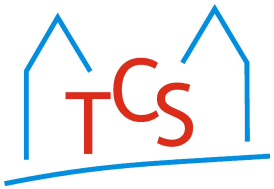
On String Compression by Context-free Grammars

Rüdiger Reischuk

joint work with Jan Arpe

presented at Data Compression Conference, Utah 2006



Institut für Theoretische Informatik
Universität zu Lübeck

2. Lübeck-Tartu Workshop
September 26, 2006

Research at ITCS

- algorithmic complexity

Research at ITCS

- algorithmic complexity
  - design efficient algorithms: randomized, parallel, online

Research at ITCS

- algorithmic complexity
  - design efficient algorithms: randomized, parallel, online
  - algorithmic learning
    discrete optimization, molecular biology

Research at ITCS

- algorithmic complexity
  - design efficient algorithms: randomized, parallel, online
  - algorithmic learning
    discrete optimization, molecular biology
  - complexity classes
    lower bounds on problem complexity

Research at ITCS

- algorithmic complexity
  - design efficient algorithms: randomized, parallel, online
  - algorithmic learning
    discrete optimization, molecular biology
  - complexity classes
    lower bounds on problem complexity
- formal methods and analysis: distributed systems

Research at ITCS

- algorithmic complexity
  - design efficient algorithms: randomized, parallel, online
  - algorithmic learning
    discrete optimization, molecular biology
  - complexity classes
    lower bounds on problem complexity
- formal methods and analysis: distributed systems
- security, cryptography, compression of digital data

Research at ITCS

- algorithmic complexity
  - design efficient algorithms: randomized, parallel, online
  - algorithmic learning
    discrete optimization, molecular biology
  - complexity classes
    lower bounds on problem complexity
- formal methods and analysis: distributed systems
- security, cryptography, compression of digital data
  - information hiding, steganography, watermarking

Research at ITCS

- algorithmic complexity
  - design efficient algorithms: randomized, parallel, online
  - algorithmic learning
    discrete optimization, molecular biology
  - complexity classes
    lower bounds on problem complexity
- formal methods and analysis: distributed systems
- security, cryptography, compression of digital data
  - information hiding, steganography, watermarking
  - authentication and privacy issues in networks
    string compression, signatures for digital data

Compression of Strings

- given a string **x** find a string $\mathbf{y = compress(x)}$ that specifies x in a more compact way

Compression of Strings

- given a string **x** find a string $\mathbf{y = compress(x)}$ that specifies x in a more compact way
  - 01010101 $\rightsquigarrow$ $(0\ 1)^4$

Compression of Strings

- given a string **x** find a string $\mathbf{y = compress(x)}$ that specifies x in a more compact way
  - $01010101 \rightsquigarrow (0\,1)^4$
  - $11110100001000101111 \rightsquigarrow$

Compression of Strings

- given a string **x** find a string $\mathbf{y = compress(x)}$ that specifies x in a more compact way
  - 01010101 $\rightsquigarrow$ $(0\,1)^4$
  - 11110100001000101111 $\rightsquigarrow$ *largest prime less than 1M in binary*

Compression of Strings

- given a string **x** find a string $\mathbf{y = compress(x)}$ that specifies x in a more compact way
  - $01010101 \rightsquigarrow (0\,1)^4$
  - $11110100001000101111 \rightsquigarrow$ *largest prime less than 1M in binary*
  - . .

Compression of Strings

- given a string **x** find a string $\mathbf{y = compress(x)}$ that specifies x in a more compact way
  - $01010101 \rightsquigarrow (0\,1)^4$
  - $11110100001000101111 \rightsquigarrow$ *largest prime less than 1M in binary*
  - . .

1. universal schemes

Compression of Strings

- given a string **x** find a string $\mathbf{y = compress(x)}$ that specifies x in a more compact way
  - 01010101 $\rightsquigarrow$ $(0\,1)^4$
  - 11110100001000101111 $\rightsquigarrow$ *largest prime less than 1M in binary*
  - . .

1. universal schemes
   - $|\mathrm{compress}(x)|$ = Kolmogorov complexity $K(x)$ not computable!

Compression of Strings

- given a string **x** find a string $\mathbf{y = compress(x)}$ that specifies x in a more compact way
  - $01010101 \rightsquigarrow (0\,1)^4$
  - $1111010000101000101111 \rightsquigarrow$ *largest prime less than 1M in binary*
  - . .

1. universal schemes
   - $|\mathrm{compress}(x)| = K(x)$ Kolmogorov complexity
     not computable!
2. ressource bounded versions

Compression of Strings

- given a string **x** find a string $\mathbf{y = compress(x)}$ that specifies x in a more compact way
  - 01010101 $\rightsquigarrow$ $(0\,1)^4$
  - 1111010000101000101111 $\rightsquigarrow$ *largest prime less than 1M in binary*
  - . .

1. universal schemes
   - $|\mathrm{compress}(x)| \;=$ Kolmogorov complexity $K(x)$
     not computable!
2. ressource bounded versions
   - $K^T(x)$ for some time bound T
     still very expensive to estimate!

Compression of Strings

- given a string **x** find a string $\mathbf{y = compress(x)}$ that specifies x in a more compact way
  - $01010101 \rightsquigarrow (0\,1)^4$
  - $1111010000101000101111 \rightsquigarrow$ *largest prime less than 1M in binary*
  - . .

1. universal schemes
   - $|\mathrm{compress}(x)| =$ Kolmogorov complexity $K(x)$
     not computable!
2. ressource bounded versions
   - $K^T(x)$ for some time bound T
     still very expensive to estimate!
3. more efficient, practical methods:

Compression of Strings

- given a string **x** find a string $\mathbf{y = compress(x)}$ that specifies x in a more compact way
  - $01010101 \rightsquigarrow (0\,1)^4$
  - $11110100001000101111 \rightsquigarrow$ *largest prime less than 1M in binary*
  - . .

1. universal schemes
   - $|\mathrm{compress}(x)| =$ Kolmogorov complexity $K(x)$
     not computable!
2. ressource bounded versions
   - $K^T(x)$ for some time bound T
     still very expensive to estimate!
3. more efficient, practical methods:
   - Lempel-Ziv methodology: dictionaries with pointers

Compression of Strings

- given a string **x** find a string $\mathbf{y = compress(x)}$ that specifies x in a more compact way
  - $01010101 \rightsquigarrow (0\,1)^4$
  - $1111010000101000101111 \rightsquigarrow$ *largest prime less than 1M in binary*
  - . .

1. universal schemes
   - $|\mathrm{compress}(x)| =$ Kolmogorov complexity $K(x)$
     not computable!
2. ressource bounded versions
   - $K^T(x)$ for some time bound T
     still very expensive to estimate!
3. more efficient, practical methods:
   - Lempel-Ziv methodology: dictionaries with pointers
   - simple computational model: formal grammars

Grammar-based Compression

given a string $x \in \Sigma^*$

Grammar-based Compression

given a string $x \in \Sigma^*$

1. find a context-free grammar G_x such that

Grammar-based Compression

given a string $x \in \Sigma^*$

1. find a context-free grammar G_x such that $L(G_x) = \{x\}$

Grammar-based Compression

given a string $x \in \Sigma^*$

1. find a context-free grammar G_x such that $L(G_x) = \{x\}$
2. design a suitable encoding of G_x: $\mathrm{code}(G_x)$

Grammar-based Compression

given a string $x \in \Sigma^*$

1. find a context-free grammar G_x such that $L(G_x) = \{x\}$
2. design a suitable encoding of G_x: $\mathrm{code}(G_x)$
3. **compress(x) := code(G_x)**

Grammar-based Compression

given a string $x \in \Sigma^*$

1. find a context-free grammar G_x such that $L(G_x) = \{x\}$
2. design a suitable encoding of G_x: $\mathrm{code}(G_x)$
3. **compress(x) := code(G_x)**

further requirements for G_x:

Grammar-based Compression

given a string $x \in \Sigma^*$

1. find a context-free grammar G_x such that $L(G_x) = \{x\}$
2. design a suitable encoding of G_x: $\mathrm{code}(G_x)$
3. **compress(x) := code(G_x)**

further requirements for G_x:

- deterministic and acyclic

Grammar-based Compression

given a string $x \in \Sigma^*$

1. find a context-free grammar G_x such that $L(G_x) = \{x\}$
2. design a suitable encoding of G_x: $\mathrm{code}(G_x)$
3. **compress(x) := code(G_x)**

further requirements for G_x:

- deterministic and acyclic

in short: „G_x is a grammar for x.“

Grammar-based Compression

given a string $x \in \Sigma^*$

1. find a context-free grammar G_x such that $L(G_x) = \{x\}$
2. design a suitable encoding of G_x: $\mathrm{code}(G_x)$
3. **compress(x) := code(G_x)**

further requirements for G_x:

- deterministic and acyclic

in short: „G_x is a grammar for x.“

main concern in the following: find a **small grammar G_x for x**!

