Two-way finite automata & Pebble automata

Written by Liat Peterfreund

Two-way finite automata

A two way deterministic finite automata (2DFA)

is a quintuple $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where:

- Q, Σ, q_0, F are as before
- $\delta : Q \times \Sigma \to Q \times \{L, R\}$
  - If $\delta(q, a) = (p, L)$ then in state q, scanning the input symbol a, the $2DFA$ enters state p and moves its head left one square.
  - If $\delta(q, a) = (p, R)$ then in state q, scanning the input symbol a, the $2DFA$ enters state p and moves its head right one square.

COMMENTS:

- In describing the behavior of one-way finite automata, we extend δ to $Q \times \Sigma^*$. This notion is insufficient for $2DFA$ because the head may move left.
- Instead we introduce the notion of instantaneous description (ID) of $2DFA$ which describes the input string, current state and current position if the input head.
- The relation $\vdash_M$ on ID's such that $I_1 \vdash_M I_2$ if and only if M can go from I_1 to I_2 in one move.
- [3] An ID of M is a sting in $\Sigma^* Q \Sigma^*$.

- The ID wqx whereas $w, x \in \Sigma^*$ and $q \in Q$ is intended to represent :
  1) wx is the input string
  2) q is the current state
  3) the input head is scanning the first symbol of x (If $x = \varepsilon$ then the input head has moved off the right hand end of the input).

- We define the relation $\vdash_M$ or just $\vdash$ if M is understood, by

  1) $a_1 a_2 \ldots a_{i-1} q a_i \ldots a_n \vdash a_1 a_2 \ldots a_{i-1} a_i p a_{i+1} \ldots a_n$ whenever $\delta(q, a_i) = (p, R)$

  2) $a_1 a_2 \ldots a_{i-2} a_{i-1} q a_i \ldots a_n \vdash a_1 a_2 \ldots a_{i-2} p a_{i-1} a_i \ldots a_n$ whenever $\delta(q, a_i) = (p, L)$ and $i > 1$ (The condition $i > 1$ prevents any action in the event that the tape head would move off the left hand end of the tape).

- Note that no move is possible if $i = n+1$ (the tape head has moved off the right hand end).

Let $\vdash^*$ be the reflexive and transitive closure of $\vdash$.

$I_1 \vdash^* I_k$ whenever $I_1 \vdash I_2 \vdash \ldots \vdash I_k$ for some $I_2, \ldots, I_{k-1}$ we define $L(M) = \{w \mid q_0 w \vdash^* wp, p \in F\}$. That is w is accepted by M if, starting in state q_0 with w on the input tape and the head at the left end of w, M eventually enters a final state at the same time it falls off the right hand of the input tape.

Example:

Consider a $2DFA$ M that behaves as follows: starting in state q_0, M repeats a cycle of moves wherein the tape head moves right until two ones have been encountered then left until encountering a zero at which point state q_0 is reentered and the cycle repeated. More precisely M has three states all of which are final. δ is given as follows:

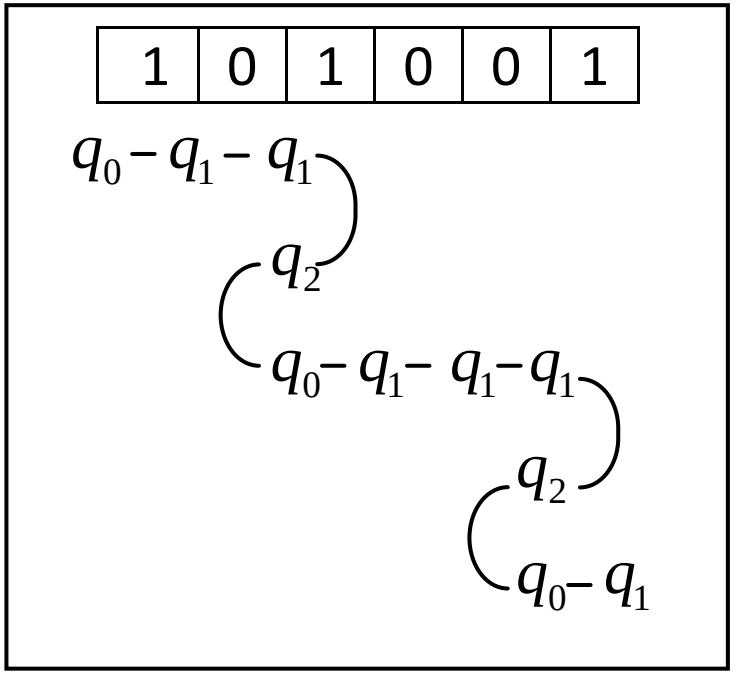
	0	1
q_0	(q_0, R)	(q_1, R)
q_1	(q_1, R)	(q_2, L)
q_2	(q_0, R)	(q_2, L)

Consider the input 101001:

$$
\begin{aligned}
q_0 101001 &\vdash 1q_1 01001 \\
&\vdash 10q_1 1001 \\
&\vdash 1q_2 01001 \\
&\vdash 10q_0 1001 \\
&\vdash 101q_1 001 \\
&\vdash 1010q_1 01 \\
&\vdash 10100q_1 1 \\
&\vdash 1010q_2 01 \\
&\vdash 10100q_0 1 \\
&\vdash 101001q_1
\end{aligned}
$$

than $101001 \in L(M)$

Crossing sequences



The list of states below each boundary between squares is termed a ***crossing sequence***.

