Reasoning about subwords and subsequences

Ph. Schnoebelen

Laboratoire Méthodes Formelles (LMF) – ENS Paris-Saclay

GT DAAL – Apr. 2024 – Rennes

Outline the talk

Subwords not so well understood algorithmically

Let me tell you about a few simple and not-so-simple problems:

1. Subwords in compressed words
2. The Post embedding problem
3. Computing with subword-closed languages
4. Solving subword constraints
5. Describing words by their subwords

WORDS AND THEIR SUBWORDS

AB	is a prefix of	ABRACADABRA	
BRA	is a factor of	ABRACADABRA	(also a suffix)
ABBA	is a subword of	ABRACADABRA	

WORDS AND THEIR SUBWORDS

AB	is a prefix of	ABRACADABRA	
BRA	is a factor of	ABRACADABRA	(also a suffix)
ABBA	is a subword of	ABRACADABRA	

Subwords and factors are both very natural and fundamental notions.

But mathematically and algorithmically, subwords are trickier than factors.

Words and their subwords

AB	is a prefix of	ABRACADABRA	
BRA	is a factor of	ABRACADABRA	(also a suffix)
ABBA	is a subword of	ABRACADABRA	

Subwords and factors are both very natural and fundamental notions.

But mathematically and algorithmically, subwords are trickier than factors.

E.g. ABRACADABRA has 12 different prefixes, 55 different factors and 1304 different subwords.

WORDS AND THEIR SUBWORDS

AB	is a prefix of	ABRACADABRA	
BRA	is a factor of	ABRACADABRA	(also a suffix)
ABBA	is a subword of	ABRACADABRA	

Subwords and factors are both very natural and fundamental notions.

But mathematically and algorithmically, subwords are trickier than factors.

E.g. ABRACADABRA has 12 different prefixes, 55 different factors and 1304 different subwords.

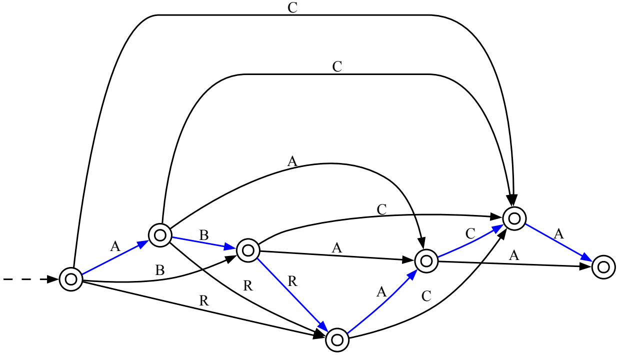
My own perspective is not language theory or combinatorics. I want to show you a few problems on subwords that appear “naturally” in formal methods and program verification.

How do you compute the number of distinct subwords of w?

(Does ABRACADABRA really has 1304 different subwords?)

First puzzle: counting

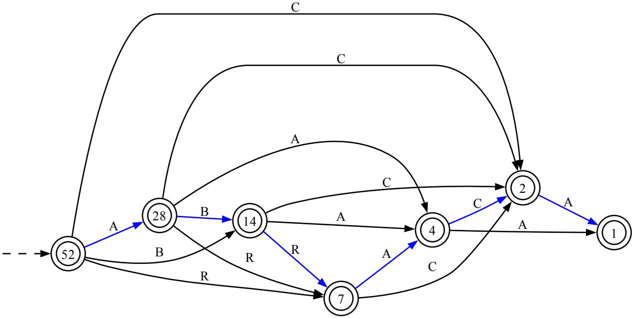
Build the subword automaton!



(here for ABRACA)

FIRST PUZZLE: COUNTING

Build the subword automaton. And count!



(here for ABRACA)

SECOND PUZZLE: TESTING

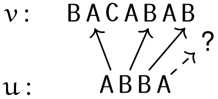
How do you check that $u \preccurlyeq v$?

(Here and later $\preccurlyeq$ denotes the subword ordering)

How do you check that $u \preccurlyeq v$?

(Here and later $\preccurlyeq$ denotes the subword ordering)

OK, the problem is trivial. Compute leftmost embedding:

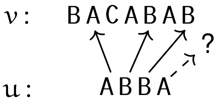


Actually this is easier than checking whether u is a factor of v.

How do you check that $u \preccurlyeq v$?

(Here and later $\preccurlyeq$ denotes the subword ordering)

OK, the problem is trivial. Compute leftmost embedding:



Actually this is easier than checking whether u is a factor of v.

However I had to check whether $U \preccurlyeq V$ for compressed words...

SLP-COMPRESSED WORDS

Straight Line Programs are the standard mathematical model for compressed “words” (i.e., text files, databases, genomes, log files, ..)

Equivalently, SLP $\equiv$ acyclic deterministic context-free grammar.

$$
\begin{aligned}
X_0 &:= \text{c h a} \\
X_1 &:= X_0 \text{ n t} \\
X_2 &:= X_0 \text{ s s e} \\
X_3 &:= X_1 X_2 \text{ r} \\
X_4 &:= X_2 \text{ u r s a } X_3
\end{aligned}
$$

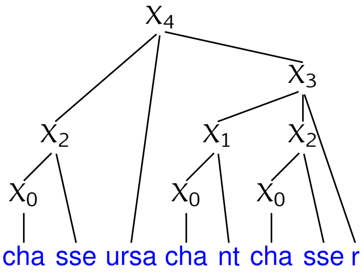
An SLP expands into a single word, of potentially exponential length.

SLP-COMPRESSED WORDS

Straight Line Programs are the standard mathematical model for compressed "words" (i.e., text files, databases, genomes, log files, ..)

Equivalently, SLP $\equiv$ acyclic deterministic context-free grammar.

$$
\begin{aligned}
X_0 &:= \text{c h a} \\
X_1 &:= X_0 \text{ n t} \\
X_2 &:= X_0 \text{ s s e} \\
X_3 &:= X_1 X_2 \text{ r} \\
X_4 &:= X_2 \text{ u r s a } X_3
\end{aligned}
$$



An SLP expands into a single word, of potentially exponential length.

SLP-COMPRESSED WORDS

Many efficient algorithms exist for SLPs (i.e., no expansion):

- Compute $X[\ell]$, letter at position ℓ
- Count number of occurrences of letter a
- Build SLP for $X[n \cdots m]$
- Decide if $X \in L(A)$ for some FSA A
- Find all occurrences of X as a factor of Y (pattern matching)
- Find largest palindrome inside X, etc.

See 2012 survey by Markus Lohrey

SLP-COMPRESSED WORDS

Many efficient algorithms exist for SLPs (i.e., no expansion):

- Compute $X[\ell]$, letter at position ℓ
- Count number of occurrences of letter a
- Build SLP for $X[n \cdots m]$
- Decide if $X \in L(A)$ for some FSA A
- Find all occurrences of X as a factor of Y (pattern matching)
- Find largest palindrome inside X, etc.

See 2012 survey by Markus Lohrey

However checking whether $X \preccurlyeq Y$ is PP-hard (Lifshits, Lohrey)

SLP-COMPRESSED WORDS

Many efficient algorithms exist for SLPs (i.e., no expansion):

- Compute $X[\ell]$, letter at position ℓ
- Count number of occurrences of letter a
- Build SLP for $X[n \cdots m]$
- Decide if $X \in L(A)$ for some FSA A
- Find all occurrences of X as a factor of Y (pattern matching)
- Find largest palindrome inside X, etc.

See 2012 survey by Markus Lohrey

However checking whether $X \preccurlyeq Y$ is PP-hard (Lifshits, Lohrey)

Luckily my verification problem only used SLPs of a particular form, and I could rely on:

Theorem

- Deciding whether $X \preccurlyeq u_1^{n_1} \cdots u_k^{n_k}$ where X is a SLP, $u_1, \ldots, u_k$ are words and $n_1, \ldots, n_k$ are integers can be done in polynomial-time.
- Same for deciding whether $u_1^{n_1} \cdots u_k^{n_k} \preccurlyeq X$.

SLP-COMPRESSED WORDS

Checking whether $X \preccurlyeq Y$ is PP-hard (Lifshits, Lohrey)

Luckily my verification problem only used SLPs of a particular form, and I could rely on:

Theorem
• Deciding whether $X \preccurlyeq u_1^{n_1} \cdots u_k^{n_k}$ where X is a SLP, $u_1, \ldots, u_k$ are words and $n_1, \ldots, n_k$ are integers can be done in polynomial-time.
• Same for deciding whether $u_1^{n_1} \cdots u_k^{n_k} \preccurlyeq X$.

Perspectives. Many interesting open problems in this area. More generally: what are good algorithms for testing embedding over various data structures?

Post Correspondence Problem
… but with subwords!

Joint work with Pierre Chambart & Prateek Karandikar

A NEW PROBLEM

Post Correspondence Problem:
Input: two morphisms $u, v : \Sigma^* \to \Gamma^*$
Question: is there $x \in \Sigma^+$ with $u(x) = v(x)$?

Post Embedding Problem:
Input: ... same ...
Question: is there $x \in \Sigma^+$ with $u(x) \leqslant v(x)$?

Regular Post Embedding Problem:
Input: ... and a regular $R \in \mathit{Reg}(\Sigma)$
Question: is there $x \in R$ with $u(x) \leqslant v(x)$?

Equivalently: given a rational relation $R \subseteq \Gamma^* \times \Gamma^*$, does $R \cap \leqslant \neq \emptyset$?

(*Side puzzle*: Is $\nleqslant \cap \ngeqslant$ a rational relation?)

A NEW PROBLEM

Post Correspondence Problem:
Input: two morphisms $u, v : \Sigma^* \to \Gamma^*$
Question: is there $x \in \Sigma^+$ with $u(x) = v(x)$?

Post Embedding Problem:
Input: ... same ...
Question: is there $x \in \Sigma^+$ with $u(x) \preccurlyeq v(x)$?

Regular Post Embedding Problem:
Input: ... and a regular $R \in \mathit{Reg}(\Sigma)$
Question: is there $x \in R$ with $u(x) \preccurlyeq v(x)$?

Equivalently: given a rational relation $R \subseteq \Gamma^* \times \Gamma^*$, does $R \cap \preccurlyeq \neq \emptyset$?

(*Side puzzle*: Is $\not\preccurlyeq \cap \not\succcurlyeq$ a rational relation?)

A NEW PROBLEM

Post Correspondence Problem:
Input: two morphisms $u,v:\Sigma^* \to \Gamma^*$
Question: is there $x \in \Sigma^+$ with $u(x) = v(x)$?

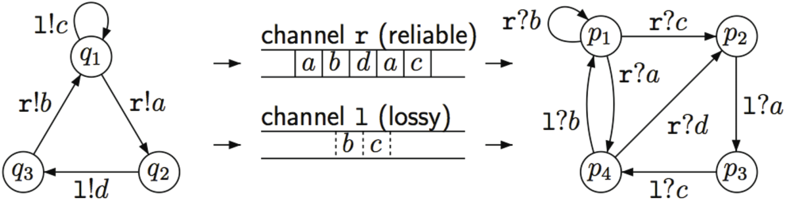
Post Embedding Problem:
Input: ... same ...
Question: is there $x \in \Sigma^+$ with $u(x) \preccurlyeq v(x)$?

Regular Post Embedding Problem:
Input: ... and a regular $R \in \mathit{Reg}(\Sigma)$
Question: is there $x \in R$ with $u(x) \preccurlyeq v(x)$?

Equivalently: given a rational relation $R \subseteq \Gamma^* \times \Gamma^*$, does $R \cap \preccurlyeq \neq \varnothing$?

(*Side puzzle*: Is $\not\preccurlyeq \cap \not\succcurlyeq$ a rational relation?)

MOTIVATIONS: UCS (UNIDIRECTIONAL CHANNEL SYSTEMS)

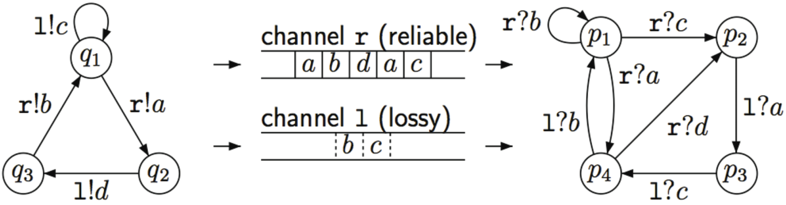


UCSs appeared while classifying networks mixing reliable and lossy fifo channels. Now has applications in logics for querying graphs [Barceló, Figueira, Libkin, LICS 2012].

Main question: Is reachability decidable for UCSs?

NB: Reachability is decidable if you change direction of one channel (ring with a lossy component). It is undecidable if you add a third channel in any way.

MOTIVATIONS: UCS (UNIDIRECTIONAL CHANNEL SYSTEMS)



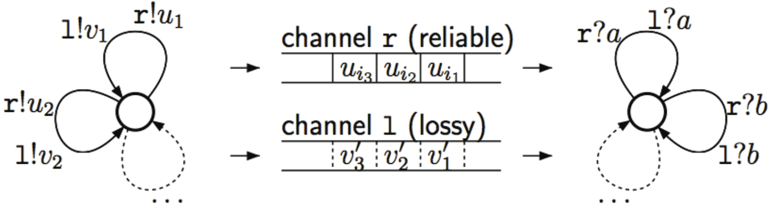
UCSs appeared while classifying networks mixing reliable and lossy fifo channels. Now has applications in logics for querying graphs [Barceló, Figueira, Libkin, LICS 2012].

Main question: Is reachability decidable for UCSs?

NB: Reachability is decidable if you change direction of one channel (ring with a lossy component). It is undecidable if you add a third channel in any way.

UCS CAN SOLVE PEP

Let $(u_1,v_1),(u_2,v_2),\ldots$ be a Post Embedding instance.



Sender guesses solution, Receiver validates it.

