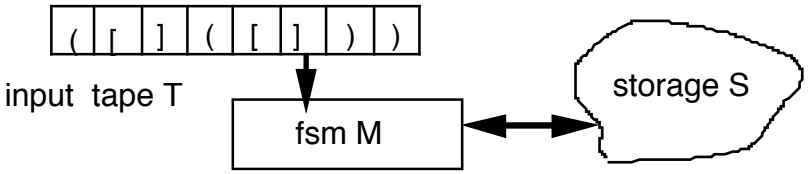
4. Finite automata with external storage

Concepts of this chapter: Finite automata with external storage of unbounded capacity, subject to various access restrictions that limit the automa's power of computation. Counter automata (CA), pushdown automata (PDA), non-equivalence of deterministic (DPDA) and non-deterministic PDAs (DPDA), linear bounded automata (LBA). Different access restrictions naturally lead to automata of different power, expressed by the relationship: CA < DPDA < NPDA < LBA. Remarkably, a further relaxation of access restrictions quickly leads to a universal model of computation, i.e. one as powerful as any other model (of discrete computation). We refer to the 2-PDA (a PDA with 2 stacks), the QM or Post machine (a finite state machine and a queue), and the Turing machine (TM). These will be discussed in more detail in later chapters.

4.1 Finite automata with external storage - concepts, conventions, notation

FSM M controls access to an input tape T (if any) and a storage device S. General form of a transition:

(current state of M, currently scanned symbol on T, currently scanned symbol on S)
-> (new state of M, newly written symbol on S, motion of the read/write head on S)



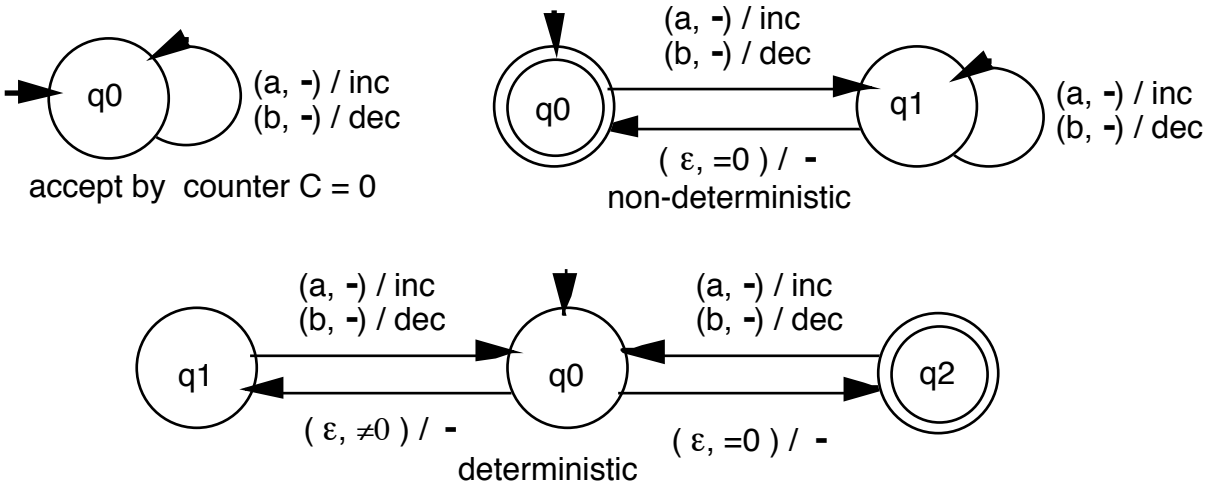
The two important parameters are:
- size of the storage device (unbounded, except for LBA, where storage size depends on the size of the input)
- access operations (e.g. read, write, motion of the read/write head, test for empty).

Details may vary greatly, e.g:
There may be no input tape T. The input to be processed is written in a designated area of the storage device.
Structure of the storage device, e.g: one or more tapes (single-ended or double-ended), 2-d grid, etc.
Acceptance: by accepting state or by designated content of the storage device (e.g. empty).
Tape T and storage S may have separate alphabets, which we call A, B.
We will use different conventions chosen to make each example simple.

4.2 Counter automata

A counter automaton consists of a fsm M augmented by a register (counter) C that can hold a single integer of arbitrary size. C is initialized to zero. M can increment and decrement the counter, and test it for 0. Thus, M's memory is unbounded, and arguments of the type "FAs can't count", as formalized in the pumping lemma of Ch 3, don't apply. In principle, any amount of data can be coded into a single integer. For example, a sequence of 3 integers a, b, c can encoded reversibely by the single integer $2^a 3^b 5^c$ (Goedel numbering). Thus, a CA has all the storage capacity one might need. Its computational power is restricted by access operations limited to increment, decrement, and test for 0. Due to these, counter automata "can't do much more than count".

