Chapter Thirteen: Stack Machines

Stack Machines

- Stacks are ubiquitous in computer programming, and they have an important role in formal language as well.
- A stack machine is a kind of automaton that uses a stack for auxiliary data storage.
  - The size of the stack is unbounded—it never runs out of space—and that gives stack machines an edge over finite automata.
  - In effect, stack machines have infinite memory, though they must use it in stack order.
- The set of languages that can be defined using a stack machine is exactly the same as the set of languages that can be defined using a CFG: the context-free languages.

Outline

Stacks

- A stack machine maintains an unbounded stack of symbols
- We'll represent these stacks as strings
- Left end of the string is the top of the stack
  - For example, *abc* is a stack with *a* on top and *c* on the bottom
  - Popping *abc* gives you the symbol *a*, leaving *bc* on the stack
  - Pushing *b* onto *abc* produces the stack *babc*

Stack Machine Moves

- A stack machine is an automaton for defining languages, but unlike DFA and NFA: no states!
- It is specified by a table that shows the moves it is allowed to make. For example:

read	pop	push
a	*c*	*a b c*

- Meaning:
  - If the current input symbol is *a*, and
  - if the symbol on top of the stack is *c*, it may make this move:
  - pop off the *c*, push *abc*, and advance to the next input symbol

Leaving The Stack Unchanged

- Every move pops one symbol off, then pushes a string of zero or more symbols on
- To specify a move that leaves the stack unchanged, you can explicitly push the popped symbol back on:

read	pop	push
a	*c*	*c*

- Meaning:
  - If the current input symbol is *a*, and
  - if the symbol on top of the stack is *c*, it may make this move:
  - pop off the *c*, push it back on, and advance to the next input symbol

Popping The Stack

- Every move pushes a string onto the stack
- To specify a move that pops but does not push, you can explicitly push the empty string:

read	pop	push
a	*c*	ε

- Meaning:
  - If the current input symbol is *a*, and
  - if the symbol on top of the stack is *c*, it may make this move:
  - pop off the *c*, push nothing in its place, and advance to the next input symbol

Moves On No Input

- The first column can be ε
- Like a ε-transition in an NFA, this specifies a move that is made without reading an input symbol

read	pop	push
ε	*c*	*a b*

- Meaning:
  - Regardless of what the next input symbol (if any) is,
  - if the symbol on top of the stack is *c*, it may make this move:
  - pop off the *c*, and push *ab* in its place

Stack Machines

- A stack machine starts with a stack that contains just one symbol, the start symbol *S*
- On each move it can alter its stack, but only as we have seen: only in stack order
- Like an NFA, a stack machine may be nondeterministic: it may have more than one sequence of legal moves on a given input
- A string is in the language if there is at least one sequence of legal moves that reads the entire input string and ends with the stack empty

Example

	read	pop	push
1.	ε	S	$a\ b$
2.	a	S	$e\ f$
3.	a	S	ε

- Consider input *a* (and, as always, initial stack *S*):
- Three possible sequences of moves
  - Move 1 first: no input is read and the stack becomes *ab*; then stuck, rejecting since input not finished and stack not empty
  - Move 2 first: *a* is read and the stack becomes *ef*; rejecting since stack not empty
  - Move 3 first: *a* is read and the stack becomes empty; accepting

Outline

- 13.1 Stack Machine Basics
- 13.2 A Stack Machine for $\{a^n b^n\}$
- 13.3 A Stack Machine for $\{xx^R\}$
- 13.4 Stack Machines, Formally Defined
- 13.5 Example: Equal Counts
- 13.6 Example: A Regular Language
- 13.7 A Stack Machine for Every CFG
- 13.8 A CFG For Every Stack Machine

Strategy For $\{a^n b^n\}$

- We'll make a stack machine that defines the language $\{a^n b^n\}$
- As always, the stack starts with *S*
- Reading the input string from left to right:
  1. For each *a* you read, pop off the *S*, push a 1, then push the *S* back on top
  2. In the middle of the string, pop off the *S*; at this point the stack contains just a list of zero or more 1s, one for each *a* that was read
  3. For each *b* you read, pop a 1 off the stack
- This ends with all input read and the stack empty, if and only if the input was in $\{a^n b^n\}$

