Approximate Computing in Formal Languages

Petr Dvořáček

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2, 612 66 Brno - Královo Pole
idvoracek@fit.vutbr.cz

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

10th December 2015

Content

Approximate computing

- Motivation
- Usage

Approximate computing in formal languages

- Cover languages and automata
- Regular expression approximation
- FSM covering *L2*, *L1*, and languages beyond *L0*
- Solving NP-complete problem with DTM in polynomial time

Conclusion

Approximate Computing – motivation

Definition

- Tradeoff between quality of result and efficiency.
- A common characteristic: a perfect result is not necessary and an approximate or less-than-optimal result is sufficient

Definition

- Tradeoff between quality of result and efficiency.
- A common characteristic: a perfect result is not necessary and an approximate or less-than-optimal result is sufficient.

Kaushik Roy's energy task

$$\frac{923}{21} > 1.75$$

$$\frac{923}{21} > 45.27$$

Approximate Computing – motivation

Definition

- Tradeoff between quality of result and efficiency.
- A common characteristic: a perfect result is not necessary and an approximate or less-than-optimal result is sufficient.

Kaushik Roy‘s energy task

$$\frac{923}{21} > 1.75$$

$$\frac{923}{21} > 45.27$$

Approximations are **natural**!

Approximate Computing – usage

- Signal and image processing
- Text search
- Clustering
- Data analysis
- Robotics
- Classification
- Neural networks
- Probabilistic computing
- Networking
- Hardware (cache)

The list is endless.

Approximate Computing – usage

- Signal and image processing – human perception is limited
- Text search
- Clustering
- Data analysis
- Robotics
- Classification
- Neural networks
- Probabilistic computing
- Networking
- Hardware (cache)

The list is endless.

- Signal and image processing – human perception is limited
- Text search
- Clustering
- Data analysis
- Robotics
- Classification
- Neural networks
- Probabilistic computing
- Networking
- Hardware (cache)

No golden result

The list is endless.

Approximate Computing – usage

- Signal and image processing – human perception is limited
- Text search
- Clustering
- Data analysis
- Robotics
- Classification
- Neural networks
- Probabilistic computing
- Networking
- Hardware (cache)

Text search, Clustering, Data analysis: No golden result

Robotics, Classification, Neural networks, Probabilistic computing, Networking, Hardware (cache): Perfect result is not always possible

The list is endless.

Reference Language – L

Approximate Language – LA such as $|LA \cap L| \geq 1$

Reference Language – L

Approximate Language – LA such as $|LA \cap L| \geq 1$

Error Language

Lets have two languages L and LA.

$E = L\ xor\ LA = (L - LA) \cup (LA - L)$

Error language contains only unique strings.

Reference Language – L

Approximate Language – LA such as $|LA \ \cap L| \geq 1$

Error Language

Lets have two languages L and LA.

$E \ = L \; xor \; LA = \ (L \ - LA) \cup (LA \ - L)$

Error language contains only unique strings.

Cover Languages

We can create *LA* to accept all words from language *L* and other strings. Thus we say *LA* covers *L*.

Because $L \ - LA = \ \emptyset$ then $|L| < |LA|$.

Reference Language – L

Approximate Language – LA such as $|LA \cap L| \geq 1$

Error Language

Lets have two languages L and LA.

$E = L\ xor\ LA = (L - LA) \cup (LA - L)$

Error language contains only unique strings.

Cover Languages

We can create *LA* to accept all words from language *L* and other strings. Thus we say *LA* covers *L*.

Because $L - LA = \emptyset$ then $|L| < |LA|$.