Grammar-based Compression

given a string $x \in \Sigma^*$

1. find a context-free grammar G_x such that $L(G_x) = \{x\}$
2. design a suitable encoding of G_x: $\mathrm{code}(G_x)$
3. **compress(x) := code(G_x)**

further requirements for G_x:

- deterministic and acyclic

in short: „G_x is a grammar for x.“

main concern in the following: find a **small grammar G_x for x**!
coding of G_x can be done by standard methods

Example 1

greedy strategy:

iteratively replace longest subwords that appear more than once by a nonterminal

Example 1

Input

$x = \texttt{fischers} \sqcup \texttt{fritz} \sqcup \texttt{fischt} \sqcup \texttt{frische} \sqcup \texttt{fische}$

Example 1

Input

$x = \texttt{fischers}\sqcup\texttt{fritz}\sqcup\texttt{fischt}\sqcup\texttt{frische}\sqcup\texttt{fische}$

Grammar G_x for x

$\texttt{S} \rightarrow \texttt{fischers}\sqcup\texttt{fritz}\sqcup\texttt{fischt}\sqcup\texttt{frische}\sqcup\texttt{fische}$

Example 1

Input

$x = \texttt{fischers}\sqcup\texttt{fritz}\sqcup\texttt{fischt}\sqcup\texttt{frische}\sqcup\texttt{fische}$

Grammar G_x for x

$\texttt{S} \rightarrow \texttt{fischers}\sqcup\texttt{fritz}\sqcup\texttt{fischt}\sqcup\texttt{frische}\sqcup\texttt{fische}$

Example 1

Input

$x = \texttt{fischers}\sqcup\texttt{fritz}\sqcup\texttt{fischt}\sqcup\texttt{frische}\sqcup\texttt{fische}$

Grammar G_x for x

$$\begin{aligned} \texttt{S} &\longrightarrow \texttt{A}\texttt{rs}\sqcup\texttt{fritz}\sqcup\texttt{fischt}\sqcup\texttt{frische}\sqcup\texttt{A} \\ \texttt{A} &\longrightarrow \texttt{fische} \end{aligned}$$

Example 1

Input

$x = \texttt{fischers}\sqcup\texttt{fritz}\sqcup\texttt{fischt}\sqcup\texttt{frische}\sqcup\texttt{fische}$

Grammar G_x for x

$$\begin{aligned} \texttt{S} &\longrightarrow \texttt{Ars}\sqcup\texttt{fritz}\sqcup\texttt{fischt}\sqcup\texttt{frische}\sqcup\texttt{A} \\ \texttt{A} &\longrightarrow \texttt{fische} \end{aligned}$$

Example 1

Input

$x = \texttt{fischers} \sqcup \texttt{fritz} \sqcup \texttt{fischt} \sqcup \texttt{frische} \sqcup \texttt{fische}$

Grammar G_x for x

$$\begin{aligned} \texttt{S} &\longrightarrow \texttt{Ars} \sqcup \texttt{fritz} \sqcup \texttt{Bt} \sqcup \texttt{frische} \sqcup \texttt{A} \\ \texttt{A} &\longrightarrow \texttt{Be} \\ \texttt{B} &\longrightarrow \texttt{fisch} \end{aligned}$$

Example 1

Input

$x = \texttt{fischers}\sqcup\texttt{fritz}\sqcup\texttt{fischt}\sqcup\texttt{frische}\sqcup\texttt{fische}$

Grammar G_x for x

$$\begin{aligned} \texttt{S} &\longrightarrow \texttt{Ars}\sqcup\texttt{fritz}\sqcup\texttt{Bt}\sqcup\texttt{frische}\sqcup\texttt{A} \\ \texttt{A} &\longrightarrow \texttt{Be} \\ \texttt{B} &\longrightarrow \texttt{fisch} \end{aligned}$$

Example 1

Input

$x = \texttt{fischers} \sqcup \texttt{fritz} \sqcup \texttt{fischt} \sqcup \texttt{frische} \sqcup \texttt{fische}$

Grammar G_x for x

$$
\begin{aligned}
\texttt{S} &\longrightarrow \texttt{ArsCtz} \sqcup \texttt{BtCsche} \sqcup \texttt{A} \\
\texttt{A} &\longrightarrow \texttt{Be} \\
\texttt{B} &\longrightarrow \texttt{fisch} \\
\texttt{C} &\longrightarrow \sqcup\texttt{fri}
\end{aligned}
$$

Example 1

Input

$x = \texttt{fischers} \sqcup \texttt{fritz} \sqcup \texttt{fischt} \sqcup \texttt{frische} \sqcup \texttt{fische}$

Grammar G_x for x

$$
\begin{array}{lcl}
\texttt{S} & \longrightarrow & \texttt{ArsCtz} \sqcup \texttt{BtCsche} \sqcup \texttt{A} \\
\texttt{A} & \longrightarrow & \texttt{Be} \\
\texttt{B} & \longrightarrow & \texttt{fisch} \\
\texttt{C} & \longrightarrow & \sqcup\texttt{fri}
\end{array}
$$

Example 1

Input

$x = \texttt{fischers}\sqcup\texttt{fritz}\sqcup\texttt{fischt}\sqcup\texttt{frische}\sqcup\texttt{fische}$

Grammar G_x for x

$$
\begin{array}{lcl}
\texttt{S} & \longrightarrow & \texttt{ArsCtz}\sqcup\texttt{BtCDe}\sqcup\texttt{A} \\
\texttt{A} & \longrightarrow & \texttt{Be} \\
\texttt{B} & \longrightarrow & \texttt{fiD} \\
\texttt{C} & \longrightarrow & \sqcup\texttt{fri} \\
\texttt{D} & \longrightarrow & \texttt{sch}
\end{array}
$$

Example 1

Input

$x = \texttt{fischers} \sqcup \texttt{fritz} \sqcup \texttt{fischt} \sqcup \texttt{frische} \sqcup \texttt{fische}$

Grammar G_x for x

$$
\begin{array}{lcl}
\texttt{S} & \longrightarrow & \texttt{ArsCtz} \sqcup \texttt{BtCDe} \sqcup \texttt{A} \\
\texttt{A} & \longrightarrow & \texttt{Be} \\
\texttt{B} & \longrightarrow & \texttt{fiD} \\
\texttt{C} & \longrightarrow & \sqcup\texttt{fri} \\
\texttt{D} & \longrightarrow & \texttt{sch}
\end{array}
$$

Compression Rate

$|x| = 36$

Example 1

Input

$x = \texttt{fischers} \sqcup \texttt{fritz} \sqcup \texttt{fischt} \sqcup \texttt{frische} \sqcup \texttt{fische}$

Grammar G_x for x

$$
\begin{array}{lcl}
\texttt{S} & \longrightarrow & \texttt{ArsCtz} \sqcup \texttt{BtCDe} \sqcup \texttt{A} \\
\texttt{A} & \longrightarrow & \texttt{Be} \\
\texttt{B} & \longrightarrow & \texttt{fiD} \\
\texttt{C} & \longrightarrow & \sqcup\texttt{fri} \\
\texttt{D} & \longrightarrow & \texttt{sch}
\end{array}
$$

Compression Rate

$$|x| = 36 \quad \rightsquigarrow \quad |G_x| = 26$$

Optimization Problem
Minimum Grammar Compression

MGC

Optimization Problem
Minimum Grammar Compression

MGC

- input: $x \in \Sigma^*$

Optimization Problem
Minimum Grammar Compression

MGC

- input: $x \in \Sigma^*$
- solution: grammar G_x for x

Optimization Problem
Minimum Grammar Compression

MGC

- input: $x \in \Sigma^*$
- solution: grammar G_x for x
- goal: minimize

$$|G_x| := \sum_{A \to \alpha \text{ in } G_x} |\alpha|$$

Optimization Problem
Minimum Grammar Compression

MGC

- input: $x \in \Sigma^*$
- solution: grammar G_x for x
- goal: minimize

$$|\mathbf{G_x}| := \sum_{\mathbf{A \to \alpha \text{ in } G_x}} |\alpha|$$

- notation: $\mathbf{m^*(x)} := \min |G_x|$

Optimization Problem
Minimum Grammar Compression

k-MGC bounded alphabet size

- input: $x \in \Sigma^*$, $|\Sigma| \leq k$
- solution: grammar G_x for x
- goal: minimize

$$|\mathbf{G_x}| := \sum_{\mathbf{A \to \alpha \text{ in } G_x}} |\alpha|$$

- notation: $\mathbf{m^*(x)} := \min |G_x|$

Optimization Problem
Minimum Grammar Compression

2-MGC binary alphabet

- input: $x \in \Sigma^*$, $|\Sigma| = 2$
- solution: grammar G_x for x
- goal: minimize

$$|\mathbf{G_x}| := \sum_{\mathbf{A \to \alpha \text{ in } G_x}} |\alpha|$$

- notation: $\mathbf{m^*(x)} := \min |G_x|$

Known Results about MGC

- $m^*(x) \geq \Omega(\log |x|)$

Known Results about MGC

- $m^*(x) \geq \Omega(\log |x|)$
- G_x is a grammar for x of size $m \implies \forall\, \ell \in [1, \ldots, |x|]$ number of diff. substrings of length ℓ in x is at most $m \cdot \ell$ [Lehman 2002]

Known Results about MGC

- $m^*(x) \geq \Omega(\log |x|)$
- G_x is a grammar for x of size $m \implies \forall\, \ell \in [1, \ldots, |x|]$ number of diff. substrings of length ℓ in x is at most $m \cdot \ell$ [Lehman 2002]
- MGC is $\mathcal{NP}$-complete [Storer 1977]

Known Results about MGC

- $m^*(x) \geq \Omega(\log |x|)$
- G_x is a grammar for x of size $m \implies \forall\, \ell \in [1, \ldots, |x|]$ number of diff. substrings of length ℓ in x is at most $m \cdot \ell$ [Lehman 2002]
- MGC is $\mathcal{NP}$-complete [Storer 1977]
- k-MGC is $\mathcal{NP}$-complete for $k \geq 3$

Known Results about MGC

- $m^*(x) \geq \Omega(\log |x|)$
- G_x is a grammar for x of size $m \implies \forall\, \ell \in [1, \ldots, |x|]$ number of diff. substrings of length ℓ in x is at most $m \cdot \ell$ [Lehman 2002]
- MGC is $\mathcal{NP}$-complete [Storer 1977]
- k-MGC is $\mathcal{NP}$-complete for $k \geq 3$

- Open question: is 2-MGC $\mathcal{NP}$-complete as well?

