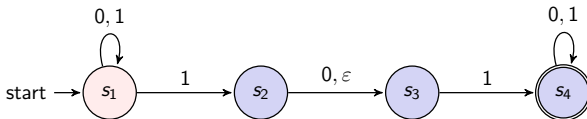


CSCI 3434: Theory of Computation

Lecture 4: Regular Expressions

Ashutosh Trivedi



Department of Computer Science
UNIVERSITY OF COLORADO BOULDER

What are Regular Languages?

- An alphabet $\Sigma = \{a, b, c\}$ is a **finite** set of letters,
- The set of all **strings** (aka, words) Σ^* over an alphabet Σ can be recursively defined as: as :
 - Base case: $\epsilon \in \Sigma^*$ (empty string),
 - Induction: If $w \in \Sigma^*$ then $wa \in \Sigma^*$ for all $a \in \Sigma$.
- A **language** L over some **alphabet** Σ is a **set** of **strings**, i.e. $L \subseteq \Sigma^*$.
- Some examples:
 - $L_{\text{even}} = \{w \in \Sigma^* : w \text{ is of even length}\}$
 - $L_{a^*b^*} = \{w \in \Sigma^* : w \text{ is of the form } a^n b^m \text{ for } n, m \geq 0\}$
 - $L_{a^n b^n} = \{w \in \Sigma^* : w \text{ is of the form } a^n b^n \text{ for } n \geq 0\}$
 - $L_{\text{prime}} = \{w \in \Sigma^* : w \text{ has a prime number of } a\text{'s}\}$
- **Deterministic finite state automata** define languages that require finite resources (states) to recognize.

What are Regular Languages?

- An alphabet $\Sigma = \{a, b, c\}$ is a **finite** set of letters,
- The set of all **strings** (aka, words) Σ^* over an alphabet Σ can be recursively defined as: as :
 - Base case: $\varepsilon \in \Sigma^*$ (empty string),
 - Induction: If $w \in \Sigma^*$ then $wa \in \Sigma^*$ for all $a \in \Sigma$.
- A **language** L over some **alphabet** Σ is a **set** of **strings**, i.e. $L \subseteq \Sigma^*$.
- Some examples:
 - $L_{\text{even}} = \{w \in \Sigma^* : w \text{ is of even length}\}$
 - $L_{a^*b^*} = \{w \in \Sigma^* : w \text{ is of the form } a^n b^m \text{ for } n, m \geq 0\}$
 - $L_{a^n b^n} = \{w \in \Sigma^* : w \text{ is of the form } a^n b^n \text{ for } n \geq 0\}$
 - $L_{\text{prime}} = \{w \in \Sigma^* : w \text{ has a prime number of } a\text{'s}\}$
- **Deterministic finite state automata** define languages that require finite resources (states) to recognize.

Definition (Regular Languages)

We call a language **regular** if it can be **accepted** by a deterministic finite state automaton.

Why they are “Regular”

- A number of widely different and equi-expressive formalisms precisely capture the same class of languages:
 - Deterministic finite state automata
 - Nondeterministic finite state automata (also with ε -transitions)
 - Kleene’s [regular expressions](#), also appeared as [Type-3 languages](#) in Chomsky’s hierarchy [Cho59].
 - [Monadic second-order logic](#) definable languages [Bö0, Elg61, Tra62]
 - Certain Algebraic connection (acceptability via finite semi-group) [RS59]

Why they are “Regular”

- A number of widely different and equi-expressive formalisms precisely capture the same class of languages:
 - Deterministic finite state automata
 - Nondeterministic finite state automata (also with ε -transitions)
 - Kleene’s **regular expressions**, also appeared as **Type-3 languages** in Chomsky’s hierarchy [Cho59].
 - **Monadic second-order logic** definable languages [Bö0, Elg61, Tra62]
 - Certain Algebraic connection (acceptability via finite semi-group) [RS59]

We have already seen that:

Theorem (DFA=NFA= ε -NFA)