Observations:

- If a $2DFA$ accepts its input, no crossing sequence may have a repeated state with the head moving in the same direction. Otherwise the $2DFA$ being deterministic would be in a loop.
- The first time a boundary is crossed, the head must be moving right.
- Subsequent crossings must be in alternate directions:
  - ✓ odd-numbered elements of the crossing sequence represents right moves
  - ✓ even-numbered elements of the crossing sequence represents left moves

- If the input is accepted then all crossing sequences are of odd length
- A crossing sequence $q_1, \ldots, q_k$ is said to be valid if :
  - ✓ it is of odd length
  - ✓ no two odd-numbered and no two even-numbered elements are identical
- The number of valid crossing sequences is finite (a $2DFA$ with s states can have valid crossing sequence of length at most 2s-1).

Theorem:

$2DFA$ and DFA (deterministic finite automata) have the same expressive power.

Proof:

- We construct a NFA (non-deterministic finite automata) that will simulate the $2DFA$ M and whose states are the valid crossing sequences of M .
- In order to construct the transition function we first examine the relationship between adjacent crossing sequences.

- Suppose we are given an isolated tape square holding the symbol a and are also given valid crossing sequences $q_1, q_2, \ldots, q_k$ and $p_1, p_2, \ldots, p_l$ at the left and the right boundaries of the square, respectively.
  - ✓ Note that there may be no input strings that can be attached to the left and right of symbol a to actually produce these two crossing sequences

- We can test two sequences for local compatibility as follows:
  - ✓ If the tape head moves left from the square holding a in state q_i, restart the automaton on the square holding a in state q_{i+1}
  - ✓ If the tape head moves right from the square holding a in state p_i, restart the automaton on the square holding a in state p_{i+1}

We take $q_1, q_2, \ldots, q_k$ to appear at the left boundary of a and $p_1, p_2, \ldots, p_l$ at the right boundary of a then,

- ✓ $q_1, q_2, \ldots, q_k$ right-match $p_1, p_2, \ldots, p_l$ on a if these sequences are consistent, assuming we initially reach a in state q_1 moving right
- ✓ $q_1, q_2, \ldots, q_k$ left-match $p_1, p_2, \ldots, p_l$ on a if these sequences are consistent, assuming we initially reach a in state p_1 moving left

We define right-matching and left matching pairs of crossing sequences recursively in (1) through (5):

(1) The null sequence left- and right-matches the null sequence. (if we never reach the square holding a, then it is consistent that the boundaries on neither side should be crossed).

(2) If $q_3, \ldots, q_k$ right-matches $p_1, p_2, \ldots, p_l$ and $\delta(q_1, a) = (q_2, L)$ then $q_1, q_2, \ldots, q_k$ right-matches $p_1, p_2, \ldots, p_l$ (if the first crossing of the left boundary is in state q_1 and the head immediately moves left in state q_2, then if we follow these two crossings by any consistent behavior starting from another crossing of the left boundary, we obtain a consistent pair of sequences with first crossing moving right i.e. a right matched pair).

(3) If $q_2, \ldots, q_k$ left-matches $p_2, \ldots, p_l$ and $\delta(q_1, a) = (p_1, R)$ then $q_1, q_2, \ldots, q_k$ right-matches $p_1, p_2, \ldots, p_l$.

(4) If $q_1, \ldots, q_k$ left-matches $p_3, \ldots, p_l$ and $\delta(p_1, a) = (p_2, R)$ then $q_1, \ldots, q_k$ left-matches $p_1, p_2, \ldots, p_l$.

(5) If $q_2, \ldots, q_k$ right matches $p_2, \ldots, p_l$ and $\delta(p_1, a) = (q_1, L)$ then $q_1, \ldots, q_k$ left-matches $p_1, p_2, \ldots, p_l$.

The construction:

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a $2DFA$. We contruct a NFA $M' = (Q', \Sigma, \delta', q_0', F')$ which accepts $L(M)$ such that:

1) Q' consists of all valid crossing sequences for M .
2) q_0' is the crossing sequence consisting of q_0 alone.
3) F' is the set of all crossing sequences of length one consisting of a state in F .
4) $\delta'(c, a) = \left\{ d \mid d \text{ is a valid crossing sequence that is right matched by } c \text{ on input } a \right\}$

- ✓ The intuitive idea is that M' puts together pieces of computations of M as it scans the input string. This is done by guessing successive crossing sequences. If M' has guessed that c is the crossing sequence at a boundary, and a is the next input symbol then M' can guess any valid crossing sequence that c right matches on input a.

