not contained in $V \cup \Sigma$), and during the computation the stack provides a way to store the symbols and variables needed to carry out a derivation with respect to G.

If you consider how derivations of strings by a grammar G and the operation of the corresponding PDA P work, it will be evident that P accepts precisely those strings that can be generated by G. We start with just the start variable on the stack (in addition to the bottom of the stack marker). In general, if a variable appears on the top of the stack, we can pop it off and replace it with any string of symbols and variables appearing on the right-hand side of a rule for the variable that was popped; and if a symbol appears on the top of the stack we essentially just match it up with an input symbol—so long as the input symbol matches the symbol on the top of the stack we can pop it off, move to the next input symbol, and process whatever is left on the stack. We can move to the accept state whenever the stack is empty (meaning that just the bottom of the stack marker is present), and if all of the input symbols have been read we accept. This situation is representative of the input string having been derived by the grammar.

Every language recognized by a PDA is context free

We will now argue that every language recognized by a PDA is context free. There is a method through which a given PDA can actually be converted into an equivalent CFG, but it is messy and the intuition tends to get lost in the details. Here we will summarize a different way to prove that every language recognized by a PDA is context free that is pretty simple, given the tools that we have already collected in our study of context-free languages. If you wanted to, you could turn this proof into an explicit construction of a CFG for a given PDA, and it would not be all that different from the method just mentioned, but we will focus just on the proof and not on turning it into an explicit construction.

Suppose we have a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$. The transition function δ takes the form

$$\delta : Q \times \big(\Sigma \cup \updownarrow\Gamma \cup \{\varepsilon\}\big) \to \mathcal{P}(Q), \tag{11.22}$$

so if we wanted to, we could think of P as being an NFA for some language over the alphabet $\Sigma \cup \updownarrow\Gamma$. Slightly more formally, let N be the NFA defined as

$$N = \big(Q, \Sigma \cup \updownarrow\Gamma, \delta, q_0, F\big); \tag{11.23}$$

we do not even need to change the transition function because it already has the right form of a transition function for an NFA over the alphabet $\Sigma \cup \updownarrow\Gamma$. Also define $B = \mathrm{L}(N) \subseteq (\Sigma \cup \updownarrow\Gamma)^*$ to be the language recognized by N. In general, the strings in B include symbols in both Σ and $\updownarrow\Gamma$. Even though symbols in $\updownarrow\Gamma$ may be present in the strings accepted by N, there is no requirement on these strings to actually represent a valid use of a stack, because N does not have a stack with which to check this condition.

Now let us consider a second language $C \subseteq (\Sigma \cup \updownarrow\Gamma)^*$. This will be the language consisting of all strings over the alphabet $\Sigma \cup \updownarrow\Gamma$ having the property that by deleting every symbol in Σ, a valid stack string is obtained. We already discussed the fact that the language consisting of all valid stack strings is context free, and so it follows from Proposition 11.6 that the language C is also context free.

Next, we consider the intersection $D = B \cap C$. Because D is the intersection of a regular language and a context-free language, it is context free. The strings in D actually correspond to valid computations of the PDA P that lead to an accept state; but in addition to the input symbols in Σ that are read by P, these strings also include symbols in $\updownarrow\Gamma$ that represent transitions of P that involve stack operations.

The language D is therefore not the same as the language A, but it is closely related; A is the language that is obtained from D by deleting all of the symbols in $\updownarrow\Gamma$ and leaving the symbols in Σ alone. Because we know that D is context free, it therefore follows that A is context free by Proposition 11.5, which is what we wanted to prove.