Manipulating and simplifying REs

- These techniques are based on the equivalence of languages REs define.
- The meaning of every RE is a language, i.e, it is a set.

Identities for union of REs

- Union is commutative, associative, and idempotent:

  commutative: $A + B = B + A$
  associative: $(A + B) + C = A + (B + C)$
  idempotent: $A + A = A$

- $\emptyset$ is the identity for union: $A + \emptyset = \emptyset + A = A$
- If $L(B) \subseteq L(A)$, then $A + B = A$. E.g.: $a^* + aa = a^*$, since $L(aa) \subseteq L(a^*)$

Identities for concatenation

- Concatenation is associative: $(AB)C = A(BC)$.
- ϵ is the identity for concatenation: $A\epsilon = \epsilon A = A$.
- $\emptyset$ is *zero* for concatenation: $A\emptyset = \emptyset A = \emptyset$.
- Concatenation distributes over union:

$$(A + B)C = (AC) + (BC)$$
$$C(A + B) = (CA) + (CB)$$

Identities for Kleene star

- $\emptyset^* = \epsilon, \epsilon^* = \epsilon$
- $(A^*)^* = A^*$: Notice $L((A^*)^*)$ contains concatenations of A^*, which are in turn concatenations of $L(A)$, i.e., A^*.
- If, $L(A^*) \subseteq L(B^*)$, then $A^*B^* = B^*$, e.g.:

$$A^*A^* = A^*, \qquad \text{since } L(A^*) \subseteq L(A^*)$$
$$A^*(A+B)^* = (A+B)^*, \qquad \text{since } L(A^*) \subseteq L((A+B)^*)$$

- Similarly, if $L(B^*) \subseteq L(A^*)$, then $A^*B^* = A^*$.
- $(A+B)^* = (A^*B^*)^*$.
- If $L(B) \subseteq L(A^*)$, then $(A+B)^* = A^*$, e.g., $(a+\epsilon)^* = a^*$, since $\{\epsilon\} \subseteq L(a^*)$.

Simplifying an expression

$$
\begin{array}{lllllll}
 & ((a^* + \emptyset)^* & +aa) & (b+ & bb)^* & b^* & ((a+b)^*b^* + ab)^* \quad \{L(\emptyset) \subseteq L(a^*)\} \\
= & ((a^*)^* & +aa) & (b+ & bb)^* & b^* & ((a+b)^*b^* + ab)^* \\
= & (a^* & +aa) & (b+ & bb)^* & b^* & ((a+b)^*b^* + ab)^* \quad \{L(aa) \subseteq L(a^*)\} \\
= & a^* & & (b+ & bb)^* & b^* & ((a+b)^*b^* + ab)^* \quad \{L(bb) \subseteq L(b^*)\} \\
= & a^* & & b^* & & b^* & ((a+b)^*b^* + ab)^* \\
= & a^* & & b^* & & & ((a+b)^*b^* + ab)^* \quad \{L(b^*) \subseteq L((a+b)^*)\} \\
= & a^* & & b^* & & & ((a+b)^* + ab)^* \quad \{L(ab) \subseteq L((a+b)^*)\} \\
= & a^* & & b^* & & & ((a+b)^*)^* \\
= & a^* & & b^* & & & (a+b)^* \quad \{L(b^*) \subseteq L((a+b)^*)\} \\
= & a^* & & & & & (a+b)^* \quad \{L(a^*) \subseteq L((a+b)^*)\} \\
= & & & & & & (a+b)^*
\end{array}
$$

Properties of Regular Languages

- *Regular and Non Regular languages*: The language a^*b^* is regular. The language $\{a^n b^n Ln \geq 0\}$ is not regular. (intuition: it is not possible, given a finite number of states, to count the a's and then compare that count with the # of b's).
  - How can we show that a language is regular?
  - How can we show that a language is not regular?
- *Closure properties*: can be used to help us prove that a language is not regular.

Showing that a language is regular

Every finite language L is regular. Proof:

- If $L = \emptyset$, then it is defined by the RE $\emptyset$. So it is regular.
- If L is a finite set of strings $\{s_1, s_2, \cdots, s_n\}$, then it is defined by the RE $s_1 + s_2 + \cdots + s_n$. So it is regular.