- ✓ If the guessed computation results in M moving off the right end of the input in an accepting state, then M' accepts.

Claim:

$$L(M) = L(M')$$

Proof:

$L(M) \subseteq L(M')$:

Let $w \in L(M)$. Look at the crossing sequences generated by an accepting computation of M on w. Each crossing sequence right-matches the one at the next boundary, so M' can guess the proper crossing sequence (among its other guesses) and accept i.e. $w \in L(M')$.

$L(M') \subseteq L(M)$:

- ✓ Let $w \in L(M')$.
- ✓ Consider the crossing sequence $c_0, \ldots, c_n$ of M corresponding the states of M' as M' scans $w = a_1, \ldots, a_n$.
- ✓ For each $0 \leq i < n$ c_i right matches c_{i+1} on a_i
- ✓ We can construct an accepting computation of M on input w by determining when the head reverses direction.

- ✓ We prove by induction on i that M' on reading $a_1 \ldots a_i$ can enter state $c_i = [q_1, \ldots, q_k]$ only if the following two conditions hold:

  1) M started in state q_0 on $a_1 \ldots a_i$ will first move right from position i in state q_1

  2) for $j = 2, 4, \ldots$, if M is started at position i in state q_j, M will eventually move right from position i in state q_{j+1} (this implies k must be odd).

- ✓ Basis: $i = 0$ then $c_0 = [q_0]$. (1) is satisfied since M begins its computation by "moving right from position 0" in state q_0. (2) holds vacuously.

- ✓ Assume the hypothesis is true for $i-1$. Suppose that M' on reading $a_1 \ldots a_i$ can enter state $c_i = [p_1, \ldots, p_l]$ from state $c_{i-1} = [q_1, \ldots, q_k]$.
  - Since k and l are odd, and c_{i-1} right matches c_i on a_i there must exist an odd j such that in state q_j on input a_i, M moves right. Let j_1 be the smallest such j. By definition of "right matches" it follows that $\delta(q_{j_1}, a_1) = (p_1, R)$. This proves (1).
  - We will prove (2) by induction:
    - Basis: vacuously
    - Inductive step: by the definition of "right-matches" (rule (3)) $[q_{j_1+1}, \ldots, q_k]$ left-matches $[p_2, \ldots, p_l]$

- Now if $\delta(p_j, a_i) = (p_{j+1}, R)$ for all even j, then (2) follows immediately.
- In the case that for some smallest even j_2, $\delta(p_{j_2}, a_i) = (q, L)$, then by definition of "left-matches" (rule (5)) q must be q_{j_1+1} and $[q_{j_1+2}, \ldots, q_k]$ right matches $[p_{j_2+1}, \ldots, p_l]$
- By induction hypothesis, (2) holds for $[q_{j_1+2}, \ldots, q_k]$ and $[p_{j_2+1}, \ldots, p_l]$

- With the induction hypothesis for all i established, the fact that $c_n = [p]$ for some $p \in F$ implies that M accepts $a_1 \ldots a_n$.

Reminder - Example:

Consider a $2DFA$ M that behaves as follows: starting in state q_0, M repeats a cycle of moves wherein the tape head moves right until two ones have been encountered then left until encountering a zero at which point state q_0 is reentered and the cycle repeated. More precisely M has three states all of which are final. δ is given as follows:

	0	1
q_0	(q_0, R)	(q_1, R)
q_1	(q_1, R)	(q_2, L)
q_2	(q_0, R)	(q_2, L)

Consider the construction of $NFA\ M'$ equivalent to the $2DFA\ M$:

