FINITE STATE MACHINES
FOR STRINGS OVER INFINITE ALPHABETS

DANIEL OHAYON
Danielohayon444@gmail.com
FIRST SOME DEFINITIONS

- $D$ is an infinite alphabet.
- A $D$-string $w$ is a finite sequence $d_1, d_2, \ldots, d_n$ where each $d_i \in D$.
- $\text{dom}(w) = \{1, \ldots, |w|\}$ \hspace{1cm} $\text{val}_w(i) = d_i$
- Two special symbols to represent the beginning and end of a string are $\triangleright, \triangleleft \notin D$. And thus the extended string is $w \triangleright v \triangleleft$ where $v \in D^*$
- $\text{dom}^+(w) = \{0, \ldots, |w| + 1\}$
- $\text{val}_w(0) = \triangleright, \text{val}_w(|w| + 1) = \triangleleft$
FINITE MEMORY AUTOMATA

- AKA Register Automate
- Finite state machine
- Finite number of registers
- Each register holds a value from $D$
- When processing a string, it compares the symbol on the current position with values in the registers.
FINITE MEMORY AUTOMATA

- Formal Definition of Finite Memory Automata.
- A nondeterministic two-way $k$-register automation $B$ over $D$ is a tuple $(Q, q_0, F, \tau_0, P)$ where
- $Q$ is a finite set of states.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final states.
- $\tau_0 : \{1, \ldots, k\} \rightarrow D \cup \{\triangleright, \triangleleft\}$ is the initial register assignment.
- $P$ is a finite set of transitions of the forms $(i, q) \rightarrow (q', d)$ or $q \rightarrow (q', i, d)$. Here $i \in \{1, \ldots, k\}$, $q, q' \in Q$, $d \in \{\text{stay, left, right}\}$.
- If there are no left-transitions then the automata is one-way.
CONFIGURATION

• Given a string $w$, a configuration of $B$ on $w$ is a tuple $[j, q, \tau]$ where $j \in \text{dom}^+(w)$, $q \in Q$, $\tau: \{1, ..., k\} \to D \cup \{\triangleright, \triangleleft\}$.

• Initial configuration: $\gamma_0 := [1, q_0, \tau_0]$

• $\gamma = [j, q, \tau] \vdash \gamma' = [j', q', \tau']$ if there is a transition $(i, q) \to (q', d)$ s.t. $\tau = \tau'$, $j = j'$ or $j' = j - 1$ or $j' = j + 1$ according to $d$ or there is a transition $q \to (q', i, d)$ s.t. $j = j'$ or $j' = j - 1$ or $j' = j + 1$ according to $d$ and $\tau'$ is obtained from $\tau$ by setting $\tau'(i)$ to $\text{val}_w(j)$.

• We denote the transitive closure of $\vdash$ by $\vdash^*$.

• A string $w$ is accepted by $B$ if $\exists \gamma$ s.t. $\gamma_0 \vdash^* \gamma$, and $\gamma$ is an accepting configuration.

• An automation is deterministic if in each configuration at most one transition applies.
PEBBLE AUTOMATA

• Pebbles are numbered from 1 to \( k \) and pebble \( i + 1 \) can only be placed when pebble \( i \) is present on the string.
• Like wise pebble \( i \) can only be lifted when pebble \( i + 1 \) is not present.
• The highest numbered pebble present on the string acts like the head of the automation.
PEBBLE AUTOMATA

- Formal Definition: A nondeterministic two-way k-pebble automation $A$ over $D$ is a tuple $(Q, q_0, F, T)$ where

  - $Q$ is a finite set of states. $q_0$ is the initial state. $F \subseteq Q$ is the set of final states
  - $T$ is a finite set of transitions of the form $\alpha \rightarrow \beta$ where
    - $\alpha$ is of the form $(i, s, P, V, q)$ or $(i, P, V, q)$ where $i \in \{1, \ldots, k\}$, $s \in D \cup \{\triangleright, \triangleleft\}$
      and $P, V \subseteq \{1, \ldots, i - 1\}$
    - $\beta$ is of the form $(q, d)$ with $q \in Q$
      and $d \in \{\text{stay}, \text{left}, \text{right}, \text{place} - \text{pebble}, \text{lift} - \text{pebble}\}$
PEBBLE AUTOMATA CONFIGURATION

• Given a string \( w \), a configuration of \( A \) on \( w \) is of the form \( \gamma = [i, q, \theta] \) where \( i \in \{1, \ldots, k\} \), \( q \in Q \), \( \theta : \{1, \ldots, i\} \to \text{dom}^+(w) \). We call \( \theta \) a pebble assignment. And \( i \) the depth of the configuration also denoted as \( \text{depth}(\gamma) \).