Finite Languages

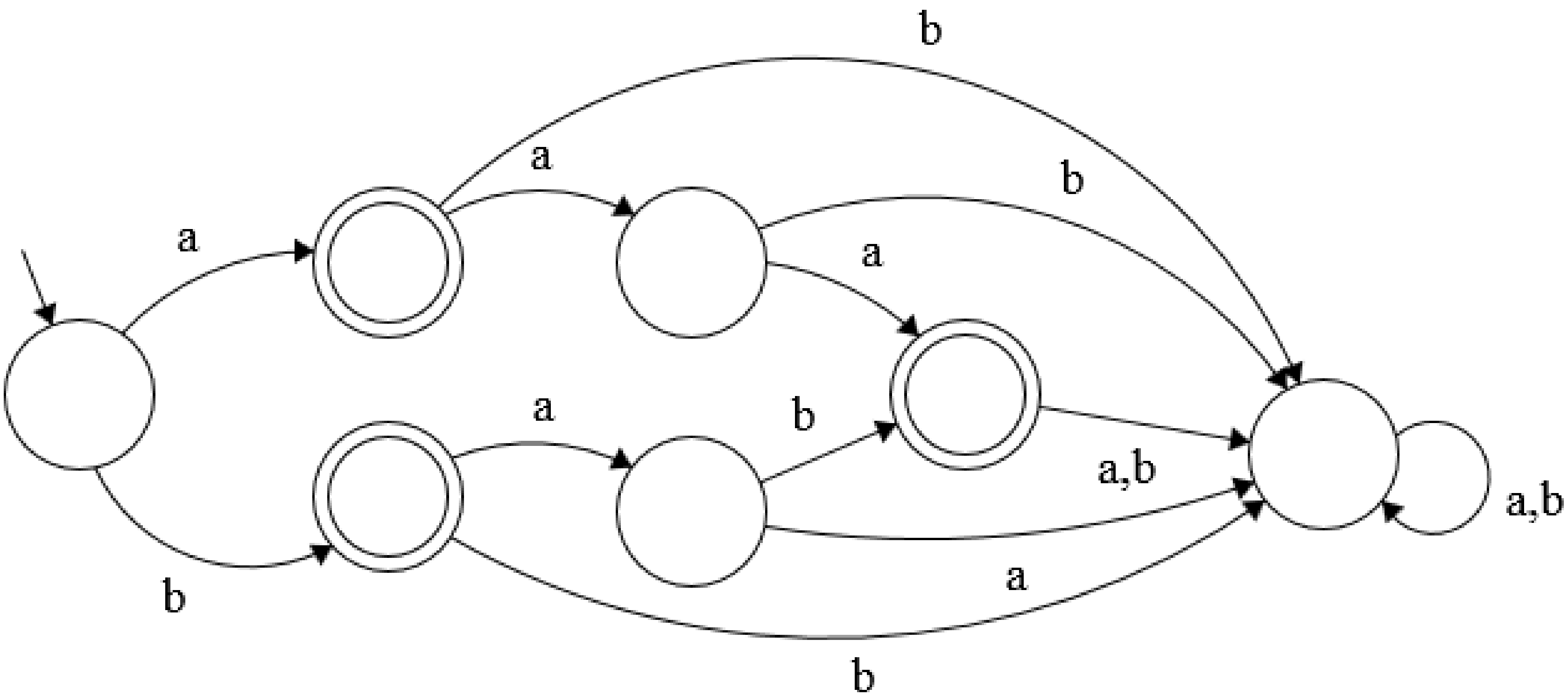
Creation of approximate **finite** language from the reference language *L* which is infinite.

A cover automaton for a **finite** language *L* is a FSM that accepts all words in *L* and possibly other words that are longer than any word in *L*.