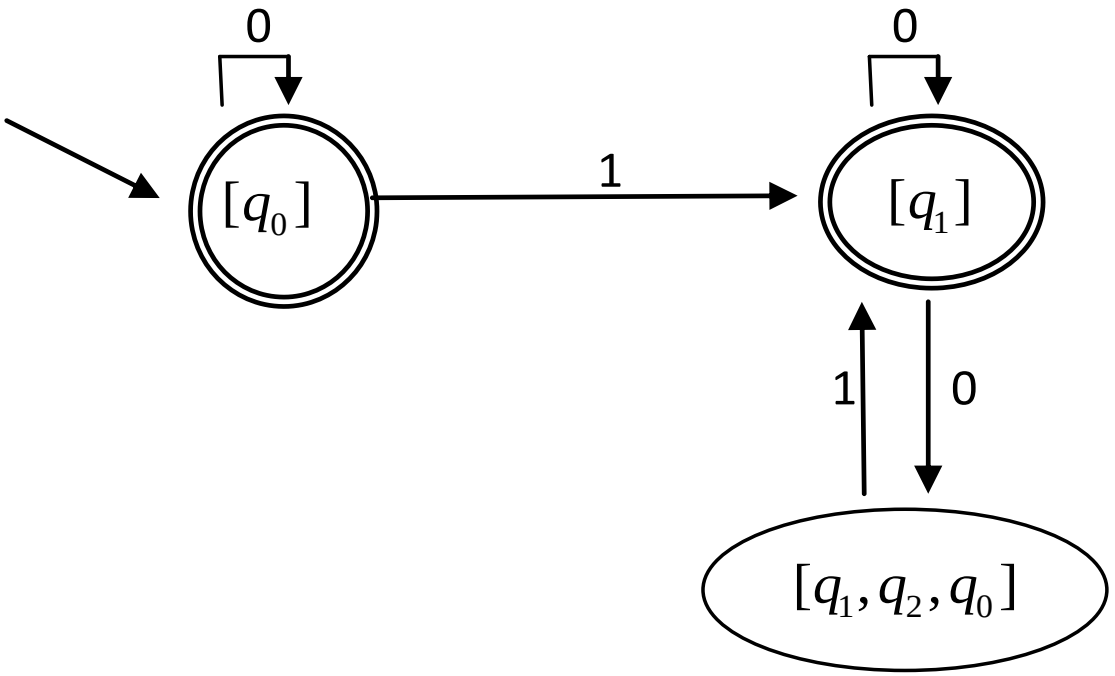
- Since q_2 is only entered on a left move and q_1 and q_3 are only entered on right moves, all even numbered components of valid crossing sequences must be q_2.
- Since a valid crossing sequence must be of odd length, and no two odd numbered states can be the same, nor can two even-numbered states be the same, there are only four crossing sequences of interest:

$[q_0], [q_1], [q_0, q_2, q_1], [q_1, q_2, q_0]$.

- Their right matches:

Valid crossing sequences	Right matches on 0	Right matches on 1
$[q_0]$	$[q_0]$	$[q_1]$
$[q_1]$	$[q_1],[q_1,q_2,q_0]$	-
$[q_0,q_2,q_1]$	-	-
$[q_1,q_2,q_0]$	-	$[q_1]$

Thus the automaton is:



Pebble Automata

Preface:

- We consider strings over infinite alphabet D.
- A D-string is a finite sequence $d_1 \ldots d_n$ where $n \geq 0$ and $\forall 1 \leq i \leq n : d_i \in D$.
- $\mathrm{dom}(w) = \{1, 2, \ldots, |w|\}$.
- For $i \in \mathrm{dom}(w)$ we write $val_w(i)$ for d_i.
- A language is a set of D-strings.

- We delimit input strings by two special symbols $\triangleleft, \triangleright$ for the left and for the right of the string, neither of which are in D.
- Therefore the automata work on extended strings of the form $w = \triangleright u \triangleleft$ where $u \in D^*$.
- The positions of $\triangleright, \triangleleft$ are 0 and $|w|+1$ respectively
- We write $dom^+(w)$ for the extended set $\{0,\ldots,|w|+1\}$ of positions
- We likewise extend the function val_w by defining $val_w(0) = \triangleright$ and $val_w(|w|+1) = \triangleleft$

- Pebble automata are finite state machines equipped with a finite number of pebbles.
- The use of pebbles is restricted by a stack discipline (pebble $i+1$ can only be placed when pebble i is present on the string and pebble i can only be lifted when pebble $i+1$ is not present on the string).
- The highest numbered pebble present on the string acts as the head of the automaton.

- A transition depends on:
  - The current state
  - The pebbles placed on the current position of the head
  - Equality type of the data values under the placed pebbles.
- The transition relation specifies:
  - Change of state
  - Movement of the head
  - Possibly whether the head pebble is removed or a new pebble is placed

Definition:

A nondeterministic two-way k-pebble automaton A over D (infinite alphabet) is a tuple (Q, q_0, F, T) where:

- Q is a finite set of states
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is a set of finite states

- T is a finite set of transitions of the form $\alpha \to \beta$ where
  - α is of the form (i, s, P, V, q) or (i, P, V, q) where $i \in \{1, \ldots, k\}$, $s \in D \cup \{\triangleleft, \triangleright\}$, $P, V \subseteq \{1, \ldots, i-1\}$
  - β is of the form (q, d) with $q \in Q$ and $d \in \begin{Bmatrix} \text{stay,left,right,place-new-pebble,} \\ \text{lift-current-pebble} \end{Bmatrix}$

Given a string w, a **configuration** of A on w is of the form $\gamma = [i, q, \theta]$ where $i \in \{1, \ldots, k\}$, $q \in Q$ and $\theta : \{1, \ldots, i\} \to \mathrm{dom}^{+}(w)$.

We call θ pebble assignment and i the depth of configuration (and of pebble assignment).

