On pebble automata

And decision problems

Asaf Yeshurun
Definitions – words and alphabet

\[ \Sigma = \text{a finite alphabet of labels} \]
\[ D = \text{an infinite set of data values} \]

We define a word \( w \) as a finite sequence:
\[ w = (\sigma_1 a_1)(\sigma_2 a_2)\ldots(\sigma_n a_n) \text{ where } \sigma_i \in \Sigma \text{ and } a_i \in D \]

The input word of the automaton is of the form \( \langle w \rangle \) where \( \langle \) and \( \rangle \) are called the left and right end marker and do not belong to neither \( \Sigma \) nor \( D \).
Definitions – pebble automata

A one-way alternating weak k-pebble automaton (k-PA) over $\Sigma$ is a system $A = \langle \Sigma, Q, q_0, F, \mu, U \rangle$ whose components are:

- $Q, q_0 \in Q$ and $F \subseteq Q$ are a finite set of states, the initial state, and the set of final states.
- $U \subseteq Q - F$ is the set of universal states.
- $\mu \subseteq C \times D$ is the set of transitions, where
  - $C$ is a set whose elements are of the form $(i, \sigma, V, q)$, where $1 \leq i \leq k$, $\sigma \in \Sigma$, $V \subseteq \{i + 1, \ldots, k\}$ and $q \in Q$.
  - $D$ is a set whose elements are of the form $(q, \text{act})$ where $q \in Q$ and act is either stay, right, place-pebble or lift-pebble.
Given a word \( w = (\sigma_1 a_1 \sigma_2 a_2 \ldots \sigma_n a_n) \), a configuration of \( A \) on \( w \) is a triple \([i, q, \theta]\), where \( i \in \{1, \ldots, k\} \), \( q \in Q \), and \( \theta : \{i, i + 1, \ldots, k\} \rightarrow \{0, 1, \ldots, n, n + 1\} \).

The function \( \theta \) defines the position of the pebbles and is called the pebble assignment.

The initial configuration is \( \gamma_0 = [k, q_0, \theta_0] \),
where \( \theta_0(k) = 0 \) is the initial pebble assignment.

A transition \((i, \sigma, V, p) \rightarrow (p', act)\) applies to a configuration \([j, q, \theta]\), if:
1. \( i = j \) and \( p = q \),
2. \( V = \{l > i : a_{\theta(l)} = a_{\theta(i)}\} \),
3. \( a_{\theta(i)} = \sigma \).
The acceptance criteria are based on the notion of leads to acceptance. For every configuration $\gamma = [i, q, \theta]$:
- if $q \in F$, then $\gamma$ leads to acceptance.
- if $q \in U$, then $\gamma$ leads to acceptance if and only if for all configurations $\gamma'$ such that $\gamma \vdash \gamma'$, $\gamma$ leads to acceptance;
- if $q \notin F \cup U$, then $\gamma$ leads to acceptance if and only if there is at least one configuration $\gamma'$ such that $\gamma \vdash \gamma'$ and $\gamma'$ leads to acceptance.

A $\Sigma$-data word $w$ is accepted by $A$, if $\gamma_0$ leads to acceptance.

The automaton $A$ is non-deterministic, if the set $U = \emptyset$, and it is deterministic, if there is exactly one transition that applies for each configuration.
Theorem:
For all $k \geq 1$: alternating, non-deterministic and deterministic weak k-PA have the same recognition power.

Proof:
For every one-way alternating weak 2-PA, we will construct its equivalent one-way deterministic weak 2-PA. This is done in two steps:

1. we transform the one-way alternating weak 2-PA into its equivalent one-way non-deterministic weak 2-PA.

2. we transform the one-way non-deterministic weak 2-PA into its equivalent one-way deterministic weak 2-PA.
Let $A = \langle Q, q_0, F, \mu \rangle$ be a non-deterministic weak 2-PA. We normalize the behavior of $A$ as follows:

**N1.** For every configuration $\gamma$ of $A$, there exists a transition in $\mu$ that applies to it.

**N2.** There is no stay transition in $A$.

**N3.** The automaton can only enter a final state when the control is in pebble 2. Furthermore, it does so only after it reads the right-end marker.

