

(XML Schema Languages and) Succinctness of Regular Expressions

Wouter Gelade

Hasselt University and
transnational University of Limburg

July 24, 2008

Regular Languages and Expressions

- One of the most fundamental concepts of (theoretical) computer science.
- Applications: Pattern matching, XML, . . .

Regular Languages and Expressions

- One of the most fundamental concepts of (theoretical) computer science.
- Applications: Pattern matching, XML,

Succinctness

- Regular expressions to automata.
- Operations on automata (state complexity).

Regular Languages and Expressions

- One of the most fundamental concepts of (theoretical) computer science.
- Applications: Pattern matching, XML,

Succinctness

- Regular expressions to automata.
- Operations on automata (state complexity).
- Automata to regular expressions?
- Operations on regular expressions???

XML Schema Languages

- Define sets of XML documents.
- Many languages: DTD, XML Schema, Relax NG, Pattern-based schemas, ...
- Use (extended) regular expressions to define structure of documents.

XML Schema Languages

- Define sets of XML documents.
- Many languages: DTD, XML Schema, Relax NG, Pattern-based schemas, ...
- Use (extended) regular expressions to define structure of documents.

Translations

Translations among schema languages reduce to:

- applying operations on regular expressions (complement, intersection); and
- *Translate away* additional operators.

Outline

- 1 Complement
- 2 Automata to Regular Expressions
- 3 Intersection
- 4 Interleaving

REs with complement

Theorem [Stockmeyer, Meyer '73]

RE(\neg) are non-elementary more succinct than standard regular expressions.

REs with complement

Theorem [Stockmeyer, Meyer '73]

RE(\neg) are non-elementary more succinct than standard regular expressions.

Single Complementation?

Given a regular expression r , what is the complexity of constructing a regular expression defining $\Sigma^* \setminus L(r)$.

Naive Algorithm

Proposition

Given a regular expression r , a regular expression s defining $\Sigma^* \setminus L(r)$ can be constructed in time $2^{2^{\mathcal{O}(|r|)}}$.

Algorithm

Given a regular expression r :

- Construct an NFA A with $L(A) = L(r)$. (polynomial)
- Construct a DFA B with $L(B) = \Sigma^* \setminus L(A)$. (exponential)
- Construct a RE s with $L(s) = L(B) = \Sigma^* \setminus L(r)$. (exponential)

Lower Bound

Theorem

For every $n \in \mathbb{N}$, there is a regular expression r_n of size $\mathcal{O}(n)$ such that any regular expression r defining $\Sigma^* \setminus L(r_n)$ is of size at least 2^{2^n} .

Lower Bound

Theorem

For every $n \in \mathbb{N}$, there is a regular expression r_n of size $\mathcal{O}(n)$ such that any regular expression r defining $\Sigma^* \setminus L(r_n)$ is of size at least 2^{2^n} .

Remark

All lower bounds, unless mentioned otherwise, are over a fixed-size (binary) alphabet.

Lower Bound

Theorem

For every $n \in \mathbb{N}$, there is a regular expression r_n of size $\mathcal{O}(n)$ such that any regular expression r defining $\Sigma^* \setminus L(r_n)$ is of size at least 2^{2^n} .

Remark

All lower bounds, unless mentioned otherwise, are over a fixed-size (binary) alphabet.

Remark 2

Several results independently obtained by Gruber and Holzer.

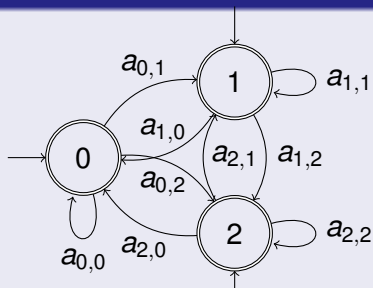
A Language by Ehrenfeucht and Zeiger

Definition

For every $n \in \mathbb{N}$, let Z_n be defined by the complete DFA on n states with

- only initial and final states; and
- a different label on every edge. ($\Sigma_n = \{a_{i,j} \mid 0 \leq i, j < n\}$)