*A language is accepted by a **deterministic finite automaton** if and only if it is accepted by a **non-deterministic finite automaton**.*

Why they are “Regular”

- A number of widely different and equi-expressive formalisms precisely capture the same class of languages:
 - Deterministic finite state automata
 - Nondeterministic finite state automata (also with ϵ -transitions)
 - Kleene’s **regular expressions**, also appeared as **Type-3 languages** in Chomsky’s hierarchy [Cho59].
 - **Monadic second-order logic** definable languages [Bö60, Elg61, Tra62]
 - Certain Algebraic connection (acceptability via finite semi-group) [RS59]

We have already seen that:

Theorem (DFA=NFA= ϵ -NFA)

*A language is accepted by a **deterministic finite automaton** if and only if it is accepted by a **non-deterministic finite automaton**.*

In this lecture we introduce **Regular Expressions**, and prove:

Theorem (REGEX=DFA)

*A language is accepted by a **deterministic finite automaton** if and only if it is accepted by a **regular expression**.*

Regular Expressions (RegEx)

- textual ([declarative](#)) way to represent regular languages (compare automata)
- Users of UNIX-based systems will already be familiar with these expressions:
 - **ls lecture*.pdf**
 - **rm -rf *.***
 - **grep automat* /usr/share/dict/words**
 - Also used in AWK, expr, Emacs and vi searches,
 - Lexical analysis tools like **flex** use it for defining [tokens](#)

Regular Expressions (RegEx)

- textual (**declarative**) way to represent regular languages (compare automata)
- Users of UNIX-based systems will already be familiar with these expressions:
 - **ls lecture*.pdf**
 - **rm -rf *.***
 - **grep automat* /usr/share/dict/words**
 - Also used in AWK, expr, Emacs and vi searches,
 - Lexical analysis tools like **flex** use it for defining **tokens**
- Some useful **String-set operations**:
 - **union** $L \cup M \stackrel{\text{def}}{=} \{w : w \in L \text{ or } w \in M\}$
 - **concatenation** $L.M \stackrel{\text{def}}{=} \{u.v : u \in L \text{ and } v \in M\}$
 - **self-concatenation** let $L^2 \stackrel{\text{def}}{=} L.L$, similarly L^3, L^4, \dots . Also $L^0 \stackrel{\text{def}}{=} \{\varepsilon\}$.
 - S. C. Kleene proposed notation L^* to denote **closure** of self-concatenation operation, i.e. $L^* \stackrel{\text{def}}{=} \cup_{i \geq 0} L^i$.
 - Examples $L = \{\varepsilon\}$ and $L = \{0, 1\}$

Regular Expressions: Inductive Definition

For a regular expression E we write $L(E)$ for its language. The set of valid regular expressions $RegEx$ can be defined recursively as the following:

	Syntax	Semantics
(empty string)	$\varepsilon \in RegEx$	$L(\varepsilon) = \{\varepsilon\}$
(empty set)	$\emptyset \in RegEx$	$L(\emptyset) = \emptyset$
(single letter)	$\mathbf{a} \in RegEx$	$L(\mathbf{a}) = \{a\}$
(variable)	$L \in RegEx$	where L is a language variable.
(union)	$E + F \in RegEx$	$L(E + F) = L(E) \cup L(F)$
(concatenation)	$E.F \in RegEx$	$L(E.F) = L(E).L(F)$
(Kleene Closure)	$E^* \in RegEx$	$L(E^*) = (L(E))^*$
(Parenthetic Expression)	$(E) \in RegEx$	$L((E)) = L(E).$

Precedence Rules: $* > . > +$

Example : $01^* + 1^*0^* \stackrel{\text{def}}{=} (0.(1^*)) + ((1^*).(0^*))$

Regular Expressions: Examples

Find regular expressions for the following languages:

- The set of all strings with an even number of 0's
- The set of all strings of even length (length multiple of k)
- The set of all strings that begin with 110
- The set of all strings containing exactly three 1's
- The set of all strings divisible by 2
- The set of strings where third last symbol is 1

Regular Expressions: Examples

Find regular expressions for the following languages:

- The set of all strings with an even number of 0's
- The set of all strings of even length (length multiple of k)
- The set of all strings that begin with 110
- The set of all strings containing exactly three 1's
- The set of all strings divisible by 2
- The set of strings where third last symbol is 1
- Practice writing regular expressions for the languages accepted by finite state automata.