$$L = \{a, b, aa, aaa, bab\}$$

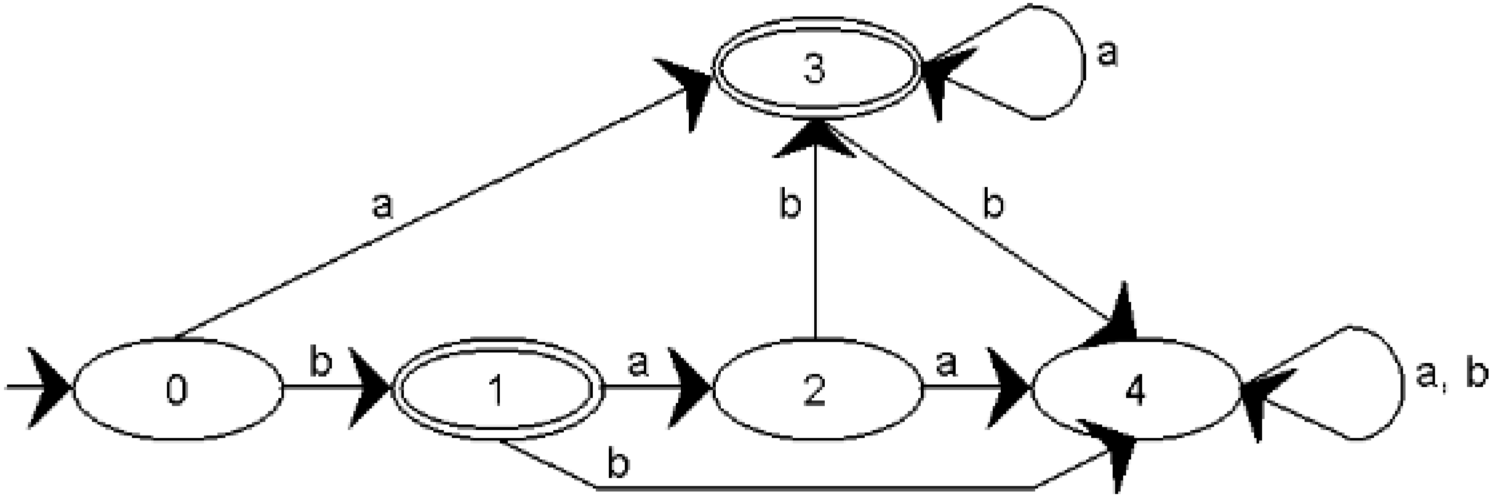
A cover automaton for a **finite** language *L* is a FSM that accepts all words in *L* and possibly other words that are longer than any word in *L*.

$$L = \{a, b, aa, aaa, bab\}$$

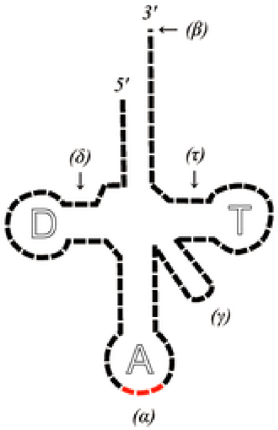


A cover automaton for a **finite** language *L* is a FSM that accepts all words in *L* and possibly other words that are longer than any word in *L*.

$$L = \{a, b, aa, aaa, bab\}$$
$$LA = \{aa^*, b, baba^*\}$$

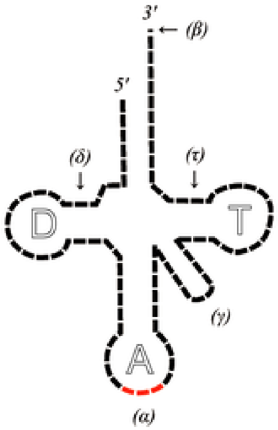


Task Find all the tRNA genes in DNA of bacterium e. coli.



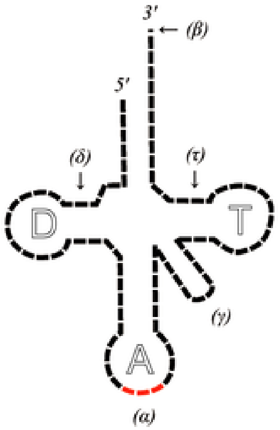
Task Find all the ~~tRNA genes~~ in ~~DNA of bacterium e. coli~~.

substrings string



Task Find all the ~~tRNA genes~~ in ~~DNA of bacterium e. coli~~.

substrings string



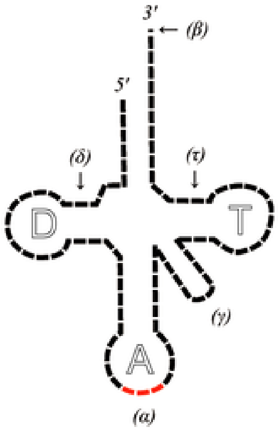
Substrings can be found by the RE:

(.{14}A[AG].{1,3}G.{11,14}[ATC]T(...)[AG].{11,31}GTTC[AG]A.[TC]C.{12}CCA)
|(TGG.{12}G[AG].T[TC]GAAC.{11,31}[TC](...)A[ATG].{11,14}C.{1,3}[TC]T.{14})

Regular expression approximation

Task Find all the ~~tRNA genes~~ in ~~DNA of bacterium e. coli~~.

substrings string



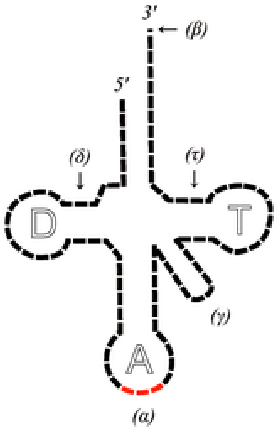
Substrings can be found by the RE:

(.{14}A**[AG]**.{1,3}G.{11,14}**[ATC]**T(...)**[AG]**.{11,31}GTTC**[AG]**A.**[TC]**C.{12}CCA)
|(TGG.{12}G**[AG]**.T**[TC]**GAAC.{11,31}**[TC]**(...)A**[ATG]**.{11,14}C.{1,3}**[TC]**T.{14})

Experiment	*Found*	*Filtered*
a) Fully functional RE	94	85

Task Find all the ~~tRNA genes~~ in ~~DNA of bacterium e. coli~~.

substrings string



Substrings can be found by the RE:

(.{14}A**[AG]**.{1,3}G.{11,14}**[ATC]**T(...)**[AG]**.{11,31}GTTC**[AG]**A.**[TC]**C.{12}CCA)
|(TGG.{12}G**[AG]**.T**[TC]**GAAC.{11,31}**[TC]**(...)A**[ATG]**.{11,14}C.{1,3}**[TC]**T.{14})

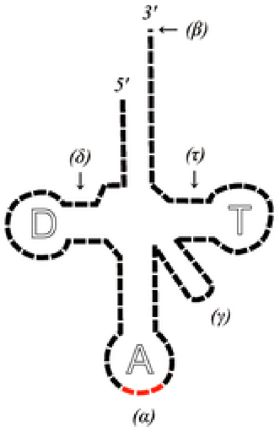
(.{14}A..{1,3}G.{11,14}.T(...)[AG].{11,31}GTTC.A..C.{12}CCA)
|(TGG.{12}G..T.GAAC.{11,31}.(...)A..{11,14}C.{1,3}.T.{14})

Experiment	*Found*	*Filtered*
a) Fully functional RE	94	85
b) Let . replace all brackets	118	85

Regular expression approximation

Task Find all the ~~tRNA genes~~ in ~~DNA of bacterium e. coli~~.

substrings string



Substrings can be found by the RE:

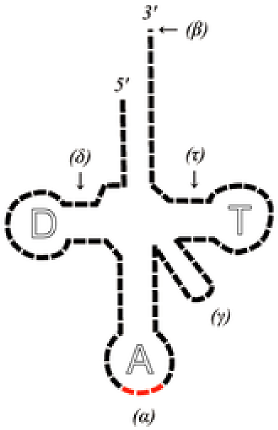
(.{14}A[AG].{1,3}G.{11,14}[ATC]T(...)[AG]**.{11,31}**GTTC[AG]A.[TC]C.{12}CCA)
|(TGG.{12}G[AG].T[TC]GAAC**.{11,31}**[TC](...)A[ATG].{11,14}C.{1,3}[TC]T.{14})

(.{14}A[AG].{1,3}G.{11,14}[ATC]T(...)[AG]**.{11,21}**GTTC[AG]A.[TC]C.{12}CCA)
|(TGG.{12}G[AG].T[TC]GAAC**.{11,21}**[TC](...)A[ATG].{11,14}C.{1,3}[TC]T.{14})