**N4.** Immediately after pebble 2 moves right, pebble 1 is placed.

**N5.** Pebble 1 is lifted only when it reaches the right-end marker.

**N6.** Immediately after pebble 1 is lifted, pebble 2 moves right.

With the normalizations **N1–N6**, there is no non-deterministism in choosing which action to take. Non-determinism is now limited only in deciding which states to take. So we got to a point where

For each $i = 1, 2$ if $(i, P, V, p) \rightarrow (q_1, act_1)$ and $(i, P, V, p) \rightarrow (q_2, act_2)$, then $act_1 = act_2$.

And now we can take the power set of the states of $A$ to deterministically simulate $A$. 

Non-deterministic to deterministic
Let $A = \langle \Sigma, Q, q_0, \mu, F, U \rangle$ be a one-way alternating weak 2-PA. We normalize $A$ the following way:

**A1.** For every $p \in U$, if $(i, \sigma, V, p) \rightarrow (q, act) \in \mu$, then $act = stay$.

**A2.** Every pebble can be lifted only after it reads the right-end marker.

**A3.** Only pebble 2 can enter a final state and it does so only after it reads the right-end marker.

We assume that $Q$ is partitioned into $Q_1 \cup Q_2$ where $Q_i$ is the set of states where pebble $i$ is the head pebble. For each $i = 1, 2$ we can further partition each $Q_i$ into four sets of states: $Q_{i,\text{stay}}, Q_{i,\text{right}}, Q_{i,\text{place}}, Q_{i,\text{lift}}$, such that for every $i = 1, 2$, $\sigma \in \Sigma$, $V \subseteq \{1, 2\}$, and $q, p \in Q$:

**A4.** If $q \in Q_{i,\text{stay}}$ and $(i, \sigma, V, p) \rightarrow (q, act) \in \mu$, then $act = stay$.

**A5.** If $q \in Q_{i,\text{right}}$ and $(i, \sigma, V, p) \rightarrow (q, act) \in \mu$, then $act = right$.

**A6.** If $q \in Q_{i,\text{place}}$ and $(i, \sigma, V, p) \rightarrow (q, act) \in \mu$, then $act = place$ pebble.

**A7.** If $q \in Q_{i,\text{lift}}$ and $(i, \sigma, V, p) \rightarrow (q, act) \in \mu$, then $act = lift$ pebble.
The non-determinization process itself is a simulation of all possible computation paths of $A$. On an input $w = (\sigma_1) \cdot (\sigma_2) \cdot \ldots (\sigma_n)$, due to branching, the automaton $A$ can be in several states when it reaches a certain position $i$, where $1 \leq i \leq n$. Since the number of states is finite, to simulate the run of $A$ on $w$, it is then sufficient to remember all these states and simulates all possible transitions from these states.

Formally, we define the non-deterministic weak 2-PA $A' = \langle \Sigma, Q', q'_0, \mu', F' \rangle$, where:

- $Q = 2^{Q_2} \cup (2^{Q_2} \times 2^{Q_1})$
- $q'_0 = \{q_0\}$
- $F' = 2^F - \{\phi\}$. 
The set $\mu$ contains the following transitions:

When pebble 2 is the head, for every $S \subseteq Q_2$ and for every $\sigma \in \Sigma$:

If $S$ contains a state $q \in U$, then:

$$(2, \sigma, \emptyset, S) \rightarrow ((S - \{q\}) \cup U_q, stay) \in \mu'$$

where $U_q = \{p | (2, \sigma, \emptyset, q) \rightarrow (p, stay) \in \mu\}$.

If $S$ contains a state $q \in Q_2, stay$ and $S \cap U = \emptyset$, then:

$$(2, \sigma, V, S) \rightarrow ((S - \{q\}) \cup N_q, stay) \in \mu'$$

for every $N_q \subseteq \{p | (i, \sigma, \emptyset, q) \rightarrow (p, stay) \in \mu\}$ and $N_q = \emptyset$. 

Alternating to non-deterministic (3)
Alternating to non-deterministic (4)

If $S$ contains a state $q \in Q_{2,place}$ and $S \cap Q_{2,stay} = \emptyset$, then:

$$(2, \sigma, \emptyset, S) \rightarrow ((S - \{q\}, \{p\}), \text{place pebble}) \in \mu'$$

where $(2, \sigma, \emptyset, q) \rightarrow (p, \text{place pebble}) \in \mu$.