Example: Z_3



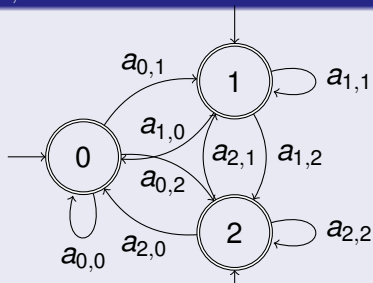
A Language by Ehrenfeucht and Zeiger

Definition

For every $n \in \mathbb{N}$, let Z_n be defined by the complete DFA on n states with

- only initial and final states; and
- a different label on every edge. ($\Sigma_n = \{a_{i,j} \mid 0 \leq i, j < n\}$)

Example: $a_{1,0}a_{0,2}a_{2,2} \in Z_3$



Lower Bound: Proof Sketch

Theorem [Ehrenfeucht and Zeiger '76]

Any regular expression defining Z_n must be of size at least 2^{n-1} .

Lower Bound: Proof Sketch

Theorem [Ehrenfeucht and Zeiger '76]

Any regular expression defining Z_n must be of size at least 2^{n-1} .

Corollary

Any regular expression defining Z_{2^n} must be of size at least 2^{2^n-1} .

Lower Bound: Proof Sketch

Theorem [Ehrenfeucht and Zeiger '76]

Any regular expression defining Z_n must be of size at least 2^{n-1} .

Corollary

Any regular expression defining Z_{2^n} must be of size at least 2^{2^n-1} .

End of Proof

Construct regular expression of size $\mathcal{O}(n)$ defining $\Sigma^* \setminus Z_{2^n}$.

Lower Bound: Proof Sketch

Theorem [Ehrenfeucht and Zeiger '76]

Any regular expression defining Z_n must be of size at least 2^{n-1} .

Corollary

Any regular expression defining Z_{2^n} must be of size at least 2^{2^n-1} .

End of Proof

Construct regular expression of size $\mathcal{O}(n)$ defining $\Sigma^* \setminus Z_{2^n}$.

Problem

The alphabet of Z_{2^n} is of size $(2^n)^2$.

Lower Bound: Proof Sketch

Binary Encoding of Z_n

For every $a_{i,j} \in \Sigma_n$ define

$$\rho_n(a_{i,j}) = \text{enc}(j)\text{enc}(i)\#,$$

where $\text{enc}(i)$ and $\text{enc}(j)$ are the $\lceil \log(n) \rceil$ -bit encodings of i and j .

Extend ρ_n to strings as $\rho_n(a_{i_0,i_1} \cdots a_{i_{k-1},i_k}) = \rho_n(a_{i_0,i_1}) \cdots \rho_n(a_{i_{k-1},i_k})$.

Lower Bound: Proof Sketch

Binary Encoding of Z_n

For every $a_{i,j} \in \Sigma_n$ define

$$\rho_n(a_{i,j}) = \text{enc}(j)\$ \text{enc}(i)\#,$$

where $\text{enc}(i)$ and $\text{enc}(j)$ are the $\lceil \log(n) \rceil$ -bit encodings of i and j .
 Extend ρ_n to strings as $\rho_n(a_{i_0,i_1} \cdots a_{i_{k-1},i_k}) = \rho_n(a_{i_0,i_1}) \cdots \rho_n(a_{i_{k-1},i_k})$.

The Language K_n : Definition

$K_n = \{\rho_n(w) \mid w \in Z_n\}$ (over the alphabet $\Sigma = \{0, 1, \$, \#\}$).

Lower Bound: Proof Sketch

Binary Encoding of Z_n

For every $a_{i,j} \in \Sigma_n$ define

$$\rho_n(a_{i,j}) = \text{enc}(j)\$ \text{enc}(i)\#,$$

where $\text{enc}(i)$ and $\text{enc}(j)$ are the $\lceil \log(n) \rceil$ -bit encodings of i and j .
 Extend ρ_n to strings as $\rho_n(a_{i_0,i_1} \cdots a_{i_{k-1},i_k}) = \rho_n(a_{i_0,i_1}) \cdots \rho_n(a_{i_{k-1},i_k})$.