Known Results about MGC

- $m^*(x) \geq \Omega(\log |x|)$
- G_x is a grammar for x of size $m \implies \forall\, \ell \in [1, \ldots, |x|]$ number of diff. substrings of length ℓ in x is at most $m \cdot \ell$ [Lehman 2002]
- MGC is $\mathcal{NP}$-complete [Storer 1977]
- k-MGC is $\mathcal{NP}$-complete for $k \geq 3$

- Open question: is 2-MGC $\mathcal{NP}$-complete as well?
- $\implies$ transformation between alphabets

Know Results: Approximability

- several greedy strategies have been proposed
  performance can be very bad for specific examples

Know Results: Approximability

- several greedy strategies have been proposed
  performance can be very bad for specific examples
- MGC is approximable with a factor $\mathbf{O(\log |x|)}$
  [Charikar, Lehman, Liu, Panigraphy, Prabhakaran, Rasala, Sahai, Shelat: STOC'02] + [Rytter: CPM'02]

Know Results: Approximability

- several greedy strategies have been proposed
  performance can be very bad for specific examples
- MGC is approximable with a factor $\mathbf{O(\log |x|)}$
  [Charikar, Lehman, Liu, Panigraphy, Prabhakaran, Rasala, Sahai, Shelat: STOC'02] + [Rytter: CPM'02]
- Open question:
  – better upper bounds – even for simpler problems?
  – improvements for k-MGC?

Know Results: Approximability

- several greedy strategies have been proposed
  performance can be very bad for specific examples
- MGC is approximable with a factor $\mathbf{O(\log |x|)}$
  [Charikar, Lehman, Liu, Panigraphy, Prabhakaran, Rasala, Sahai, Shelat: STOC'02] + [Rytter: CPM'02]
- Open question:
  – better upper bounds – even for simpler problems?
  – improvements for k-MGC?
- MGC is not $\mathbf{1 + \frac{1}{8568}}$-approximable (unless $\mathcal{P} = \mathcal{NP}$)
  [Lehman, Shelat: SODA'02]

Know Results: Approximability

- several greedy strategies have been proposed
  performance can be very bad for specific examples
- MGC is approximable with a factor $\mathbf{O(\log |x|)}$
  [Charikar, Lehman, Liu, Panigraphy, Prabhakaran, Rasala, Sahai, Shelat: STOC'02] + [Rytter: CPM'02]
- Open question:
  – better upper bounds – even for simpler problems?
  – improvements for k-MGC?
- MGC is not $\mathbf{1 + \frac{1}{8568}}$-approximable (unless $\mathcal{P} = \mathcal{NP}$)
  [Lehman, Shelat: SODA'02]
- Open question:
  – lower bound for k-MGC?
  – nonconstant lower bound for MGC?

Transformation between Alphabets

How do compression properties transfer between alphabets?

- Γ arbitrary alphabet, Σ alphabet of size k

Transformation between Alphabets

How do compression properties transfer between alphabets?

- Γ arbitrary alphabet, Σ alphabet of size k
- $\varphi : \Gamma^* \longrightarrow \Sigma^*$ is an ℓ-blockcode

Transformation between Alphabets

How do compression properties transfer between alphabets?

- Γ arbitrary alphabet, Σ alphabet of size k
- $\varphi : \Gamma^* \longrightarrow \Sigma^*$ is an ℓ-blockcode
  - iff homomorphism + injective

Transformation between Alphabets

How do compression properties transfer between alphabets?

- Γ arbitrary alphabet, Σ alphabet of size k
- $\varphi : \Gamma^* \longrightarrow \Sigma^*$ is an ℓ-blockcode
  - iff homomorphism + injective
  - $|\varphi(\gamma)| = \ell$ for all $\gamma \in \Gamma$

Transformation between Alphabets

How do compression properties transfer between alphabets?

- Γ arbitrary alphabet, Σ alphabet of size k
- $\varphi : \Gamma^* \longrightarrow \Sigma^*$ is an ℓ-blockcode
  - iff homomorphism + injective
  - $|\varphi(\gamma)| = \ell$ for all $\gamma \in \Gamma$
- Question:

Transformation between Alphabets

How do compression properties transfer between alphabets?

- Γ arbitrary alphabet, Σ alphabet of size k
- $\varphi : \Gamma^* \longrightarrow \Sigma^*$ is an ℓ-blockcode
  - iff homomorphism + injective
  - $|\varphi(\gamma)| = \ell$ for all $\gamma \in \Gamma$
- Question:
  - relations between G_x and $G_{\varphi(x)}$?

Transformation between Alphabets

How do compression properties transfer between alphabets?

- Γ arbitrary alphabet, Σ alphabet of size k
- $\varphi : \Gamma^* \longrightarrow \Sigma^*$ is an ℓ-blockcode
  - iff homomorphism + injective
  - $|\varphi(\gamma)| = \ell$ for all $\gamma \in \Gamma$
- Question:
  - relations between G_x and $G_{\varphi(x)}$?
  - in particular, $m^*(x) \leftrightarrow m^*(\varphi(x))$?

From G_x to $G_{\varphi(x)}$

Example 2

string	*transformation*
$x = \mathtt{abcaabc}$	$\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
	$\varphi(\mathtt{c}) = 0100$

From G_x to $G_{\varphi(x)}$

Example 2

string
$x = \mathtt{abcaabc}$

transformation
$\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
$\varphi(\mathtt{c}) = 0100$

G_x :

$$\mathtt{S} \rightarrow \mathtt{PaP}$$
$$\mathtt{P} \rightarrow \mathtt{abc}$$

From G_x to $G_{\varphi(x)}$

Example 2

string
$x = \mathtt{abcaabc}$

transformation
$\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
$\varphi(\mathtt{c}) = 0100$

G_x :	S	$\rightarrow$	PaP	S_a	$\rightarrow$	1T1
	P	$\rightarrow$	abc	S_b	$\rightarrow$	T10
				S_c	$\rightarrow$	01T
				T	$\rightarrow$	00

From G_x to $G_{\varphi(x)}$

Example 2

string
$x = \mathtt{abcaabc}$

transformation
$\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
$\varphi(\mathtt{c}) = 0100$

G_x :	S	$\rightarrow$	PaP	S_a	$\rightarrow$	1T1
	P	$\rightarrow$	abc	S_b	$\rightarrow$	T10
				S_c	$\rightarrow$	01T
				T	$\rightarrow$	00

$G_{\varphi(x)}$:	S	$\rightarrow$	$\mathtt{PS_aP}$	$\mathtt{S_a}$	$\rightarrow$	1T1
	P	$\rightarrow$	$\mathtt{S_aS_bS_c}$	$\mathtt{S_b}$	$\rightarrow$	T10
				$\mathtt{S_c}$	$\rightarrow$	01T
				T	$\rightarrow$	00

From G_x to $G_{\varphi(x)}$

Example 2

string
$x = \mathtt{abcaabc}$

transformation
$\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
$\varphi(\mathtt{c}) = 0100$

G_x :	S	$\rightarrow$	PaP	S_a	$\rightarrow$	1T1
	P	$\rightarrow$	abc	S_b	$\rightarrow$	T10
				S_c	$\rightarrow$	01T
				T	$\rightarrow$	00

$G_{\varphi(x)}$:	S	$\rightarrow$	PS_aP	S_a	$\rightarrow$	1T1
	P	$\rightarrow$	$S_aS_bS_c$	S_b	$\rightarrow$	T10
				S_c	$\rightarrow$	01T
				T	$\rightarrow$	00

$$|G_{\varphi(x)}| = |G_x| + |G_\varphi| = 6 + 11 = 17$$

From G_x to $G_{\varphi(x)}$

Theorem

G_x grammar for x, G_φ grammar for $\varphi(\Gamma)$

From G_x to $G_{\varphi(x)}$

Theorem

G_x grammar for x, G_φ grammar for $\varphi(\Gamma)$
$\Longrightarrow$ $G_{\varphi(x)} = G_x \cup G_\varphi$ is a grammar for $\varphi(x)$, that means

$$|G_{\varphi(x)}| \leq |G_x| + |G_\varphi|$$

From G_x to $G_{\varphi(x)}$

Theorem

G_x grammar for x, G_φ grammar for $\varphi(\Gamma)$
$\Longrightarrow$ $G_{\varphi(x)} = G_x \cup G_\varphi$ is a grammar for $\varphi(x)$, that means

$$|G_{\varphi(x)}| \leq |G_x| + |G_\varphi|$$

$$\mathbf{m^*(\varphi(x)) \ \leq \ m^*(x) + m^*(\varphi(\Gamma))}$$

From $G_{\varphi(x)}$ to G_x

Example 3

$x = \mathtt{abcaabc}$ $\qquad$ $\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
$\varphi(\mathtt{c}) = 0100$

$G_{\varphi(x)}$:

S	$\longrightarrow$	PAP		A	$\longrightarrow$	1T1
P	$\longrightarrow$	ABC		B	$\longrightarrow$	T10
				C	$\longrightarrow$	01T
				T	$\longrightarrow$	00