If $S \subseteq Q_{2,right}$ then:

$$(2, \sigma, \emptyset, S) \rightarrow (S', \text{right}) \in \mu'$$

where $S' = \{p \mid (2, \sigma, \emptyset, q) \rightarrow (p, \text{right}) \in \mu \text{ and } q \in S\}$. 
The transitions when pebble 1 is the head are as follows:

For every \( S_2 \subseteq Q_2, \ S_1 \subseteq Q_1, \ V \subseteq \{2\} \) and for every \( \sigma \in \Sigma \):

If \( S_1 \) contains a state \( q \in U \), then:
\[
(1, \sigma, V, (S_2, S_1)) \rightarrow ((S_2, (S_1-{q}) \cup U_q), \text{stay}) \in \mu'
\]
where \( U_q = \{p \mid (1, \sigma, V, q) \rightarrow (p, \text{stay}) \in \mu\} \).

If \( S_1 \) contains a state \( q \in Q_{1\text{,stay}} \) and \( S \cap U = \emptyset \), then:
\[
(1, \sigma, V, (S_2, S_1)) \rightarrow ((S_1, (S_1-{q}) \cup N_q), \text{stay}) \in \mu'
\]
for every \( N_q \subseteq \{p \mid (1, \sigma, V, q) \rightarrow (p, \text{stay}) \in \mu\} \) and \( N_q = \emptyset \).
From here the proof that \( L(A) = L(A') \) is pretty straightforward.
The emptiness problem is **decidable** for weak 2-PA, but, the emptiness problem is **undecidable** for weak 3-PA.

The proof of decidability for weak 2-PA will be presented in the following slides.

The proof of undecidability for the weak 3-PA (the most difficult proof in this lecture in my opinion) will be presented in the end of the presentation.
Proof of emptiness decidability of 2-PA (1)

We will prove that for every weak 2-PA $A$, there exists a one-way alternating 1-RA $A'$ such that $L(A') = L(A)$. Moreover, the construction of $A'$ from $A'$ is effective.

(This theorem is true not only for 2-PA and 1-RA but for every $k$-PA and $(k-1)$-RA)

Now we immediately obtain the decidability of the emptiness problem for weak 2-PA because the same problem for one-way alternating 1-RA is known to be decidable.
Proof of emptiness decidability of 2-PA (2)

First, we normalize the behavior of $A$ in a similar fashion to what we did in previous proofs:

**N1:** Pebble 1 is lifted only after it reads the right-end marker.

**N2:** The automaton can only enter a final state when the control is in pebble 2. Furthermore, it does so only after pebble 2 reads the right-end marker.

**N3:** Immediately after pebble 2 moves right, pebble 1 is placed.

**N4:** Immediately after pebble 1 is lifted, pebble 2 moves right.

After the above normalizations a run of $A$ on a word $w$ can be represented by the following tree:
Proof of emptiness decidability of 2-PA (3)
Proof of emptiness decidability of 2-PA (4)

Now the simulation of $A$ by a one-way alternating 1-RA $A'$ becomes straightforward. The automaton $A'$ is defined as follows.
- The states of $A$ are elements of $Q \cup (Q \times Q)$
- The initial state is $q_0$
- The set of final states is $F \cup \{(p, p): p \in Q\}$.

For each placement of pebble 1 on position $i$, the automaton performs a “Guess–Split–Verify” procedure which will be explained in the following slide.
The Guess-Split-Verify procedure:

1. From the state $q_i$, the automaton $A'$ guesses the state in which pebble 1 is eventually lifted, meaning the state $p_i$. It stores $p_i$ in its internal state and simulates the transition $(2, \sigma_i, \emptyset, \emptyset, q_i) \rightarrow (p_i, i, place pebble) \in \mu$ to enter into the state $(p_{i,i}, p_i)$.