The Language K_n : Definition

$K_n = \{\rho_n(w) \mid w \in Z_n\}$ (over the alphabet $\Sigma = \{0, 1, \$, \#\}$).

Example

- $w = a_{0,2}a_{2,1}a_{1,3} \in Z_4$ and thus,
- $\rho_n(w) = 10\$00\#01\$10\#11\$01\# \in K_4$.

Lower Bound: Proof Sketch

Theorem

Any regular expression defining K_n is of size at least 2^n .

Corollary

Any regular expression defining K_{2^n} must be of size at least 2^{2^n} .

Lower Bound: Proof Sketch

Theorem

Any regular expression defining K_n is of size at least 2^n .

Corollary

Any regular expression defining K_{2^n} must be of size at least 2^{2^n} .

Defining the Complement of K_{2^n}

- Expression is disjunction of expressions capturing all mistakes in a string. For instance:

Lower Bound: Proof Sketch

Theorem

Any regular expression defining K_n is of size at least 2^n .

Corollary

Any regular expression defining K_{2^n} must be of size at least 2^{2^n} .

Defining the Complement of K_{2^n}

- Expression is disjunction of expressions capturing all mistakes in a string. For instance:
- String does not end with #: $\Sigma^*(0 + 1 + \$)$.

Lower Bound: Proof Sketch

Theorem

Any regular expression defining K_n is of size at least 2^n .

Corollary

Any regular expression defining K_{2^n} must be of size at least 2^{2^n} .

Defining the Complement of K_{2^n}

- Expression is disjunction of expressions capturing all mistakes in a string. For instance:
- String does not end with #: $\Sigma^*(0 + 1 + \$)$.
- String has two corresponding bits which are not equal ($10\$00\#01\$10\#11\$00\#$):

Lower Bound: Proof Sketch

Theorem

Any regular expression defining K_n is of size at least 2^n .

Corollary

Any regular expression defining K_{2^n} must be of size at least 2^{2^n} .

Defining the Complement of K_{2^n}

- Expression is disjunction of expressions capturing all mistakes in a string. For instance:
- String does not end with #: $\Sigma^*(0 + 1 + \$)$.
- String has two corresponding bits which are not equal
 $(10\$00\#01\$10\#11\$00\#)$:
 $((0 + 1)^* + \Sigma^*\#(0 + 1)^*)1\Sigma^{3n+2}0\Sigma^* + \dots$
- ...

Outline

- 1 Complement
- 2 Automata to Regular Expressions**
- 3 Intersection
- 4 Interleaving

Automata to Regular Expressions

Theorem [McNaughton and Yamada '60]

Given an NFA A , a regular expression defining $L(A)$ can be constructed in time $2^{O(n)}$.

Automata to Regular Expressions

Theorem [McNaughton and Yamada '60]

Given an NFA A , a regular expression defining $L(A)$ can be constructed in time $2^{\mathcal{O}(n)}$.

Theorem [Ehrenfeucht and Zeiger '76]

- Any regular expression defining Z_n must be of size at least 2^{n-1} .
- There is a DFA of size $\mathcal{O}(n^2)$ accepting Z_n .

Automata to Regular Expressions

Theorem [McNaughton and Yamada '60]

Given an NFA A , a regular expression defining $L(A)$ can be constructed in time $2^{\mathcal{O}(n)}$.

Theorem [Ehrenfeucht and Zeiger '76]

- Any regular expression defining Z_n must be of size at least 2^{n-1} .
- There is a DFA of size $\mathcal{O}(n^2)$ accepting Z_n .

Corollary

In the translation from DFAs to regular expressions, an exponential blow-up can not be avoided.

Automata to Regular Expressions

Theorem

- 1 Any regular expression defining K_n is of size at least 2^n .
- 2 There is a DFA A_n of size $\mathcal{O}(n^2 \log n)$ defining K_n .

Automata to Regular Expressions

Theorem

- 1 Any regular expression defining K_n is of size at least 2^n .
- 2 There is a DFA A_n of size $\mathcal{O}(n^2 \log n)$ defining K_n .