• The initial configuration is \( \gamma_0 = [1, q_0, \theta_0] \) where \( \theta_0(1) = 0 \).

• A transition \((i, s, P, V, p) \to \beta\) applies to a configuration \( \gamma = [j, q, \theta] \) if
  1) \( i = j \), \( p = q \)
  2) \( V = \{l < i \mid \text{val}_w(\theta(l)) = \text{val}_w(\theta(i))\} \)
  3) \( P = \{l < i \mid \theta(l) = \theta(i)\} \) and
  4) \( \text{val}_w(\theta(i)) = s \)

• A transition \((i, P, V, q) \to \beta\) applies to \( \gamma \) if (1) – (3) hold and no transition \((i, s, P, V, q) \to \beta\) applies.
PEBBLE AUTOMATA TRANSITION RELATION

- We define the transition relation $\gamma = [i, q, \theta] \vdash \gamma' = [i', q', \theta']$ as follows: iff there is a transition $\alpha \rightarrow (p, d)$ that applies to $\gamma$ such that $q' = p$ and $\theta'(j) = \theta(j)$ for all $j < i$ 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 $d = \text{place – pebble}$ then $i' = i + 1$ and $\theta'(i + 1) = \theta'(i) = \theta(i)$
  - if $d = \text{lift – pebble}$ then $i' = i - 1$

- When a PA lifts pebble $i$ the control is transferred to pebble $i - 1$ there for even a $1D - PA$ can make several left to right sweeps.
PEBBLE AUTOMATA EXAMPLE

• $L = (d_1, ..., d_n | n \geq 0, \exists i, j \text{ s.t. } i \neq j \text{ and } d_i = d_j)$

• Our pebble automata $A = (Q, q_1, F, T)$ is a two pebble automata and defined as follows.

• $Q = \{q_1, q_2, q_\rightarrow, q_{acc}\}$, $F = \{q_{acc}\}$ and $T$ consists of the following transitions:
  - (1) $(1, \emptyset, \emptyset, q_1) \rightarrow (q_1, \text{Right})$
  - (2) $(1, \emptyset, \emptyset, q_1) \rightarrow (q_\rightarrow, \text{Place - Pebble})$
  - (3) $(2, \{1\}, \{1\}, q_\rightarrow) \rightarrow (q_2, \text{Right})$
  - (4) $(2, \emptyset, \emptyset, q_2) \rightarrow (q_2, \text{Right})$
  - (5) $(2, \emptyset, \{1\}, q_2) \rightarrow (q_{acc}, \text{Stay})$.

• (1) – move right, (2) – decides to place a pebble, (3) – after the pebble is placed $A$ moves to the right. (4) – continue moving right, (5) – if $A$ sees a symbol equal to the symbol under the first pebble it moves to the final state.
We will now describe a $2N - PA$ $A$ that accepts all words $w$ where $w$ is of length at least two and there exists a position $i$ where the set of symbols occurring before $i$ is disjoint from the set of symbols after $i$. For example $abb \in L(A)$ and $abab \notin L(A)$.

$A = (Q, q_0, F, T)$ where $Q = \{q_0, \ldots, q_7\}$ and $F = \{q_5\}$ and $T$. 
PEBBLE AUTOMATA EXAMPLE

• $A = (Q, q_0, F, T)$ where $Q = \{q_0, ... , q_7\}$ and $F = \{q_5\}$ and $T$ is:

• $(1, \triangleright, \emptyset, \emptyset, q_0) \rightarrow (q_0, \text{right})$
  $(1, \emptyset, \emptyset, q_0) \rightarrow (q_1, \text{right})$
  $(1, \lt, \emptyset, \emptyset, q_0) \rightarrow (q_2, \text{stay})$
  $(1, \emptyset, \emptyset, q_1) \rightarrow (q_1, \text{right})$
  $(1, \emptyset, \emptyset, q_1) \rightarrow (q_3, \text{place – pebble})$

• $(2, \{1\}, \{1\}, q_3) \rightarrow (q_4, \text{left})$
  $(2, \emptyset, \emptyset, q_3) \rightarrow (q_4, \text{left})$
  $(2, \triangleright, \emptyset, \emptyset, q_4) \rightarrow (q_5, \text{stay})$
  $(2, \emptyset, \emptyset, q_4) \rightarrow (q_6, \text{place – pebble})$