Stack Machine For $\{a^n b^n\}$

	read	pop	push
1.	a	S	S 1
2.	ε	S	ε
3.	b	1	ε

- That strategy again:
  1. For each *a* you read, pop off the *S*, push a 1, then push the *S* back on top
  2. In the middle of the string, pop off the *S*; at this point the stack contains just a list of zero or more 1s, one for each *a* that was read
  3. For each *b* you read, pop a 1 off the stack

	read	pop	push
1.	*a*	*S*	*S* 1
2.	ε	*S*	ε
3.	*b*	1	ε

- Accepting *aaabbb*:
  - Start: input: $\underline{a}aabbb$; stack: $\underline{S}$
  - Move 1: input: $a\underline{a}abbb$; stack: $\underline{S}1$
  - Move 1: input: $aa\underline{a}bbb$; stack: $\underline{S}11$
  - Move 1: input: $aaa\underline{b}bb$; stack: $\underline{S}111$
  - Move 2: input: $aaa\underline{b}bb$; stack: $\underline{1}11$
  - Move 3: input: $aaab\underline{b}b$; stack: $\underline{1}1$
  - Move 3: input: $aaabb\underline{b}$; stack: $\underline{1}$
  - Move 3: input: *aaabbb*_; stack empty

	read	pop	push
1.	*a*	*S*	*S* 1
2.	ε	*S*	ε
3.	*b*	1	ε

- A rejecting sequence for *aaabbb*:
  - Start: input: $\underline{a}aabbb$; stack: $\underline{S}$
  - Move 1: input: $a\underline{a}abbb$; stack: $\underline{S}1$
  - Move 2: input: $a\underline{a}abbb$; stack: $\underline{1}$
  - No legal move from here
- But, as we've seen, there is an accepting sequence, so *aaabbb* is in the language defined by the stack machine
- What happens with string aabbb and aab?

Nondeterminism

- This stack machine can pop the S off the top of the stack at any time
- But there is only one correct time: it must be popped off in the middle of the input string
- This uses the nondeterminism of stack machines
- We can think of these machines as making a guess about where the middle of the input is
- All the sequences with a wrong guess reject
- But the one sequence that makes the right guess accepts, and one is all it takes

Outline

- 13.1 Stack Machine Basics
- 13.2 A Stack Machine for $\{a^n b^n\}$
- 13.3 A Stack Machine for $\{xx^R\}$
- 13.4 Stack Machines, Formally Defined
- 13.5 Example: Equal Counts
- 13.6 Example: A Regular Language
- 13.7 A Stack Machine for Every CFG
- 13.8 A CFG For Every Stack Machine

The 4-Tuple

- A *stack machine* M is a 4-tuple $M = (\Gamma, \Sigma, S, \delta)$
  - Γ is the stack alphabet
  - Σ is the input alphabet
  - $S \in \Gamma$ is the initial stack symbol
  - $\delta \in ((\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow P(\Gamma^*)$ is the transition function
- The stack alphabet and the input alphabet may or may not have symbols in common

Transition Function

- Type is $\delta \in ((\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow P(\Gamma^*)$
- That is, in $\delta(x,y) = Z$:
  - x is an input symbol or ε
  - y is a stack symbol
  - The result Z is a set of strings of stack symbols
- The result is a set because the stack machine is nondeterministic
- For a given input symbol x and top-of-stack symbol y, there may be more than one move
- So, there may be more than one string that can be pushed onto the stack in place of y

Example

	read	pop	push
1 .	ε	S	$a\ b$
2 .	a	S	$e\ f$
3 .	a	S	ε

- $M = (\Gamma, \Sigma, S, \delta)$ where
  - $\Gamma = \{S, a, b, e, f\}$
  - $\Sigma = \{a\}$
  - $\delta(\varepsilon, S) = \{ab\}$
    $\delta(a, S) = \{\varepsilon, ef\}$