From $G_{\varphi(x)}$ to G_x

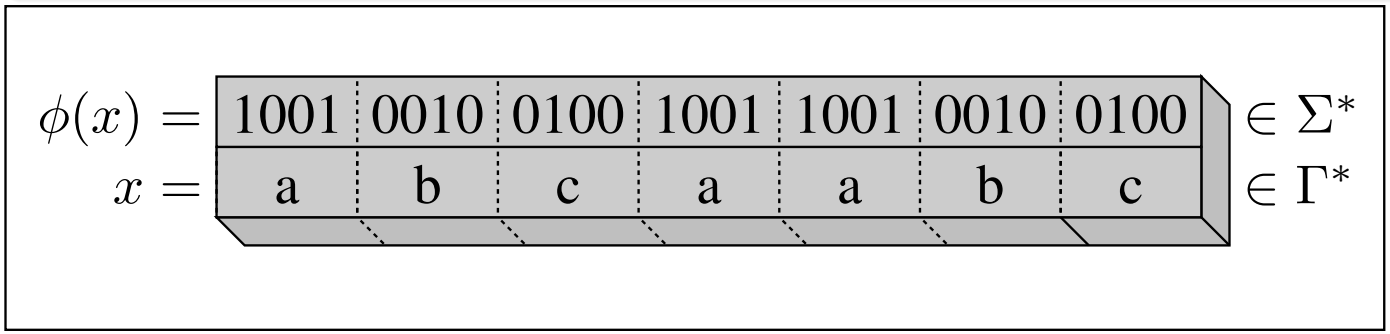
Example 3

$x = \mathtt{abcaabc}$

$\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
$\varphi(\mathtt{c}) = 0100$

$G_{\varphi(x)}$:

S	$\longrightarrow$	PAP	
P	$\longrightarrow$	ABC	
A	$\longrightarrow$	1T1	
B	$\longrightarrow$	T10	
C	$\longrightarrow$	01T	
T	$\longrightarrow$	00	



From $G_{\varphi(x)}$ to G_x

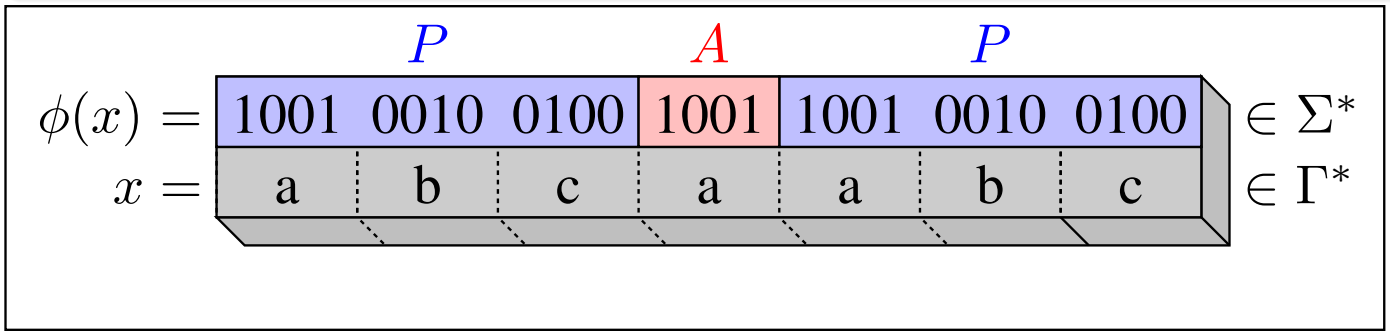
Example 3

$x = \mathtt{abcaabc}$

$\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
$\varphi(\mathtt{c}) = 0100$

$G_{\varphi(x)}$:

S	$\longrightarrow$	PAP	
P	$\longrightarrow$	ABC	
A	$\longrightarrow$	1T1	
B	$\longrightarrow$	T10	
C	$\longrightarrow$	01T	
T	$\longrightarrow$	00	

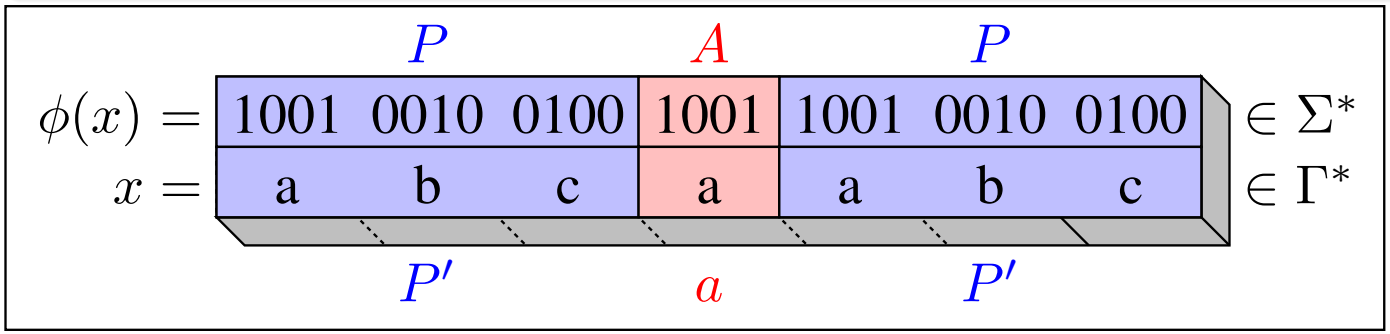


From $G_{\varphi(x)}$ to G_x

Example 3

$x = \mathtt{abcaabc}$ $\qquad$ $\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
$\varphi(\mathtt{c}) = 0100$

G_x: $\mathtt{S}' \rightarrow \mathtt{P}'\mathtt{a}\mathtt{P}'$
$\mathtt{P}' \rightarrow \mathtt{abc}$



From $G_{\varphi(x)}$ to G_x

Example 3

$x = \mathtt{abcaabc}$ $\qquad$ $\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
$\varphi(\mathtt{c}) = 0100$

$$
\begin{array}{lcl}
G_x: \quad \mathtt{S}' & \to & \mathtt{P}'\mathtt{a}\mathtt{P}' \\
\mathtt{P}' & \to & \mathtt{abc}
\end{array}
$$

$$|G_x| = |G_{\varphi(x)}| - |G_\varphi| = 17 - 11 = 6$$

From $G_{\varphi(x)}$ to G_x

Example 3

$x = \mathtt{abcaabc}$ $\qquad$ $\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
$\varphi(\mathtt{c}) = 0100$

G_x: $\mathtt{S}' \rightarrow \mathtt{P}'\mathtt{a}\mathtt{P}'$
$\mathtt{P}' \rightarrow \mathtt{abc}$

$$|G_x| = |G_{\varphi(x)}| - |G_\varphi| = 17 - 11 = 6$$

? always true ?

$$|G_x| \leq |G_{\varphi(x)}| - m^*(\varphi(\Gamma)) \quad ?$$

From $G_{\varphi(x)}$ to G_x

Example 4

$x = \mathtt{abcaabc}$ $\qquad$ $\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
$\varphi(\mathtt{c}) = 0100$

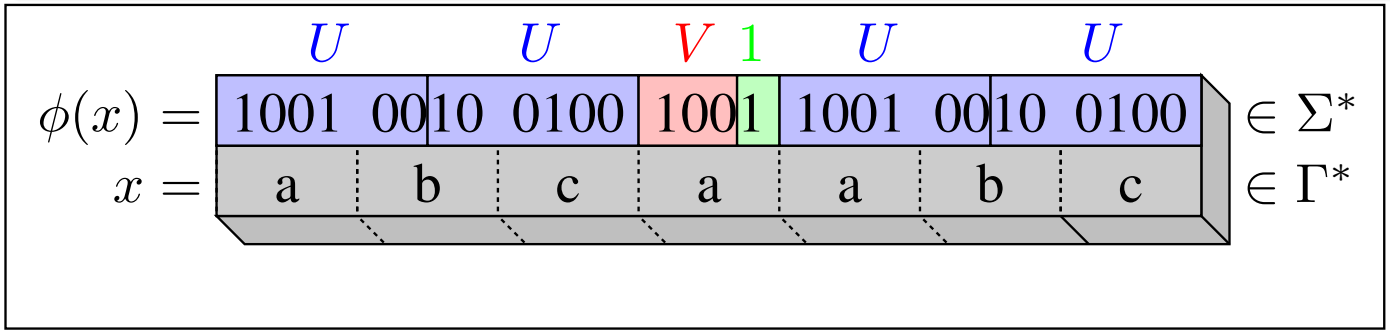
$G_{\varphi(x)}$: $\quad \mathtt{S} \to \mathtt{UUV1UU} \qquad \mathtt{U} \to \mathtt{VV} \qquad \mathtt{V} \to 100$

From $G_{\varphi(x)}$ to G_x

Example 4

$x = \mathtt{abcaabc}$ $\qquad$ $\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
$\varphi(\mathtt{c}) = 0100$

$G_{\varphi(x)}$: $\mathtt{S} \to \mathtt{UUV1UU}$ $\qquad$ $\mathtt{U} \to \mathtt{VV}$
$\mathtt{V} \to 100$



From $G_{\varphi(x)}$ to G_x

Example 4

$x = \mathtt{abcaabc}$

$\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
$\varphi(\mathtt{c}) = 0100$

$G_{\varphi(x)}$: $\mathtt{S} \longrightarrow \mathtt{UUV1UU}$

$\mathtt{U} \longrightarrow \mathtt{VV}$
$\mathtt{V} \longrightarrow 100$

$$|G_{\varphi(x)}| - m^*(\varphi(\Gamma)) = 11 - 11 = 0$$

From $G_{\varphi(x)}$ to G_x

Example 4

$x = \mathtt{abcaabc}$ $\qquad$ $\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
$\varphi(\mathtt{c}) = 0100$

$G_{\varphi(x)}$: $\quad \mathtt{S} \to \mathtt{UUV1UU} \qquad \mathtt{U} \to \mathtt{VV} \qquad \mathtt{V} \to 100$

$$|G_{\varphi(x)}| - m^*(\varphi(\Gamma)) = 11 - 11 = 0$$

$$|G_x| = 6 \quad ???$$

From $G_{\varphi(x)}$ to G_x

Example 4

$x = \mathtt{abcaabc}$ $\qquad$ $\varphi(\mathtt{a}) = 1001,\ \varphi(\mathtt{b}) = 0010$
$\varphi(\mathtt{c}) = 0100$