Initial configuration: $\gamma_0 := [1, q_0, \theta_0]$ where $\theta_0(1) = 0$.

Accepting configuration: $[i, q, \theta]$ where $\theta \in F$.

A transition $(i, s, P, V, q) \rightarrow \beta$ applies to a configuration $\gamma = [j, q, \theta]$ if:

1) $i = j, p = q$
2) $V = \{l < i \mid val_w(\theta(l)) = val_w(\theta(i))\}$
3) $P = \{l < i \mid \theta(l) = \theta(i)\}$
4) $val_w(\theta(i)) = s$

A transition $(i, P, V, q) \rightarrow \beta$ applies to a configuration $\gamma = [j, q, \theta]$ if 1-3 holds and no transition $(i, s, P, V, q) \rightarrow \beta$ applies to γ.

We define the transition relation $\vdash$ as follows:

$[i,q,\theta]\vdash[i',q',\theta']$ iff there is a transition $\alpha\rightarrow(p,d)$ that applies to γ such that $q'=p$ and $\forall j<i:\theta'(j)=\theta(j)$ and:

- If $d=\text{stay}$, then $i'=i$ and $\theta'(i)=\theta(i)$
- If $d=\text{left}$, then $i'=i$ and $\theta'(i)=\theta(i)-1$
- If $d=\text{right}$, then $i'=i$ and $\theta'(i)=\theta(i)+1$
- If $d=\text{place-new-pebble}$, then $i'=i+1$ and $\theta'(i)=\theta'(i+1)=\theta(i)$
- If $d=\text{lift-current-pebble}$, then $i'=i-1$

An automaton is **deterministic** if in each configuration at most one transition applies.

If there are no left transitions then the automaton is **one-way**.

The **language** of the automaton: A string w is accepted by A if $\gamma_0 \vdash^* \gamma$ for some accepting configuration γ.

Example 1:

Let $L = \{d_1 ... d_n \mid n \geq 0, \exists i, j : i \neq j \wedge d_i = d_j\}$.

We define a 2-pebble-automata A accepting this language.

$A = (Q, q_0, F, T)$ is defined as follows:

- $Q = \{q_1, q_2, q_{\rightarrow}, q_{acc}\}$
- $F = \{q_{acc}\}$
- T consists of the following transitions:

(1) $(1, \varnothing, \varnothing, q_1) \rightarrow (q_1, \text{right})$

(2) $(1, \varnothing, \varnothing, q_1) \rightarrow (q_{\rightarrow}, \text{place-new-pebble})$

(3) $(2, \{1\}, \{1\}, q_{\rightarrow}) \rightarrow (q_2, \text{right})$

(4) $(2, \varnothing, \varnothing, q_2) \rightarrow (q_2, \text{right})$

(5) $(2, \varnothing, \{1\}, q_2) \rightarrow (q_{acc}, \text{stay})$

Intuitive explanation:

(1-2) A stays in state q_1 while moving to the right or places a pebble

(3) After the placement of the pebble A moves one position to the right

(4-5) A continues moving to the right and when it reads a symbol equal to the symbol under the first pebble, it moves to the final state

Example 2:

Let $L = \{u\$v \mid u, v \in D^* \wedge \$ \notin D \wedge [u] \subseteq [v]\}$ whereas $[u], [v]$ are the sets of all the letters of u, v respectively.

We define a 2-pebble-automata A accepting this language that operates as follows:

- The first pebble passes through all of the positions of the word u (until it encounters the sign $\$$) and for each such position it checks whether there is a position in v with the same letter.
- The above check is done by placing the second pebble and iterating through all of the positions in v - if there is such a position it lifts the pebble and continues with the next position in u, if there is not it rejects.
- When the first pebble reaches the $\$$ it moves to an accepting state.

Formally, $A = (Q, q_0, F, T)$ whereas $Q = \{q_0, q_u, q_{uv}, q_v, q_{rej}, q_{acc}\}$, $F = \{q_{acc}\}$ and T contains the following transitions:

$(a1)(1, \varnothing, \varnothing, q_0) \rightarrow (q_u, \text{right})$

$(b1)(1, \$, \varnothing, \varnothing, q_u) \rightarrow (q_{acc}, \text{stay})$

$(c1)(1, \varnothing, \varnothing, q_u) \rightarrow (q_{uv}, \text{place-new-pebble})$

$(a2)(2, \{1\}, \{1\}, q_{uv}) \rightarrow (q_{uv}, \text{right})$

$(b2)(2, \$, \varnothing, \varnothing, q_{uv}) \rightarrow (q_v, \text{right})$

$(c2)(2, \varnothing, \varnothing, q_{uv}) \rightarrow (q_{uv}, \text{right})$

$(d2)(2, \varnothing, \{1\}, q_{uv}) \rightarrow (q_{uv}, \text{right})$

$(e2)(2, \varnothing, \{1\}, q_v) \rightarrow (q_0, \text{lift-current-pebble})$

$(f2)(2, \triangleleft, \varnothing, \varnothing, q_v) \rightarrow (q_{rej}, \text{stay})$