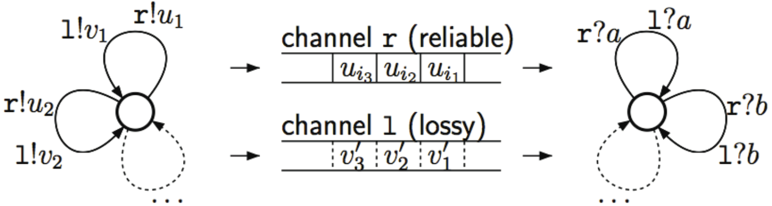
NB: Sender can guess a solution in regular R.

NB: Reciprocally, PEP can express the existence of a UCS run.

Our plan: Check relevant literature (mostly Finnish) for answer. Was naive.

UCS CAN SOLVE PEP

Let $(u_1,v_1),(u_2,v_2),\ldots$ be a Post Embedding instance.



Sender guesses solution, Receiver validates it.

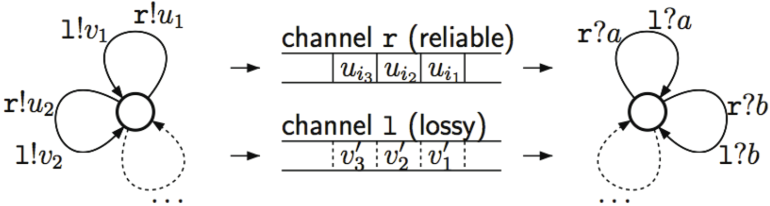
NB: Sender can guess a solution in regular R.

NB: Reciprocally, PEP can express the existence of a UCS run.

Our plan: Check relevant literature (mostly Finnish) for answer. Was naive.

UCS CAN SOLVE PEP

Let $(u_1, v_1), (u_2, v_2), \ldots$ be a Post Embedding instance.



Sender guesses solution, Receiver validates it.

NB: Sender can guess a solution in regular R.

NB: Reciprocally, PEP can express the existence of a UCS run.

Our plan: Check relevant literature (mostly Finnish) for answer. Was naive.

PEP with $R = \Sigma^+$ is trivial

Assume $u(x_1x_2) \preccurlyeq v(x_1x_2)$. Then $u(x_1) \preccurlyeq v(x_1)$ or $u(x_2) \preccurlyeq v(x_2)$

Hence a PEP instance has a solution iff it has a length-one solution

Σ	1	2	3
v_i	ac	$aaba$	$cbab$
u_i	aa	$baba$	ca

With $R = \Sigma^+$, PEP is decidable in logspace

Trickier with $R \neq \Sigma^+$.

Side puzzle: Take $R \stackrel{\text{def}}{=} \Sigma^* 1 \Sigma^*$. Is there a solution in R?
Answer: ??

PEP WITH $R = \Sigma^+$ IS TRIVIAL

Assume $u(x_1x_2) \preccurlyeq v(x_1x_2)$. Then $u(x_1) \preccurlyeq v(x_1)$ or $u(x_2) \preccurlyeq v(x_2)$

Hence a PEP instance has a solution iff it has a length-one solution

Σ	1	2	3
v_i	ac	$aaba$	$cbab$
u_i	aa	$baba$	ca

With $R = \Sigma^+$, PEP is decidable in logspace

Trickier with $R \neq \Sigma^+$.

Side puzzle: Take $R \stackrel{\text{def}}{=} \Sigma^* 1 \Sigma^*$. Is there a solution in R?
Answer: ??

PEP WITH $R = \Sigma^+$ IS TRIVIAL

Assume $u(x_1 x_2) \preccurlyeq v(x_1 x_2)$. Then $u(x_1) \preccurlyeq v(x_1)$ or $u(x_2) \preccurlyeq v(x_2)$

Hence a PEP instance has a solution iff it has a length-one solution

Σ	1	2	3
v_i	ac	$aaba$	$cbab$
u_i	aa	$baba$	ca

With $R = \Sigma^+$, PEP is decidable in logspace

Trickier with $R \neq \Sigma^+$.

Side puzzle: Take $R \stackrel{\text{def}}{=} \Sigma^* 1 \Sigma^*$. Is there a solution in R?
Answer: ??

PROVING THE ABSENCE OF SOLUTIONS

Take $R \stackrel{\text{def}}{=} \Sigma^* 1 \Sigma^*$ for

v_i	ac	aaba	cbab
u_i	aa	baba	ca

Lem 1. $(a+b)u_x \not\leqslant v_x$ for all $x \in \Sigma^*$:

– $x = \varepsilon$: ? Would need $a \leqslant \varepsilon$ or $b \leqslant \varepsilon$.

– $x = 1.y$: $(a+b)aa\,u_y \leqslant ac\,v_y$? Would need $aa\,u_y \leqslant v_y$.

– $x = 2.y$: $(a+b)baba\,u_y \leqslant aaba\,v_y$? Would need $ba\,u_y \leqslant v_y$.

– $x = 3.y$: $(a+b)ca\,u_y \leqslant cbab\,v_y$? Would need $ca\,u_y \leqslant v_y$.

Lem 2. $u_x \not\leqslant b^* v_x$ for all $x \in R$:

– $x = \varepsilon$: not in R

– $x = 1.y$: $aa\,u_y \leqslant b^*ac\,v_y$? Would need $a\,u_y \leqslant v_y$.

– $x = 2.y$: $baba\,u_y \leqslant b^*aaba\,v_y$? Would need $u_y \leqslant v_y$ with $y \in R$.

– $x = 3.y$: $ca\,u_y \leqslant b^*cbab\,v_y$? Would need $u_y \leqslant b\,v_y$ with $y \in R$.

Coro. There is no $x \in R$ with $u(x) \leqslant v(x)$

Proving the absence of solutions

Take $R \stackrel{\text{def}}{=} \Sigma^* 1 \Sigma^*$ for

v_i	ac	aaba	cbab
u_i	aa	baba	ca

Lem 1. $(a+b)u_x \not\leqslant v_x$ for all $x \in \Sigma^*$:

– $x = \varepsilon$: ? Would need $a \leqslant \varepsilon$ or $b \leqslant \varepsilon$.

– $x = 1.y$: $(a+b)aa\,u_y \leqslant ac\,v_y$? Would need $aa\,u_y \leqslant v_y$.

– $x = 2.y$: $(a+b)baba\,u_y \leqslant aaba\,v_y$? Would need $ba\,u_y \leqslant v_y$.

– $x = 3.y$: $(a+b)ca\,u_y \leqslant cbab\,v_y$? Would need $ca\,u_y \leqslant v_y$.

Lem 2. $u_x \not\leqslant b^* v_x$ for all $x \in R$:

– $x = \varepsilon$: not in R

– $x = 1.y$: $aa\,u_y \leqslant b^*ac\,v_y$? Would need $a\,u_y \leqslant v_y$.

– $x = 2.y$: $baba\,u_y \leqslant b^*aaba\,v_y$? Would need $u_y \leqslant v_y$ with $y \in R$.

– $x = 3.y$: $ca\,u_y \leqslant b^*cbab\,v_y$? Would need $u_y \leqslant b\,v_y$ with $y \in R$.

Coro. There is no $x \in R$ with $u(x) \leqslant v(x)$

PROVING THE ABSENCE OF SOLUTIONS

Take $R \stackrel{\text{def}}{=} \Sigma^* 1 \Sigma^*$ for

v_i	ac	aaba	cbab
u_i	aa	baba	ca

Lem 1. $(a+b)u_x \not\preccurlyeq v_x$ for all $x \in \Sigma^*$:
– $x = \varepsilon$: ? Would need $a \preccurlyeq \varepsilon$ or $b \preccurlyeq \varepsilon$.
– $x = 1.y$: $(a+b)aa\,u_y \preccurlyeq ac\,v_y$? Would need $aa\,u_y \preccurlyeq v_y$.
– $x = 2.y$: $(a+b)baba\,u_y \preccurlyeq aaba\,v_y$? Would need $ba\,u_y \preccurlyeq v_y$.
– $x = 3.y$: $(a+b)ca\,u_y \preccurlyeq cbab\,v_y$? Would need $ca\,u_y \preccurlyeq v_y$.

Lem 2. $u_x \not\preccurlyeq b^* v_x$ for all $x \in R$:
– $x = \varepsilon$: not in R
– $x = 1.y$: $aa\,u_y \preccurlyeq b^* ac\,v_y$? Would need $a\,u_y \preccurlyeq v_y$.
– $x = 2.y$: $baba\,u_y \preccurlyeq b^* aaba\,v_y$? Would need $u_y \preccurlyeq v_y$ with $y \in R$.
– $x = 3.y$: $ca\,u_y \preccurlyeq b^* cbab\,v_y$? Would need $u_y \preccurlyeq b\,v_y$ with $y \in R$.

Coro. There is no $x \in R$ with $u(x) \preccurlyeq v(x)$

PROVING THE ABSENCE OF SOLUTIONS

Take $R \stackrel{\text{def}}{=} \Sigma^* 1 \Sigma^*$ for

v_i	ac	aaba	cbab
u_i	aa	baba	ca

Lem 1. $(a+b)u_x \not\leqslant v_x$ for all $x \in \Sigma^*$:

– $x = \varepsilon$: ? Would need $a \leqslant \varepsilon$ or $b \leqslant \varepsilon$.

– $x = 1.y$: $(a+b)aa\,u_y \leqslant ac\,v_y$? Would need $aa\,u_y \leqslant v_y$.

– $x = 2.y$: $(a+b)baba\,u_y \leqslant aaba\,v_y$? Would need $ba\,u_y \leqslant v_y$.

– $x = 3.y$: $(a+b)ca\,u_y \leqslant cbab\,v_y$? Would need $ca\,u_y \leqslant v_y$.

Lem 2. $u_x \not\leqslant b^* v_x$ for all $x \in R$:

– $x = \varepsilon$: not in R

– $x = 1.y$: $aa\,u_y \leqslant b^* ac\,v_y$? Would need $a\,u_y \leqslant v_y$.

– $x = 2.y$: $baba\,u_y \leqslant b^* aaba\,v_y$? Would need $u_y \leqslant v_y$ with $y \in R$.

– $x = 3.y$: $ca\,u_y \leqslant b^* cbab\,v_y$? Would need $u_y \leqslant b\,v_y$ with $y \in R$.

Coro. There is no $x \in R$ with $u(x) \leqslant v(x)$

PROVING THE ABSENCE OF SOLUTIONS

Take $R \stackrel{\text{def}}{=} \Sigma^* 1 \Sigma^*$ for

v_i	ac	aaba	cbab
u_i	aa	baba	ca

Lem 1. $(a+b)u_x \not\preccurlyeq v_x$ for all $x \in \Sigma^*$:

– $x = \varepsilon$: ? Would need $a \preccurlyeq \varepsilon$ or $b \preccurlyeq \varepsilon$.
– $x = 1.y$: $(a+b)\mathtt{aa}\,u_y \preccurlyeq \mathtt{ac}\,v_y$? Would need $aa\,u_y \preccurlyeq v_y$.
– $x = 2.y$: $(a+b)\mathtt{baba}\,u_y \preccurlyeq \mathtt{aaba}\,v_y$? Would need $ba\,u_y \preccurlyeq v_y$.
– $x = 3.y$: $(a+b)\mathtt{ca}\,u_y \preccurlyeq \mathtt{cbab}\,v_y$? Would need $ca\,u_y \preccurlyeq v_y$.

Lem 2. $u_x \not\preccurlyeq b^* v_x$ for all $x \in R$:

– $x = \varepsilon$: not in R
– $x = 1.y$: $\mathtt{aa}\,u_y \preccurlyeq b^*\mathtt{ac}\,v_y$? Would need $a\,u_y \preccurlyeq v_y$.
– $x = 2.y$: $\mathtt{baba}\,u_y \preccurlyeq b^*\mathtt{aaba}\,v_y$? Would need $u_y \preccurlyeq v_y$ with $y \in R$.
– $x = 3.y$: $\mathtt{ca}\,u_y \preccurlyeq b^*\mathtt{cbab}\,v_y$? Would need $u_y \preccurlyeq b\,v_y$ with $y \in R$.

Coro. There is no $x \in R$ with $u(x) \preccurlyeq v(x)$

PROVING THE ABSENCE OF SOLUTIONS

Take $R \stackrel{\text{def}}{=} \Sigma^* 1 \Sigma^*$ for

v_i	ac	aaba	cbab
u_i	aa	baba	ca

Lem 1. $(a+b)u_x \not\leqslant v_x$ for all $x \in \Sigma^*$:
– $x = \varepsilon$: ? Would need $a \leqslant \varepsilon$ or $b \leqslant \varepsilon$.
– $x = 1.y$: $(a+b)aa\,u_y \leqslant ac\,v_y$? Would need $aa\,u_y \leqslant v_y$.
– $x = 2.y$: $(a+b)baba\,u_y \leqslant aaba\,v_y$? Would need $ba\,u_y \leqslant v_y$.
– $x = 3.y$: $(a+b)ca\,u_y \leqslant cbab\,v_y$? Would need $ca\,u_y \leqslant v_y$.

Lem 2. $u_x \not\leqslant b^* v_x$ for all $x \in R$:
– $x = \varepsilon$: not in R
– $x = 1.y$: $aa\,u_y \leqslant b^* ac\,v_y$? Would need $a\,u_y \leqslant v_y$.
– $x = 2.y$: $baba\,u_y \leqslant b^* aaba\,v_y$? Would need $u_y \leqslant v_y$ with $y \in R$.
– $x = 3.y$: $ca\,u_y \leqslant b^* cbab\,v_y$? Would need $u_y \leqslant b\,v_y$ with $y \in R$.

Coro. There is no $x \in R$ with $u(x) \leqslant v(x)$

PEP IS DECIDABLE — FIRST PROOF

General Method: – Guess regular languages A_L and B_L associated with each of the finitely many quotients L of R.
– Check that they block solutions, i.e., that for all these L

- $A_L\, u_x \not\leqslant v_x$ for all $x \in L$,
- $u_x \not\leqslant B_L\, v_x$ for all $x \in L$.