$G_{\varphi(x)}$: $\mathtt{S} \to \mathtt{UUV1UU}$ $\qquad$ $\mathtt{U} \to \mathtt{VV}$
$\mathtt{V} \to 100$

$$|G_{\varphi(x)}| - m^*(\varphi(\Gamma)) = 11 - 11 = 0$$

$$|G_x| = 6 \quad ???$$

$$|G_x| \leq |G_{\varphi(x)}| \quad ???$$

From $G_{\varphi(x)}$ to G_x: lower bound

Small alphabets may compress much better

Theorem

For every ℓ there exists an alphabet Γ of size 2^ℓ,
a binary blockcode $\varphi : \Gamma^* \longrightarrow \{0,1\}^*$ and
a string $x \in \Gamma^*$ of size about $2^{2\ell}$ such that

$$\mathbf{m^*(x) \ \geq \ \Omega(\ell \cdot m^*(\varphi(x)))}$$

From $G_{\varphi(x)}$ to G_x: lower bound

Small alphabets may compress much better

Theorem

For every ℓ there exists an alphabet Γ of size 2^ℓ,
a binary blockcode $\varphi : \Gamma^* \longrightarrow \{0,1\}^*$ and
a string $x \in \Gamma^*$ of size about $2^{2\ell}$ such that

$$\mathbf{m^*(x) \ \geq \ \Omega(\ell \cdot m^*(\varphi(x)))}$$

$$m^*(\varphi(x)) \ \leq \ O(2^\ell), \qquad m^*(\varphi(\Gamma)) \ \leq \ O(2^\ell), \qquad m^*(x) \ \geq \ \Omega(\ell \cdot 2^\ell)$$

From $G_{\varphi(x)}$ to G_x: upper bound

This loss of compression is the worst for ℓ-blockcodes

Theorem

If φ is an ℓ-blockcode and $G_{\varphi(x)}$ a grammar for a coded string $\varphi(x)$

$\Longrightarrow \exists$ grammar G_x for x with $|G_x| \leq 2 \cdot \ell \cdot |G_{\varphi(x)}|$

From $G_{\varphi(x)}$ to G_x: upper bound

This loss of compression is the worst for ℓ-blockcodes

Theorem

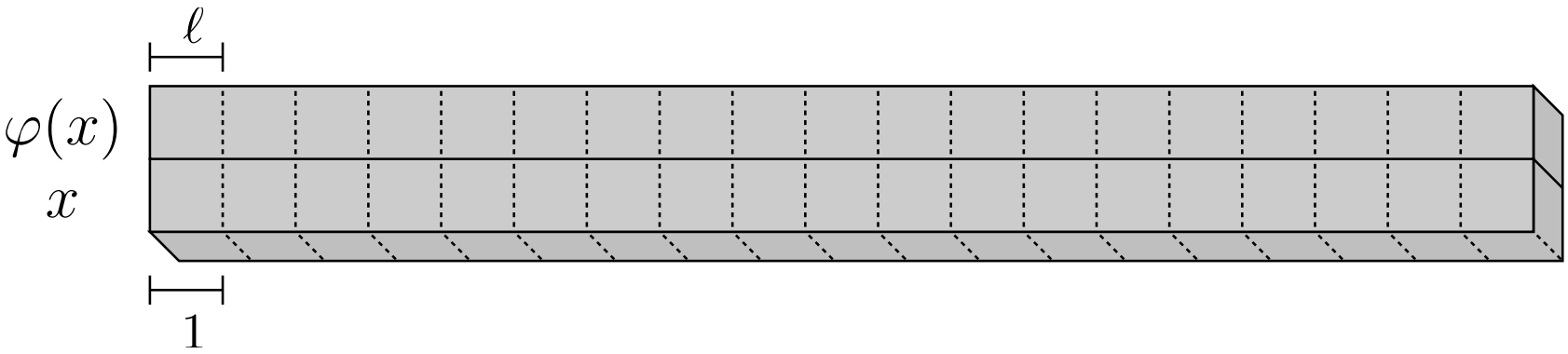
If φ is an ℓ-blockcode and $G_{\varphi(x)}$ a grammar for a coded string $\varphi(x)$

$$\Longrightarrow \exists \text{ grammar } G_x \text{ for } x \text{ with } |G_x| \leq 2 \cdot \ell \cdot |G_{\varphi(x)}|$$

$$\mathbf{m^*(x) \leq 2 \cdot \ell \cdot m^*(\varphi(x)) \quad \forall\, x}$$

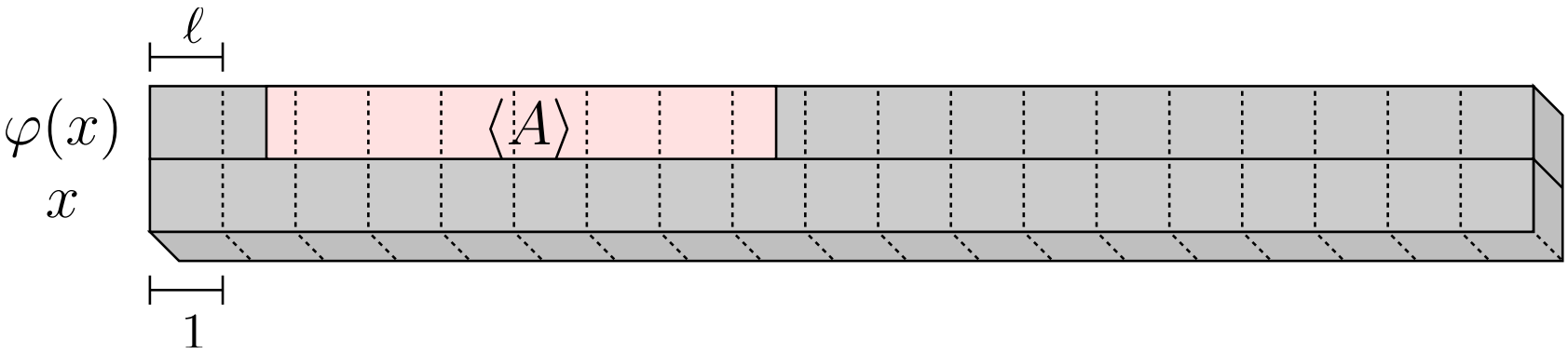
From $G_{\varphi(x)}$ to G_x: upper bound - idea of proof

in $G_{\varphi(x)}$: $A \to \alpha$



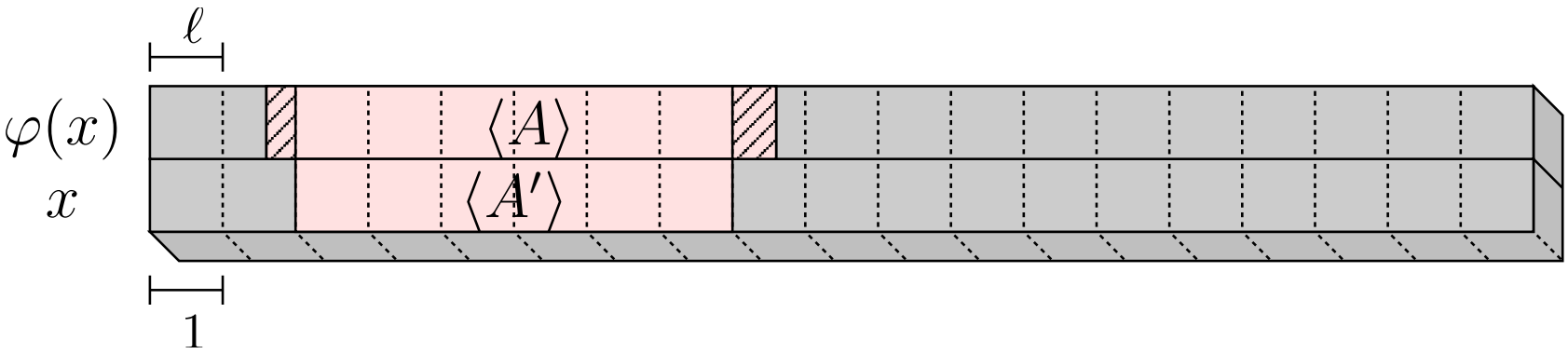
From $G_{\varphi(x)}$ to G_x: upper bound - idea of proof