Regular Expressions: Examples

Find regular expressions for the following languages:

- The set of all strings with an even number of 0's
- The set of all strings of even length (length multiple of k)
- The set of all strings that begin with 110
- The set of all strings containing exactly three 1's
- The set of all strings divisible by 2
- The set of strings where third last symbol is 1
- Practice writing regular expressions for the languages accepted by finite state automata.
- Can we generalize this intuitive construction?
- Can we construct a DFA/NFA for a regular expression?

Finite Automata to Regular Expressions

Theorem

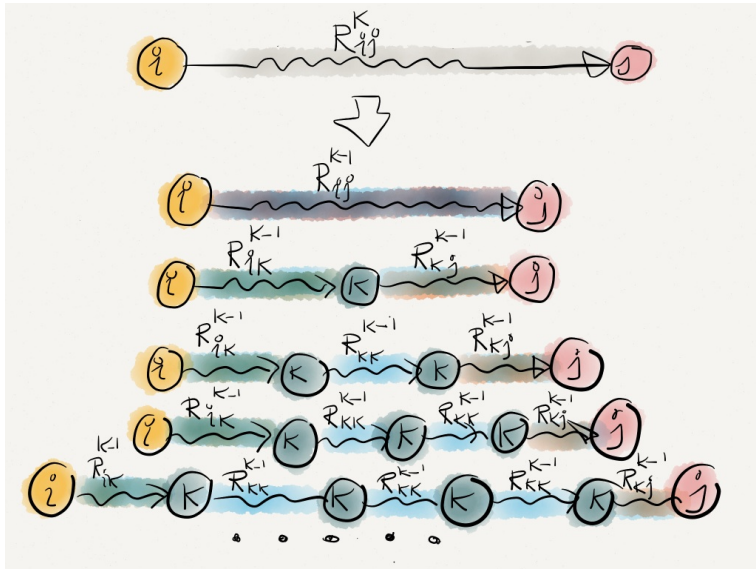
For every deterministic finite automaton A there exists a regular expression E_A such that $L(A) = L(E_A)$.

Proof.

- Let states of automaton A be $\{1, 2, \dots, n\}$.
- Consider $R_{i,j}^{(k)}$ be the regular expression whose language is the set of labels of path from i to j without visiting any state with label larger than k .
- (Basis): $R_{i,j}^{(0)}$ collects labels of direct paths from i to j ,
 - $R_{i,j}^{(0)} = a_1 + a_2 + \dots + a_n$ if $\delta(i, a_k) = j$ for $1 \leq k \leq n$
 - if $i = j$ then it also includes ε .
- (Induction): Compute $R_{i,j}^{(k)}$ using $R_{i,j}^{(k-1)}$'s.



Computing $R_{ij}^{(k)}$ using $R_{ij}^{(k-1)}$



Finite Automata to Regular Expressions

Theorem

For every deterministic finite automaton A there exists a regular expression E_A such that $L(A) = L(E_A)$.

Proof.

- Let states of automaton A be $\{1, 2, \dots, n\}$.
- Consider $R_{i,j}^{(k)}$ be the regular expression whose language is the set of labels of path from i to j without visiting any state with label larger than k .
- (Basis): $R_{i,j}^{(0)}$ collects labels of direct paths from i to j ,
 - $R_{i,j}^{(0)} = a_1 + a_2 + \dots + a_n$ if $\delta(i, a_k) = j$ for $1 \leq k \leq n$
 - if $i = j$ then it also includes ε .
- (Induction): Compute $R_{i,j}^{(k)}$ using $R_{i,j}^{(k-1)}$'s.

$$R_{i,j}^{(k)} = R_{i,j}^{(k-1)} + R_{i,k}^{(k-1)} \cdot (R_{k,k}^{(k-1)})^* \cdot R_{k,j}^{(k-1)}.$$

- E_A is $R_{i_0, f_1}^{(n)} + R_{i_0, f_2}^{(n)} + \dots + R_{i_0, f_k}^{(n)}$.