– Check finally that $\varepsilon \in A_R$.
– Deduce that the PEP instance has no solutions.

Note 1: The method is effective: the checks mostly involve regularity-preserving operations on regular languages.
Note 2: The method is complete: the largest blocking languages are upward-closed, hence regular (Higman, Haines).

Hence PEP is co-r.e. Since it is obviously also r.e., it is decidable.

PEP IS DECIDABLE — FIRST PROOF

General Method: – Guess regular languages A_L and B_L associated with each of the finitely many quotients L of R.
– Check that they block solutions, i.e., that for all these L
- $A_L\, u_x \not\leqslant v_x$ for all $x \in L$,
- $u_x \not\leqslant B_L\, v_x$ for all $x \in L$.

– Check finally that $\varepsilon \in A_R$.
– Deduce that the PEP instance has no solutions.

Note 1: The method is effective: the checks mostly involve regularity-preserving operations on regular languages.
Note 2: The method is complete: the largest blocking languages are upward-closed, hence regular (Higman, Haines).

Hence PEP is co-r.e. Since it is obviously also r.e., it is decidable.

PEP IS DECIDABLE — FIRST PROOF

General Method: – Guess regular languages A_L and B_L associated with each of the finitely many quotients L of R.
– Check that they block solutions, i.e., that for all these L
- $A_L\, u_x \not\leqslant v_x$ for all $x \in L$,
- $u_x \not\leqslant B_L\, v_x$ for all $x \in L$.

– Check finally that $\varepsilon \in A_R$.
– Deduce that the PEP instance has no solutions.

Note 1: The method is effective: the checks mostly involve regularity-preserving operations on regular languages.
Note 2: The method is complete: the largest blocking languages are upward-closed, hence regular (Higman, Haines).

Hence PEP is co-r.e. Since it is obviously also r.e., it is decidable.

Second proof: Higman's Lemma + Effectivity

Higman's Lemma:
any infinite sequence $w_1, w_2, \ldots, w_m, \ldots$ of words in Γ^*
contains an infinite increasing subsequence $w_{i_1} \leqslant w_{i_2} \leqslant \cdots \leqslant w_{i_m} \cdots$

Question: Can one bound i_2?

Finitary version of Higman's Lemma: There is a computable function H such that for any k-controlled sequence $w_1, w_2, \ldots, w_L$ of words in Γ^* the following holds:
if $L \geqslant H(n, k, \Gamma)$ then there is an increasing subsequence $w_{i_1} \leqslant w_{i_2} \leqslant \cdots \leqslant w_{i_n}$ of length n

NB: "k-controlled" $\overset{\text{def}}{\Leftrightarrow} |w_i| \leqslant i \times k$ for all $i = 1, 2, \ldots$

Proof: Tree of k-controlled sequences has finite branching. Cut each branch when it has an increasing subsequence. Apply Kőnig's Lemma.

SECOND PROOF: HIGMAN'S LEMMA + EFFECTIVITY

Higman's Lemma:
any infinite sequence $w_1, w_2, \ldots, w_m, \ldots$ of words in Γ^*
contains an infinite increasing subsequence $w_{i_1} \leqslant w_{i_2} \leqslant \cdots \leqslant w_{i_m} \cdots$

Question: Can one bound i_2?

Finitary version of Higman's Lemma: There is a computable function H such that for any k-controlled sequence $w_1, w_2, \ldots, w_L$ of words in Γ^* the following holds:
if $L \geqslant H(n, k, \Gamma)$ then there is an increasing subsequence $w_{i_1} \leqslant w_{i_2} \leqslant \cdots \leqslant w_{i_n}$ of length n
NB: "k-controlled" $\overset{\text{def}}{\Leftrightarrow}$ $|w_i| \leqslant i \times k$ for all $i = 1, 2, \ldots$

Proof: Tree of k-controlled sequences has finite branching. Cut each branch when it has an increasing subsequence. Apply Kőnig's Lemma.

Second proof: Higman's Lemma + Effectivity

Higman's Lemma:
any infinite sequence $w_1, w_2, \ldots, w_m, \ldots$ of words in Γ^*
contains an infinite increasing subsequence $w_{i_1} \leqslant w_{i_2} \leqslant \cdots \leqslant w_{i_m} \cdots$

Question: Can one bound i_2?

Finitary version of Higman's Lemma: There is a computable function H such that for any k-controlled sequence $w_1, w_2, \ldots, w_L$ of words in Γ^* the following holds:
if $L \geqslant H(n, k, \Gamma)$ then there is an increasing subsequence $w_{i_1} \leqslant w_{i_2} \leqslant \cdots \leqslant w_{i_n}$ of length n

NB: "k-controlled" $\overset{\text{def}}{\Leftrightarrow}$ $|w_i| \leqslant i \times k$ for all $i = 1, 2, \ldots$

Proof: Tree of k-controlled sequences has finite branching. Cut each branch when it has an increasing subsequence. Apply Kőnig's Lemma.

Second proof: Higman's Lemma + Effectivity

Higman's Lemma:
any infinite sequence $w_1, w_2, \ldots, w_m, \ldots$ of words in Γ^*
contains an infinite increasing subsequence $w_{i_1} \leqslant w_{i_2} \leqslant \cdots \leqslant w_{i_m} \cdots$

Question: Can one bound i_2?

Finitary version of Higman's Lemma: There is a computable function H such that for any k-controlled sequence $w_1, w_2, \ldots, w_L$ of words in Γ^* the following holds:
if $L \geqslant H(n, k, \Gamma)$ then there is an increasing subsequence
$w_{i_1} \leqslant w_{i_2} \leqslant \cdots \leqslant w_{i_n}$ of length n

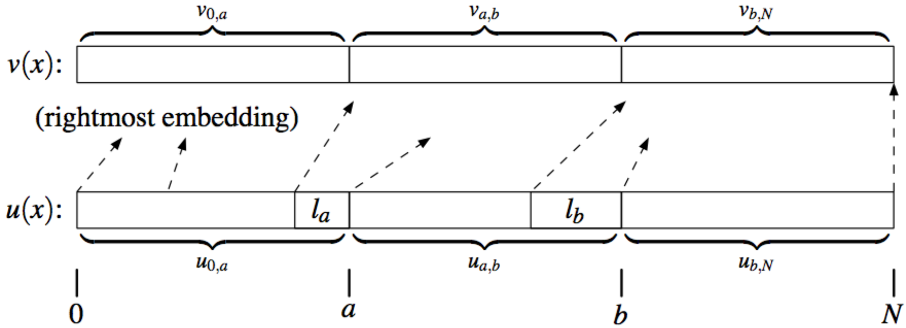
NB: "k-controlled" $\overset{\text{def}}{\Leftrightarrow}$ $|w_i| \leqslant i \times k$ for all $i = 1, 2, \ldots$

Proof: Tree of k-controlled sequences has finite branching. Cut each branch when it has an increasing subsequence. Apply Kőnig's Lemma.

CUTTING THROUGH PEP SOLUTIONS

For x a length-N solution, write $u_{i,j}, \ldots$ for $u(x[i,j)), \ldots$

For $i \in \{0,\ldots,N\}$, say $x[0,i)$ is a good prefix if $u_{i,N} \preccurlyeq v_{i,N}$. Then let l_i be the longest suffix of $u_{0,i}$ such that $l_i.u_{i,N} \preccurlyeq v_{i,N}$, call it “left margin”

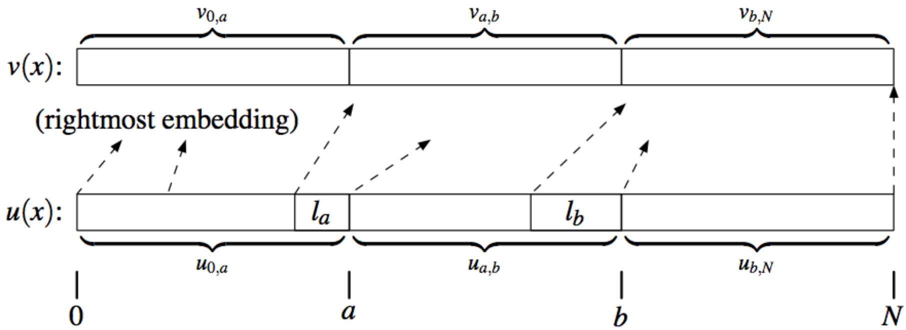


Lem. If $a < b$ are good and $l_a \preccurlyeq l_b$ then $x' = x[0,a).x[b,N)$ is a solution

Cutting through PEP solutions

For x a length-N solution, write $u_{i,j}, \ldots$ for $u(x[i,j)), \ldots$

For $i \in \{0,\ldots,N\}$, say $x[0,i)$ is a good prefix if $u_{i,N} \preccurlyeq v_{i,N}$. Then let l_i be the longest suffix of $u_{0,i}$ such that $l_i.u_{i,N} \preccurlyeq v_{i,N}$, call it "left margin"



Lem. If $a < b$ are good and $l_a \preccurlyeq l_b$ then $x' = x[0,a).x[b,N)$ is a solution

WHEN DO WE HAVE $l_a \preccurlyeq l_b$?

- Let $M = H(n_R + 1, K_u, |\Gamma|)$ with $K_u = \max_{i \in \Sigma} |u(i)|$

- If x has $> M$ good prefixes, it has a sequence $l_{a_0} \preccurlyeq l_{a_1} \preccurlyeq \cdots \preccurlyeq l_{a_{n_R}}$
Proof: the $(l_i)_{i \text{ good}}$ are K_u-controlled

- If $N > 2M$ then either x has $> M$ good prefixes or it has $> M$ bad prefixes, which are mirrors of good prefixes.

Lem. If a solution $x \in R$ is longer than $2M$, then there is a shorter solution $x' \in R$

Proof: Take x' is $x[0,a)x[b,N)$ for $a < b$ with $l_a \preccurlyeq l_b$ and $x[0,a) \sim_R x[0,b)$

Coro. PEP is decidable

When do we have $l_a \preccurlyeq l_b$?

- Let $M = H(n_R + 1, K_u, |\Gamma|)$ with $K_u = \max_{i \in \Sigma} |u(i)|$

- If x has $> M$ good prefixes, it has a sequence $l_{a_0} \preccurlyeq l_{a_1} \preccurlyeq \cdots \preccurlyeq l_{a_{n_R}}$
Proof: the $(l_i)_{i \text{ good}}$ are K_u-controlled

- If $N > 2M$ then either x has $> M$ good prefixes or it has $> M$ bad prefixes, which are mirrors of good prefixes.

Lem. If a solution $x \in R$ is longer than $2M$, then there is a shorter solution $x' \in R$

Proof: Take x' is $x[0,a)x[b,N)$ for $a < b$ with $l_a \preccurlyeq l_b$ and $x[0,a) \sim_R x[0,b)$

Coro. PEP is decidable

WHEN DO WE HAVE $l_a \preccurlyeq l_b$?

- Let $M = H(n_R + 1, K_u, |\Gamma|)$ with $K_u = \max_{i \in \Sigma} |u(i)|$

- If x has $> M$ good prefixes, it has a sequence $l_{a_0} \preccurlyeq l_{a_1} \preccurlyeq \cdots \preccurlyeq l_{a_{n_R}}$
Proof: the $(l_i)_{i \text{ good}}$ are K_u-controlled

- If $N > 2M$ then either x has $> M$ good prefixes or it has $> M$ bad prefixes, which are mirrors of good prefixes.

Lem. If a solution $x \in R$ is longer than $2M$, then there is a shorter solution $x' \in R$

Proof: Take x' is $x[0,a)x[b,N)$ for $a < b$ with $l_a \preccurlyeq l_b$ and $x[0,a) \sim_R x[0,b)$

Coro. PEP is decidable

When do we have $l_a \preccurlyeq l_b$?

- Let $M = H(n_R + 1, K_u, |\Gamma|)$ with $K_u = \max_{i \in \Sigma} |u(i)|$

- If x has $> M$ good prefixes, it has a sequence $l_{a_0} \preccurlyeq l_{a_1} \preccurlyeq \cdots \preccurlyeq l_{a_{n_R}}$
Proof: the $(l_i)_{i \text{ good}}$ are K_u-controlled

- If $N > 2M$ then either x has $> M$ good prefixes or it has $> M$ bad prefixes, which are mirrors of good prefixes.

Lem. If a solution $x \in R$ is longer than $2M$, then there is a shorter solution $x' \in R$

Proof: Take x' is $x[0,a)x[b,N)$ for $a < b$ with $l_a \preccurlyeq l_b$ and $x[0,a) \sim_R x[0,b)$

Coro. PEP is decidable

When do we have $l_a \preccurlyeq l_b$?

- Let $M = H(n_R + 1, K_u, |\Gamma|)$ with $K_u = \max_{i \in \Sigma} |u(i)|$

- If x has $> M$ good prefixes, it has a sequence $l_{a_0} \preccurlyeq l_{a_1} \preccurlyeq \cdots \preccurlyeq l_{a_{n_R}}$
Proof: the $(l_i)_{i \text{ good}}$ are K_u-controlled

- If $N > 2M$ then either x has $> M$ good prefixes or it has $> M$ bad prefixes, which are mirrors of good prefixes.

Lem. If a solution $x \in R$ is longer than $2M$, then there is a shorter solution $x' \in R$

Proof: Take x' is $x[0,a)x[b,N)$ for $a < b$ with $l_a \preccurlyeq l_b$ and $x[0,a) \sim_R x[0,b)$

Coro. PEP is decidable

Extensions and Variants

$\exists^\infty$PEP is decidable.

#PEP is computable.