in $G_{\varphi(x)}$: $A \to \alpha$



From $G_{\varphi(x)}$ to G_x: upper bound - idea of proof

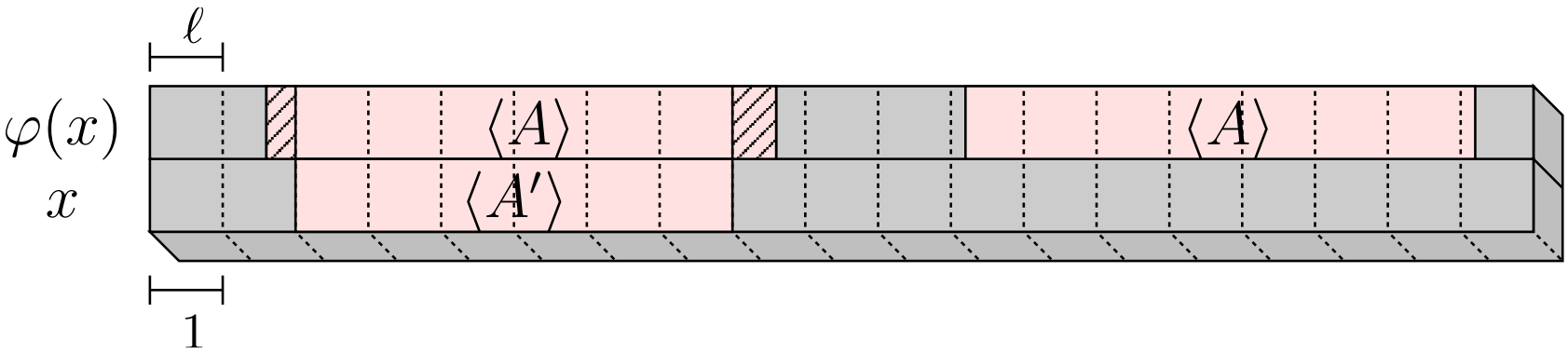
in $G_{\varphi(x)}$: $A \to \alpha$



in G_x: $A' \to \alpha'$

From $G_{\varphi(x)}$ to G_x: upper bound - idea of proof

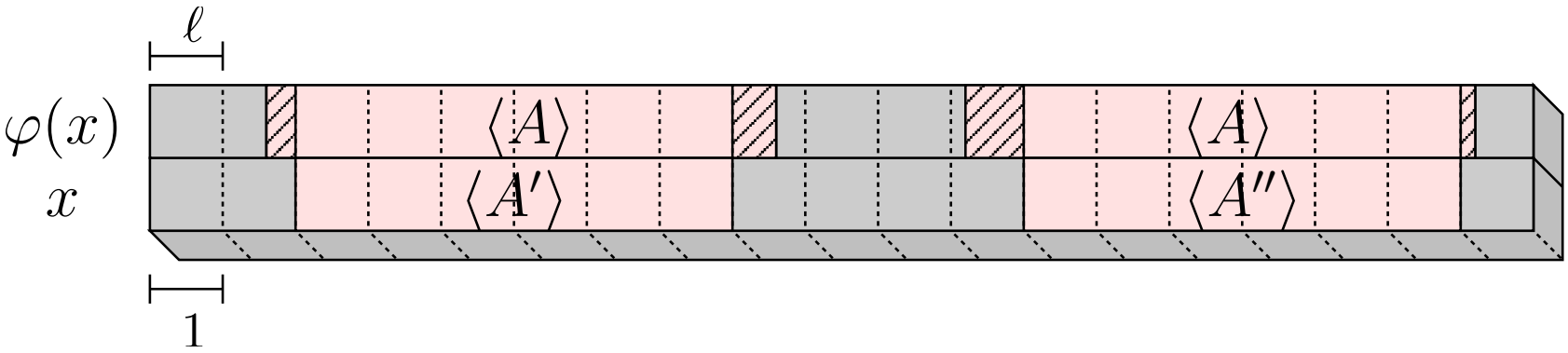
in $G_{\varphi(x)}$: $A \to \alpha$



in G_x: $A' \to \alpha'$

From $G_{\varphi(x)}$ to G_x: upper bound - idea of proof

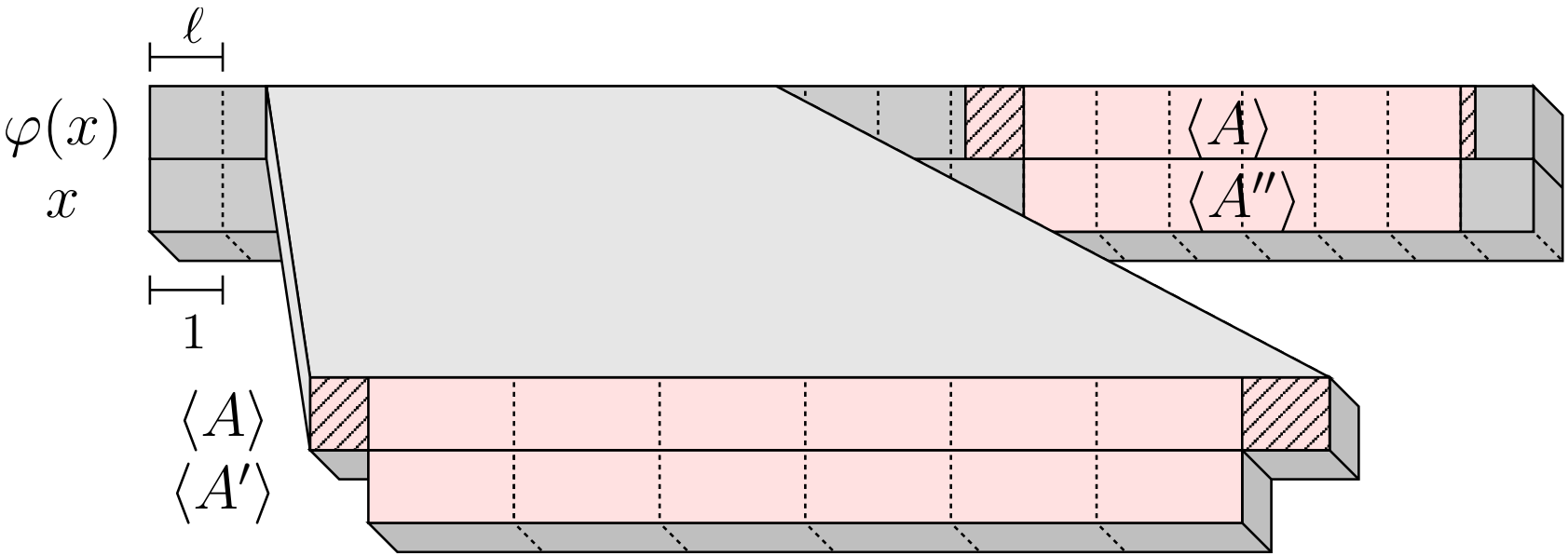
in $G_{\varphi(x)}$: $A \to \alpha$



in G_x: $A' \to \alpha'$ $A'' \to \alpha''$ ... max. ℓ copies

From $G_{\varphi(x)}$ to G_x: upper bound - idea of proof