Instantaneous Descriptions

- At any point in a stack machine's operation, its future depends on two things:
  - That part of the input string that is still to be read
  - The current contents of the stack
- An instantaneous description (ID) for a stack machine is a pair (x, y) where:
  - $x \in \Sigma^*$ is the unread part of the input
  - $y \in \Gamma^*$ is the current stack contents
- As always, the left end of the string y is considered to be the top of the stack

A One-Move Relation On IDs

- We will write $I \mapsto J$ if I is an ID and J is ID that follows from I after one move of the stack machine
- Technically: $\mapsto$ is a relation on IDs, defined by the δ function for the stack machine as follows:
  - Regular transitions: $(ax, Bz) \mapsto (x, yz)$ if and only if $y \in \delta(a,B)$
  - ε-transitions: $(x, Bz) \mapsto (x, yz)$ if and only if $y \in \delta(\varepsilon,B)$.
- Note no move is possible when stack is empty

Zero-Or-More-Move Relation

- As we did with grammars and NFAs, we extend this to a zero-or-more-move $\mapsto^*$
- Technically, $\mapsto^*$ is a relation on IDs, with $I \mapsto^* J$ if and only if there is a sequence of zero or more relations that starts with I and ends with J
- Note this is reflexive by definition: we always have $I \mapsto^* I$ by a sequence of zero moves

A Stack Machine's Language

- The language accepted by a stack machine is the set of input strings for which there is at least one sequence of moves that ends with the whole string read and the stack empty
- Technically, $L(M) = \{x \in \Sigma^* \mid (x, S) \mapsto^* (\varepsilon, \varepsilon)\}$

Previous Example

	read	pop	push
1.	*a*	*S*	*S* 1
2.	ε	*S*	ε
3.	*b*	1	ε

- Accepting *aaabbb*:
  - Start: input: $\underline{a}aabbb$; stack: $\underline{S}$
  - Move 1: input: $a\underline{a}abbb$; stack: $\underline{S}1$
  - Move 1: input: $aa\underline{a}bbb$; stack: $\underline{S}11$
  - Move 1: input: $aaa\underline{b}bb$; stack: $\underline{S}111$
  - Move 2: input: $aaa\underline{b}bb$; stack: $\underline{1}11$
  - Move 3: input: $aaab\underline{b}b$; stack: $\underline{1}1$
  - Move 3: input: $aaabb\underline{b}$; stack: $\underline{1}$
  - Move 3: input: *aaabbb*_; stack empty

Example, Continued

	read	pop	push
1.	a	S	S 1
2.	ε	S	ε
3.	b	1	ε

- $M = (\{a,b,S\}, \{a,b\}, S, \delta)$, where
  - $\delta(a,S) = \{S1\}$ $\delta(\varepsilon,S) = \{\varepsilon\}$ $\delta(b,1) = \{\varepsilon\}$
- The accepting sequence of moves for *abbbba* is
  - $(aaabbb, S) \mapsto (aabbb, S1) \mapsto (abbb, S11) \mapsto (bbb, S111)$
    $\mapsto (bbb, 111) \mapsto (bb, 11) \mapsto (b, 1) \mapsto (\varepsilon, \varepsilon)$
- $(aaabbb, S) \mapsto^* (\varepsilon, \varepsilon)$ and so $aaabbb \in L(M)$

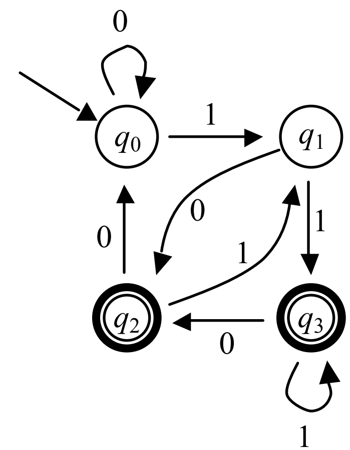
Outline

Simulating DFAs

- A stack machine can easily simulate any DFA
  - Use the same input alphabet
  - Use the states as stack symbols
  - Use the start state as the start symbol
  - Use a transition function that keeps exactly one symbol on the stack: the DFA's current state
  - Allow accepting states to be popped; that way, if the DFA ends in an accepting state, the stack machine can end with an empty stack