And for infinite languages, we have developed 4 techniques for showing that a regular language L is regular:

- exhibit a DFA for L,
- exhibit an NFA for L,
- exhibit an ϵ-NFAfor L, or
- exhibit an RE for L.

Showing that a language is not regular

- How can we show that there are no RL descriptors for a given language L?
- It is not sufficient to argue that we tried to find a descriptor and we failed. Perhaps we did not look in the right place...
- We need a technique that does not rely on our cleverness (or lack of it).
- And this technique relies on the following observation: every RL L can be accepted by FA M with a finite number of states.
- If L is infinite, then there must be at least one loop in M.
- And sufficiently long strings in L consist of one or more repeating patterns (recognized by the loop in M).

The Pumping Lemma, formally

Let L be regular. Then $\exists n, \forall w \in L : |w| \geq n \Rightarrow w = xyz$ such that

- $y \neq \epsilon$
- $|xy| \leq n$
- $\forall k \geq 0, xy^k z \in L$

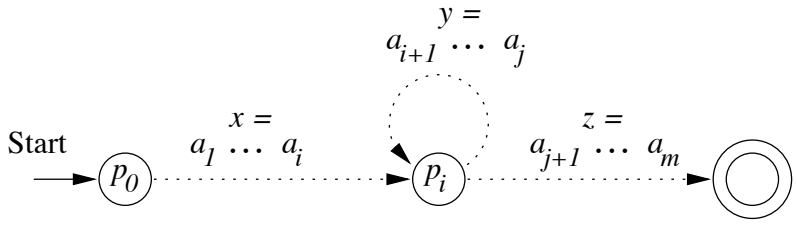
That is, we can always find a string y that can be either deleted or "pumped", i.e., repeated any number of times, resulting in a string in L.

Proving the pumping lemma

Suppose L is regular. Then L is recognized by some DFA A with, say n states.

- Let $w = a_1 a_2 \cdots a_m \in L, m \geq n$.
- Let $p_i = \hat{\delta}(q_0, a_1 a_2 \cdots a_i)$, i.e., p_i is the state A is in after reading i symbols of w.
- Consider what happens if the DFA receives the input $w = xy^k z$, where

$$x = a_1 a_2 \cdots a_i$$
$$y = a_{i+1} a_{i+2} \cdots a_j$$
$$z = a_{j+1} a_{j+2} \cdots a_m$$



Clearly $xy^k z \in L$, for any $k \geq 0$.

How to apply the pumping lemma

- You assume the language L in question is regular.
- You consider a certain pumping length p. (Do not bind p to a certain number. I.e., don't say "Let $p = 3$", for instance. Leave it general.)
- You find a string $|w| \geq p$ in the language. (Pick a string that you think you will not be able to pump).
- Split the string $w = xyz$, so that $|y| > 0$ and $|xy| \leq p$.
- Show that there is no possible split of w in which $\forall i \geq 0$, $xy^iz \in L$. In other words, for every split of w there will be a value of i where $xy^iz \notin L$.

Pumping Lemma: example

Proving that a language is not a regular language.
Example: Let $L = \{0^n 1^n : n \geq 1\}$. Show that L is not regular.

- Assume L is regular. Try to prove that the PL holds for a certain pumping length p.
- Choose $w = 0^p 1^p$. (Hence satisfying the conditions $w \in L$ and $|w| \geq p$).
- Split w into $w = xyz$ so that $|xy| \leq p$ and $|y| > 0$.
- If y contain only zeros, or straddles the middle point, or contains only 1s, then by pumping y we'll obtain a string not in L.
- Therefore there is a contradiction. L is not regular.

Some Closure Properties of Regular Languages

Let L and M be regular languages. Then the following languages are all regular:

- Union: $L \cup M$
- Intersection: $L \cap M$
- Complement: $\overline{N}$
- Difference: $L - M$
- Reversal: $L^R = \{w^R : w \in L\}$
- Kleene star: L^*
- Concatenation: $L.M$

Closure properties (cnt.)

- Union:

  Theorem: For any regular languages R and M, $R \cup M$ is regular.

  Proof: Let $R = L(E)$ and $M = L(F)$. Then $L(E + F) = L \cup M$ by definition.