• $(3, \{2\}, \{2\}, q_6) \rightarrow (q_6, \text{right})$
  $(3, \emptyset, \emptyset, q_6) \rightarrow (q_6, \text{right})$
  $(3, \emptyset, \{2\}, q_6) \rightarrow (q_6, \text{right})$
  $(3, \{1\}, \{1\}, q_6) \rightarrow (q_7, \text{right})$
  $(3, \emptyset, \{1\}, q_7) \rightarrow (q_7, \text{right})$
  $(3, \emptyset, \emptyset, q_7) \rightarrow (q_4, \text{right})$
  $(2, \lt, \emptyset, \emptyset, q_7) \rightarrow (q_3, \text{lift – pebble})$
• Kaminski and Francez already showed that emptiness of $1N − RA$ is decidable.

• And that for a two $1N − RA; A, B$ with two registers it’s decidable whether $L(A) \subseteq L(B)$.

• We will now show that universality of $1N − RA$ is undecidable. This implies that equivalence (and hence containment) for arbitrary $1N − RAs$ is undecidable.

• In our proofs we use a reduction from the Post Correspondence Problem (PCP) which is well known to be undecidable.
POST CORRESPONDENCE PROBLEM (PCP)

• An instance of PCP is a sequence of pairs \((x_1, y_1), \ldots, (x_n, y_n)\), where \(x_i, y_i \in \{a, b\}^*\) for \(i = 1, \ldots, n\). This instance has a solution if there are \(m \in \mathbb{N}\) and \(\alpha_1, \ldots, \alpha_m \in \{1, \ldots, n\}\) such that \(x_{\alpha_1} \cdots x_{\alpha_m} = y_{\alpha_1} \cdots y_{\alpha_m}\).

• For example for the pairs \((a, baa), (ab, aa), (bba, bb)\) a solution to this problem would be \((3, 2, 3, 1)\) because:

\[
x_3x_1x_3x_2 = bba \cdot ab \cdot bba \cdot a = bbaabbbbaa = bb \cdot aa \cdot bb \cdot baa = y_3y_1y_3y_2
\]

• The PCP problem asks whether a given instance of the problem has a solution.
PCP ENCODING

• The input string is \( w = u\#v \) where \# is a delimiter and \( u \) and \( v \) are strings representing candidate solution \((x_{\alpha_1}, \ldots, x_{\alpha_m}; y_{\beta_1}, \ldots, y_{\beta_l})\). A candidate is a solution if:
  
  (1) \( l = m \) and for each \( i, \alpha_i = \beta_i \)

  (2) both strings are the same meaning the corresponding positions \( x_{\alpha_1}, \ldots, x_{\alpha_m} \) and \( y_{\beta_1}, \ldots, y_{\beta_l} \) carry the same symbol.

• Each item \( x_{\alpha_j} \) is encoded as a string of the form \( \&\gamma\alpha_j\delta_1a_1\ldots\delta_ka_k \). To achieve uniqueness all \( \gamma \) and \( \delta \) values occur only once in \( u \).

• A string \( w = u\#v \) is correct if it’s syntactically correct and
  
  (1) for each \( \gamma \) the number to the right of \( \gamma \) is the same in \( u \) and \( v \).

  (2) for each \( \delta \) the symbol to the right of \( \delta \) is the same in \( u \) and \( v \).
UNDECIDABILITY OF UNIVERSALITY OF A $1N - RA$

- We will describe a $1N - RA A$ that only accepts a string $w$ iff it is not syntactically correct or does not represent a solution. Hence $A$ accepts all inputs $\iff$ the PCP instance has no solution. The following automation tries to guess an error in the encoding represented by $w$.

- (1) Is of the wrong form.
  
  (a) $w$ is not of the form $w = u\#v$ or $u$ or $v$ is not of the form $(\&\gamma a_j \delta_1 a_1 ... \delta_k a_k)^*$
  
  (b) $x_i \neq a_1 ... a_k$ in some entry $\&\gamma a_j \delta_1 a_1 ... \delta_k a_k$ of $u$ or $v$

- (2) The $\gamma$ projections are wrong.
  
  (a) The first or last $\gamma$ in $u$ differs from the first $\gamma$ in $v$.
  
  (b) Two $\gamma'$s in $u$ or $v$ are the same.
  