Alternative Method—Eliminating States

Shortcomings of previous reduction:

- The previous method works in all the settings, but is expensive (up to n^3 expressions, with a **factor of 4 blowup** in each step).
- For each i, j, i', j' , both $R_{i,j}^{(k+1)}$ and $R_{i',j'}^{(k+1)}$ store expression $(R_{k,k}^{(k)})^*$. This **duplication** can be avoided.

Alternative (more intuitive) method:

- A “beast” in the middle: **Finite automata with regular expressions**
- Remove all states except final and initial states in an “intuitive” way.
- Trivial to write regular expressions for DFA with only two states: an initial and a final one.
- The regular expression is union of this construction for every final state.
- **Example**

figure2

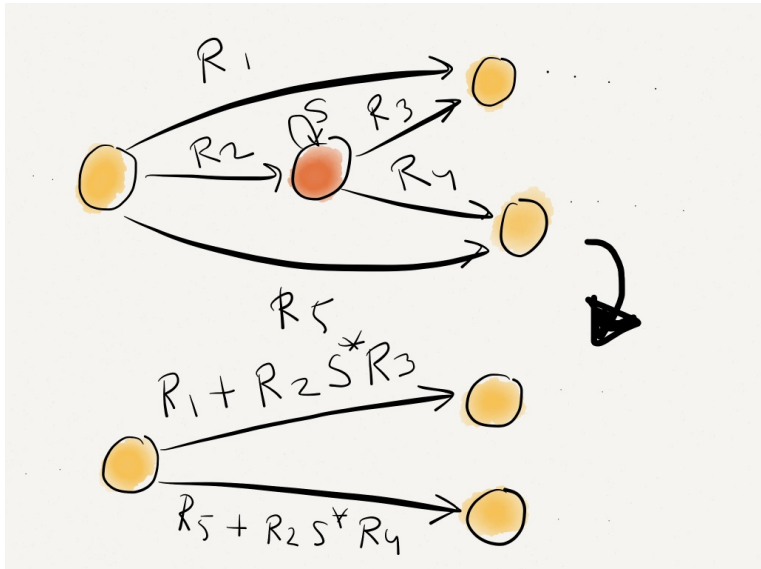
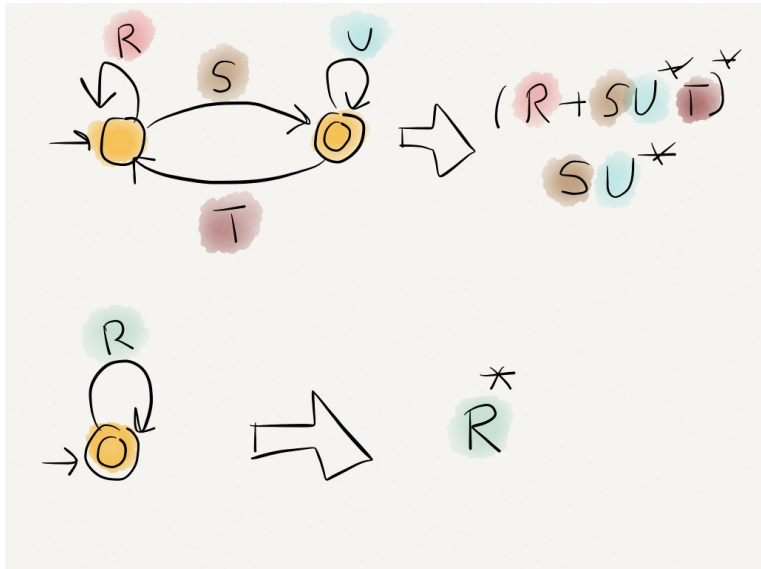