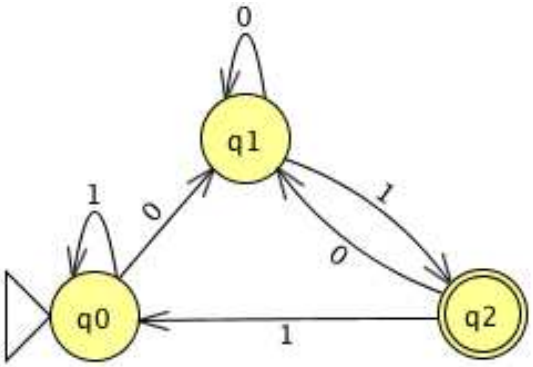
- Complement:

  Theorem: If R is a regular language over Σ, then so is $\overline{R} = \Sigma^* - R$.

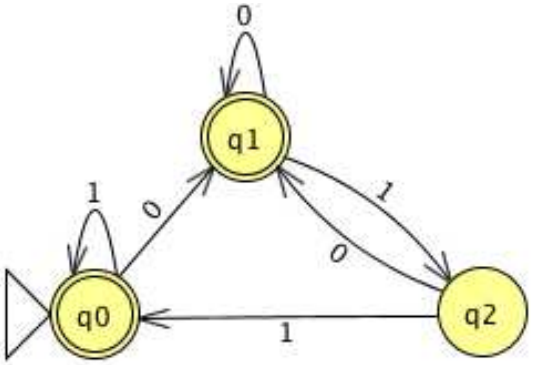
  Proof: Let R be recognized by a DFA $A = (Q, \Sigma, \delta, q_0, F)$. Let $B = (Q, \Sigma, \delta, q_0, Q - F)$. Now $L(B) = \overline{R}$.

Example

Let L be recognized by the DFA below



$\overline{L}$ is recognized by



Closure properties (cont.)

- Intersection:

Theorem: If L and M are regular, then so is $L \cap M$.

Proof 1: By DeMorgan's law $L \cap M = \overline{\overline{L} \cup \overline{M}}$. We already know that regular languages are closed under complement and union.

Proof 2: Let L be the language of two DFAs

$$A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$$

and M be the language of

$$A_M = (Q_M, \delta, \delta_M, q_M, F_M)$$

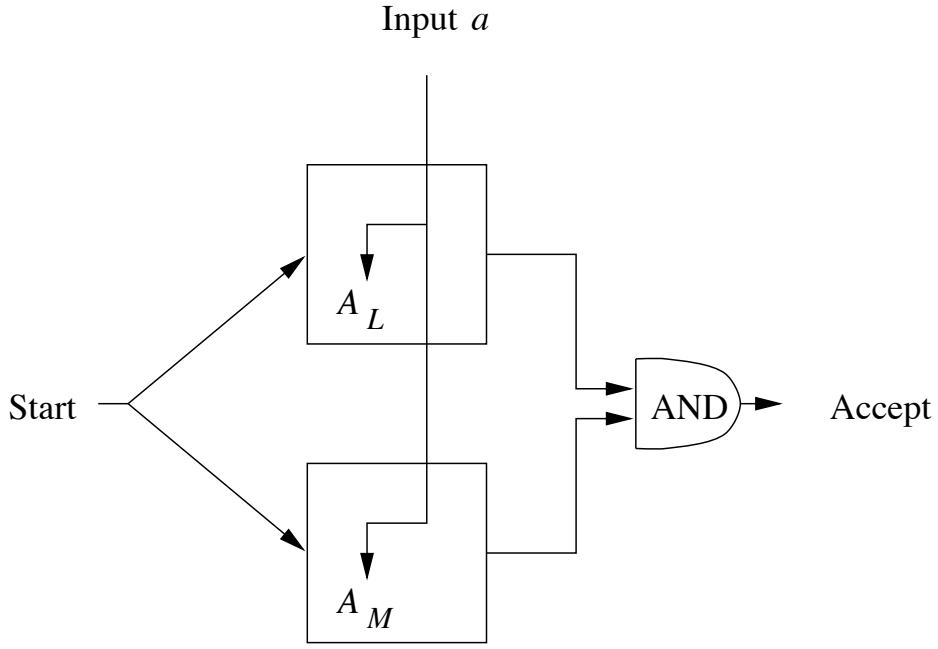
We construct an automaton that simulates A_L and A_M in parallel.