- $\forall x \in R : u(x) \leqslant v(x)$ and $\forall^\infty x \in R : u(x) \leqslant v(x)$ are decidable
- $\exists x \in \Sigma^+ : \big(u_1(x) \leqslant v_1(x) \wedge u_2(x) \leqslant v_2(x)\big)$
and $\exists x \in \Sigma^+ : \big(u_1(x) \leqslant v_1(x) \wedge u_2(x) \not\leqslant v_2(x)\big)$ are undecidable
- $\forall x \in R\ \exists y \in R' : u(xy) \leqslant v(xy)$ is undecidable

Bottom line. PEP is $\mathbf{F}_{\omega^\omega}$-complete. Nice problem to use in reductions.

Fourth problem: subword-closures and superword-closures

Compute the set of subwords
(or of superwords) of a language?

Joint work with Prateek Karandikar & Mathias Niewerth

Closed languages

A language $L \subseteq \Sigma^*$ is

- Upward closed, if $x \in L$ and $x \preccurlyeq y$ implies $y \in L$.
- Downward closed, if $x \in L$ and $y \preccurlyeq x$ implies $y \in L$.

Examples:

- The set of all superwords of $aacb$ is upward closed, this is $\Sigma^* a \Sigma^* a \Sigma^* c \Sigma^* b \Sigma^*$.
- $\{w : |w|_c > 0 \wedge |w|_a \geqslant 2\}$ is upward closed.
- The set of all subwords of $aabbab$ is downward closed.
- $(a+b)^*(c+\varepsilon) + c^*$ is downward closed.

CLOSED LANGUAGES

A language $L \subseteq \Sigma^*$ is

- Upward closed, if $x \in L$ and $x \preccurlyeq y$ implies $y \in L$.
- Downward closed, if $x \in L$ and $y \preccurlyeq x$ implies $y \in L$.

Examples:

- The set of all superwords of $aacb$ is upward closed, this is $\Sigma^* a \Sigma^* a \Sigma^* c \Sigma^* b \Sigma^*$.
- $\{w : |w|_c > 0 \wedge |w|_a \geqslant 2\}$ is upward closed.
- The set of all subwords of $aabbab$ is downward closed.
- $(a+b)^*(c+\varepsilon)+c^*$ is downward closed.

Closed languages

A language $L \subseteq \Sigma^*$ is

- Upward closed, if $x \in L$ and $x \preccurlyeq y$ implies $y \in L$.
- Downward closed, if $x \in L$ and $y \preccurlyeq x$ implies $y \in L$.

Examples:

- The set of all superwords of $aacb$ is upward closed, this is $\Sigma^* a \Sigma^* a \Sigma^* c \Sigma^* b \Sigma^*$.
- $\{w : |w|_c > 0 \wedge |w|_a \geqslant 2\}$ is upward closed.
- The set of all subwords of $aabbab$ is downward closed.
- $(a + b)^*(c + \varepsilon) + c^*$ is downward closed.

CLOSURES

The upward closure of L is the smallest upward closed language which includes L:

$$\uparrow L = \{x : \exists y \in L \; y \preccurlyeq x\}$$

For example, $\uparrow\varnothing = \varnothing$. But $\uparrow\{\varepsilon\} = \Sigma^*$.
$\uparrow\{x : |x|_a$ is even and $|x|_b$ is odd$\} = \Sigma^* b \Sigma^*$.

The downward closure of L is the smallest downward closed language which includes L:

$$\downarrow L = \{x : \exists y \in L \; x \preccurlyeq y\}$$

For example, $\downarrow\left[(aba)^*(bb)^+(bc)^*\right] = (a+b)^*(b+c)^*$.

CLOSURES

The upward closure of L is the smallest upward closed language which includes L:

$$\uparrow L = \{x : \exists y \in L \; y \preccurlyeq x\}$$

For example, $\uparrow\varnothing = \varnothing$. But $\uparrow\{\varepsilon\} = \Sigma^*$.
$\uparrow\{x : |x|_a$ is even and $|x|_b$ is odd$\} = \Sigma^* b \Sigma^*$.

The downward closure of L is the smallest downward closed language which includes L:

$$\downarrow L = \{x : \exists y \in L \; x \preccurlyeq y\}$$

For example, $\downarrow\left[(aba)^*(bb)^+(bc)^*\right] = (a+b)^*(b+c)^*$.

REGULARITY

Every upward closed language has a finite set of minimal elements (by Higman's Lemma), and so is regular (/rational/recognizable).
By complementation, every downward closed language is regular.

Central problem

Computing with closed languages, for example:

- Given L, compute $\uparrow\mathrm{L}$ and $\downarrow\mathrm{L}$.
- Given $\mathrm{L}_1, \mathrm{L}_2$, decide whether $\uparrow\mathrm{L}_1 = \uparrow\mathrm{L}_2$ etc.

This problem exists in many variants, depending on the situation at hand.

Regularity

Every upward closed language has a finite set of minimal elements (by Higman's Lemma), and so is regular (/rational/recognizable).
By complementation, every downward closed language is regular.

Central problem

Computing with closed languages, for example:

- Given L, compute $\uparrow L$ and $\downarrow L$.
- Given L_1, L_2, decide whether $\uparrow L_1 = \uparrow L_2$ etc.

This problem exists in many variants, depending on the situation at hand.

Regularity

Every upward closed language has a finite set of minimal elements (by Higman's Lemma), and so is regular (/rational/recognizable). By complementation, every downward closed language is regular.

Central problem

Computing with closed languages, for example:

- Given L, compute $\uparrow L$ and $\downarrow L$.
- Given L_1, L_2, decide whether $\uparrow L_1 = \uparrow L_2$ etc.

This problem exists in many variants, depending on the situation at hand.

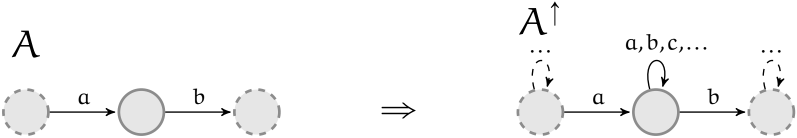
I will consider state complexity, when L is regular.

"State complexity", denoted $n_D(L)$ and $n_N(L)$ = minimal number of states of a DFA (resp. NFA) that recognizes L. Also: $n_U(L)$, $n_A(L)$, ...

Upward closure with NFAs

Assume that L is recognized by A.

An NFA for $\uparrow L$, denoted $A^{\uparrow}$, can be obtained from A by adding self-loops with all letters on all states.



UPWARD CLOSURE - EXAMPLE

Consider an alphabet $\Sigma = \{a_1, \ldots, a_k\}$, and the language

$$E_k = \{a_1 a_1, a_2 a_2, \ldots, a_k a_k\}$$

It is recognized by the following:

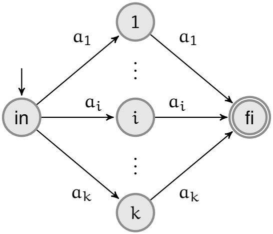
This is in fact deterministic and has $k + 2$ states.

UPWARD CLOSURE - EXAMPLE

Consider an alphabet $\Sigma = \{a_1, \ldots, a_k\}$, and the language

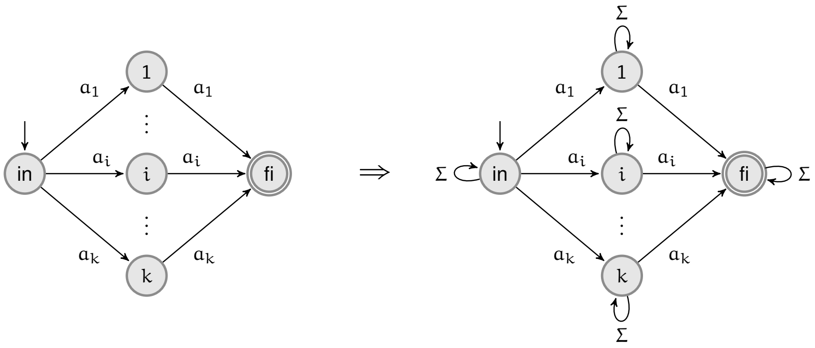
$$E_k = \{a_1 a_1, a_2 a_2, \ldots, a_k a_k\}$$

It is recognized by the following:



This is in fact deterministic and has $k + 2$ states.

Add a self-loop with all letters on every state, to get upward closure:



No longer deterministic!

Upward closure - example

$$E_k = \{a_1 a_1, a_2 a_2, \ldots, a_k a_k\}$$

$\uparrow E_k =$ "some letter appears at least twice"

A DFA for $\uparrow E_k$ must remember the set of letters read so far, and so needs at least 2^k states.

Concl. An exponential blowup may be necessary (and is always sufficient) when computing a DFA for $A^{\uparrow}$.

UPWARD CLOSURE - EXAMPLE

$$E_k = \{a_1 a_1, a_2 a_2, \ldots, a_k a_k\}$$

$$\uparrow E_k = \text{"some letter appears at least twice"}$$

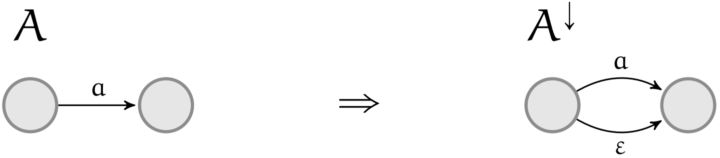
A DFA for $\uparrow E_k$ must remember the set of letters read so far, and so needs at least 2^k states.

Concl. An exponential blowup may be necessary (and is always sufficient) when computing a DFA for $A^{\uparrow}$.

Downward closure with NFAs

Assume that L is recognized by A.

An NFA for $\downarrow L$, denoted $A^{\downarrow}$, is obtained from A by adding an ε-transition parallel to every transition.



DOWNWARD CLOSURE - EXAMPLE

Consider an alphabet Σ with k letters, and the language

$$D_k = \bigcup_{a \in \Sigma} a \cdot \big(\Sigma \setminus \{a\}\big)^*$$

It is recognized by the following:

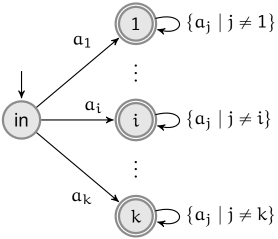
This is deterministic and has $k + 1$ states.

Downward closure - example

Consider an alphabet Σ with k letters, and the language

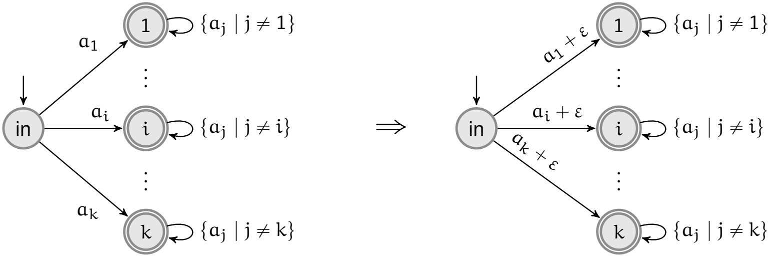
$$D_k = \bigcup_{a \in \Sigma} a \cdot \left(\Sigma \setminus \{a\}\right)^*$$

It is recognized by the following:



This is deterministic and has $k + 1$ states.

Add ε-edges parallel to every edge, to get downward closure:



No longer deterministic!

Downward closure - DFAs

$$D_k = \bigcup_{a\in\Sigma} a \cdot \big(\Sigma \setminus \{a\}\big)^*$$

$$\downarrow D_k = D_k \cup \text{“some letter does not appear”} = \bigcup_{a\in\Sigma} (a+\varepsilon)(\Sigma\setminus\{a\})^*$$

A DFA for $\downarrow D_k$ must remember the set of letters seen so far (ignoring the first letter), and so has at least 2^k states.

Concl. An exponential blowup may be necessary (and is always sufficient) when computing a DFA for $A^{\downarrow}$. *(Same as for upward closure)*

Downward closure - DFAs

$$D_k = \bigcup_{a \in \Sigma} a \cdot \left(\Sigma \setminus \{a\}\right)^*$$

$$\downarrow D_k = D_k \cup \text{“some letter does not appear”} = \bigcup_{a \in \Sigma} (a + \varepsilon)(\Sigma \setminus \{a\})^*$$

A DFA for $\downarrow D_k$ must remember the set of letters seen so far (ignoring the first letter), and so has at least 2^k states.

Concl. An exponential blowup may be necessary (and is always sufficient) when computing a DFA for $A^{\downarrow}$. *(Same as for upward closure)*

History of the question

- Gruber, Holzer, and Kutrib explicitly raised the question (Fund. Inf. 2009) and showed a $2^{\Omega(\sqrt{n}\log(n))}$ lower bound, for DFAs.
- Okhotin improved these bounds (Fund. Inf. 2010), gave exact bounds for upward closure on unbounded alphabets, and gave exponential $2^{\Omega(\sqrt{n})}$ lower bounds for a three-letter alphabet.
- Brzozowski and Jirásková (2010) gave exact upper bounds for upward and downward closures on unbounded alphabets.
- It turns out that Héam (ITA 2002) already had an exponential $r^{\sqrt{n}}$ lower bound —with $r = \left(\frac{1+\sqrt{5}}{2}\right)^{\frac{\sqrt{2}}{2}}$— for upward closure with a two-letter alphabet while studying “shuffle ideals”.