$(g2)(2, \varnothing, \varnothing, q_v) \rightarrow (q_v, \text{right})$

Intuitive explanation:

$(a1)$ is used to skip a sign

$(b1), (c1)$ are used to iterate all over u 's elements

$(a2) - (d2)$ skip the end of u and the $\$$

$(e2) - (f2)$ iterates through v and compares the sign under the current pebble with the sign under pebble 1

Example 3:

- We define a two-way nondeterministic pebble automaton with three pebbles that accepts all words w where $|w| \geq 2$ and there exists some position i in $dom(w)$ such that the set of symbols occurring at positions smaller than i is disjoint from the set of symbols occurring at position larger than or equal to i.
- For example the automaton accepts $\triangleright abb \triangleleft$ and rejects $\triangleright abab \triangleleft$

- The automaton will work as follows:
  - The first pebble is used nondeterministically to guess i
  - The second pebble is used to step through position to the left of i one at a time, from $i-1$ to $\triangleleft$
  - For each such position of the second pebble, the third pebble is used to check that all symbols at positions $\geq i$ are distinct from the symbol under the second pebble.
  - The input is excepted if the second pebble can reach the leftmost symbol $\triangleleft$

- Formal definition: $A = \langle Q, q_o, F, T \rangle$ where $Q = \{q_0, \ldots, q_7\}$, $F = \{q_5\}$ and T is the set of transitions:

$(a1)\ \ (1,\triangleright,\varnothing,\varnothing,q_0)\to(q_0,\text{right})$ } skip $\triangleright$

$(b1)\ \ (1,\varnothing,\varnothing,q_0)\to(q_1,\text{right})$ } $i>1$

$(c1)\ \ (1,\triangleleft,\varnothing,\varnothing,q_1)\to(q_2,\text{stay})$ } reached the end without guessing i

$(d1)\ \ (1,\varnothing,\varnothing,q_1)\to(q_1,\text{right})$

$(e1)\ \ (1,\varnothing,\varnothing,q_1)\to(q_3,\text{place-new-pebble})$

} guess i

$(a2)\ \ (2,\{1\},\{1\},q_3)\to(q_4,\text{left})$ } move to position $i-1$

$(b2)\ \ (2,\varnothing,\varnothing,q_3)\to(q_4,\text{left})$ } move one position left (after lifting pebble 3)

$(c2)\ \ (2,\triangleright,\varnothing,\varnothing,q_4)\to(q_5,\text{stay})$ } finished reading positions $\le i-1$

$(d2)\ \ (2,\varnothing,\varnothing,q_4)\to(q_6,\text{place-new-pebble})$ } "save" position