As an example, each of the following counter automata accepts the language of all strings over A = {a, b} with an equal number of a's and b's:



A transition is activated by a pair (input symbol, state of counter) and triggers an action on the counter. Testable states of the counter are '=0' and '≠0', and counter actions are increment 'inc' and decrement 'dec'. Instead of reading an input symbol, M can also act based on the counter state alone, which we denote as 'reading the nullstring ε". M may ignore the state of the counter, which we denote by '-' = "don't care". Finally, M may choose to execute no action on the counter C, which we denote by '-' = "don't act".

HW 4.1: Parenthesis expressions. Polish or parenthesis-free notation for arithmetic expressions.
a) Consider the language L1 of correct parenthesis expressions over the alphabet A = { (,)}. Examples of correct expressions: (( )), ( ) ( ), (( )) ( ), and the nullstring ε. Either: exhibit a counter automaton M1 that accepts the language L1 , or show that no such counter automaton exists.
b) Consider the language L2 of correct parenthesis expressions over the alphabet A = { (,), [,]}, involving two pairs of parentheses. Examples of correct expressions: ([ ]), ( ) [ ] ( ), (( ) []) ( ), and the nullstring ε. Example of an incorrect expression: ([ )]. Either: exhibit a counter automaton M2 that accepts the language L2 , or argue that no such counter automaton exists.
c) Polish or parenthesis-free notation for arithmetic expressions come in 2 versions: prefix and suffix notation. Consider operands designated by a single letter, say x, y, or z, and the 4 binary operators +, -, *, /. In prefix notation the operator is written before its 2 operands, in suffix notation after. x+y becomes +xy or xy+. Design a counter automaton P that recognizes correct prefix expressions, and a counter automaton S that recognizes correct suffix epressions.

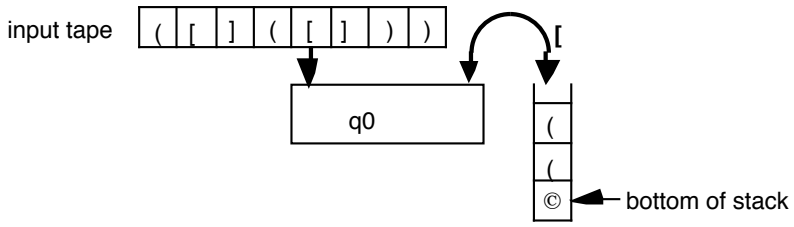
HW 4.2: Consider CAs whose counter can only hold a non-negative integer, call them CA+. Show that the class CA+ is equally powerful as the class CA whose automata can hold an arbitrary integer, positive or negative. Illustrate your construction by comparing a CA M and a CA+ M+, each of which accepts the language of all strings over {0, 1} that have an equal number 0s and 1s.

4.3 Deterministic pushdown automata (DPDA)

PDA: FSM controls a stack. Unbounded memory, limited last-in-first-out (LIFO) access.
Stack operations: push, pop, test empty. Abbreviation: $A\varepsilon = A \cup \{ \varepsilon \}$, $B\varepsilon = B \cup \{ \varepsilon \}$.

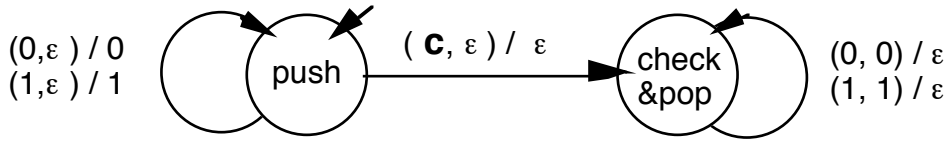
Df DPDA: M= (Q, A, B, f, q0, F) or M= (Q, A, B, f, q0) in the version "accept by empty stack"
Q: set of states, q0: initial state, $F \subseteq Q$: final or accepting states
A: input alphabet; B: stack alphabet (convention: includes a for all $a \in A$, and bottom of stack symbol ¢)
Transition fct f: $Q \times A\varepsilon \times B\varepsilon \to Q \times B^*$, transitions $(q, a, b) \to (q', v)$, $a \in A\varepsilon$, $b \in B\varepsilon$, $v \in B^*$.
When M is shown in a diagram, this transition is drawn as an arrow from q to q' labeled a, b -> v or a, b / v.
Reading a stack symbol implies 'pop'. If the stack is not consulted, read ε from the stack.

Ex: Parenthesis expressions involving 2 pairs of parentheses,( ) and [ ], are recognized by a 1-state controler that accepts by empty stack (an additional error or trap state, and error transitions, are not shown).



Transition function f: current state, input symbol, pop top of stack -> next state, push stack
q, (, ε -> q, (; q, [, ε -> q, [; q,), (-> q, ε ; q,], [-> q, ε

Ex: M accepts (via empty stack) **palindromes with a center marker c**. L = {w c $w_{reverse}$ | w ∈ {0, 1}* }.



LIFO access makes PDAs computationally weak! No deterministic PDA can recognize palindromes **without** an explicit center marker. It doesn't know when to stop pushing input symbols onto the stack, and to start popping. Non-determinism adds some power, but doesn't remove the LIFO bottleneck completely.