  (c) $\gamma_1$ and $\gamma_2$ are successors in $u$ but not in $v$. 
UNDECIDABILITY OF UNIVERSALITY OF A $1N - RA$

- (3) The $\delta$-projections are wrong.
- (4) $w$ is syntactically correct but does not represent a solution
  
  (a) The $\alpha$ value for some $\gamma$ in $u$ is different from the corresponding $\beta$ value in $v$.
  
  (b) The symbol after some $\delta$ in $u$ is different from the corresponding symbol in $v$.

- Clearly $w$ is not a solution if one of these conditions holds. In addition each single condition is easy to check and the class of languages definable by $1N - RAs$ is closed under union.

- Hence if universality of a $1N - RA$ like the one we just built was decidable then the PCP problem would be decidable and that’s a contradiction.
$L(A) = \emptyset$ IS UNDECIDABLE FOR WEAK 1D $- PA$

- **Proof:** The weak $1D - PA$ $A$ first checks whether the input is of the desired form. Then it accepts only if the input encodes a solution of the PCP instance.
- As pebbles can only be moved to the right we keep the first pebble on the first position. $A$ operates as follows:
- (1) Checks whether $u$ and $v$ are of the form $(\&\gamma \alpha_j \delta_1 a_1 \ldots \delta_k a_k)^*$ and that $x_i = a_1 \ldots a_k$ for each $\&\gamma \alpha_j \delta_1 a_1 \ldots \delta_k a_k$ in $u$ and in $v$ respectively.
- (2) Checks that $w$ is syntactically correct by
  (a) All $\gamma$’s in $u$ and $v$ are different
  (b) The first and last $\gamma$ in $u$ equals the first and last $\gamma$ in $v$ respectively.
  (c) If $\gamma_1$ and $\gamma_2$ are successors in $u$ then they are successors in $v$.
  (c) Verify that the $\delta$’s also form an index in an analogous way.
(3) To check that $w$ also represents a solution of the PCP instance:

(a) The $\alpha$ value for some $\gamma$ in $u$ is equal to the corresponding $\beta$ value in $v$.

(b) The symbol after some $\delta$ in $u$ is equal to the corresponding symbol in $v$.

• Hence the PCP instance has a solution iff $L(A)$ is non empty.
A string $w$ is represented by the logical structure with domain $\text{dom}^+(w)$.

The natural ordering $<$ on the domain, and a function $\text{val}: \text{dom}(w) \to D$.

An atomic formula is of the form $x < y, \text{val}(x) = \text{val}(y), \text{or } \text{val}(x) = d$ for $d \in D \cup \{\geq, <\}$.

The logic $MSO^*$ is obtained by adding quantification over unary predicates on $\text{dom}^+(w)$ and no quantification over $D$ is allowed.

A sentence $\varphi$ defines a set of strings via $L(\varphi) := \{w \in D | < w \models \varphi\}$.

As an example $\forall x \forall y (x \neq y \rightarrow \text{val}(x) \neq \text{val}(y))$
• For every formula in $MSO^*$ (Monadic second order logic) there is a finite automation that accepts the same language. Thus every language expressed by a $MSO^*$ formula is regular.

• Example:

$$\exists x, y \ s.t. \ x < y \land \text{val}(x) = a \land \text{val}(y) = b.$$
2D − RA ⊈ MSO*

- Proof: Consider a string \( w = u\#v \) where \( u, v \in (D - \{\#\})^* \). Define \( N_u \) and \( N_v \) as the set of symbols in \( u, v \). Denote by \( n_u \) and \( n_v \) their cardinalities.

- We will show a 2D − RA that accepts \( w = u\#v \) iff \( n_u = n_v \) while there is no such MSO* sentence.

- Definition: \( lmo_w(d) \) is the index of the leftmost occurrence of \( d \) in \( w \).

- Suppose \( N_u = \{a_1, \ldots, a_n\} \) and \( N_v = \{b_1, \ldots, b_m\} \) where \( n = n_u, m = n_v \) and for every \( i < j: lmo_u(a_i) < lmo_u(a_j) \) and in \( u \).

- To check if \( w \) is valid \( A \) starts by visiting \( lmo_u(a_1), lmo_v(b_1), \ldots, lmo_u(a_n), lmo_v(b_m) \).

- If \( a_n \) and \( b_m \) are not reached simultaneously it rejects.
AUTOMAT DESCRIPTION

• It remains to explain how the $2D - RA$ can visit $lmo_u(a_1), lmo_v(b_1), \ldots, lmo_u(a_n), lmo_v(b_m)$ in sequence.