2. The automaton $A'$ splits its computation into two branches.
   In one branch, assuming the guess $p_i$ is correct, $A'$ moves right and enters into the state $q_{i+1}$, simulating the transition $(2, \emptyset, p_i) \rightarrow (q_{i+1}, right)$.
   After this, it recursively performs the Guess–Split–Verify procedure for the next placement of pebble 1 on position $i+1$.
   In the other branch $A'$ stores the data value $d_i$ in its register and simulates the run of pebble 1 on $(\sigma_1, d_1) (\sigma_2, d_2) \ldots (\sigma_n, d_n)$, starting from the state $p_{i,i}$, to verify that the guess $p_i$ is correct.
Proof of emptiness decidability of 2-PA (6)

During the simulation, the states of $A'$ are $(p_{i,i}, p_i), \ldots, (p_{i,n+1}, p_i)$. A accepts when the simulation ends in the state $(p_i, p_i)$, that is, when the guess $p_i$ is “correct.”
Now let’s study the time complexity of three specific problems related to weak 2-PA:

**Emptiness problem.** Given a weak 2-PA $A$, is $L(A) = \emptyset$?

**Labeling problem.** Given a deterministic weak 2-PA $A$ over the labels $\Sigma$ and a sequence of data values $d_1, d_2, ..., d_n \in D^n$, is there a sequence of labels $\sigma_1, \sigma_2, ..., \sigma_n \in \Sigma^n$ such that $(\sigma_1/d_1)(\sigma_2/d_2)...(\sigma_n/d_n) \in L(A)$?

**Data value membership problem.** Given a deterministic weak 2-PA $A$ over the labels $\Sigma$ and a sequence of finite labels $\sigma_1, \sigma_2, ..., \sigma_n \in \Sigma^n$, is there a sequence of data values $d_1, ..., d_n \in D^n$ such that $(\sigma_1/d_1)(\sigma_2/d_2)...(\sigma_n/d_n) \in L(A)$?
The complexity of the Emptiness problem will not be shown in this presentation. As it turns out, the Emptiness problem is not primitive recursive, which intuitively means it doesn’t behave “too nicely”.

The Labeling problem and the Data value membership problem both turn out to be NP-complete, as we will prove in the following slides.
Labeling problem complexity

To show the NP-hardness of the labeling problem, we will define a reduction from graph 3-colorability.

In the following we may assume that the data values are taken from the set of natural numbers. Let $V = \{1, \ldots, n\}$ and $E = \{(i_1, j_1), \ldots, (i_m, j_m)\}$. Assuming that D contains the natural numbers, we take $i_1j_1i_2j_2 \ldots i_mj_m$ as the sequence of data values.

Then, we construct a weak 2-PA $A$ over the alphabet $\Sigma = \{V_R, V_G, V_B\}$ that accepts data words of even length in which the following hold:
- For all odd position $x$, the label on position $x$ is different from the label on position $x + 1$.
- For every two positions $x$ and $y$, if they have the same data value, then they have the same label.

Thus, the graph $G$ is 3-colorable if and only if there exists $\sigma_1, \sigma_2 \ldots \sigma_{2m} \in \{V_R, V_G, V_B\}^*$ such that $
(\sigma_1)_{i_1}(\sigma_2)_{j_1} \ldots (\sigma_{2m-1})_{i_m}(\sigma_{2m})_{j_m} \in L(A)$, and the NP-completeness of the labeling problem follows.
The NP-hardness of data value membership problem can be established in a similar spirit. The reduction is from the following variant of graph 3-colorability, called 3-colorability with constraint. Given a graph $G = (V, E)$ and three integers $n_r, n_g, n_b$, is the graph $G$ 3-colorable with the colors $R, G$ and $B$ such that the numbers of vertices colored with $R, G$ and $B$ are $n_r, n_g, n_b$ respectively?

The reduction to data value membership problem is as follows:
Let $V = \{1, \ldots, n\}$ and $E = \{(i_1, j_1), \ldots, (i_m, j_m)\}$. We define $\Sigma = \{V_R, V_G, V_B, v_1, \ldots, v_n\}$ and take $v_{i_1}v_{j_1} \ldots v_{i_m}v_{j_m}V_R \ldots V_RV_G \ldots V_GV_B \ldots V_B$ as a sequence of finite labels.
Then, we construct a weak 2-PA over $\Sigma$ that accepts data words of the form:

$$(v_{i_1}c_1d_1)(v_{j_1}c_md_m)(v_{R}a_1)(v_{R}a_{n_r})(v_{G}a_{1'})(v_{G}a_{n_g'})(v_{B}a_{1''})(v_{B}a_{n_b''})$$