figure3



Regular Expressions to Finite Automata

Theorem

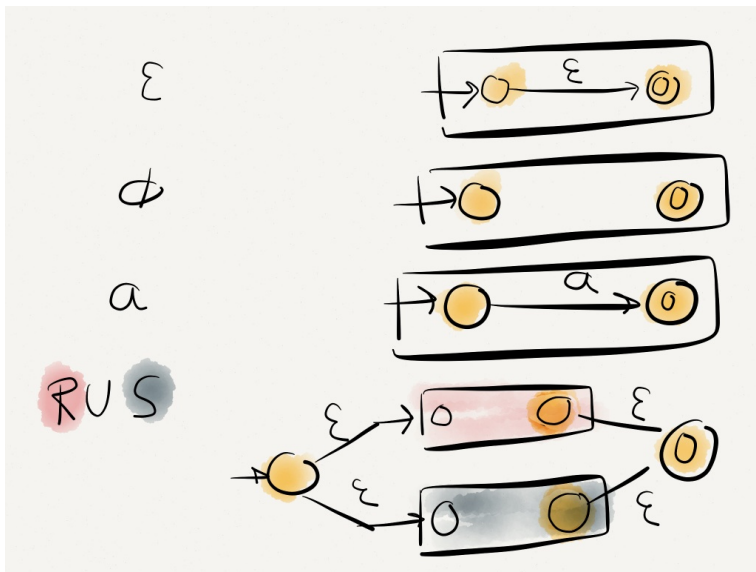
For every regular expression E there exists a deterministic finite automaton A_E such that $L(E) = L(A_E)$.

Proof.

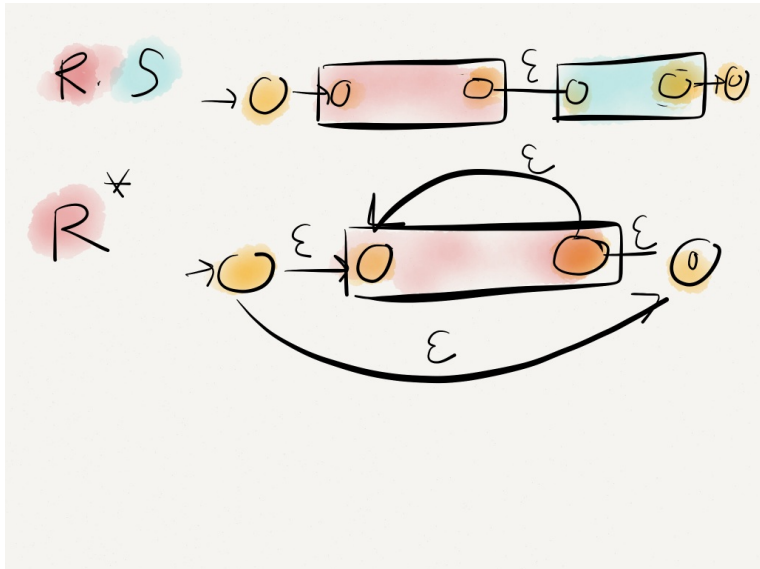
- Via induction on the structure of the regular expressions we show a reduction to nondeterministic finite automata with ε -transitions.
- Result follows from the equivalence of such automata with DFA.



Regular Expressions to Finite Automata



Regular Expressions to Finite Automata



Syntactic Sugar for Regular Expressions in Unix

$[a_1 a_2 a_3 \dots a_k]$	for	$a_1 + a_2 + \dots + a_k$
$.$	for	$a + b + \dots + z + A + \dots$
$ $	for	$+$
$R\{5\}$	for	$RRRRR$
$R+$	for	$\cup_{i \geq 1} R\{i\}$
$R?$	for	$\varepsilon + R$

Also $[A-Za-z0-9]$, $[:digits:]$, etc.