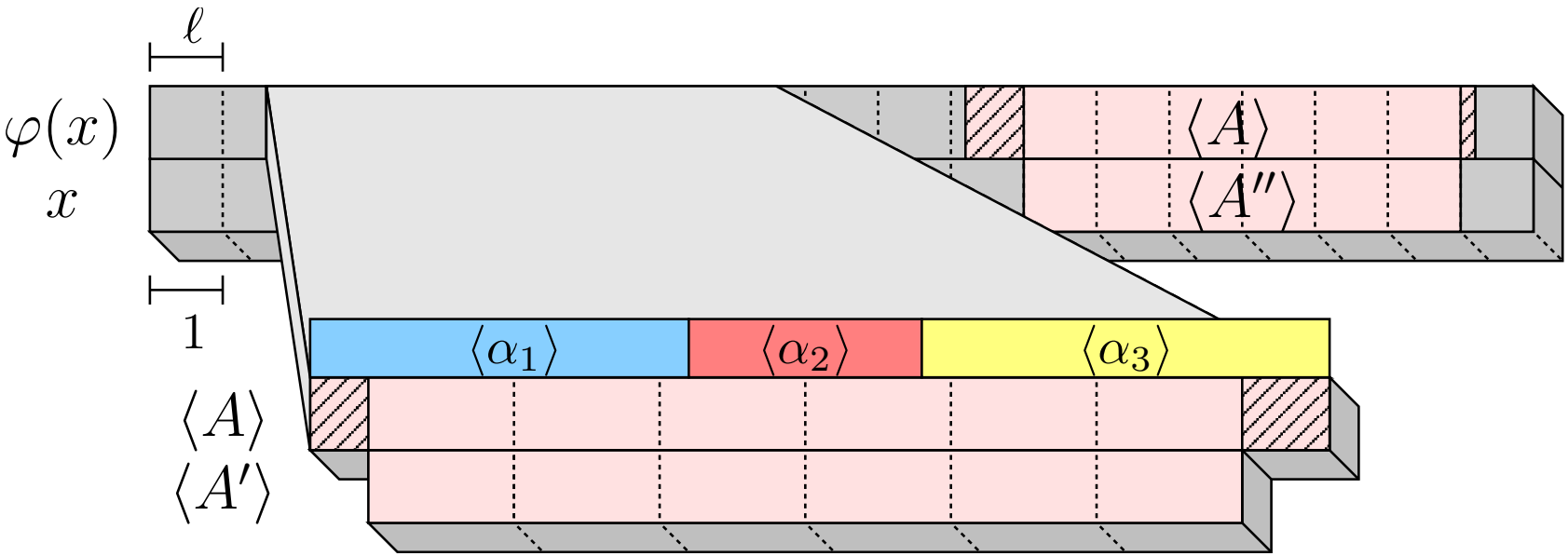
in $G_{\varphi(x)}$: $A \to \alpha$



in G_x: $A' \to \alpha'$

From $G_{\varphi(x)}$ to G_x: upper bound - idea of proof

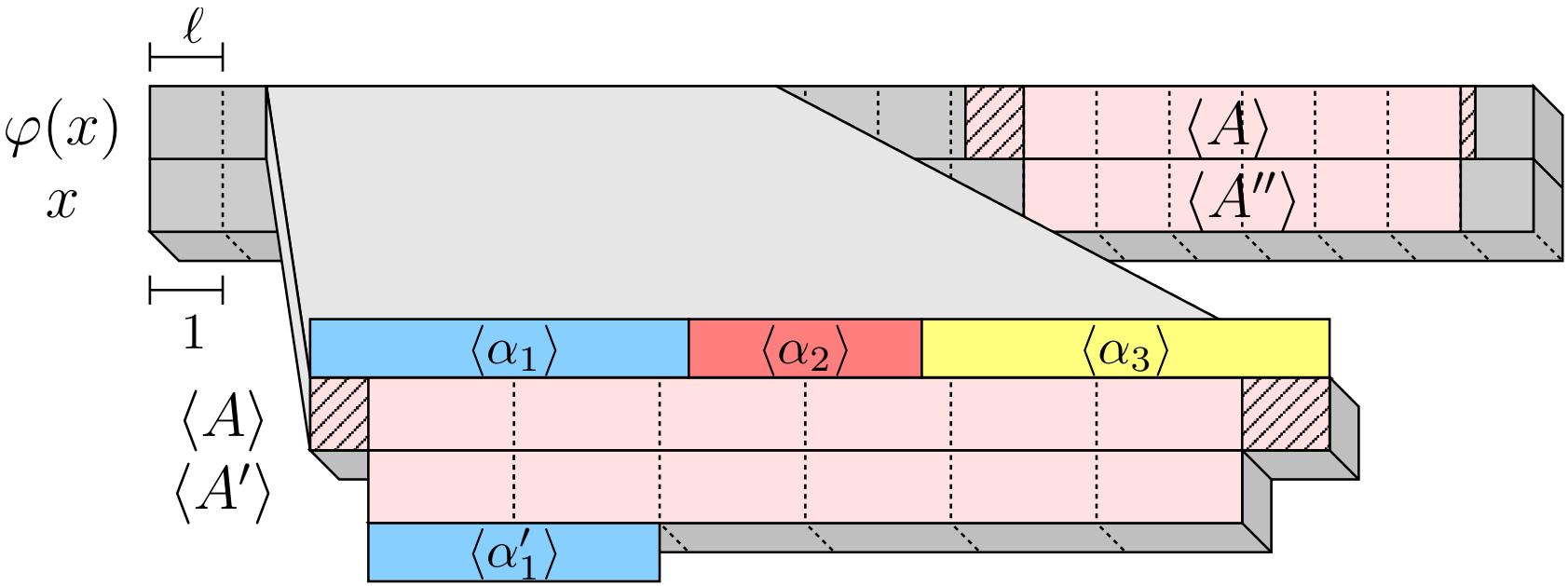
in $G_{\varphi(x)}$: $A \to \alpha = \alpha_1\alpha_2\alpha_3$



in G_x: $A' \to \alpha'$

From $G_{\varphi(x)}$ to G_x: upper bound - idea of proof

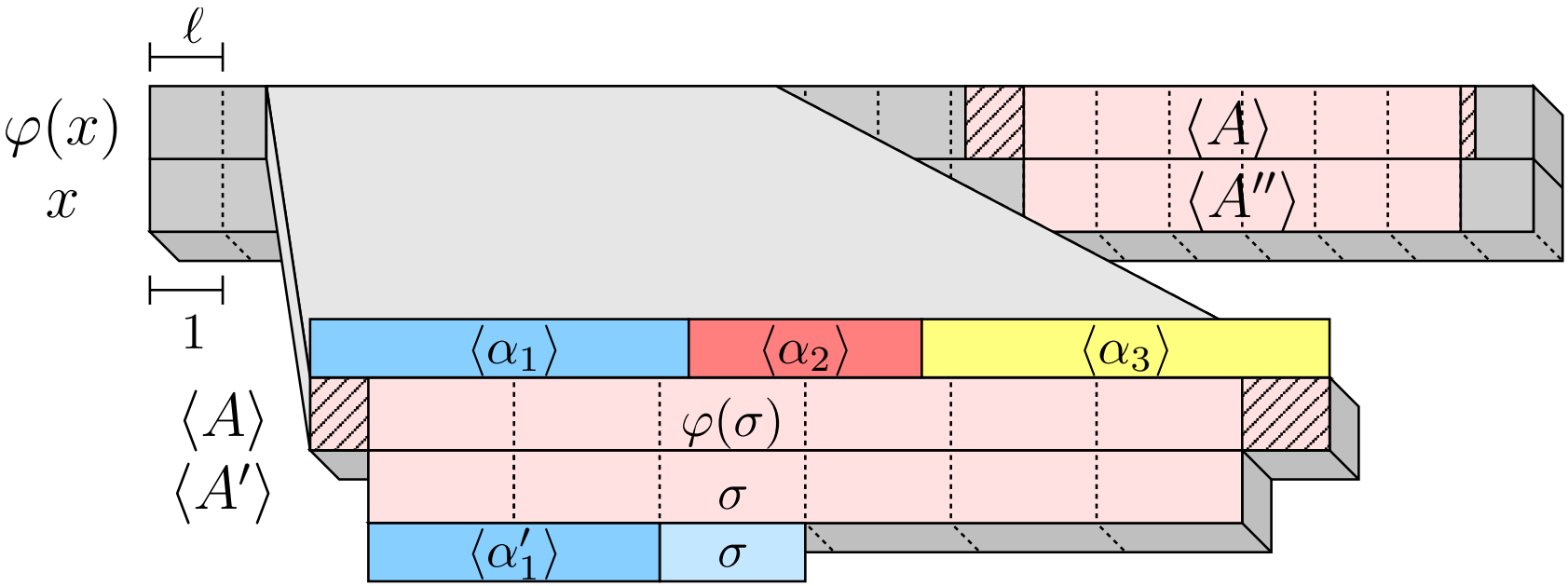
in $G_{\varphi(x)}$: $A \to \alpha = \alpha_1\alpha_2\alpha_3$



in G_x: $A' \to \alpha' = \alpha'_1$

From $G_{\varphi(x)}$ to G_x: upper bound - idea of proof

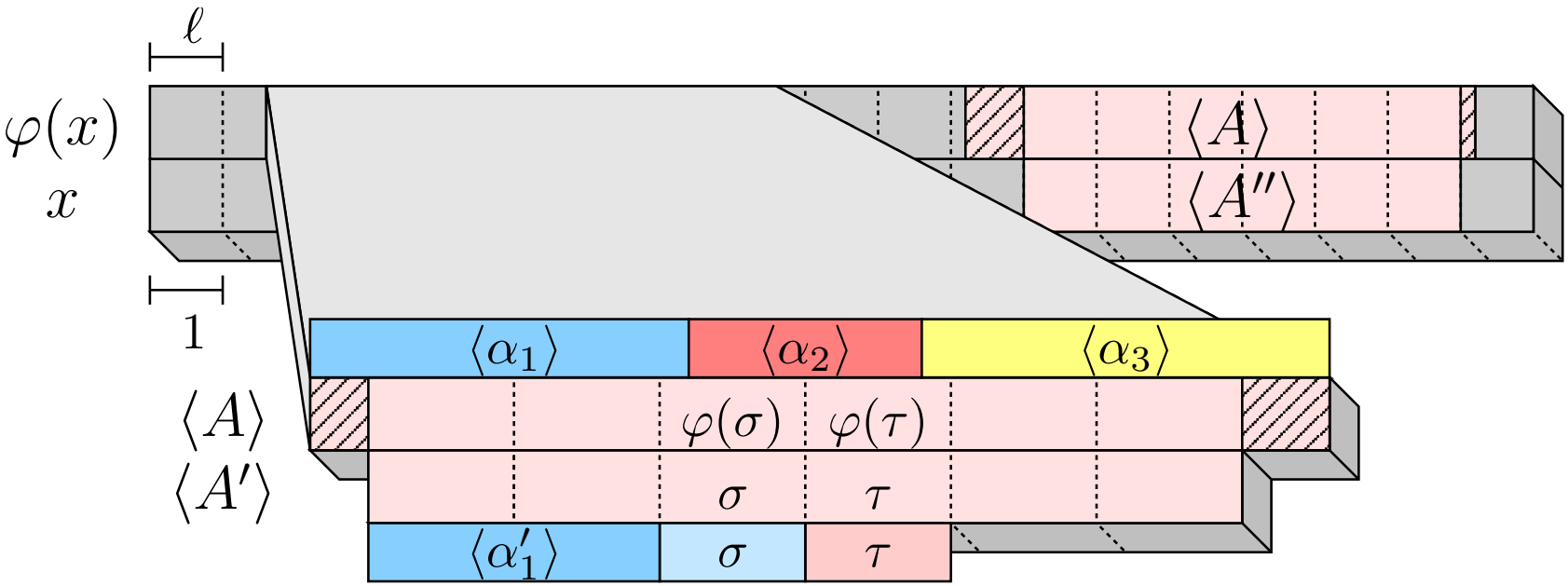
in $G_{\varphi(x)}$: $A \to \alpha = \alpha_1\alpha_2\alpha_3$