Where:

- $v_{i_1}, v_{j_1}, \ldots, v_{i_m}, v_{j_m} \in \{v_1, \ldots, v_n\}$
- In the sub-word $(v_{i_1}c_1d_1)(v_{j_1}c_md_m)(v_{j_m}c_md_m)$, every two positions with the same labels have the same data value.
- The data values $a_1 \ldots a_{n_r} a_1' \ldots a_{n_g} a_1'' \ldots a_{n_b}''$ are pairwise different;
- For each $i = 1 \ldots m$ the values $c_i, d_i$ appear among $a_1 \ldots a_{n_r} a_1' \ldots a_{n_g} a_1'' \ldots a_{n_b}''$ such that the following holds:
Data value membership problem complexity (3)

- If $c_i$ appears either among $a_1 \ldots a_{n_r}$ then $d_i$ appears either among $a'_1 \ldots a'_{n_g}$ or $a''_1 \ldots a''_{n_b}$.
- If $c_i$ appears either among $a'_1 \ldots a'_{n_g}$ then $d_i$ appears either among $a_1 \ldots a_{n_r}$ or $a''_1 \ldots a''_{n_b}$.
- If $c_i$ appears either among $a''_1 \ldots a''_{n_b}$ then $d_i$ appears either among $a_1 \ldots a_{n_r}$ or $a'_1 \ldots a'_{n_g}$.

Now the graph $G$ is 3-colorable with the constraints $n_r, n_g, n_b$ if and only if there exists $c_1d_1 \ldots c_md_ma_1 \ldots a_{n_r}a'_1 \ldots a'_{n_g}a''_1 \ldots a''_{n_b}$ such that:

$$(v_{i_1})(v_{j_1}) \ldots (v_{i_m})(v_{j_m})(v_R)(v_R) \ldots (v_G)(v_G) \ldots (v_B)(v_B) \ldots (v_B)$$

is accepted by $A$.

The NP-completeness of the data value membership problem then follows.
Top view weak k-PA

Top view weak PA are weak PA where the equality test is performed only between the data values seen by the last and the second to last placed pebbles. That is, if pebble $i$ is the head pebble, then it can only compare the data value it reads with the data value read by pebble $i + 1$.

We can also define an alternating version of top view weak $k$-PA. However, just like in the case of weak $k$-PA, alternating, non-deterministic and deterministic top view weak $k$-PA have the same recognition power.

Furthermore the emptiness problem, the labeling problem, and the data value membership problem have the same complexity lower bound for top view weak $k$-PA, for each $k = 2, 3, ...$
Theorem: For every top view weak k-PA $A$, there is a one-way alternating 1-RA $A'$ such that $L(A') = L(A)$. Moreover, the construction of $A'$ is effective.

Proof: The proof is similar to the proof of equivalence between a weak 2-PA and a weak 1-RA. Each placement of a pebble is simulated by “Guess–Split–Verify” procedure. Since each pebble $i$ can only compare its data value with the one seen by pebble $i + 1$, $A$ does not need to store the data values seen by pebbles $i + 2, \ldots k$. It only needs to store the data value seen by pebble $i + 1$, thus, one register is sufficient for the simulation.

From this theorem we immediately obtain the decidability of the emptiness problem for top view weak k-PA.
Top view weak PA with unbounded number of pebbles

One can also discuss and formally define a top-view weak PA with an unbounded number of pebbles.

It is straightforward to show that 1-way deterministic 1-RA can be simulated by top view weak unbounded PA: Each time the register automaton changes the content of the register, the top view weak unbounded PA places a new pebble.

Furthermore, top view weak unbounded PA can be simulated by 1-way alternating 1-RA. Each time a pebble is placed, the register automaton performs the procedure “Guess–Split–Verify”. Thus, the emptiness problem for top view unbounded weak PA is still decidable.
As promised, we will now prove that the emptiness problem for weak 3-PA is undecidable.