$(a3)\ \ (3,\{2\},\{2\},q_6)\to(q_6,\text{right})$

$(b3)\ \ (3,\varnothing,\varnothing,q_6)\to(q_6,\text{right})$

$(c3)\ \ (3,\varnothing,\{2\},q_6)\to(q_6,\text{right})$

$(d3)\ \ (3,\{1\},\{1\},q_6)\to(q_7,\text{right})$

} move right to position $i+1$

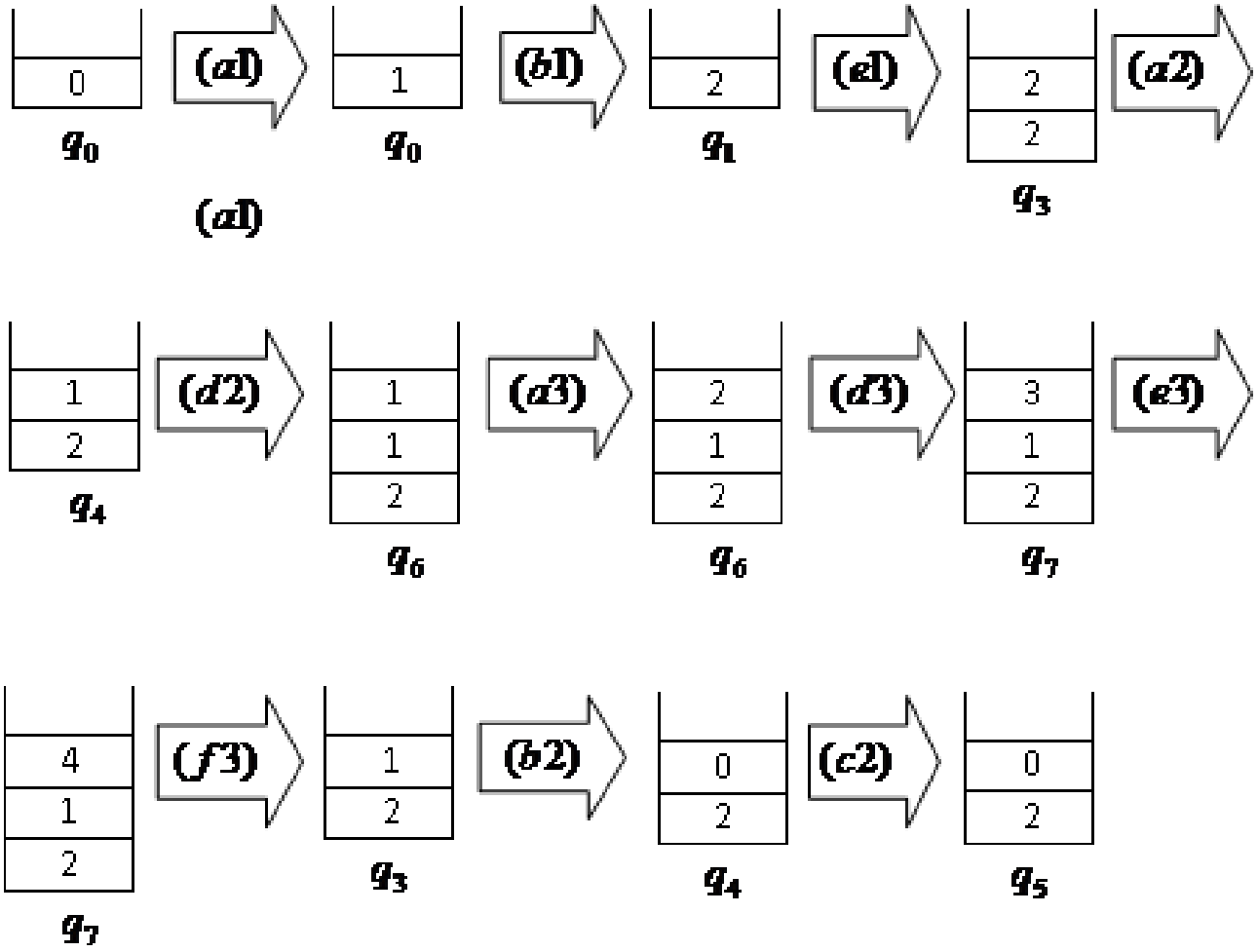
$(e3)\ \ (3,\varnothing,\{1\},q_7)\to(q_7,\text{right})$

$(f3)\ \ (3,\triangleleft,\varnothing,\varnothing,q_7)\to(q_3,\text{lift-pebble})$

$(g3)\ \ (3,\varnothing,\varnothing,q_7)\to(q_7,\text{right})$

} scan positions > i

- For example, an accepting run on $\triangleright abb \triangleleft$ will be:



- In the above definition, new pebbles are placed at the position if the most recent pebble. An alternative would be to place new pebbles at the beginning of the string. While the choice makes no difference in the two-way case, it is significant in the one way case. We refer to the model as defined above as **weak** pebble automata and to the latter as **strong** pebble automata. Strong pebble automata are formally defined by setting $\theta'(i+1)=0$ in the place-new-pebble case of the definition of the transition relation.

- Our next result shows that the variants of strong pebble automata, one-way and two-way, collapse.

Theorem:

Two-way weak 2-pebble automata and two-way strong 2-pebble automata have the same expressive power.

Proof:

$\Rightarrow$ Let $A = \langle Q, q_o, F, T \rangle$ be a two-way strong 2-pebble automaton. We want to construct a two-way weak 2-pebble automaton that will simulate A's behavior.

Idea of construction: For every state in $q \in Q$ we will add a new state q' and transition that such that the automaton head will move left until it encounters the left delimiter and then move to the appropriate state.

$\Leftarrow$ Let $A = \langle Q, q_o, F, T \rangle$ be a two-way weak 2-pebble automaton. We construct a two-way 2-pebble strong automaton that will simulate A's behavior. The only difference between the automata is in placing a new pebble.

Idea of construction: For every state in $q \in Q$ we will add a new state q' and transition that such that the automaton head will move right until it encounters the first pebble and then move to the appropriate state.

Theorem:

Two-way strong 2-pebble automata and one-way strong 2-pebble automata have the same expressive power.

Proof:

A one-way strong 2-pebble automaton is a particular case of two-way strong automaton.

Claim: For each two-way non-deterministic pebble automata A there is a strong one-way deterministic pebble automata that accepts the same language.

Proof:

Assumptions:

(1) A lifts pebble only in the right delimiter

(2) A accepts its input only in configurations [1, q, θ], i.e., with only one pebble.

By virtually adding two steps we view an accepting computation as consisting of:

(1) a first step in which the first pebble is placed at the left delimiter, i.e., position 0.

(2) a computation in which always at least one pebble is present.

(3) a final step in which the only remaining pebble is removed.