Experiment	*Found*	*Filtered*
a) Fully functional RE	94	85
b) Let . replace all brackets	118	85
c) Let make var. loop shorter	80	70

Task Find all the ~~tRNA genes~~ in ~~DNA of bacterium e. coli~~.

substrings string



Substrings can be found by the RE:

(.{14}A[AG].{1,3}G.{11,14}[ATC]T(...)[AG]**.{11,31}**GTTC[AG]A.[TC]C.{12}CCA)
|(TGG.{12}G[AG].T[TC]GAAC**.{11,31}**[TC](...)A[ATG].{11,14}C.{1,3}[TC]T.{14})

(.{14}A**.**.{1,3}G.{11,14}**.**T(...)[AG]**.{11,21}**GTTC.A**.**.C.{12}CCA)
|(TGG.{12}G**.**.T**.**GAAC**.{11,21}.**(...)A**.**.{11,14}C.{1,3}**.**T.{14})

Experiment	*Found*	*Filtered*
a) Fully functional RE	94	85
b) Let . replace all brackets	118	85
c) Let make var. loop shorter	80	70
d) Combination of b, c	109	70

Let have CFL $L = \{a^n b^n | \, n \geq 0\}$

Let have CFL $L = \{a^n b^n | \, n \geq 0\}$

Approximating with a Finite-State Calculus (principle)

Construct a grammar G of L.

$$S \rightarrow aSb \quad [1]$$
$$\rightarrow \varepsilon \quad [2]$$

In G find all cycling rules ($S \rightarrow aS, \; S \rightarrow Sb$) by using right-hand-rule, seven formulae, and dotted rules.

Let have CFL $L = \{a^n b^n | \, n \geq 0\}$

Approximating with a Finite-State Calculus (principle)

Construct a grammar G of L.

$$S \rightarrow aSb \quad [1]$$

$$\rightarrow \varepsilon \quad [2]$$

In G find all cycling rules ($S \rightarrow aS,\ S \rightarrow Sb$) by using right-hand-rule, seven formulae, and dotted rules.

Final cover language

$$LA = \{a^+ b^+ \mid \varepsilon\}$$

Cons: We can find better approximation: $\{aa^+ bb^+ \mid ab \mid \varepsilon\}$

Syntax analysis

Let have context free grammar G1

$$
\begin{aligned}
E &\rightarrow (E) \\
  &\rightarrow E * E \\
  &\rightarrow E + E \\
  &\rightarrow i
\end{aligned}
$$

Syntax analysis

Let have context free grammar G1

$$
\begin{aligned}
E &\rightarrow (E) \\
&\rightarrow E * E \\
&\rightarrow E + E \\
&\rightarrow i
\end{aligned}
$$

Do we write programs like this?

a = (a+b)*c;

Syntax analysis

Let have context free grammar G1

$$
\begin{aligned}
E &\rightarrow (E) \\
&\rightarrow E * E \\
&\rightarrow E + E \\
&\rightarrow i
\end{aligned}
$$

Do we write programs like this?

a = (a+b)*c;

Or like this?

a = (((… (a + b) * c) …)));

Syntax analysis

Let have context free grammar G1

$$
\begin{aligned}
E &\rightarrow (E) \\
&\rightarrow E * E \\
&\rightarrow E + E \\
&\rightarrow i
\end{aligned}
$$

Do we write programs like this?

a = (a+b)*c;

Or like this?

a = (((… (a + b) * c) …)));

We don’t use infinite number of parenthesis.

Syntax analysis

Let have context free grammar G1

$$
\begin{aligned}
E &\rightarrow (E) \\
&\rightarrow E * E \\
&\rightarrow E + E \\
&\rightarrow i
\end{aligned}
$$

G1 can be covered with G2

$$
\begin{aligned}
A &\rightarrow (B & \bar{A} &\rightarrow +A & B &\rightarrow (C & \bar{B} &\rightarrow +B & C &\rightarrow \ldots \\
&\rightarrow i\bar{A} & &\rightarrow *A & &\rightarrow i)\bar{A} & &\rightarrow *B & & \\
& & &\rightarrow \varepsilon & &\rightarrow i\bar{B} & & & &
\end{aligned}
$$

Cons: Breaks syntax tree – no priorities given.