Corollary

In the translation from DFAs to regular expressions, an exponential blow-up can not be avoided, even when the alphabet is **fixed**.

Automata to Regular Expressions

Theorem

- 1 Any regular expression defining K_n is of size at least 2^n .
- 2 There is a DFA A_n of size $\mathcal{O}(n^2 \log n)$ defining K_n .

Corollary

In the translation from DFAs to regular expressions, an exponential blow-up can not be avoided, even when the alphabet is **fixed**.

Theorem [Gruber, Holzer '08]

For every $n \in \mathbb{N}$, there is a DFA A_n of size $\mathcal{O}(n)$, over a **fixed** alphabet, such that any regular expression r defining $L(A_n)$ is of size at least 2^n .

Outline

- 1 Complement
- 2 Automata to Regular Expressions
- 3 Intersection**
- 4 Interleaving

REs with Intersection

Proposition

Let r be a $\text{RE}(\cap)$. A (standard) regular expression s defining $L(r)$ can be constructed in time $2^{2^{O(|r|)}}$.

REs with Intersection

Proposition

Let r be a $\text{RE}(\cap)$. A (standard) regular expression s defining $L(r)$ can be constructed in time $2^{2^{\mathcal{O}(|r|)}}$.

Theorem

Let $n \in \mathbb{N}$. There exist expressions r_1, \dots, r_m , each of size $\mathcal{O}(n)$, such that any regular expression defining $\bigcap_{i \leq m} L(r_i)$ is of size at least 2^{2^n} .

REs with Intersection

Proposition

Let r be a $\text{RE}(n)$. A (standard) regular expression s defining $L(r)$ can be constructed in time $2^{2^{O(|r|)}}$.

Theorem

Let $n \in \mathbb{N}$. There exist expressions r_1, \dots, r_m , each of size $O(n)$, such that any regular expression defining $\bigcap_{i \leq m} L(r_i)$ is of size at least 2^{2^n} .

Proof Idea

- Construct expressions describing properties any string in (a variant of) K_{2^n} must have.
- Variant of K_{2^n} is defined by intersection of expressions.

A fixed number of expressions

Upper bound

For any **fixed** $k \in \mathbb{N}$, let r_1, \dots, r_k be regular expressions. A regular expression defining $\bigcap_{i \leq k} L(r_i)$ can be constructed in time $2^{\mathcal{O}(|r|^k)}$.

A fixed number of expressions

Upper bound

For any **fixed** $k \in \mathbb{N}$, let r_1, \dots, r_k be regular expressions. A regular expression defining $\bigcap_{i \leq k} L(r_i)$ can be constructed in time $2^{\mathcal{O}(|r|^k)}$.

Theorem [Gruber, Holzer '08]

For every $n \in \mathbb{N}$, there are regular expressions r_n and s_n of size $\mathcal{O}(n)$ such that any regular expression defining $L(r_n) \cap L(s_n)$ is of size at least 2^n .

Proof

Definition

- The **star height** of a regular expression r , denoted $\text{sh}(r)$, is the maximal number of nested stars in r .
- $\text{sh}((a^*b)^* + c^*) = 2$, $\text{sh}(a^{***}) = 3$
- The **star height** of a regular language L , denoted $\text{sh}(L)$, is the minimal star height among all regular expressions defining L .
- $\text{sh}(L(a^{***})) = \text{sh}(a^*) = 1$

Proof

Lemma [Gruber, Holzer '08]

Let L be a regular language. Every regular expression defining L must be of **size** at least $2^{1/3(\text{sh}(L)-1)} - 1$.

Proof

Lemma [Gruber, Holzer '08]

Let L be a regular language. Every regular expression defining L must be of **size** at least $2^{1/3(\text{sh}(L)-1)} - 1$.

Proof

- Let $r_n = (b^* ab^* a \dots ab^*)^*$ (n a's) and $s_n = (a^* ba^* b \dots ba^*)^*$ (n b's).
- $\text{sh}(L(r_n) \cap L(s_n)) = n$. [Eggan '63, Gruber and Holzer '08]
- \Rightarrow Any regular expression defining $L(r_n) \cap L(s_n)$ must be of size exponential in n .