LOWER BOUND FOR DOWNWARD CLOSURE WITH n LETTERS

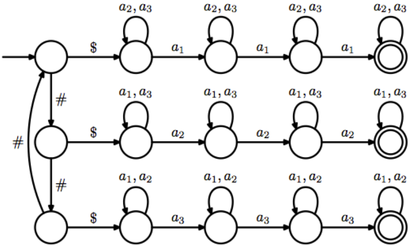
Fundamenta Informaticae 91 (2009) 105–121
DOI 10.3233/FI-2009-0035
IOS Press

More on the Size of Higman-Haines Sets: Effective Constructions*

Hermann Gruber†
Markus Holzer
Martin Kutrib

Theorem 3.2. For every $n \geq 1$, there exists a language L_n over an $(n+2)$-letter alphabet accepted by a DFA of size $(n+2)(n+1)^2$, such that any DFA accepting $\textsc{Down}(L_n)$ is at least of size $2^{\Omega(n \log n)}$.

$$L_n = \{\, \#^j \$ w \mid w \in A^*, j \geq 0, i = j \bmod n, |w|_{a_{i+1}} = n \,\}.$$

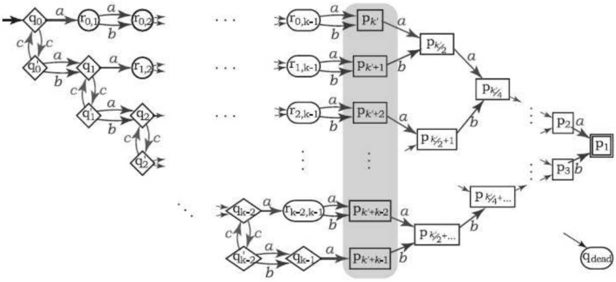


LOWER BOUND FOR DOWNWARD CLOSURE WITH 3 LETTERS

Fundamenta Informaticae 99 (2010) 325–338
DOI 10.3233/FI-2010-252
IOS Press

On the State Complexity of Scattered Substrings and Superstrings*

Alexander Okhotin†



STATE COMPLEXITY OF UPWARD CLOSURE

Proposition (Okhotin) [Upper bound]. 1. If A is an n-state NFA then $n_D(\uparrow L(A)) \leqslant 2^{n-2} + 1$.

Proof 1. Let $A = (\Sigma, Q, \delta, I, F)$ be an n-state NFA for $L = L(A)$. We assume that $I \cap F = \varnothing$ (and $I \neq \varnothing \neq F$) otherwise L contains ε (or is empty) and $\uparrow L$ is trivial.
Since $A^{\uparrow}$ has loops on all its states and for any letter, applying the powerset construction yields a DFA where $P \xrightarrow{a} P'$ implies $P \subseteq P'$, hence any state P reachable from I includes I. Furthermore, if P is accepting (i.e., $P \cap F \neq \varnothing$) and $P \xrightarrow{a} P'$, then P' is accepting too, hence all accepting states recognize exactly Σ^* and are equivalent. Then there can be at most $2^{|Q \setminus (I \cup F)|}$ states in the powerset automaton that are both reachable and not accepting. To this we add 1 for the accepting states since they are all equivalent. Finally $n_D(\uparrow L) \leqslant 2^{n-2} + 1$ since $|I \cup F|$ is at least 2 as we observed.

STATE COMPLEXITY OF UPWARD CLOSURE

Proposition (Okhotin) [Upper bound]. 1. If A is an n-state NFA then $n_D(\uparrow L(A)) \leqslant 2^{n-2} + 1$.

Proof 1. Let $A = (\Sigma, Q, \delta, I, F)$ be an n-state NFA for $L = L(A)$. We assume that $I \cap F = \varnothing$ (and $I \neq \varnothing \neq F$) otherwise L contains ε (or is empty) and $\uparrow L$ is trivial.
Since $A^{\uparrow}$ has loops on all its states and for any letter, applying the powerset construction yields a DFA where $P \xrightarrow{a} P'$ implies $P \subseteq P'$, hence any state P reachable from I includes I. Furthermore, if P is accepting (i.e., $P \cap F \neq \varnothing$) and $P \xrightarrow{a} P'$, then P' is accepting too, hence all accepting states recognize exactly Σ^* and are equivalent. Then there can be at most $2^{|Q \smallsetminus (I \cup F)|}$ states in the powerset automaton that are both reachable and not accepting. To this we add 1 for the accepting states since they are all equivalent. Finally $n_D(\uparrow L) \leqslant 2^{n-2} + 1$ since $|I \cup F|$ is at least 2 as we observed.

STATE COMPLEXITY OF UPWARD CLOSURE

Proposition (after Okhotin) [Lower bound]. 1. If A is an n-state NFA then $n_D(\uparrow L(A)) \leqslant 2^{n-2} + 1$.
2. Furthermore, for any $n > 1$ there exists a language L_n with $n_N(L_n) = n$ and $n_D(\uparrow L_n) = n_U(\uparrow L_n) = 2^{n-2} + 1$.

For the lower bound, $L_n = E_{n-2}$ works!

Recall that $E_k = \{a_1 a_1, a_2 a_2, \ldots, a_k a_k\}$ is recognized by a DFA wwith $k + 2$ states.
And that a DFA for $\uparrow E_k$ (= "*some letter appears at least twice*") needs $2^k + 1$ states.

State complexity of upward closure

Proposition (after Okhotin) [Lower bound]. 1. If A is an n-state NFA then $n_D(\uparrow L(A)) \leqslant 2^{n-2} + 1$.
2. Furthermore, for any $n > 1$ there exists a language L_n with $n_N(L_n) = n$ and $n_D(\uparrow L_n) = n_U(\uparrow L_n) = 2^{n-2} + 1$.

For the lower bound, $L_n = E_{n-2}$ works!

Recall that $E_k = \{a_1 a_1, a_2 a_2, \ldots, a_k a_k\}$ is recognized by a DFA wwith $k + 2$ states.
And that a DFA for $\uparrow E_k$ (= "*some letter appears at least twice*") needs $2^k + 1$ states.

State complexity of downward closure

Proposition (after Brzozowski & Jirásková) [Upper bound]. 1. If A is an n-state NFA with only one initial state (in particular when A is a DFA) then $n_D(\downarrow L(A)) \leqslant 2^{n-1}$.

Proof 1. We assume, w.l.o.g., that all states in $A = (\Sigma, Q, \delta, \{q_{\text{init}}\}, F)$ are reachable from the single initial state. From A one derives an NFA $A^{\downarrow}$ for $\downarrow L(A)$ by adding ε-transitions to A.

With these ε-transitions, the language recognized from a state $q \in Q$ is a subset of the language recognized from q_{init}. Hence, in the powerset automaton obtained by determinizing $A^{\downarrow}$, all states $P \subseteq Q$ that contain q_{init} are equivalent and recognize exactly $\downarrow L(A)$.

There also are 2^{n-1} states in 2^Q that do not contain q_{init}. Thus $2^{n-1} + 1$ bounds the number of non-equivalent states in the powerset automaton of $A^{\downarrow}$, and this includes a sink state (namely $\emptyset \in 2^Q$) that will be omitted in the canonical minimal DFA for $\downarrow L(A)$.

STATE COMPLEXITY OF DOWNWARD CLOSURE

Proposition (after Brzozowski & Jirásková) [Upper bound]. 1. If A is an n-state NFA with only one initial state (in particular when A is a DFA) then $n_D(\downarrow L(A)) \leqslant 2^{n-1}$.

Proof 1. We assume, w.l.o.g., that all states in $A = (\Sigma, Q, \delta, \{q_{\text{init}}\}, F)$ are reachable from the single initial state. From A one derives an NFA $A^{\downarrow}$ for $\downarrow L(A)$ by adding ε-transitions to A.

With these ε-transitions, the language recognized from a state $q \in Q$ is a subset of the language recognized from q_{init}. Hence, in the powerset automaton obtained by determinizing $A^{\downarrow}$, all states $P \subseteq Q$ that contain q_{init} are equivalent and recognize exactly $\downarrow L(A)$.

There also are 2^{n-1} states in 2^Q that do not contain q_{init}. Thus $2^{n-1} + 1$ bounds the number of non-equivalent states in the powerset automaton of $A^{\downarrow}$, and this includes a sink state (namely $\varnothing \in 2^Q$) that will be omitted in the canonical minimal DFA for $\downarrow L(A)$.

STATE COMPLEXITY OF DOWNWARD CLOSURE

Proposition [Lower bound]. 1. If A is an n-state NFA with only one initial state (in particular when A is a DFA) then $n_D(\downarrow L(A)) \leqslant 2^{n-1}$.
2. Furthermore, for any $n > 1$ there exists a language L'_n with $n_D(L'_n) = n$ and $n_D(\downarrow L'_n) = n_U(\downarrow L'_n) = 2^{n-1}$.

For the lower bound, D_{n-1} works!

Recall that $D_k = \bigcup_{a \in \Sigma} a.(\Sigma \setminus a)^*$ is recognized by a DFA with $k+1$ states.

And that a DFA for $\downarrow D_k$ needs 2^k states.

We can show that 2^k states are required for $\downarrow D_k$ even using Unambiguous NFAs.

NB: For NFAs with several initial states, a DFA for $A^\downarrow$ may need $2^n - 1$ states.

State complexity of downward closure

Proposition [Lower bound]. 1. If A is an n-state NFA with only one initial state (in particular when A is a DFA) then $n_D(\downarrow L(A)) \leqslant 2^{n-1}$.
2. Furthermore, for any $n > 1$ there exists a language L'_n with $n_D(L'_n) = n$ and $n_D(\downarrow L'_n) = n_U(\downarrow L'_n) = 2^{n-1}$.

For the lower bound, D_{n-1} works!

Recall that $D_k = \bigcup_{a \in \Sigma} a.(\Sigma \setminus a)^*$ is recognized by a DFA with $k+1$ states.

And that a DFA for $\downarrow D_k$ needs 2^k states.

We can show that 2^k states are required for $\downarrow D_k$ even using Unambiguous NFAs.

NB: For NFAs with several initial states, a DFA for $A^\downarrow$ may need $2^n - 1$ states.

LOWER BOUNDS FOR A TWO-LETTER ALPHABET

Proposition. For languages over a 2-letter alphabet, $n_D(\uparrow L)$ and $n_D(\downarrow L)$ are in $2^{\Omega(n^{1/3})}$, where $n = n_D(L)$.

We use the same family of witness languages to show both lower bounds.

Idea. Encode a larger alphabet by a 2-letter alphabet. Be careful about the interaction with the subword relation.

$$H = \{n, n+1, \ldots, 2n\}$$

For $i \in H$, $c(i) = a^i b^{3n-i}$.

$$L = \{c(i)^n : i \in H\}$$

Lower bounds for a two-letter alphabet

Proposition. For languages over a 2-letter alphabet, $n_D(\uparrow L)$ and $n_D(\downarrow L)$ are in $2^{\Omega(n^{1/3})}$, where $n = n_D(L)$.

We use the same family of witness languages to show both lower bounds.

Idea. Encode a larger alphabet by a 2-letter alphabet. Be careful about the interaction with the subword relation.

$$H = \{n, n+1, \ldots, 2n\}$$

For $i \in H$, $c(i) = a^i b^{3n-i}$.

$$L = \{c(i)^n : i \in H\}$$

$$
\begin{aligned}
H &= \{n, n+1, \ldots, 2n\} \\
c(i) &= a^i b^{3n-i} \\
L &= \{c(i)^n : i \in H\}
\end{aligned}
$$

For $n = 2$, $H = \{2, 3, 4\}$, and

$$
\begin{aligned}
L = \{&aabbbbaabbbb, \\
&aaabbbaaabbb, \\
&aaaabbaaaabb\}
\end{aligned}
$$

For general n,

$$H = \{n, n+1, \dots, 2n\}$$
$$c(i) = a^i b^{3n-i}$$
$$L = \{c(i)^n : i \in H\}$$

L has a DFA with $3n^3 + 1$ states, but both $\uparrow L$ and $\downarrow L$ need more than $\binom{n+1}{n/2}$ states. This is $\approx \frac{2^{n+3/2}}{\sqrt{\pi n}}$, i.e. $2^{\Omega(n)}$ states.

Proof idea: for any two different halves $X = \{p_1, \dots, p_{n/2}\}$ and $Y = \{q_1, \dots, q_{n/2}\}$ of H, the words $w_X \stackrel{\text{def}}{=} c(p_1) \cdots c(p_{n/2})$ and $w_Y \stackrel{\text{def}}{=} c(q_1) \cdots c(q_{n/2})$ must reach different states in any DFA for $\downarrow L$.

For $\uparrow L$, one considers $w'_X \stackrel{\text{def}}{=} c(p_1)c(p_1) \cdots c(p_{n/2})c(p_{n/2})$ and $w'_Y \stackrel{\text{def}}{=} c(q_1)c(q_1) \cdots c(q_{n/2})c(q_{n/2})$.

For general n,

$$\begin{aligned} H &= \{n, n+1, \ldots, 2n\} \\ c(i) &= a^i b^{3n-i} \\ L &= \{c(i)^n : i \in H\} \end{aligned}$$

L has a DFA with $3n^3 + 1$ states, but both $\uparrow L$ and $\downarrow L$ need more than $\binom{n+1}{n/2}$ states. This is $\approx \frac{2^{n+3/2}}{\sqrt{\pi n}}$, i.e. $2^{\Omega(n)}$ states.

Proof idea: for any two different halves $X = \{p_1, \ldots, p_{n/2}\}$ and $Y = \{q_1, \ldots, q_{n/2}\}$ of H, the words $w_X \stackrel{\text{def}}{=} c(p_1) \cdots c(p_{n/2})$ and $w_Y \stackrel{\text{def}}{=} c(q_1) \cdots c(q_{n/2})$ must reach different states in any DFA for $\downarrow L$.

For $\uparrow L$, one considers $w'_X \stackrel{\text{def}}{=} c(p_1)c(p_1) \cdots c(p_{n/2})c(p_{n/2})$ and $w'_Y \stackrel{\text{def}}{=} c(q_1)c(q_1) \cdots c(q_{n/2})c(q_{n/2})$.

COMPLEXITY OF DECISION PROBLEMS

Proposition.
Deciding whether $L(A)$ is upward-closed or downward-closed is PSPACE-complete over NFAs (NL-complete over DFAs), even in the 2-letter alphabet case.

Proposition (Bachmeier+Luttenberger+Schlund, 2015).
1. Deciding whether $\downarrow L(A) \subseteq \downarrow L(B)$ or whether $\uparrow L(A) \subseteq \uparrow L(B)$ is coNP-complete when A and B are NFAs.
2. Deciding $\downarrow L(A) = \downarrow L(B)$ or $\uparrow L(A) = \uparrow L(B)$ is coNP-hard even when A and B are DFAs over a two-letter alphabet.
3. These problems are NL-complete when restricting to NFAs over a 1-letter alphabet.