Example

- $M = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, q_0, \delta)$



  - $\delta(0,q_0) = \{q_0\}$ $\delta(1,q_0) = \{q_1\}$
  - $\delta(0,q_1) = \{q_2\}$ $\delta(1,q_1) = \{q_3\}$
  - $\delta(0,q_2) = \{q_0\}$ $\delta(1,q_2) = \{q_1\}$
  - $\delta(0,q_3) = \{q_2\}$ $\delta(1,q_3) = \{q_3\}$
  - $\delta(\varepsilon,q_2) = \{\varepsilon\}$ $\delta(\varepsilon,q_3) = \{\varepsilon\}$
- Accepting sequence for 0110:
  - $(0110, q_0) \mapsto (110, q_0) \mapsto (10, q_1) \mapsto (0, q_3) \mapsto (\varepsilon, q_2) \mapsto (\varepsilon, \varepsilon)$

DFA To Stack Machine

- Such a construction can be used to make a stack machine equivalent to any DFA
- It can be done for NFAs too
- It tells us that the languages definable using a stack machine include, at least, all the regular languages
- In fact, regular languages are a snap: we have an unbounded stack we barely used
- We won't give the construction formally, because we can do better…

Outline

- 13.1 Stack Machine Basics
- 13.2 A Stack Machine for $\{a^n b^n\}$
- 13.3 A Stack Machine for $\{xx^R\}$
- 13.4 Stack Machines, Formally Defined
- 13.5 Example: Equal Counts
- 13.6 Example: A Regular Language
- 13.7 A Stack Machine for Every CFG
- 13.8 A CFG For Every Stack Machine

From CFG To Stack Machine

- A CFG defines a string rewriting process
- Start with *S* and rewrite repeatedly, following the rules of the grammar until fully terminal
- We want a stack machine that accepts exactly those strings that could be generated by the given CFG
- Our strategy for such a stack machine:
  - Do a derivation, with the string in the stack
  - Match the derived string against the input

Strategy

- Two types of moves:
  1. A move for each production $X \rightarrow y$
  2. A move for each terminal $a \in \Sigma$
- The first type lets it do any derivation
- The second matches the derived string and the input
- Their execution is interlaced:
  - type 1 when the top symbol is nonterminal
  - type 2 when the top symbol is terminal

read	pop	push
ε	X	y
a	a	ε

Example: $\{xx^R \mid x \in \{a,b\}^*\}$

$S \rightarrow aSa \mid bSb \mid \varepsilon$

	read	pop	push
1.	ε	S	aSa
2.	ε	S	bSb
3.	ε	S	ε
4.	a	a	ε
5.	b	b	ε

- Derivation for *abbbba:*

  $S \Rightarrow aSb \Rightarrow abSba \Rightarrow abbSbba \Rightarrow abbbba$

- Accepting sequence of moves on *abbbba:*

  $(abbbba, S) \mapsto_1 (abbbba, aSa) \mapsto_4 (bbbba, Sa) \mapsto_2 (bbbba, bSba) \mapsto_5$
  $(bbba, Sba) \mapsto_2 (bbba, bSbba) \mapsto_5 (bba, Sbba) \mapsto_3 (bba, bba) \mapsto_5$
  $(ba, ba) \mapsto_5 (a, a) \mapsto_4 (\varepsilon, \varepsilon)$

Lemma 13.7

If $G = (V, \Sigma, S, P)$ is any context-free grammar, there is some stack machine M with $L(M) = L(G)$.