• Clearly $lmo_u(a_1), lmo_v(b_1)$ are the first positions of $u$ and $v$, respectively.

• If $A$ has the values of $a_i$ and $b_i$ stored in its registers it can compute $a_{i+1}$ and $b_{i+1}$.

• It first moves its head to position $lmo_w(a_i)$.

• Now it tests, for all positions $lmo_w(a_i) + j, j > 0$, starting with $j = 1$, whether they carry a leftmost occurrence of a symbol $d$. 
AUTOMAT DESCRIPTION

• it stores $d$ in a register, and goes from $lmo_w(a_i) + j$ to the left until it either sees a $d$ or reaches the left end of the string.
  (1) It finds an occurrence of $d$ then it moves to check $lmo_w(a_i) + j + 1$.
  (2) It doesn't find an occurrence of $d$ then $a_{i+1}$ is defined.

• Now we show that there is no $MSO^*$ sentence.

• If we assume there is one $\varphi^*$. Let $C$ be the set of $D$-symbols occurring in $\varphi^*$.

• Suppose that $u \# v$ is admissible iff
  (1) $Nu \cap Nv = \emptyset$.
  (2) Each $D$-symbol occurs at most once in $u$ or $v$.
  (3) No symbol in $C$ occurs in $u$ or $v$. 
Let $\varphi$ be obtained from $\varphi^*$ by replacing each occurrence of $\text{val}(x) = \text{val}(y)$ by $x = y$, and every occurrence of $\text{val}(x) = d$ s.t. $d \in C$ by $false$, if $d \neq \#$.

For every admissible string $d_1 \cdots d_n e_1 \cdots e_m$, $a_n \# a_m \models \varphi$

iff $d_1 \cdots d_n \# e_1 \cdots e_m \models \varphi$ iff $d_1 \cdots d_n \# e_1 \cdots e_m \models \varphi^*$ iff $n = m$.

Hence, $\{a^n \# a^n \mid n \in N\}$ would be MSO-definable and therefore regular. This leads to the desired contradiction.
$1N – RA \subseteq MSO^*$

- Proof: Let $B = (Q, q_0, F, \tau_0, T)$ be a $1N – RA$. Assume for the moment that no transition of $B$ encodes a stay-move.

- We describe the construction of an MSO* formula that accepts $w$ iff $B$ accepts $w = w_1, ..., w_n$.

- First of all, $\varphi$ guesses, for each position $i$ of $w$, the state that $B$ is in after reading $w_i$.

  - $i \in S_q$ means that the state of $B$ after reading $w_i$ is $q$.

- Next, $\varphi$ guesses, again for each position $i$, which transition $B$ applies.

  - $i \in T_t$ means that transition $t$ is applied when $B$ reads $w_i$.
Now assume that there is an accepting computation of $B$ on $w$ and that $(S_q)_{q \in Q}$ and $(T_t)_{t \in T}$ are chosen accordingly.

The register $j$ content of $B$ before reading position $i$ can be determined as follows:

1. It is either determined from the initial register assignment $\tau_0$
2. It is the symbol $w_l$, where $l = \max\{m < i \mid m \in T_t \text{ where } t \text{ is of the form } q \rightarrow (q', j, d)\}$

It is straightforward to express this in $MSO^*$

With the ability to determine register contents it is now easy to check that $(S_q)_{q \in Q}$ and $(T_t)_{t \in T}$ are consistent with the transition relation of $B$. 

$1N - RA \subseteq MSO^*$
It remains to describe how we can deal with stay-transitions

Let \( r \) be the number of transitions

Call stay-transitions of the form \( (i, q) \rightarrow (q', stay) \) and \( p \rightarrow (p', i, stay) \), type\(-1\) and type\(-2\) transitions,

A type\(-2\) transition can only be followed by type\(-1\) transitions

A consecutive sequence of more than \( r \) type\(-1\) transitions always contains a cycle \( \Rightarrow \) this sequence can be reduced to one containing less than \( r + 1 \) type\(-1\) transitions

\[ 1N − RA \subseteq MSO^* \]
So the construction sketched above can take stay-transitions into account by quantifying over sets $T_{\bar{t}}$, as opposed to sets $T_t$, where $\bar{t}$ is a sequence of at most $r$ transitions.

The latter information can then be used to check the register content.

Consequently, consistency with the transition relation can be enforced.