Proposition (Rampersad+Shallit+Xu, Fund. Inf. 2012).
Deciding whether $\downarrow L(A) = \Sigma^*$ when A is a NFA is NL-complete.

How do we solve inequations?

Joint work with Prateek Karandikar, Simon Halfon & Georg Zetzsche

The First-Order Logic of Subwords

We consider $\mathsf{FO}(A^*;\preccurlyeq)$ formulas, like

$$\forall u,v,w: u \preccurlyeq v \wedge v \preccurlyeq w \implies u \preccurlyeq w \qquad (\varphi_1)$$

$$\forall u: \mathtt{ab} \preccurlyeq u \wedge \mathtt{ba} \preccurlyeq u \implies \mathtt{aa} \preccurlyeq u \vee \mathtt{bb} \preccurlyeq u \qquad (\varphi_2)$$

$$\exists u: \mathtt{abcd} \preccurlyeq u \wedge \mathtt{bcde} \preccurlyeq u \wedge \mathtt{abcde} \not\preccurlyeq u \qquad (\varphi_3)$$

$$\forall u,v: \exists w: \begin{pmatrix} & u \preccurlyeq w \wedge v \preccurlyeq w \\ \wedge & \forall t: [u \preccurlyeq t \wedge v \preccurlyeq t \implies w \preccurlyeq t] \end{pmatrix} \qquad (\varphi_4)$$

$$\exists u_1,\ldots,u_n \in a^+b^+: \bigwedge_{1\leqslant i<j\leqslant n} u_i \perp u_j \qquad (\varphi_{5,n})$$

NB1: Whether $A^* \models \varphi$ may depend on A.

NB2: φ_5 actually uses $\mathsf{FO}(A^*;\preccurlyeq,R_1,R_2,\ldots)$, the logic enriched with regular predicates.

Validity (also Truth) problem for logics of words

Problem: Given A and a sentence φ in $\mathsf{FO}(A^*;\preccurlyeq)$, is φ true?

- $\mathsf{FO}(A^*;\leqslant_{\text{prefix}})$ —even $\mathsf{MSO}(A^*,\leqslant_{\text{prefix}})$— is decidable (Rabin 1969)
- $\mathsf{FO}(A^*;\cdot)$: undecidable (Quine, 1946) but the Σ_1 fragment is decidable: cf. word equations (Makanin, 1977; Büchi & Senger, 1986/7; Plandowski 1999; Jeż 2017)
- $\mathsf{FO}(A^*;\leqslant_{\text{infix}})$: undecidable (Kuske 2006)

What about $\mathsf{FO}(A^*;\preccurlyeq)$?

VALIDITY (ALSO TRUTH) PROBLEM FOR LOGICS OF WORDS

Problem: Given A and a sentence φ in $\mathsf{FO}(A^*;\preccurlyeq)$, is φ true?

- $\mathsf{FO}(A^*;\leqslant_{\text{prefix}})$ —even $\mathsf{MSO}(A^*,\leqslant_{\text{prefix}})$— is decidable (Rabin 1969)
- $\mathsf{FO}(A^*;\cdot)$: undecidable (Quine, 1946) but the Σ_1 fragment is decidable: cf. word equations (Makanin, 1977; Büchi & Senger, 1986/7; Plandowski 1999; Jeż 2017)
- $\mathsf{FO}(A^*;\leqslant_{\text{infix}})$: undecidable (Kuske 2006)

What about $\mathsf{FO}(A^*;\preccurlyeq)$?

What about $\mathsf{FO}(A^*; \preccurlyeq)$?

Comon & Treinen, 1994: small extension $\mathsf{FO}(A^*; \preccurlyeq, p_a)$ (with prefixing function $p_a : u \mapsto a \cdot u$) is undecidable, even the Σ_4 fragment, on a 3-letter alphabet.

Kuske, 2006: $\mathsf{FO}(A^*; \preccurlyeq)$ undecidable, even the Σ_3 fragment on a 2-letter alphabet. And the Σ_1 fragment is decidable.

Kudinov, Selivanov & Yartseva, 2010: $\mathsf{FO}(A^*; \preccurlyeq)$ is computably isomorphic to $\mathsf{FO}(\omega; +, \times)$, aka first-order arithmetic.

Karandikar & Schnoebelen, 2015: The Σ_2 fragment is undecidable, even over a "small" fixed alphabet, and eventually a 2-letter alphabet.

Karandikar & Schnoebelen, 2016: The FO^2 fragment is decidable even when allowing regular predicates.

Halfon, Schnoebelen & Zetzsche, 2017: The Σ_1 fragment

FO(A^*;$\preccurlyeq$) WITH OR WITHOUT CONSTANTS?

Unlike "$\forall u, v, w : u \preccurlyeq v \preccurlyeq w \implies u \preccurlyeq w$", some formulas use **constants**, e.g., "$\mathtt{ab} \preccurlyeq u \wedge \mathtt{ba} \preccurlyeq u \implies (\mathtt{aa} \preccurlyeq u \vee \mathtt{bb} \preccurlyeq u)$"

Same for "$x \in a^+b^+$", short for "$\mathtt{ab} \preccurlyeq x \wedge \mathtt{ba} \not\preccurlyeq x \wedge \mathtt{c} \not\preccurlyeq x \wedge \cdots$"

This is FO(A^*;$\preccurlyeq$) vs. FO(A^*;$\preccurlyeq$,$w_1, w_2, \ldots$)

Anyway, constant words can be defined in FO(A^*;$\preccurlyeq$):
$\psi_e(u) \stackrel{\text{def}}{\equiv} \forall x : u \preccurlyeq x$ defines "$u = \varepsilon$"
$\psi_1(v) \stackrel{\text{def}}{\equiv} \forall x : x \preccurlyeq v \implies (\psi_e(x) \vee v \preccurlyeq x)$ defines "v is a letter or ε"

NB: We can state "$|A| = n$" and "$|A| \geqslant \aleph_0$" in FO(A^*;$\preccurlyeq$)

Defining words: we can define e.g., "$v \doteq \mathtt{aabac}$" without using constants but this is defined "modulo automorphisms of the (A^*;$\preccurlyeq$) structure".

FO(A^*;$\preccurlyeq$) WITH OR WITHOUT CONSTANTS?

Unlike "$\forall u,v,w : u \preccurlyeq v \preccurlyeq w \implies u \preccurlyeq w$", some formulas use **constants**, e.g., "$\mathtt{ab} \preccurlyeq u \land \mathtt{ba} \preccurlyeq u \implies (\mathtt{aa} \preccurlyeq u \lor \mathtt{bb} \preccurlyeq u)$"

Same for "$x \in \mathtt{a}^+\mathtt{b}^+$", short for "$\mathtt{ab} \preccurlyeq x \land \mathtt{ba} \not\preccurlyeq x \land \mathtt{c} \not\preccurlyeq x \land \cdots$"

This is FO(A^*;$\preccurlyeq$) vs. FO(A^*;$\preccurlyeq$,$w_1,w_2,...$)

Anyway, constant words can be defined in FO(A^*;$\preccurlyeq$):

$\psi_e(u) \stackrel{\text{def}}{\equiv} \forall x : u \preccurlyeq x$ defines "$u = \varepsilon$"

$\psi_l(v) \stackrel{\text{def}}{\equiv} \forall x : x \preccurlyeq v \implies (\psi_e(x) \lor v \preccurlyeq x)$ defines "v is a letter or ε"

NB: We can state "$|A| = n$" and "$|A| \geqslant \aleph_0$" in FO(A^*;$\preccurlyeq$)

Defining words: we can define e.g., "$v \doteq \mathtt{aabac}$" without using constants but this is defined "modulo automorphisms of the (A^*;$\preccurlyeq$) structure".

FO$(A^*;\preccurlyeq)$ WITH OR WITHOUT CONSTANTS?

Unlike "$\forall u,v,w: u \preccurlyeq v \preccurlyeq w \implies u \preccurlyeq w$", some formulas use **constants**, e.g., "$ab \preccurlyeq u \wedge ba \preccurlyeq u \implies (aa \preccurlyeq u \vee bb \preccurlyeq u)$"

Same for "$x \in a^+b^+$", short for "$ab \preccurlyeq x \wedge ba \not\preccurlyeq x \wedge c \not\preccurlyeq x \wedge \cdots$"

This is FO$(A^*;\preccurlyeq)$ vs. FO$(A^*;\preccurlyeq,w_1,w_2,\ldots)$

Anyway, constant words can be defined in FO$(A^*;\preccurlyeq)$:
$\psi_e(u) \stackrel{\text{def}}{\equiv} \forall x: u \preccurlyeq x$ defines "$u = \varepsilon$"
$\psi_l(v) \stackrel{\text{def}}{\equiv} \forall x: x \preccurlyeq v \Longrightarrow (\psi_e(x) \vee v \preccurlyeq x)$ defines "v is a letter or ε"

NB: We can state "$|A| = n$" and "$|A| \geqslant \aleph_0$" in FO$(A^*;\preccurlyeq)$

Defining words: we can define e.g., "$v \doteq aabac$" without using constants but this is defined "modulo automorphisms of the $(A^*;\preccurlyeq)$ structure".

Subword Constraints

"Subword Constraints" $\equiv$ the Σ_1-fragment

$$\mathrm{abc} \not\preccurlyeq u \wedge u \preccurlyeq v \wedge u \not\preccurlyeq \mathrm{baa} \wedge \cdots \wedge v \perp w$$

How do we compute a set of solutions?

Recall: "The Σ_1 fragment is decidable" (in fact NP-complete)

Yes but this was about the logic without constants!

Ok but "constants can be defined within the logic", no?

Well, we defined ε by a Π_1 formula ...

Bottom Line: we don't really know whether the Σ_1 fragment of $\mathrm{FO}(A^*; \preccurlyeq, w_1, w_2, \ldots)$ is decidable

Thm. (Halfon, Schnoebelen & Zetzsche, 2017): the fragment is undecidable

SUBWORD CONSTRAINTS

"Subword Constraints" $\equiv$ the Σ_1-fragment

$$\mathrm{abc} \not\preccurlyeq u \wedge u \preccurlyeq v \wedge u \not\preccurlyeq \mathrm{baa} \wedge \cdots \wedge v \perp w$$

How do we compute a set of solutions?

Recall: "The Σ_1 fragment is decidable" (in fact NP-complete)

Yes but this was about the logic without constants!

Ok but "constants can be defined within the logic", no?

Well, we defined ε by a Π_1 formula ...

Bottom Line: we don't really know whether the Σ_1 fragment of $\mathrm{FO}(A^*;\preccurlyeq,w_1,w_2,\ldots)$ is decidable

Thm. (Halfon, Schnoebelen & Zetzsche, 2017): the fragment is undecidable

SUBWORD CONSTRAINTS

"Subword Constraints" $\equiv$ the Σ_1-fragment

$$\mathtt{abc} \not\preccurlyeq u \wedge u \preccurlyeq v \wedge u \not\preccurlyeq \mathtt{baa} \wedge \cdots \wedge v \perp w$$

How do we compute a set of solutions?

Recall: "The Σ_1 fragment is decidable" (in fact NP-complete)

Yes but this was about the logic without constants!

Ok but "constants can be defined within the logic", no?

Well, we defined ε by a Π_1 formula ...

Bottom Line: we don't really know whether the Σ_1 fragment of $\mathrm{FO}(A^*;\preccurlyeq,w_1,w_2,\ldots)$ is decidable

Thm. (Halfon, Schnoebelen & Zetzsche, 2017): the fragment is undecidable

Σ_1-DEFINABLE PROPERTIES

Fix $A = \{a, b\}$. Here are some Σ_1-definable properties:

$$|u|_a < |v_a| \stackrel{\text{def}}{\equiv} \exists x \in a^* : x \preccurlyeq v \wedge x \not\preccurlyeq u$$
$$\equiv \exists x : b \not\preccurlyeq x \wedge x \preccurlyeq v \wedge x \not\preccurlyeq u$$

$$\exists n > 0 : u = a^{n+1} \wedge v = a^n b$$
$$\stackrel{\text{def}}{\equiv} u \in a^* \wedge v \in a^* b \wedge |v|_a < |u|_a \wedge \exists x \in a^* baa : v \not\preccurlyeq x \wedge u \preccurlyeq x$$

using $v \in a^* b \stackrel{\text{def}}{\equiv} b \preccurlyeq v \wedge ba \not\preccurlyeq v \wedge bb \not\preccurlyeq v$

and $x \in a^* baa \stackrel{\text{def}}{\equiv} baa \preccurlyeq x \wedge bb \not\preccurlyeq x \wedge baaa \not\preccurlyeq x$

$$u, v \in A^* b \wedge |u|_a = |v|_a$$
$$\stackrel{\text{def}}{\equiv} \exists x \in a^* : \exists y \in a^* b : \begin{array}{l} \exists n : x = a^{n+1} \wedge y = a^n b \\ \wedge \quad u, v \in A^* \wedge y \preccurlyeq u \not\succcurlyeq x \wedge y \preccurlyeq v \not\succcurlyeq x \end{array}$$

Σ_1-DEFINABLE PROPERTIES

Fix $A = \{a, b\}$. Here are some Σ_1-definable properties:

$$|u|_a < |v_a| \stackrel{\text{def}}{\equiv} \exists x \in a^* : x \preccurlyeq v \wedge x \not\preccurlyeq u$$
$$\equiv \exists x : b \not\preccurlyeq x \wedge x \preccurlyeq v \wedge x \not\preccurlyeq u$$

$$\exists n > 0 : u = a^{n+1} \wedge v = a^n b$$
$$\stackrel{\text{def}}{\equiv} u \in a^* \wedge v \in a^* b \wedge |v|_a < |u|_a \wedge \exists x \in a^* baa : v \not\preccurlyeq x \wedge u \preccurlyeq x$$

using $v \in a^* b \stackrel{\text{def}}{\equiv} b \preccurlyeq v \wedge ba \not\preccurlyeq v \wedge bb \not\preccurlyeq v$

and $x \in a^* baa \stackrel{\text{def}}{\equiv} baa \preccurlyeq x \wedge bb \not\preccurlyeq x \wedge baaa \not\preccurlyeq x$

$$u, v \in A^* b \wedge |u|_a = |v|_a$$
$$\stackrel{\text{def}}{\equiv} \exists x \in a^* : \exists y \in a^* b : \begin{array}{ll} & \exists n : x = a^{n+1} \wedge y = a^n b \\ \wedge & u, v \in A^* \wedge y \preccurlyeq u \not\succcurlyeq x \wedge y \preccurlyeq v \not\succcurlyeq x \end{array}$$