Outline

- 1 Complement
- 2 Automata to Regular Expressions
- 3 Intersection
- 4 Interleaving**

Interleaving or shuffle operator

Definition

- For words w, u, v , and symbols a, b :
- $w \& \varepsilon = \varepsilon \& w = w$, and
- $au \& bv = (a(u \& bv)) \cup (b(au \& v))$
- Allows the words of its operands to be *shuffled*.
- Example: $r = ab \& CD$
- $abCD, CDab, aCbD \in L(r)$, $baCD \notin L(r)$
- $L(r \& s) = \{w \mid u \in L(r), v \in L(s), w \in L(u \& v)\}$

REs with Interleaving

Proposition

Let r be a **RE**(&). A (standard) regular expression s defining $L(r)$ can be constructed in time $2^{2^{O(|r|)}}$.

REs with Interleaving

Proposition

Let r be a **RE(&)**. A (standard) regular expression s defining $L(r)$ can be constructed in time $2^{2^{\mathcal{O}(|r|)}}$.

Theorem

Let $n \in \mathbb{N}$. There exist expressions r_1, \dots, r_m , each of size $\mathcal{O}(n)$, over a **fixed** alphabet, such that any regular expression defining $L(r_1) \& \dots \& L(r_m)$ is of size at least 2^{2^n} .

REs with Interleaving

Proposition

Let r be a RE(&). A (standard) regular expression s defining $L(r)$ can be constructed in time $2^{2^{\mathcal{O}(|r|)}}$.

Theorem

Let $n \in \mathbb{N}$. There exist expressions r_1, \dots, r_m , each of size $\mathcal{O}(n)$, over a **fixed** alphabet, such that any regular expression defining $L(r_1) \& \dots \& L(r_m)$ is of size at least 2^{2^n} .

Theorem [Gruber, Holzer '08]

- Let $n \in \mathbb{N}$. There exist expressions r_1, \dots, r_m , each of **constant** size, over a **non-fixed** alphabet, such that any regular expression defining $L(r_1) \& \dots \& L(r_m)$ is of size at least 2^{2^n} .
- Expression: $(a_1 b_1)^* \& \dots \& (a_n b_n)^*$

A fixed number of expressions

Upper bound

For any **fixed** $k \in \mathbb{N}$, let r_1, \dots, r_k be regular expressions. A regular expression defining $L(r_1) \& \dots \& L(r_k)$ can be constructed in time $2^{\mathcal{O}(|r|^k)}$.

A fixed number of expressions

Upper bound

For any **fixed** $k \in \mathbb{N}$, let r_1, \dots, r_k be regular expressions. A regular expression defining $L(r_1) \& \dots \& L(r_k)$ can be constructed in time $2^{\mathcal{O}(|r|^k)}$.

Theorem [Gruber, Holzer '08]

For every $n \in \mathbb{N}$, there are regular expressions r_n and s_n of size $\mathcal{O}(n)$ such that any regular expression defining $L(r_n) \& L(s_n)$ is of size at least 2^n .

A fixed number of expressions

Upper bound

For any **fixed** $k \in \mathbb{N}$, let r_1, \dots, r_k be regular expressions. A regular expression defining $L(r_1) \& \dots \& L(r_k)$ can be constructed in time $2^{\mathcal{O}(|r|^k)}$.

Theorem [Gruber, Holzer '08]

For every $n \in \mathbb{N}$, there are regular expressions r_n and s_n of size $\mathcal{O}(n)$ such that any regular expression defining $L(r_n) \& L(s_n)$ is of size at least 2^n .

Expressions

- $r_n = (aa \cdots a)^*$, $s_n = (bb \cdots b)^*$.
- $\text{sh}(L(r_n) \& L(s_n)) = n$.

Conclusion

Conclusion

- Regular expressions are not very succinct.
- Settled some open problems of [Ellul, Krawetz, Shallit, Wang '05]
- Naive algorithms yield good complexity.