4.4 Nondeterministic pushdown automata (NPDA)

Df NPDA: M= (Q, A, B, f, q0, F) or M= (Q, A, B, f, q0) in the version "accept by empty stack"
Q, A, B, q0, F same as above, but the transition function is different: f: Q x Aε x Bε -> 2^{QxB^*}

Notice: **non-deterministic PDAs are more powerful acceptors than deterministic PDAs.**

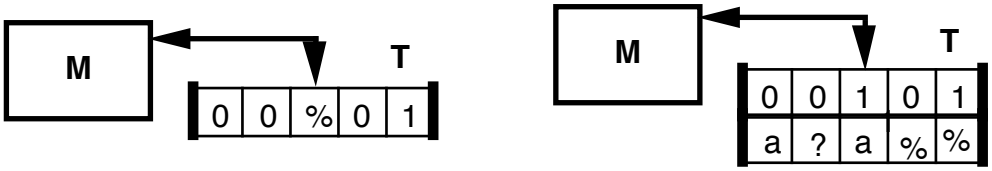
Ex: A 2-state NPDA M accepts (via empty stack) **even palindromes**, L_0 = {w $w^{reverse}$ | w ∈ {0, 1}* }.
p, 0, ε -> p, 0 ; p, 1, ε -> p, 1 p assumes M is still reading the first half of the input string
p, ε, ε -> q, ε non-deterministic guess of the mipoint
q, 0, 0 -> q, ε, q, 1, 1 -> q, ε q assumes M is reading the second half of the input string.

1) ∀ $ww^{reverse}$, w ∈ {0, 1}*, ∃ a sequence of transitions driven by $ww^{reverse}$ that results in an empty stack.
2) ∀ z ≠ $ww^{reverse}$, no sequence of transitions driven by z results in an empty stack.

Ex "LIFO bottleneck": In contrast to palindromes, it turns out that L = {ww | w ∈ {0, 1}* } is **not** accepted by any PDA (as explained in the next chapter, this implies that L is not context-free CF).

4.5 Linear bounded automata (LBA)

A linear bounded automaton is a finite automaton with read/write access to a tape T of fixed length determined by the input string w. M has a read/write head that can be moved left or right one square at a time, but cannot be moved off the tape. The name 'linear bounded' refers to the fact that T's storage capacity is a constant multiple of the capacity required to hold w.
M has a working alphabet B which contains A as a subset, and typically has additional characters used for scratch work and book-keeping. In the example of the figure at left, A = {0, 1}, B = {0, 1, %, ...}. A tape of length L has a storage capacity of L log |B| bits, which is a constant factor of log |B| / log |A| larger than the storage capacity required to store input strings over A of of the same length L . It is convenient to think of M's tape as consisting of several parallel tracks, as shown at right. A designated track contains the input string to be processed, and might be read-only. Other tracks are read/write tracks used for scratch work. In this model, M's alphabet is a Cartesian product A x B' x B" x ..of the input alphabet A and other track alphabets B', B", ..



A deterministic LBA has the components: M = (Q, A, B, f: Q x B -> Q x B x {L, R, H,..}, q_0, F}.
Q: finite state space; A: input alphabet; B tape alphabet; f: transition function; q_0: initial state; F: final states.
{L, R, H, ..}: tape actions: L = move left, R = move right, H = halt. Optional: "stay put".

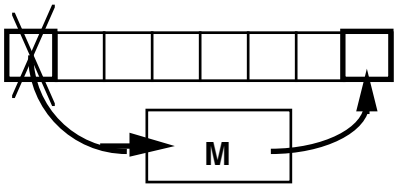
HW 4.3: The word problem for LBAs is decidable.
Describe a decision procedure which, given an LBA M and a word w, decides whether M accepts w.

4.6 Turing machines and automata of equivalent power

A Turing machine (TM, Alan Turing 1912-1954) is a FSM that controls a tape as an external storage device of unbounded size. Access operations include read/write one symbol at a time, moving the read/write head one square at a time, and possibly sensing the current end of the tape and extending it. The precise form of access operations are unimportant, as long as they are sufficiently versatile to make the machine "universal" in a sense that will be made precise in Ch 6. It is amazing how little it takes to turn some of the restricted automata discussed in this chapter into TMs!

Exercise: Consider a FSM that controls 2 stacks. Give a rigorous definition that captures this concept, and show that a "2-stack PDA" is equivalent to a FSM that controls an unbounded tape, i.e. a Turing machine.

Creative exercise: A **queue automaton** (QA) or Post machine (Emil Post 1897-1954) or Tag machine is a FSM with a single tape of unbounded length with FIFO access (first-in first-out, as opposed to the LIFO access of a stack). M reads and deletes the symbol at the head of the FIFO queue and may append a string to the tail end of the queue. Technical detail: there is a special symbol #, not part of the alphabet A of the input string, typically used as a delimiter. Perhaps surprisingly, Post machines are universal, i.e. equivalent in computational power to TMs.



End of Ch 4