Σ_1-DEFINABLE PROPERTIES

Fix $A = \{a, b\}$. Here are some Σ_1-definable properties:

$$|u|_a < |v_a| \overset{\text{def}}{\equiv} \exists x \in a^* : x \preccurlyeq v \wedge x \not\preccurlyeq u$$
$$\equiv \exists x : b \not\preccurlyeq x \wedge x \preccurlyeq v \wedge x \not\preccurlyeq u$$

$$\exists n > 0 : u = a^{n+1} \wedge v = a^n b$$
$$\overset{\text{def}}{\equiv} u \in a^* \wedge v \in a^* b \wedge |v|_a < |u|_a \wedge \exists x \in a^* baa : v \not\preccurlyeq x \wedge u \preccurlyeq x$$

using $v \in a^* b \overset{\text{def}}{\equiv} b \preccurlyeq v \wedge ba \not\preccurlyeq v \wedge bb \not\preccurlyeq v$
and $x \in a^* baa \overset{\text{def}}{\equiv} baa \preccurlyeq x \wedge bb \not\preccurlyeq x \wedge baaa \not\preccurlyeq x$

$$u, v \in A^* b \wedge |u|_a = |v|_a$$
$$\overset{\text{def}}{\equiv} \exists x \in a^* : \exists y \in a^* b : \begin{array}{l} \exists n : x = a^{n+1} \wedge y = a^n b \\ \wedge \quad u, v \in A^* \wedge y \preccurlyeq u \not\succcurlyeq x \wedge y \preccurlyeq v \not\succcurlyeq x \end{array}$$

Σ_1-DEFINABLE PROPERTIES

Fix $A = \{a, b\}$. Here are some Σ_1-definable properties:

$$|u|_a < |v_a| \overset{\text{def}}{\equiv} \exists x \in a^* : x \preccurlyeq v \wedge x \not\preccurlyeq u$$
$$\equiv \exists x : b \not\preccurlyeq x \wedge x \preccurlyeq v \wedge x \not\preccurlyeq u$$

$$\exists n > 0 : u = a^{n+1} \wedge v = a^n b$$
$$\overset{\text{def}}{\equiv} u \in a^* \wedge v \in a^* b \wedge |v|_a < |u|_a \wedge \exists x \in a^* baa : v \not\preccurlyeq x \wedge u \preccurlyeq x$$

using $v \in a^* b \overset{\text{def}}{\equiv} b \preccurlyeq v \wedge ba \not\preccurlyeq v \wedge bb \not\preccurlyeq v$

and $x \in a^* baa \overset{\text{def}}{\equiv} baa \preccurlyeq x \wedge bb \not\preccurlyeq x \wedge baaa \not\preccurlyeq x$

$$u, v \in A^* b \wedge |u|_a = |v|_a$$
$$\overset{\text{def}}{\equiv} \exists x \in a^* : \exists y \in a^* b : \begin{array}{l} \exists n : x = a^{n+1} \wedge y = a^n b \\ \wedge \quad u, v \in A^* \wedge y \preccurlyeq u \not\succcurlyeq x \wedge y \preccurlyeq v \not\succcurlyeq x \end{array}$$

Σ_1-DEFINABLE PROPERTIES

Fix $A = \{a, b\}$. Here are some Σ_1-definable properties:

$$|u|_a < |v_a| \overset{\text{def}}{\equiv} \exists x \in a^* : x \preccurlyeq v \wedge x \not\preccurlyeq u$$
$$\equiv \exists x : b \not\preccurlyeq x \wedge x \preccurlyeq v \wedge x \not\preccurlyeq u$$

$$\exists n > 0 : u = a^{n+1} \wedge v = a^n b$$
$$\overset{\text{def}}{\equiv} u \in a^* \wedge v \in a^* b \wedge |v|_a < |u|_a \wedge \exists x \in a^* baa : v \not\preccurlyeq x \wedge u \preccurlyeq x$$

using $v \in a^* b \overset{\text{def}}{\equiv} b \preccurlyeq v \wedge ba \not\preccurlyeq v \wedge bb \not\preccurlyeq v$

and $x \in a^* baa \overset{\text{def}}{\equiv} baa \preccurlyeq x \wedge bb \not\preccurlyeq x \wedge baaa \not\preccurlyeq x$

$$u, v \in A^* b \wedge |u|_a = |v|_a$$
$$\overset{\text{def}}{\equiv} \exists x \in a^* : \exists y \in a^* b : \begin{array}{l} \exists n : x = a^{n+1} \wedge y = a^n b \\ \wedge \quad u, v \in A^* \wedge y \preccurlyeq u \not\succcurlyeq x \wedge y \preccurlyeq v \not\succcurlyeq x \end{array}$$

MORE Σ_1-DEFINABLE PROPERTIES

$$\exists n : u = aaba^n b \wedge v = aba^{n+1} b \wedge w = ba^{n+2} b$$

$$\stackrel{\text{def}}{\equiv} u \in aaba^*b \wedge v = aba^*b \wedge w = ba^*b$$
$$\wedge \left[u, v, w \in A^*b \wedge |u|_a = |v|_a = |w|_a\right]$$

$$\exists n > 0 : u = ba^n b \wedge v = ba^{n+1} b$$

$$\stackrel{\text{def}}{\equiv} \exists x, y, z : \begin{array}{ll} & \exists m : x = aaba^m b \wedge y = aba^{m+1} b \wedge z = ba^{m+} b \\ \wedge & u, v \in ba^*b \wedge x \not\succcurlyeq u \preccurlyeq y \not\succcurlyeq v \preccurlyeq z \end{array}$$

$$\exists n : u = a^n \wedge v = a^{n+1} \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$v = a^{|u|_a} \equiv v = \pi_a(u) \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$|u|_a = |v|_a \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$u \in a^* \wedge v = bu \wedge w = ub \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$|w|_a = |u|_a + |v|_a \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

MORE Σ_1-DEFINABLE PROPERTIES

$$\exists n : u = aaba^n b \wedge v = aba^{n+1} b \wedge w = ba^{n+2} b$$
$$\stackrel{\text{def}}{\equiv} u \in aaba^* b \wedge v = aba^* b \wedge w = ba^* b$$
$$\wedge \left[u, v, w \in A^* b \wedge |u|_a = |v|_a = |w|_a \right]$$

$$\exists n > 0 : u = ba^n b \wedge v = ba^{n+1} b$$

$$\stackrel{\text{def}}{\equiv} \exists x, y, z : \begin{array}{l} \exists m : x = aaba^m b \wedge y = aba^{m+1} b \wedge z = ba^{m+} b \\ \wedge \; u, v \in ba^* b \wedge x \not\succcurlyeq u \preccurlyeq y \not\succcurlyeq v \preccurlyeq z \end{array}$$

$$\exists n : u = a^n \wedge v = a^{n+1} \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$v = a^{|u|_a} \equiv v = \pi_a(u) \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$|u|_a = |v|_a \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$u \in a^* \wedge v = bu \wedge w = ub \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$|w|_a = |u|_a + |v|_a \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

MORE Σ_1-DEFINABLE PROPERTIES

$$\exists n : u = aaba^n b \wedge v = aba^{n+1} b \wedge w = ba^{n+2} b$$
$$\stackrel{\text{def}}{\equiv} u \in aaba^* b \wedge v = aba^* b \wedge w = ba^* b$$
$$\wedge \left[u, v, w \in A^* b \wedge |u|_a = |v|_a = |w|_a \right]$$

$$\exists n > 0 : u = ba^n b \wedge v = ba^{n+1} b$$
$$\stackrel{\text{def}}{\equiv} \exists x, y, z : \begin{array}{l} \exists m : x = aaba^m b \wedge y = aba^{m+1} b \wedge z = ba^{m+} b \\ \wedge \;\; u, v \in ba^* b \wedge x \not\succcurlyeq u \preccurlyeq y \not\succcurlyeq v \preccurlyeq z \end{array}$$

$$\exists n : u = a^n \wedge v = a^{n+1} \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$v = a^{|u|_a} \equiv v = \pi_a(u) \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$|u|_a = |v|_a \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$u \in a^* \wedge v = bu \wedge w = ub \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$|w|_a = |u|_a + |v|_a \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

MORE Σ_1-DEFINABLE PROPERTIES

$$\exists n : u = aaba^n b \wedge v = aba^{n+1} b \wedge w = ba^{n+2} b$$

$$\stackrel{\text{def}}{\equiv} u \in aaba^* b \wedge v = aba^* b \wedge w = ba^* b$$
$$\wedge \left[u, v, w \in A^* b \wedge |u|_a = |v|_a = |w|_a \right]$$

$$\exists n > 0 : u = ba^n b \wedge v = ba^{n+1} b$$

$$\stackrel{\text{def}}{\equiv} \exists x, y, z : \begin{array}{ll} & \exists m : x = aaba^m b \wedge y = aba^{m+1} b \wedge z = ba^{m+} b \\ \wedge & u, v \in ba^* b \wedge x \not\succcurlyeq u \preccurlyeq y \not\succcurlyeq v \preccurlyeq z \end{array}$$

$$\exists n : u = a^n \wedge v = a^{n+1} \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$v = a^{|u|_a} \equiv v = \pi_a(u) \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$|u|_a = |v|_a \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$u \in a^* \wedge v = bu \wedge w = ub \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$|w|_a = |u|_a + |v|_a \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

More Σ_1-definable properties

$$\exists n : u = aaba^n b \wedge v = aba^{n+1} b \wedge w = ba^{n+2} b$$
$$\stackrel{\text{def}}{\equiv} u \in aaba^* b \wedge v = aba^* b \wedge w = ba^* b$$
$$\wedge \left[u, v, w \in A^* b \wedge |u|_a = |v|_a = |w|_a \right]$$

$$\exists n > 0 : u = ba^n b \wedge v = ba^{n+1} b$$
$$\stackrel{\text{def}}{\equiv} \exists x, y, z : \begin{array}{ll} & \exists m : x = aaba^m b \wedge y = aba^{m+1} b \wedge z = ba^{m+} b \\ \wedge & u, v \in ba^* b \wedge x \not\succcurlyeq u \preccurlyeq y \not\succcurlyeq v \preccurlyeq z \end{array}$$

$$\exists n : u = a^n \wedge v = a^{n+1} \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$v = a^{|u|_a} \equiv v = \pi_a(u) \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$|u|_a = |v|_a \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$u \in a^* \wedge v = bu \wedge w = ub \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

$$|w|_a = |u|_a + |v|_a \stackrel{\text{def}}{\equiv} \exists x, y \cdots$$

More Σ_1-definable properties

u factors as $a^{n_0} b a^{n_1} \cdots b a^{n_k}$ and $v = a^{n_k} \stackrel{\text{def}}{\equiv} \exists x, y \cdots$

$v \in a^* \wedge w = uv \stackrel{\text{def}}{\equiv} \exists x, y \cdots$

$u \leqslant_{\text{prefix}} v \stackrel{\text{def}}{\equiv} \exists y \in a^* : \exists x \leqslant v : x = uy \wedge |x|_a = |v|_a$

$w = uv \stackrel{\text{def}}{\equiv} u \leqslant_{\text{prefix}} w \wedge v \leqslant_{\text{suffix}} w \wedge \bigwedge_{c \in A} |w|_c = |u|_c + |v|_c$

$u \in (ab)^* \stackrel{\text{def}}{\equiv} \exists x : x = abu \wedge x = uab$

$|u|_a = |v|_b \stackrel{\text{def}}{\equiv} \exists x, y \cdots$

$\exists n, m : u = a^n \wedge v = a^m \wedge w = a^{n \cdot m} \stackrel{\text{def}}{\equiv} \exists x, y \cdots$

QED: Diophantine sets can be defined in the Σ_1 fragment

More Σ_1-definable properties

u factors as $a^{n_0}ba^{n_1}\cdots ba^{n_k}$ and $v = a^{n_k} \stackrel{\text{def}}{\equiv} \exists x,y \cdots$

$v \in a^* \wedge w = uv \stackrel{\text{def}}{\equiv} \exists x,y \cdots$

$u \preccurlyeq_{\text{prefix}} v \stackrel{\text{def}}{\equiv} \exists y \in a^* : \exists x \preccurlyeq v : x = uy \wedge |x|_a = |v|_a$

$$w = uv \stackrel{\text{def}}{\equiv} u \preccurlyeq_{\text{prefix}} w \wedge v \preccurlyeq_{\text{suffix}} w \wedge \bigwedge_{c \in A} |w|_c = |u|_c + |v|_c$$

$u \in (ab)^* \stackrel{\text{def}}{\equiv} \exists x : x = abu \wedge x = uab$

$|u|_a = |v|_b \stackrel{\text{def}}{\equiv} \exists x,y \cdots$

$\exists n,m : u = a^n \wedge v = a^m \wedge w = a^{n \cdot m} \stackrel{\text{def}}{\equiv} \exists x,y \cdots$

QED: Diophantine sets can be defined in the Σ_1 fragment

DECIDABLE FRAGMENTS: BOUNDED LETTER ALTERNATION

Assume that all quantifications put letter alternation bounds, i.e., have the form

$$\exists x \in a_1^* a_2^* \cdots a_k^* \qquad \forall y \in b_1^* b_2^* \cdots b_\ell^*$$

Then the full logic is **decidable** in EXPSPACE

If 1 variable is unrestricted (NB: can be reused) and all other variables are alternation bounded, the Σ_1 fragment is NP-complete, the Σ_2-fragment is undecidable

If 2 variables are unrestricted and all other variables are alternation bounded, the Σ_1 fragment is in NEXPTIME.