- Proof sketch: by construction
- Construct $M = (V \cup \Sigma, \Sigma, S, \delta)$, where
  - for all $v \in V$, $\delta(\varepsilon, v) = \{x \mid (v \to x) \in P\}$
  - for all $a \in \Sigma$, $\delta(a, a) = \{\varepsilon\}$
- M accepts x if and only if G generates x ie,

  $(x, S) \mapsto^* (\varepsilon, \varepsilon)$ if and only if $S \Rightarrow^* x$
- $L(M) = L(G)$

Summary

- We can make a stack machine for every CFL
- That's stronger than our demonstration of a stack machine for every regular language
- So now we know that the stack machines are at least as powerful as CFGs for defining languages
- Are they more powerful? Are there stack machines that define languages that are not CFLs?

Outline

- 13.1 Stack Machine Basics
- 13.2 A Stack Machine for $\{a^n b^n\}$
- 13.3 A Stack Machine for $\{xx^R\}$
- 13.4 Stack Machines, Formally Defined
- 13.5 Example: Equal Counts
- 13.6 Example: A Regular Language
- 13.7 A Stack Machine for Every CFG
- 13.8 A CFG For Every Stack Machine

From Stack Machine To CFG

- We can't just reverse the previous construction, since it produced restricted productions
- But we can use a similar idea
- The executions of the stack machine will be exactly simulated by derivations in the CFG
- To do this, we'll construct a CFG with one production for each move of the stack machine

Lemma 13.8.1

> If $M = (\Gamma, \Sigma, S, \delta)$ is any stack machine, there is context-free grammar G with $L(G) = L(M)$.

- Proof by construction
- Assume that $\Gamma \cap \Sigma = \{\}$ (without loss of generality)
- Construct $G = (\Gamma, \Sigma, S, P)$, where
  $P = \{(A \rightarrow at) \mid A \in \Gamma,\ \mathrm{a} \in \Sigma \cup \{\varepsilon\}, \text{ and } t \in \delta(\mathrm{a}, A)\}$
  where $\mathrm{t} \in \Gamma^*$
- Now leftmost derivations in G simulate runs of M:
  $S \Rightarrow^* x$ if and only if $(x, S) \mapsto^* (\varepsilon, \varepsilon)$
  for any $x \in \Sigma^*$
- So $L(G) = L(M)$

	read	pop	push
1.	a	S	SB
2.	ε	S	ε
3.	b	B	ε

1. $S \rightarrow aSB$
2. $S \rightarrow \varepsilon$
3. $B \rightarrow b$

- One-to-one correspondence:
  - Where the stack machine has $t \in \delta(a, A)$…
  - … the grammar has $A \rightarrow at$
- Accepting sequence on *aabb:*

  $(aabb, S) \mapsto_1 (abb, SB) \mapsto_1 (bb, SBB) \mapsto_2 (bb, BB) \mapsto_3 (b, B) \mapsto_3 (\varepsilon, \varepsilon)$
- Derivation of *abab:*

  $S \Rightarrow_1 aSB \Rightarrow_1 aaSBB \Rightarrow_2 aaBB \Rightarrow_3 aabB \Rightarrow_3 aabb$

Disjoint Alphabets Assumption

- The stack symbols of the stack machine become nonterminals in the CFG
- The input symbols of the stack machine become terminals of the CFG
- That's why we need to assume $\Gamma \cap \Sigma = \{\}$: symbols in a grammar must be either terminal or nonterminal, not both
- This assumption is without loss of generality because we can easily rename stack machine symbols to get disjoint alphabets…

Renaming Example

- Given a stack machine with intersecting alphabets:

	read	pop	push
1.	a	S	Sbb
2.	ε	S	ε
3.	b	b	ε

- We can rename the stack symbols (the pop and push columns only) to get disjoint alphabets:

	read	pop	push
1.	a	S	SBB
2.	ε	S	ε
3.	b	B	ε

- Then use the construction:

$$\boxed{\begin{array}{l} S \rightarrow aSBB \mid \varepsilon \\ B \rightarrow b \end{array}}$$

Theorem 13.8

A language is context free if and only if it is $L(M)$ for some stack machine M.

- Proof: follows immediately from Lemmas 13.7 and 13.8.1.
- Conclusion: CFGs and stack machines have equivalent definitional power