Proof idea (cnt.)

If A_L goes from state p to state s on reading a, and A_M goes from state q to state t on reading a, then $A_L \cap M$ will go from state (p, q) to state (s, t) on reading a.

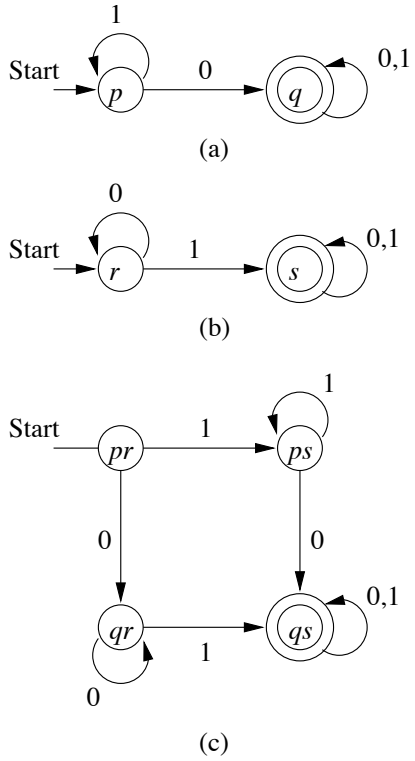
$$A = (Q_L \times Q_M, \Sigma, \delta, (q_L, q_M), F_L \times F_M)$$

where $\delta((p, q), a) = (\delta(p, a), \delta(q, a))$



Example

DFA (c) recognizes the intersection of the languages recognized by DFAs (a) and (b). To see why, obtain the languages that DFAs (a) and (b) recognizes, and verify that (c) recognizes all those strings in the intersection of those languages.



Closure properties (cnt.)

- Difference: **Theorem:** If L and M are regular languages, then so is $L - M$.

  Proof: Observe that $L - M = L \cap \overline{M}$. We already know that regular languages are closed under complement and intersection.

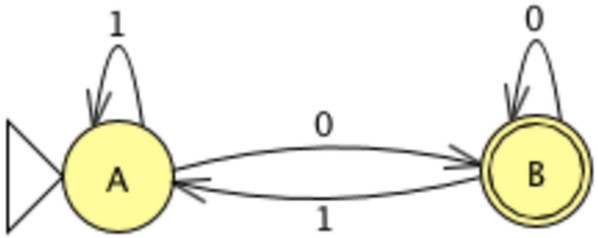
- Reverse: **Theorem:** If L is a regular language, then so is L^R.

  Proof: Let L be recognized by an FA A. Turn A into an FA for L^R, by
  1. Reversing all arcs.
  2. Make the old start state the new sole accepting state.
  3. Create a new start state p_0, with $\delta(p_0, \epsilon) = F$ (the old accepting states).

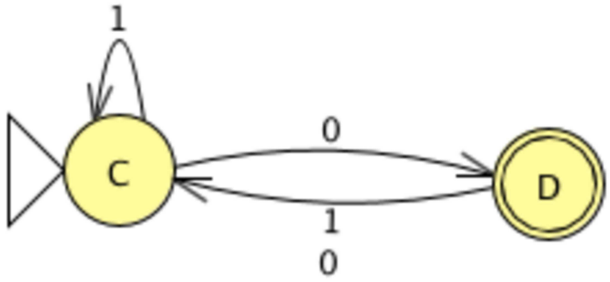
Exercise

Given the automata below, prove that $L(P) - L(Q)$ is a regular language by showing an automaton that recognizes it.

Automaton P



Automaton Q



Decision procedures/algorithms for REs

- *Equality of two descriptors*: Are two descriptors of regular languages equivalent?
- *Minimality*: is a given DFA minimal?

Equality of two descriptors

- You can test equality by hand applying algebraic properties of RLs.
- Now we shall learn how to (computationally) test whether two descriptors for regular languages are equivalent, in the sense that they define the same language.
- The plan is as follows: Suppose languages L and M are defined by two descriptors, say, a RE and an ϵ-NFA.
  1. Convert each to a DFA, say D and E.
  2. Create a new automaton $C = D \cup E$.
  3. Test if the "start states" of C are equivalent. If they are, then the descriptors define the same language.