Covering L1 with FSM

$$L = \{a^n \mid n \text{ is prime}\}$$

$L = \{a^n \mid n \text{ is prime}\}$

Approximation is based on sieve of Eratosthenes

Covering L1 with FSM

$L = \{a^n \mid n \text{ is prime}\}$

Approximation is based on sieve of Eratosthenes

Then we can construct approximate language such as:

$L_2 = \{aa\}$ $\overline{L_2} = \{aa(aa)^+\}$

$L = \{a^n \mid n \text{ is prime}\}$

Approximation is based on sieve of Eratosthenes

Then we can construct approximate language such as:

$L_2 = \{aa\}$ $\overline{L_2} = \{aa(aa)^+\}$

$L_3 = \{aaa\}$ $\overline{L_3} = \{aaa(aaa)^+\}$

$L = \{a^n \mid n \text{ is prime}\}$

Approximation is based on sieve of Eratosthenes

Then we can construct approximate language such as:

$$L_2 = \{aa\} \qquad \overline{L_2} = \{aa(aa)^+\}$$
$$L_3 = \{aaa\} \qquad \overline{L_3} = \{aaa(aaa)^+\}$$
$$L_5 = \{aaaaa\} \qquad \overline{L_5} = \{aaaaa(aaaaa)^+\}$$

$L = \{a^n \mid n \text{ is prime}\}$

Approximation is based on sieve of Eratosthenes

Then we can construct approximate language such as:

$L_2 = \{aa\}$ $\qquad \overline{L_2} = \{aa(aa)^+\}$

$L_3 = \{aaa\}$ $\qquad \overline{L_3} = \{aaa(aaa)^+\}$

$L_5 = \{aaaaa\}$ $\qquad \overline{L_5} = \{aaaaa(aaaaa)^+\}$

$L_7 = \{a^7\}$ $\qquad \overline{L_7} = \{a^7(a^7)^+\}$

$L = \{a^n \mid n \text{ is prime}\}$

Approximation is based on sieve of Eratosthenes

Then we can construct approximate language such as:

$L_2 = \{aa\}$ $\overline{L_2} = \{aa(aa)^+\}$

$L_3 = \{aaa\}$ $\overline{L_3} = \{aaa(aaa)^+\}$

$L_5 = \{aaaaa\}$ $\overline{L_5} = \{aaaaa(aaaaa)^+\}$

$L_7 = \{a^7\}$ $\overline{L_7} = \{a^7(a^7)^+\}$

$L_{11} = \{a^{11}\}$ $\overline{L_{11}} = \{a^{11}(a^{11})^+\}$

$L = \{a^n \mid n \text{ is prime}\}$

Approximation is based on sieve of Eratosthenes

Then we can construct approximate language such as:

$L_2 = \{aa\}$ $\overline{L_2} = \{aa(aa)^+\}$

$L_3 = \{aaa\}$ $\overline{L_3} = \{aaa(aaa)^+\}$

$L_5 = \{aaaaa\}$ $\overline{L_5} = \{aaaaa(aaaaa)^+\}$

$L_7 = \{a^7\}$ $\overline{L_7} = \{a^7(a^7)^+\}$

$L_{11} = \{a^{11}\}$ $\overline{L_{11}} = \{a^{11}(a^{11})^+\}$

$L_{13} = \{a^{13}\}$ $\overline{L_{13}} = \{a^{13}(a^{13})^+\}$

$L = \{a^n \mid n \text{ is prime}\}$

Approximation is based on sieve of Eratosthenes

Then we can construct approximate language such as:

$$L_2 = \{aa\} \qquad \overline{L_2} = \{aa(aa)^+\}$$
$$L_3 = \{aaa\} \qquad \overline{L_3} = \{aaa(aaa)^+\}$$
$$L_5 = \{aaaaa\} \qquad \overline{L_5} = \{aaaaa(aaaaa)^+\}$$
$$L_7 = \{a^7\} \qquad \overline{L_7} = \{a^7(a^7)^+\}$$
$$L_{11} = \{a^{11}\} \qquad \overline{L_{11}} = \{a^{11}(a^{11})^+\}$$
$$L_{13} = \{a^{13}\} \qquad \overline{L_{13}} = \{a^{13}(a^{13})^+\}$$

$$LA = L_2 \cup L_3 \cup L_5 \cup L_7 \cup L_{11} \cup L_{13} \cup \left(\{aa^+\} \setminus \left(\overline{L_2} \cup \overline{L_3} \cup \overline{L_5} \cup \overline{L_7} \cup \overline{L_{11}} \cup \overline{L_{13}}\right)\right)$$

We can construct FSM that accept approximated language.

$$L = \{a^n \mid n \text{ is prime}\}$$

$$LA = L_2 \cup L_3 \cup L_5 \cup L_7 \cup L_{11} \cup L_{13}$$
$$\cup \left(\{aa^+\} \setminus \left(\overline{L_2} \cup \overline{L_3} \cup \overline{L_5} \cup \overline{L_7} \cup \overline{L_{11}} \cup \overline{L_{13}}\right)\right)$$

$$L = \{a^n \mid n \text{ is prime}\}$$

$$LA = L_2 \cup L_3 \cup L_5 \cup L_7 \cup L_{11} \cup L_{13} \cup \left(\{aa^+\} \setminus \left(\overline{L_2} \cup \overline{L_3} \cup \overline{L_5} \cup \overline{L_7} \cup \overline{L_{11}} \cup \overline{L_{13}}\right)\right)$$

Experiment 8-bit numbers

Maximal number is 255. Then $\sqrt{255} = 16$. *LA* covers all primes to 16. Thus *LA* covers all primes to 255.

$$L = \{a^n \mid n \text{ is prime}\}$$

$$LA = L_2 \cup L_3 \cup L_5 \cup L_7 \cup L_{11} \cup L_{13} \cup \left(\{aa^+\} \setminus \left(\overline{L_2} \cup \overline{L_3} \cup \overline{L_5} \cup \overline{L_7} \cup \overline{L_{11}} \cup \overline{L_{13}}\right)\right)$$

Experiment 8-bit numbers

Maximal number is 255. Then $\sqrt{255} = 16$. *LA* covers all primes to 16. Thus *LA* covers all primes to 255.

Experiment 16-bit numbers

Maximal number is 65 535 and there is 6 542 primes. ***LA* covers them all**. It also contains 6 032 numbers which are not primes. The rest is rejected.

$$L = \{a^n \mid n \text{ is prime}\}$$

$$LA = L_2 \cup L_3 \cup L_5 \cup L_7 \cup L_{11} \cup L_{13} \cup \left(\{aa^+\} \setminus \left(\overline{L_2} \cup \overline{L_3} \cup \overline{L_5} \cup \overline{L_7} \cup \overline{L_{11}} \cup \overline{L_{13}}\right)\right)$$

Experiment 8-bit numbers *100% accuracy*

Maximal number is 255. Then $\sqrt{255} = 16$. *LA* covers all primes to 16. Thus *LA* covers all primes to 255.

Experiment 16-bit numbers *90% accuracy*

Maximal number is 65 535 and there is 6 542 primes. ***LA* covers them all**. It also contains 6 032 numbers which are not primes. The rest is rejects correctly.

Theorem

We can build FSM that covers **every** language.

(Even the languages beyond *L0*.)

FIT

Theorem

We can build FSM that covers **every** language.

(Even the languages beyond *L0*.)

Proof

Let have $\Sigma = \{a_0, a_1, \dots, a_n\}$.

Then we can build FSM from RE: $(a_0^* a_1^* \dots a_n^*) *$

This FSM accepts all strings – **accepts everything**.