Writing $[0, p, \theta_{\varnothing}]$ for a (virtual) configuration without pebble, to determine whether A accepts, one has to find out whether $[0, q_0, \theta_{\varnothing}] \vdash^* [0, q, \theta_{\varnothing}]$ for some final state q.

The latter can be done by recursively solving subproblems of the form $[i, q, \theta] \vdash^*_{>i} [i, q', \theta]$

where the subscript > i indicates that only subcomputations are considered in which, at every step, more than i pebbles are present.

More formally, we show the following claim by induction on i (starting from i = k).

Claim: For each $i \in \{0,...,\mathrm{k}\}$ and each finite set of states R, there is a strong one-way deterministic pebble automata B_i (with k pebbles) such that, whenever B_i starts from a configuration $[\mathrm{i},\mathrm{p},\theta]$, where $\mathrm{p} \in \mathrm{R}$, the next configuration of depth i of B_i is $[\mathrm{i},(\mathrm{p},\ \mathrm{S}),\theta]$, where $\mathrm{S} = \{(\mathrm{q},\mathrm{q}') \in Q \times Q \mid [\mathrm{i},\mathrm{q},\theta] \vdash_{>i}^{*} [\mathrm{i},\ \mathrm{q}',\ \theta]\}$.

The set of states of B_i includes R and $R \times 2^{R \times R}$.

The theorem indeed follows from the claim: Let i = 0 and let $R = \{p_0\}$ (the intended initial state of B_0). We obtain an automaton which ends up in a state (p_0, S) where

$$S = \{(q, q') \in Q \times Q \mid [0, q, \theta_\varnothing] \vdash_{>0}^{*} [0, q', \theta_\varnothing]\}$$

The set of final states of B_0 consists of all states (p_0, S) where S contains a pair (q_0, q) with $q \in F$.

For $i = k$ the proof of the claim is trivial, as there are no configurations of depth $> \mathrm{k}$. Hence, B_k can compute (p, S) by a stay-transition.

Therefore, let $i < k$ and suppose the claim holds for all $j > i$

- Let the input string w of length n be fixed. Hence B_i works on the string $\triangleleft w \triangleright$ with positions from $\mathrm{dom}^+(w)$.
- For $l \leq n+1$, let θ^l denote the $(i+1)$-pebble assignment that coincides with θ in the first i pebbles and for which $\theta^l(i+1) = l$.
- $S_{\rightleftharpoons}(\theta^l)$ is the set of pairs (q, q') of states such that there is a computation starting at $[i+1, q, \theta^l]$ and reaching $[i+1, q', \theta^l]$ which only includes configurations $[j, q'', \theta']$ that fulfill $j > i+1$ or ($j = i+1$ and $\theta'(i+1) \leq l$). Intuitively, this says that pebble $i+1$ is not allowed to move to the right of position l.

- We write $S_{\searrow}(\theta^l)$ for the set of pairs $(q,\ q')$ of states for which $[i+1,\ q',\ \theta^l]$ can be reached from $[i+1,\ q,\ \theta^1]$ by a subcomputation satisfying the same property.

- The set $S_{\rightleftharpoons}(\theta^l)$ can be computed as follows:
- Let $R_{\rightleftharpoons}(\theta^l)$ be the set of pairs $(q,\ q')$ for which one of the following holds:

  (a) There exist p_1, p_2 such that

  $[i+1, q, \theta^l] \vdash [i+1,\ p_1, \theta^{l-1}]$,

  $(p_1, p_2) \in S_{\rightleftharpoons}(\theta^{l-1})$ and

  $[i+1,\ p_2, \theta^{l-1}] \vdash [i+1, q', \theta^l]$.

  (b) $[i+1, q, \theta^l] \vdash^*_{>i+1} [i+1, q', \theta^l]$

  (c) $[i+1, q, \theta^l] \vdash [i+1, q', \theta^l]$

- $S_{\rightleftharpoons}(\theta^l)$ is the transitive closure of $R_{\rightleftharpoons}(\theta^l)$.
- They are computed simultaneously.
- The information needed for (a) can be computed in one left-to-right pass of pebble $i+1$.
- By induction we can assume a subautomaton B_{i+1} that computes, for each position l, the part of $R_{\rightleftharpoons}(\theta^l)$ contributed by condition (b).
- (c) and the computation of the transitive closure do not require any pebble movements.
- During the same pass, the automaton can compute, for each position l, the set $S_{\rightarrow}(\theta^l)$. The computation of $S_{\rightarrow}(\theta^l)$ makes use of the sets $S_{\rightleftharpoons}(\theta^m)$, $m \leq l$.

- From $S_{\rightleftharpoons}(\theta^{n+1})$, $S_{\rightharpoonup}(\theta^{n+1})$ and the transition relation of A one can deduce, during a lift-pebble step, the set S as in the claim. Note that $n+1$ is the position of the right delimiter and recall that A lifts its pebbles only at that position.
- This completes the proof of the claim and of the theorem.