If 3 variables are unrestricted, we can encode Diophantine sets

Decidable fragments: bounded letter alternation

Assume that all quantifications put letter alternation bounds, i.e., have the form

$$\exists x \in a_1^* a_2^* \cdots a_k^* \qquad \forall y \in b_1^* b_2^* \cdots b_\ell^*$$

Then the full logic is **decidable** in EXPSPACE

If 1 variable is unrestricted (NB: can be reused) and all other variables are alternation bounded, the Σ_1 fragment is NP-complete, the Σ_2-fragment is undecidable

If 2 variables are unrestricted and all other variables are alternation bounded, the Σ_1 fragment is in NEXPTIME.

If 3 variables are unrestricted, we can encode Diophantine sets

DECIDABLE FRAGMENTS: BOUNDED LETTER ALTERNATION

Assume that all quantifications put letter alternation bounds, i.e., have the form

$$\exists x \in a_1^* a_2^* \cdots a_k^* \qquad \forall y \in b_1^* b_2^* \cdots b_\ell^*$$

Then the full logic is **decidable** in EXPSPACE

If 1 variable is unrestricted (NB: can be reused) and all other variables are alternation bounded, the Σ_1 fragment is NP-complete, the Σ_2-fragment is undecidable

If 2 variables are unrestricted and all other variables are alternation bounded, the Σ_1 fragment is in NEXPTIME.

If 3 variables are unrestricted, we can encode Diophantine sets

DECIDABLE FRAGMENTS: BOUNDED LETTER ALTERNATION

Assume that all quantifications put letter alternation bounds, i.e., have the form

$$\exists x \in a_1^* a_2^* \cdots a_k^* \qquad \forall y \in b_1^* b_2^* \cdots b_\ell^*$$

Then the full logic is **decidable** in EXPSPACE

If 1 variable is unrestricted (NB: can be reused) and all other variables are alternation bounded, the Σ_1 fragment is NP-complete, the Σ_2-fragment is undecidable

If 2 variables are unrestricted and all other variables are alternation bounded, the Σ_1 fragment is in NEXPTIME.

If 3 variables are unrestricted, we can encode Diophantine sets

DECIDABLE FRAGMENTS: BOUNDED LETTER ALTERNATION

Assume that all quantifications put letter alternation bounds, i.e., have the form

$$\exists x \in a_1^* a_2^* \cdots a_k^* \qquad \forall y \in b_1^* b_2^* \cdots b_\ell^*$$

Then the full logic is **decidable** in EXPSPACE

If 1 variable is unrestricted (NB: can be reused) and all other variables are alternation bounded, the Σ_1 fragment is NP-complete, the Σ_2-fragment is undecidable

If 2 variables are unrestricted and all other variables are alternation bounded, the Σ_1 fragment is in NEXPTIME.

If 3 variables are unrestricted, we can encode Diophantine sets

How do we describe words
via short subwords?

Joint work with M. Veron

DEFINING WORDS VIA THEIR SUBWORDS

$$x = \text{ABBA iff } \begin{cases} & \text{AA} \preccurlyeq x & \wedge & \text{AAA} \not\preccurlyeq x & \\ \wedge & \text{BB} \preccurlyeq x & \wedge & \text{BBB} \not\preccurlyeq x & \\ \wedge & \text{BAB} \not\preccurlyeq x & \wedge & \text{AAB} \not\preccurlyeq x & \wedge \quad \text{BAA} \not\preccurlyeq x \end{cases}$$

Defining words via their subwords

$$x = \text{ABBA iff } \begin{cases} \quad \text{AA} \preccurlyeq x & \wedge \quad \text{AAA} \not\preccurlyeq x \\ \wedge \quad \text{BB} \preccurlyeq x & \wedge \quad \text{BBB} \not\preccurlyeq x \\ \wedge \quad \text{BAB} \not\preccurlyeq x & \wedge \quad \text{AAB} \not\preccurlyeq x \quad \wedge \quad \text{BAA} \not\preccurlyeq x \end{cases}$$

Thus ABBA can be defined via subword constraints of length at most 3

Defining words via their subwords

$$x = \text{ABBA iff} \begin{cases} \quad \text{AA} \preccurlyeq x & \wedge \quad \text{AAA} \not\preccurlyeq x \\ \wedge \quad \text{BB} \preccurlyeq x & \wedge \quad \text{BBB} \not\preccurlyeq x \\ \wedge \quad \text{BAB} \not\preccurlyeq x & \wedge \quad \text{AAB} \not\preccurlyeq x \quad \wedge \quad \text{BAA} \not\preccurlyeq x \end{cases}$$

Thus ABBA can be defined via subword constraints of length at most 3

Other examples: ABRACADABRA is definable with length-4 constraints, so too is THE WORKS OF SHAKESPEARE

DEFINING WORDS VIA THEIR SUBWORDS

$$x = \text{ABBA iff } \begin{cases} \quad \text{AA} \preccurlyeq x \quad \wedge \quad \text{AAA} \not\preccurlyeq x \\ \wedge \quad \text{BB} \preccurlyeq x \quad \wedge \quad \text{BBB} \not\preccurlyeq x \\ \wedge \quad \text{BAB} \not\preccurlyeq x \quad \wedge \quad \text{AAB} \not\preccurlyeq x \quad \wedge \quad \text{BAA} \not\preccurlyeq x \end{cases}$$

Thus ABBA can be defined via subword constraints of length at most 3

Other examples: ABRACADABRA is definable with length-4 constraints, so too is THE WORKS OF SHAKESPEARE

We write $h(\text{ABRACADABRA}) = 4$ and refer to the "piecewise complexity" of a word

How do you compute $h(u)$? What are its main properties?

Some more motivations

Piecewise complexity originally defined for piecewise-testable languages (Karandikar & S. 2019)
This allowed proving elementary complexity upper bounds for the aforementioned FO^2 logic of subwords

Piecewise-testable languages (Imre Simon 1972) are the languages definable by subword constraints
Also: definable in the $\mathcal{B}\Sigma_1$ fragment of the first-order logic of words
Also: the languages with a $\mathcal{J}$-trivial syntactic monoid

Here $h(u)$ and $h(L)$ is the number of variables needed in a $\mathcal{B}\Sigma_1$ formula defining u or L

These notions can be, and have been, generalized to many settings: trees, graphs, infinite words, etc.

Some more motivations

Piecewise complexity originally defined for piecewise-testable languages (Karandikar & S. 2019)
This allowed proving elementary complexity upper bounds for the aforementioned FO^2 logic of subwords

Piecewise-testable languages (Imre Simon 1972) are the languages definable by subword constraints
Also: definable in the $\mathcal{B}\Sigma_1$ fragment of the first-order logic of words
Also: the languages with a $\mathcal{J}$-trivial syntactic monoid

Here $h(u)$ and $h(L)$ is the number of variables needed in a $\mathcal{B}\Sigma_1$ formula defining u or L

These notions can be, and have been, generalized to many settings: trees, graphs, infinite words, etc.

Some more motivations

Piecewise complexity originally defined for piecewise-testable languages (Karandikar & S. 2019)
This allowed proving elementary complexity upper bounds for the aforementioned FO^2 logic of subwords

Piecewise-testable languages (Imre Simon 1972) are the languages definable by subword constraints
Also: definable in the $\mathcal{B}\Sigma_1$ fragment of the first-order logic of words
Also: the languages with a $\mathcal{J}$-trivial syntactic monoid

Here $h(u)$ and $h(L)$ is the number of variables needed in a $\mathcal{B}\Sigma_1$ formula defining u or L

These notions can be, and have been, generalized to many settings: trees, graphs, infinite words, etc.

Some definitions: Simon's congruence

Def. [Simon's congruence, 1972] $u \sim_k v$ if u and v have the same subwords of length $\leqslant k$

Def. [Simon and Sakarovich, 1983] $\delta(u,v) \stackrel{\text{def}}{=} \max\{k \mid u \sim_k v\}$

One wants to compute a distinguisher between two words u,v, or to compute $\delta(u,v)$, or to check whether $u \sim_k v$

In some applications (DNA strings, program executions, ..) the words can be very long

Simon claimed he had a linear $O(|uv|)$ algorithm. A bilinear $O(|uv| \cdot |A|)$ algorithm was given by Fleischer and Kufleitner (2018), improved to $O(|uv|)$ by Barker, Fleischmann et al. (2020).

Piecewise complexity and Simon's congruence

Def. $h(u) \stackrel{\text{def}}{=} \min\{k \mid \forall v : u \sim_k v \implies u = v\}$

E.g. for $u = \texttt{ABRACADABRA}$: $h(u) = 4$

PIECEWISE COMPLEXITY AND SIMON'S CONGRUENCE

Def. $h(u) \stackrel{\text{def}}{=} \min\{k \mid \forall v : u \sim_k v \implies u = v\}$

E.g. for $u = \texttt{ABRACADABRA}$: $h(u) = 4$

Main tool: r and ℓ "side distance" functions:

$$r(u,t) \stackrel{\text{def}}{=} \delta(u,ut) \qquad\qquad \ell(t,u) \stackrel{\text{def}}{=} \delta(tu,u)$$

Def. $h(u) \overset{\text{def}}{=} \min\{k \mid \forall v : u \sim_k v \implies u = v\}$

E.g. for $u =$ ABRACADABRA: $h(u) = 4$

Main tool: r and ℓ "side distance" functions:

$$r(u,t) \overset{\text{def}}{=} \delta(u,ut) \qquad\qquad \ell(t,u) \overset{\text{def}}{=} \delta(tu,u)$$

r and ℓ allow a reformulation of our computational problem:

$$h(u) = \max_{\substack{u=u_1u_2\\ a\in A}} r(u_1,a) + \ell(a,u_2) + 1$$

Recursive algorithm for side functions

$$r(ub,a) = \begin{cases} 0 & \text{if } a \notin ub \\ 1 + r(u,a) & \text{if } a = b \\ \min\left\{ \begin{array}{c} 1 + r(u_1,b) \\ r(u,a) \end{array} \right\} & \text{if } a \neq b \text{ and } u = u_1 a u_2 \text{ with } a \notin u_2 \end{cases}$$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
w		A	B	B	A	C	C	B	C	C	A	B	A	A	B	C
$r(i,A)$	0	1	1	1	2	1	1	1	1	1	2	2	3	4	4	3
$r(i,B)$	0	0	1	2	2	1	1	2	2	2	2	3	3	3	4	3
$r(i,C)$	0	0	0	0	0	1	2	2	3	4	2	2	2	2	2	3
$\ell(A,i)$	4	3	3	3	2	2	2	2	2	3	2	2	1	0	0	0
$\ell(B,i)$	4	5	4	3	3	3	3	2	2	2	2	1	1	1	0	0
$\ell(C,i)$	3	3	3	3	5	4	3	3	2	1	1	1	1	1	1	0

In this example, $h(w) = 6$

Prop. $h(u)$ can be computed in bilinear time $O(|A| \cdot |u|)$

Recursive algorithm for side functions

$$r(ub,a) = \begin{cases} 0 & \text{if } a \notin ub \\ 1 + r(u,a) & \text{if } a = b \\ \min\left\{ \begin{array}{c} 1 + r(u_1,b) \\ r(u,a) \end{array} \right\} & \text{if } a \neq b \text{ and } u = u_1 a u_2 \text{ with } a \notin u_2 \end{cases}$$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
w		A	B	B	A	C	C	B	C	C	A	B	A	A	B	C
$r(i,A)$	0	1	1	1	2	1	1	1	1	1	2	2	3	4	4	3
$r(i,B)$	0	0	1	2	2	1	1	2	2	2	2	3	3	3	4	3
$r(i,C)$	0	0	0	0	0	1	2	2	3	4	2	2	2	2	2	3
$\ell(A,i)$	4	3	3	3	2	2	2	2	2	3	2	2	1	0	0	0
$\ell(B,i)$	4	5	4	3	3	3	3	2	2	2	2	1	1	1	0	0
$\ell(C,i)$	3	3	3	3	5	4	3	3	2	1	1	1	1	1	1	0

In this example, $h(w) = 6$

Prop. $h(u)$ can be computed in bilinear time $O(|A| \cdot |u|)$

Recursive algorithm for side functions

$$r(ub,a) = \begin{cases} 0 & \text{if } a \notin ub \\ 1 + r(u,a) & \text{if } a = b \\ \min\left\{ \begin{array}{c} 1 + r(u_1,b) \\ r(u,a) \end{array} \right\} & \text{if } a \neq b \text{ and } u = u_1 a u_2 \text{ with } a \notin u_2 \end{cases}$$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
w		A	B	B	A	C	C	B	C	C	A	B	A	A	B	C
$r(i,A)$	0	1	1	1	2	1	1	1	1	1	2	2	3	4	4	3
$r(i,B)$	0	0	1	2	2	1	1	2	2	2	2	3	3	3	4	3
$r(i,C)$	0	0	0	0	0	1	2	2	3	4	2	2	2	2	2	3
$\ell(A,i)$	4	3	3	3	2	2	2	2	2	3	2	2	1	0	0	0
$\ell(B,i)$	4	5	4	3	3	3	3	2	2	2	2	1	1	1	0	0
$\ell(C,i)$	3	3	3	3	5	4	3	3	2	1	1	1	1	1	1	0

In this example, $h(w) = 6$

Prop. $h(u)$ can be computed in bilinear time $O(|A| \cdot |u|)$

CONCLUDING REMARKS

Subwords appear everywhere.

Surprisingly many basic questions are still unanswered, even unasked.

I have more subword-based puzzles if you're interested . . .