When two states are equivalent in a DFA

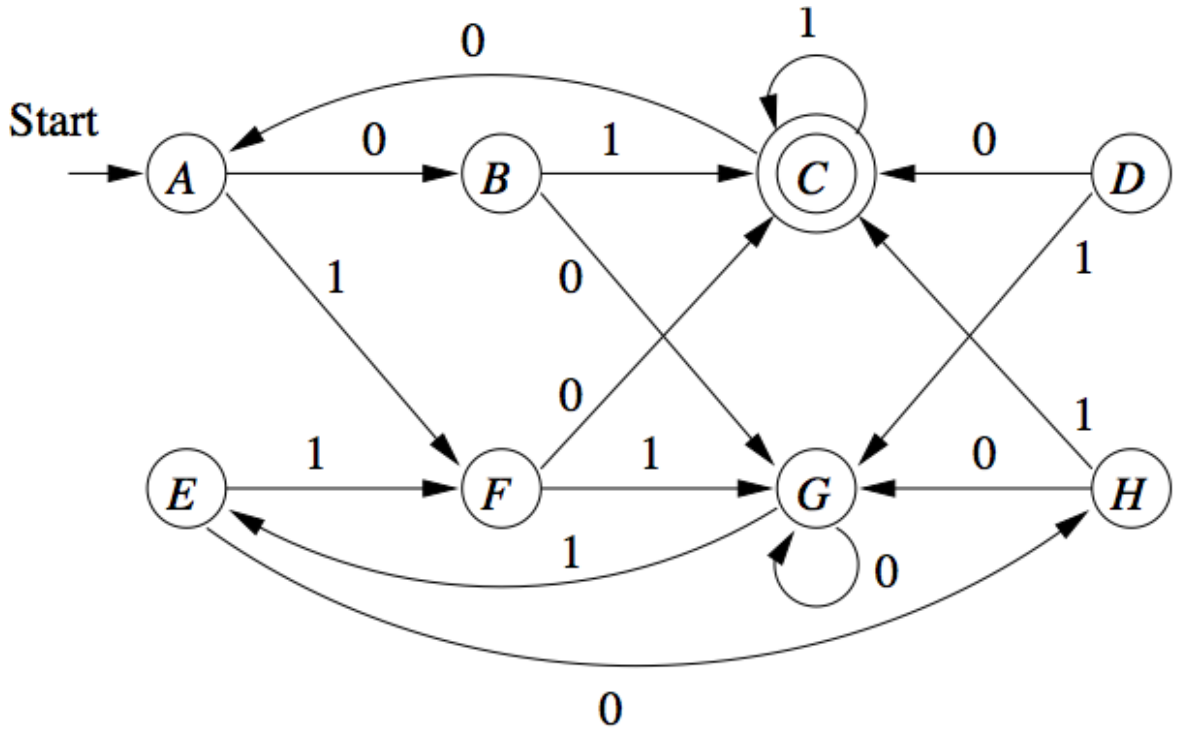
- Let $A = (Q, \delta, \Sigma, q_0, F)$ be a DFA, and $\{p, q\} \subseteq Q$.
- We say that p and q are equivalent, denoted $p \equiv q$, if:

$$\forall w \in \Sigma^*, \hat{\delta}(p, w) \in F \text{ iff } \hat{\delta}(q, w) \in F$$

- If two states p and q are not equivalent, then we say they are distinguishable, i.e.:

$$\exists w : \hat{\delta}(p, w) \in F \text{ and } \hat{\delta}(q, w) \notin F, \text{ or vice versa}$$