Consider the following language, where $\Sigma = \{\sigma, $\}$:

$$L_{ord} = \left\{ \left( \sigma \atop a_1 \right) \ldots \left( \sigma \atop a_n \right) \left( $ \atop d \right) \left( \sigma \atop a_1 \right) \ldots \left( \sigma \atop a_n \right) : a_1 \ldots a_n \text{ are pairwise different} \right\}$$

$L_{ord}$ is not accepted by weak 2-PA, but it is accepted by weak 3-PA.

The proof of undecidability of weak 3-PA will be using a reduction from PCP to a language that resembles $L_{ord}$. 
Proof of emptiness undecidability of 3-PA (2)

An instance of PCP is a sequence of pairs \((x_1, y_1) \ldots (x_n, y_n)\), where each \(x_1, y_1 \ldots x_n, y_n \in \{\alpha, \beta\}^*\). This instance has a solution if there exist indexes \(i_1 \ldots i_m \in \{1, \ldots, n\}\) such that \(x_{i_1} \cdots x_{i_m} = y_{i_1} \cdots y_{i_m}\).

Let \(\Sigma = \{1, \ldots, n, \alpha, \beta, $\}\) and \(x_i = v_{i,1} \cdots v_{i,l_i}\), for each \(i = 1, \ldots, n\).

Each string \(x_i\) is encoded as \(Enc(x_i) = (v_{i,1}/a_{i,1}) \ldots (v_{i,l_i}/a_{i,l_i})\) where \(a_{i,1} \ldots a_{i,l_i}\) are pairwise different.
Proof of emptiness undecidability of 3-PA (3)

The string $x_{i_1} \cdots x_{i_m}$ can be encoded as $(i_1 \_b_1) Enc(x_{i_1}) (i_2 \_b_2) Enc(x_{i_1}) \cdots (i_m \_b_m) Enc(x_{i_1})$ where all the data values that appear in it are pairwise different.

Similarly, we encode $y_{j_1} \cdots y_{j_m}$ as $(j_1 \_c_1) Enc(y_{j_1}) (j_2 \_c_2) Enc(y_{j_1}) \cdots (j_m \_c_m) Enc(y_{j_1})$.

We can now define our data word as:

$$w = (i_1 \_b_1) Enc(x_{i_1}) (i_2 \_b_2) Enc(x_{i_1}) \cdots (i_m \_b_m) Enc(x_{i_1}) (\_d \_c_1) Enc(y_{j_1}) (j_2 \_c_2) Enc(y_{j_1}) \cdots (j_m \_c_m) Enc(y_{j_1}).$$
Proof of emptiness undecidability of 3-PA (4)

$w$ constitutes a solution to the instance of PCP if and only if:

1. $i_1 \ldots i_m = j_1 \ldots j_m$
2. $\text{Proj}_\Sigma \left( \text{Enc}(x_{i_1}) \ldots \text{Enc}(x_{i_m}) \right) = \text{Proj}_\Sigma \left( \text{Enc}(y_{j_1}) \ldots \text{Enc}(y_{j_l}) \right)$

In order to check such properties with weak 3-PA, we demand the following additional criteria:

3. $b_1 \ldots b_m = c_1 \ldots c_l$
4. $\text{Proj}_D \left( \text{Enc}(x_{i_1}) \ldots \text{Enc}(x_{i_m}) \right) = \text{Proj}_D \left( \text{Enc}(y_{j_1}) \ldots \text{Enc}(y_{j_l}) \right)$
5. For any 2 positions $h_1$ and $h_2$ where $h_1$ is to the left of the delimiter ($\$_{d}$) and $h_2$ is to its right: if both of them has the same data value then both of them are labeled with the same label.
Proof of emptiness undecidability of 3-PA (5)

Criteria 3,4,5 imply equalities 1 and 2. (note that criteria 3,4 resemble the language $L_{ord}$).

Because the data values that appear in $Proj_D (Enc(x_{i_1}) ... Enc(x_{i_m}))$ are pairwise different, all of them are checkable by three pebbles in the weak manner. For example, to check criterion 3, the automaton does the following:

- Check that $b_1 = c_1$.
- Check that for each $i = 1, \ldots, m - 1$, there exists $j$ such that $a_i a_{i+1} = b_j b_{j+1}$.
- Finally, check that $b_m = c_i$.

Criteria 4,5 can be checked similarly.

Now the reduction is complete and we have proven that the emptiness problem for weak 3-PA is undecidable.