Applications:

Check the man page of “[grep](#)” (regular expression based search tool) and “[lex](#)” (A tool to generate regular expressions based pattern matching tool) to learn more about regular expressions on UNIX based systems.

Algebraic Laws for Regular Expressions

Associativity:

$$- L + (M + N) = (L + M) + N \text{ and } L.(M.N) = (L.M).N.$$

Commutativity:

$$- L + M = M + L. \text{ However, } L.M \neq M.L \text{ in general.}$$

Identity:

$$- \emptyset + L = L + \emptyset = L \text{ and } \varepsilon.L = L.\varepsilon = L$$

Annihilator:

$$- \emptyset.L = L.\emptyset = \emptyset$$

Distributivity:

$$- \text{left distributivity } L.(M + N) = L.M + L.N.$$

$$- \text{right distributivity } (M + N).L = M.L + N.L.$$

Idempotent $L + L = L$.

Closure Laws:

$$- (L^*)^* = L^*, \emptyset^* = \varepsilon, \varepsilon^* = \varepsilon, L^+ = LL^* = L^*L, \text{ and } L^* = L^+ + \varepsilon.$$

DeMorgan Type Law: $(L + M)^* = (L^*M^*)^*$

Verifying laws for regular expressions

Theorem

- Let E is some regular expressions with variables $L_1 L_2, \dots, L_m$.
- Let C be a regular expression where each L_i is concretized to some letters $a_1 a_2, \dots a_m$.
- Then every string w in $L(E)$ can be written as $w_1 w_2 \dots w_k$ where w_i is in some language L_{j_i} and $a_{j_1} a_{j_2} \dots a_{j_k}$ is in $L(C)$.
- In other words , the set $L(E)$ can be constructed by taking strings $a_{j_1} a_{j_2} \dots a_{j_k}$ from $L(C)$ and replacing a_{j_i} with L_{j_i} .

Proof.

A simple induction over the structure of regular expression E . □

Example

Theorem (Application)

Proof of a concretized law carries over to abstract law.

Example

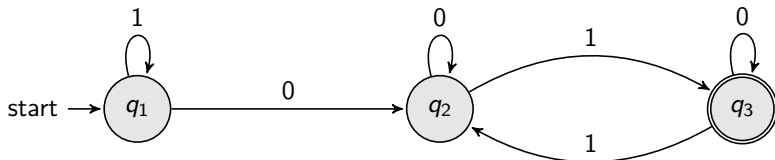
Prove that $(\varepsilon + L)^* = L^*$.

We can concretize the rule as $(\varepsilon + a)^* = a^*$. Let's prove the concretized law, and we know that the result will carry over to the abstract law.

$$\begin{aligned}(\varepsilon + a)^* &= (\varepsilon^* . a^*)^* \\ &= (\varepsilon . a^*)^* \\ &= (a^*)^* \\ &= a^*.\end{aligned}$$

First equality holds since $(L + M)^* = (L^* . M^*)^*$. The second equality holds since $\varepsilon^* = \varepsilon$. The third equality holds as ε is identity for concatenation, while the last equality follows from $(L^*)^* = L^*$.

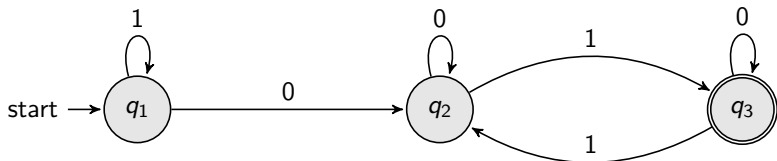
Example



	$R_{1,1}$	$R_{1,2}$	$R_{1,3}$	$R_{2,1}$	$R_{2,2}$	$R_{2,3}$	$R_{3,1}$	$R_{3,2}$	$R_{3,3}$
(0)	$1 + \epsilon$	0	\emptyset	\emptyset	$0 + \epsilon$	1	\emptyset	1	$0 + \epsilon$
(1)	1^*	1^*0	\emptyset	\emptyset	$0 + \epsilon$	1	\emptyset	1	$0 + \epsilon$
(2)	1^*	1^*00^*	1^*00^*1	\emptyset	0^*	0^*1	\emptyset	10^*	$(0 + \epsilon) + 10^*1$