in G_x: $A' \to \alpha' = \alpha'_1\sigma$

From $G_{\varphi(x)}$ to G_x: upper bound - idea of proof

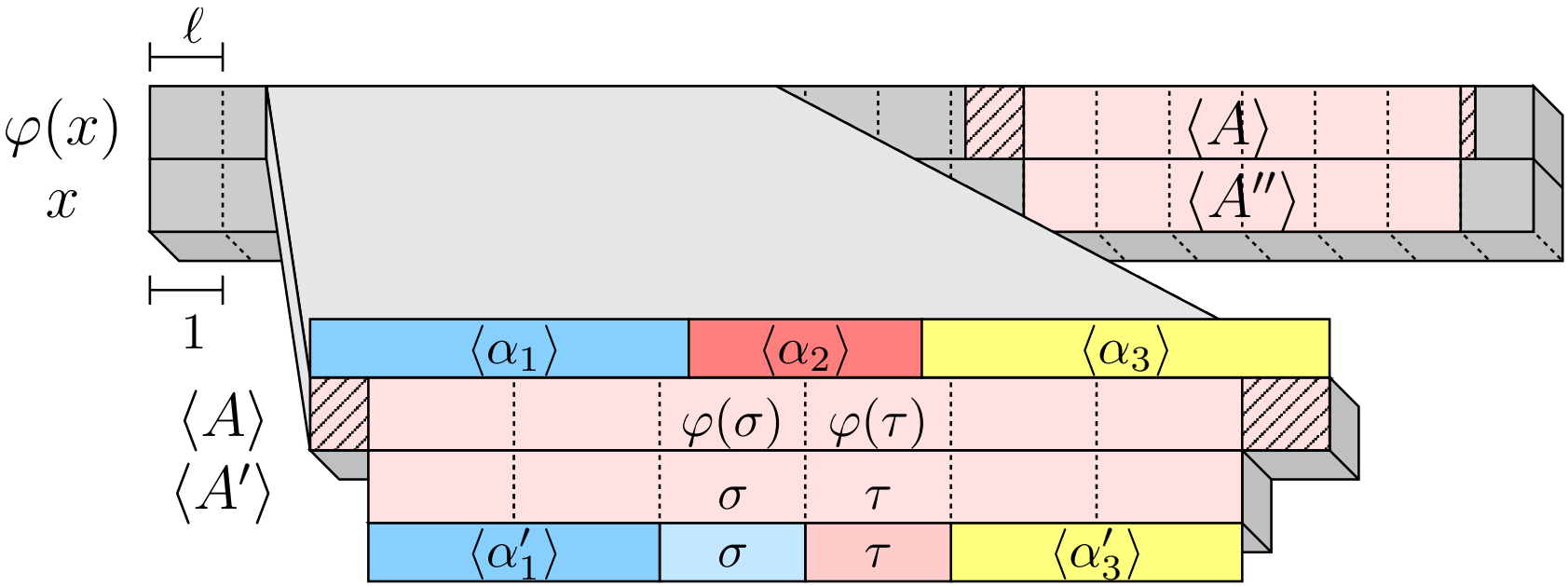
in $G_{\varphi(x)}$: $A \to \alpha = \alpha_1\alpha_2\alpha_3$



in G_x: $A' \to \alpha' = \alpha'_1\sigma\tau$

From $G_{\varphi(x)}$ to G_x: upper bound - idea of proof

in $G_{\varphi(x)}$: $A \to \alpha = \alpha_1\alpha_2\alpha_3$



in G_x: $A' \to \alpha' = \alpha'_1\sigma\tau\alpha'_3$

Better Bounds for Special Blockcodes

Overlap-free Blockcodes

$\Gamma = \{\gamma_0, \ldots, \gamma_k\}$, $k < 2^m$, $b_m(i) :=$ binary repr. of i of length m

Better Bounds for Special Blockcodes

Overlap-free Blockcodes

$\Gamma = \{\gamma_0, \ldots, \gamma_k\}$, $k < 2^m$, $b_m(i) :=$ binary repr. of i of length m

1. $\varphi(\gamma_i) := 0^m\, 1\, b_m(i)\, 1$

Better Bounds for Special Blockcodes

Overlap-free Blockcodes

$\Gamma = \{\gamma_0, \ldots, \gamma_k\}$, $k < 2^m$, $b_m(i) :=$ binary repr. of i of length m

1. $\varphi(\gamma_i) := 0^m\, 1\, b_m(i)\, 1$ overlap-free $\ell = (2m+2)$-blockcode

Better Bounds for Special Blockcodes

Overlap-free Blockcodes

$\Gamma = \{\gamma_0, \ldots, \gamma_k\}$, $k < 2^m$, $b_m(i) :=$ binary repr. of i of length m

1. $\varphi(\gamma_i) := 0^m\, 1\, b_m(i)\, 1$ overlap-free $\ell = (2m+2)$-blockcode
2. G_x grammar for $x \in \Gamma^*$,

Better Bounds for Special Blockcodes

Overlap-free Blockcodes

$\Gamma = \{\gamma_0, \ldots, \gamma_k\}$, $k < 2^m$, $b_m(i) :=$ binary repr. of i of length m

1. $\varphi(\gamma_i) := 0^m\, 1\, b_m(i)\, 1$ overlap-free $\ell = (2m+2)$-blockcode
2. G_x grammar for $x \in \Gamma^*$,
   $\Longrightarrow \exists$ grammar $G_{\varphi(x)}$ for $\varphi(x)$ with

$$|\mathbf{G_{\varphi(x)}}| \;\leq\; \mathbf{(12 + o(1)) \cdot |G_x|}$$

Better Bounds for Special Blockcodes

Overlap-free Blockcodes

$\Gamma = \{\gamma_0, \ldots, \gamma_k\}$, $k < 2^m$, $b_m(i) :=$ binary repr. of i of length m

1. $\varphi(\gamma_i) := 0^m\, 1\, b_m(i)\, 1$ overlap-free $\ell = (2m+2)$-blockcode
2. G_x grammar for $x \in \Gamma^*$,
   $\Longrightarrow \exists$ grammar $G_{\varphi(x)}$ for $\varphi(x)$ with
   $$|\mathbf{G_{\varphi(x)}}| \;\leq\; \mathbf{(12 + o(1)) \cdot |G_x|}$$
3. $G_{\varphi(x)}$ grammar for $\varphi(x)$

Better Bounds for Special Blockcodes

Overlap-free Blockcodes

$\Gamma = \{\gamma_0, \ldots, \gamma_k\}$, $k < 2^m$, $b_m(i) :=$ binary repr. of i of length m

1. $\varphi(\gamma_i) := 0^m\, 1\, b_m(i)\, 1$ overlap-free $\ell = (2m+2)$-blockcode
2. G_x grammar for $x \in \Gamma^*$,
   $\Longrightarrow \exists$ grammar $G_{\varphi(x)}$ for $\varphi(x)$ with
   $$|\mathbf{G_{\varphi(x)}}| \;\leq\; \mathbf{(12 + o(1)) \cdot |G_x|}$$
3. $G_{\varphi(x)}$ grammar for $\varphi(x)$
   $\Longrightarrow \exists$ grammar G_x for x with
   $$|\mathbf{G_x}| \;\leq\; \mathbf{2 \cdot |G_{\varphi(x)}|}$$

Approximability

Corollary

2-MGC c-approximable

Approximability

Corollary

2-MGC c-approximable
$\Rightarrow$ MGC $(24c + \varepsilon)$-approximable

Approximability

Corollary

2-MGC c-approximable
$\Rightarrow$ MGC $(24c + \varepsilon)$-approximable

Corollary

nonconstant lower bound for MGC

Approximability

Corollary

2-MGC c-approximable
$\Rightarrow$ MGC $(24c + \varepsilon)$-approximable

Corollary

nonconstant lower bound for MGC
$\Rightarrow$ nonconstant lower bound for 2-MGC

Approximability

Corollary

2-MGC c-approximable
$\Rightarrow$ MGC $(24c + \varepsilon)$-approximable

Corollary

nonconstant lower bound for MGC
$\Rightarrow$ nonconstant lower bound for 2-MGC
$\Rightarrow$ 2-MGC$\notin$ P (if $\mathcal{P} \neq \mathcal{NP}$)

Conclusion

Bounds on the Compression Ratio

- optimal compression of a string and its ℓ-blockcoding:

$$\frac{\mathbf{m^*(x)}}{\mathbf{2\ell}} \leq \mathbf{m^*(\varphi(x)) \leq m^*(x) + m^*(\varphi(\Gamma))}$$

Conclusion

Bounds on the Compression Ratio

- optimal compression of a string and its ℓ-blockcoding:

$$\frac{\mathbf{m^*(x)}}{\mathbf{2\ell}} \leq \mathbf{m^*(\varphi(x)) \leq m^*(x) + m^*(\varphi(\Gamma))}$$

- bounds are asymptotically sharp

Conclusion

Bounds on the Compression Ratio

- optimal compression of a string and its ℓ-blockcoding:

$$\frac{\mathbf{m^*(x)}}{\mathbf{2\ell}} \leq \mathbf{m^*(\varphi(x))} \leq \mathbf{m^*(x) + m^*(\varphi(\Gamma))}$$

- bounds are asymptotically sharp
- improvements for overlap-free blockcodes:

$$\mathbf{m^*(x) \in \Theta(m^*(\varphi(x)))}$$

Conclusion

Bounds on the Compression Ratio

- optimal compression of a string and its ℓ-blockcoding:

$$\frac{\mathbf{m^*(x)}}{\mathbf{2\ell}} \leq \mathbf{m^*(\varphi(x)) \leq m^*(x) + m^*(\varphi(\Gamma))}$$

- bounds are asymptotically sharp
- improvements for overlap-free blockcodes:

$$\mathbf{m^*(x) \in \Theta(m^*(\varphi(x)))}$$

- implications for the approximability

Conclusion

Open Problems

- nonconstant lower bound for MGC

Conclusion

Open Problems

- nonconstant lower bound for MGC
- $\mathcal{NP}$-completeness of 2-MGC

Conclusion

Open Problems

- nonconstant lower bound for MGC
- $\mathcal{NP}$-completeness of 2-MGC
- optimal compression in the neighborhood of a string

Conclusion

Open Problems

- nonconstant lower bound for MGC
- $\mathcal{NP}$-completeness of 2-MGC
- optimal compression in the neighborhood of a string
  - given a string x and a distance d
    with respect to a metric – for example, the edit distance –
    find a string z in the d-neighborhood of x with maximal grammar compression

Conclusion

Open Problems

- nonconstant lower bound for MGC
- $\mathcal{NP}$-completeness of 2-MGC
- optimal compression in the neighborhood of a string
  - given a string x and a distance d with respect to a metric – for example, the edit distance – find a string z in the d-neighborhood of x with maximal grammar compression
  - bounds on the relation between distances of strings and their difference in optimal grammar compression

Conclusion

Open Problems

- nonconstant lower bound for MGC
- $\mathcal{NP}$-completeness of 2-MGC
- optimal compression in the neighborhood of a string
  - given a string x and a distance d with respect to a metric – for example, the edit distance – find a string z in the d-neighborhood of x with maximal grammar compression
  - bounds on the relation between distances of strings and their difference in optimal grammar compression
  - edit grammars [CLL+02]