Examples

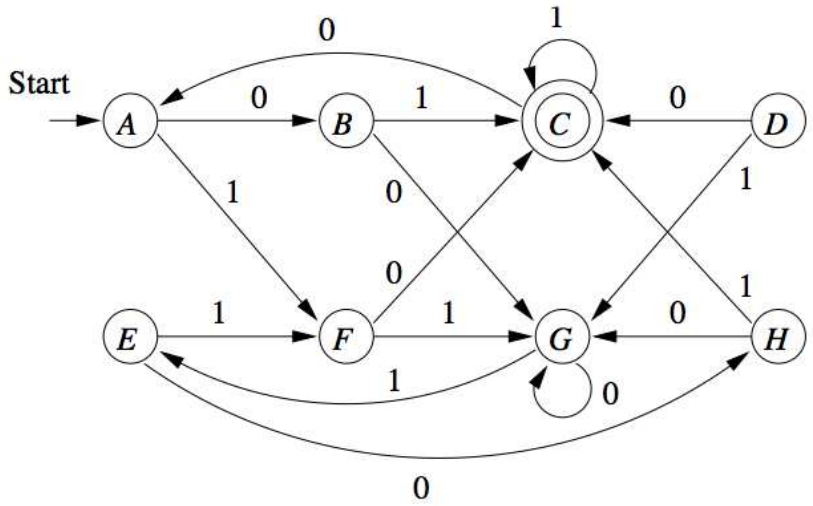


$$\hat{\delta}(C, \epsilon) \in F, \hat{\delta}(G, \epsilon) \notin F \Rightarrow C \not\equiv G$$

$$\hat{\delta}(A, 01) = C \in F, \hat{\delta}(G, 01) = E \notin F \Rightarrow A \not\equiv G$$

Examples (cnt.)

What about A and E?



$\hat{\delta}(A, \epsilon) = A \notin F, \hat{\delta}(E, \epsilon) = E \notin F$

$\hat{\delta}(A, 1) = F = \hat{\delta}(E, 1)$

Therefore $\hat{\delta}(A, 1x) = \hat{\delta}(E, 1x) = \hat{\delta}(F, x)$

$\hat{\delta}(A, 00) = G = \hat{\delta}(E, 00)$

$\hat{\delta}(A, 01) = C = \hat{\delta}(E, 01)$

Conclusion: $A \equiv E$.

- We concluded that $A \equiv E$ because no string whatsoever will distinguish A from E.
- To find states that are equivalent, we make our best efforts to find pairs of states which are distinguishable.

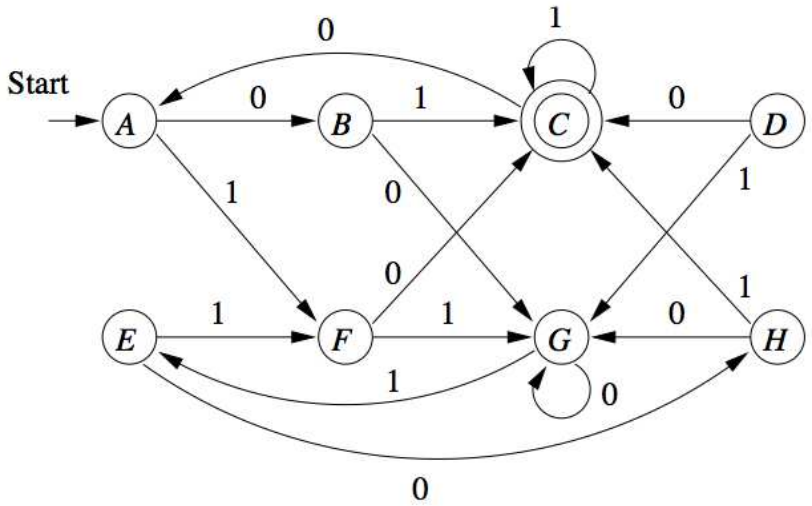
The next algorithm captures this idea.

The Table-of-Distinguishabilities Filling algorithm (TDFA)

We can compute distinguishable pairs with the following inductive algorithm:

- **Basis:** If $p \in F$ and $q \notin F$, then $p \not\equiv q$.
- **Induction:** If $\exists a \in \Sigma : \delta(r, a) \not\equiv \delta(s, a)$, then $r \not\equiv s$.

Given the DFA



The TFA yields this table

	A	B	C	D	E	F	G
B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x

The x's in the final table denote pairs of distinguishable states; blanks denote pairs of equivalent states.

Test for equivalence of RLs

- Let D and C be descriptors of regular languages .
- To test if $L(D) = L(C)$
  1. Convert both D and C to DFAs.
  2. Imagine the DFA that is the union of the two DFAs (never mind there are two start states)
  3. If the TDFA says that the two start states are distinguishable, then $L(D) \neq L(C)$, otherwise $L(D) = L(C)$.

Example

Both FA below accept $L(\epsilon + (0+1)^*0)$. Use the TDFA to verify if they recognize the same language.