$$\begin{aligned}
 R_{1,1}^{(1)} &= R_{1,1}^{(0)} + R_{1,1}^{(0)}(R_{1,1}^{(0)})^*R_{1,1}^{(0)} \\
 &= (1 + \epsilon) + (1 + \epsilon)(1 + \epsilon)^*(1 + \epsilon) \\
 &= (1 + \epsilon)\epsilon + (1 + \epsilon)(1 + \epsilon)^*(1 + \epsilon) \\
 &= (1 + \epsilon)\epsilon + (1 + \epsilon)1^*(1 + \epsilon) \\
 &= (1 + \epsilon)(\epsilon + 1^*(1 + \epsilon)) = (1 + \epsilon)(\epsilon + 1^*1 + 1^*\epsilon) \\
 &= (1 + \epsilon)(\epsilon + 1^+ + 1^*) = (1 + \epsilon)(1^* + 1^*) = (1 + \epsilon)1^* \\
 &= 11^* + 1^* = 1^+ + 1^* = 1^+ + 1^+ + \epsilon = 1^+ + \epsilon = 1^*.
 \end{aligned}$$

Example



	$R_{1,1}$	$R_{1,2}$	$R_{1,3}$	$R_{2,1}$	$R_{2,2}$	$R_{2,3}$	$R_{3,1}$	$R_{3,2}$	$R_{3,3}$
(0)	$1 + \epsilon$	0	\emptyset	\emptyset	$0 + \epsilon$	1	\emptyset	1	$0 + \epsilon$
(1)	1^*	1^*0	\emptyset	\emptyset	$0 + \epsilon$	1	\emptyset	1	$0 + \epsilon$
(2)	1^*	1^*00^*	1^*00^*1	\emptyset	0^*	0^*1	\emptyset	10^*	$(0 + \epsilon) + 10^*1$

$$\begin{aligned}R_{1,3}^{(3)} &= R_{1,3}^{(2)} + R_{1,3}^{(2)}(R_{3,3}^{(2)})^*R_{3,3}^{(2)} \\&= 1^*00^*1 + 1^*00^*1(0 + \epsilon + 10^*1)^*(0 + \epsilon + 10^*1) \\&= 1^*00^*1\epsilon + 1^*00^*1(0 + \epsilon + 10^*1)^*(0 + \epsilon + 10^*1) \\&= 1^*00^*1(\epsilon + (0 + \epsilon + 10^*1)^*(0 + \epsilon + 10^*1)) \\&= 1^*00^*1(\epsilon + (0 + 10^*1)^*(0 + \epsilon + 10^*1)) \\&= 1^*00^*1(\epsilon + (0 + 10^*1)^+ + (0 + 10^*1)^*) \\&= 1^*00^*1((0 + 10^*1)^* + (0 + 10^*1)^*) = 1^*00^*1(0 + 10^*1)^*\end{aligned}$$

-  J. R. Büchi.
Weak second-order arithmetic and finite automata.
Zeitschrift für Mathematische Logik und Grundlagen der Mathematik,
6(1–6):66–92, 1960.
-  Noam Chomsky.
On certain formal properties of grammars.
Information and Control, 2(2):137 – 167, 1959.
-  C. C. Elgot.
Decision problems of finite automata design and related arithmetics.
In Transactions of the American Mathematical Society, 98(1):21–51,
1961.
-  M. O. Rabin and D. Scott.
Finite automata and their decision problems.
IBM Journal of Research and Developmen, 3(2):114–125, 1959.
-  B. A. Trakhtenbrot.
Finite automata and monadic second order logic.
Siberian Mathematical Journal, 3:101–131, 1962.