Theorem

We can build FSM that covers **every** language.

(Even the languages beyond *L0*.)

Proof

Let have $\Sigma = \{a_0, a_1, \dots, a_n\}$.

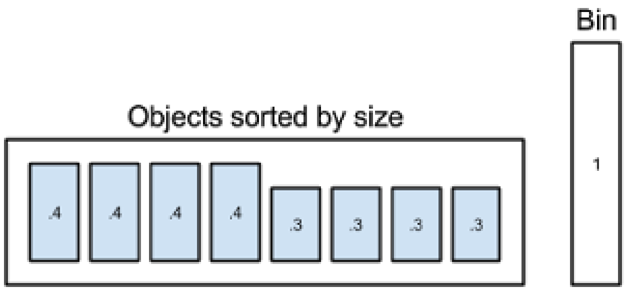
Then we can build FSM from RE: $(a_0^* a_1^* \dots a_n^*) *$

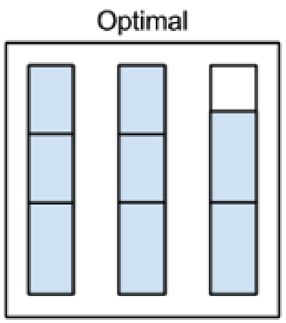
This FSM accepts all strings – **accepts everything**.

However the error language can be really large.

Bin-packing problem

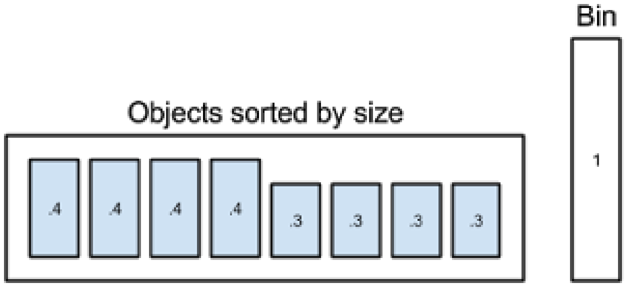
- Pack all the stuff into as few bins as possible.
- NP complete problem

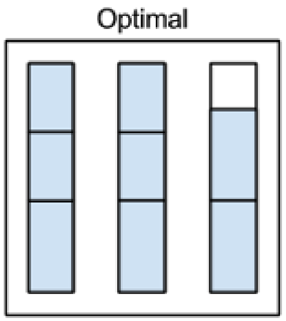




Bin-packing problem

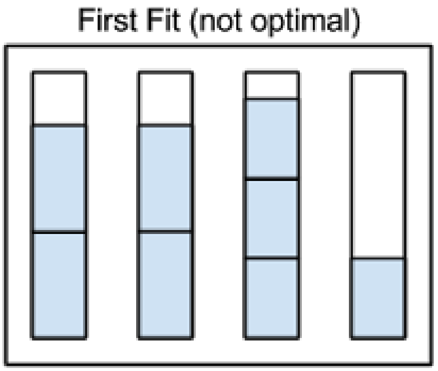
- Pack all the stuff into as few bins as possible.
- NP complete problem





Approximation – principle

- Put an item into the first bin where is space.



Approximation

- is natural
- makes our brains and machines faster
- can be used in text search (tRNA genes in DNA)

Less powerful machines can be used for approximation of more complex problems

- FSM can cover CFL $\{a^n b^n \mid n \geq 0\}$
- FSM can cover CSL $\{a^n \mid n \text{ is prime}\}$
- FSM can cover all languages (but it is not wise)
- DTM solving NP-complete problem

Thank You For Your Attention !

References

- L. Liu, *Practicality of the Vector Packing Problem*
- Edmund Grimley Evans, *Approximating Context-Free Grammars with a Finite-State Calculus*, ACL 98
- K. Roy, *Approximate Computing: An Energy-Efficient Computing Technique for Error Resilient Applications*, 2015 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)
- C. Campenau, N. Santean, S. Yu, *Minimal cover-automata for finite languages*, Workshop on Implementing Automata '98