

# Computation with Absolutely No Space Overhead

Lane Hemaspaandra<sup>1</sup>   Proshanto Mukherji<sup>1</sup>   Till Tantau<sup>2</sup>

<sup>1</sup>Department of Computer Science  
University of Rochester

<sup>2</sup>Fakultät für Elektrotechnik und Informatik  
Technical University of Berlin

Developments in Language Theory Conference, 2003



# Outline



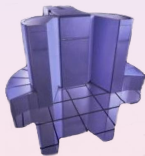
# Outline



# The Standard Model of Linear Space

tape

0	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---



Turing machine

## Characteristics

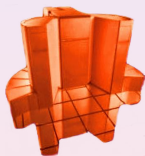
- Input fills **fixed-size** tape
- Input may be **modified**
- Tape alphabet **is larger than** input alphabet



# The Standard Model of Linear Space

tape

\$	0	1	0	0	1	0	0
----	---	---	---	---	---	---	---



Turing machine

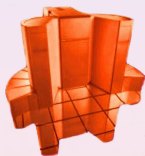
## Characteristics

- Input fills **fixed-size** tape
- Input may be **modified**
- Tape alphabet **is larger than** input alphabet

# The Standard Model of Linear Space

tape

\$	0	1	0	0	1	0	0
----	---	---	---	---	---	---	---



Turing machine

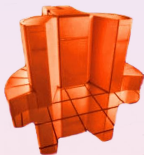
## Characteristics

- Input fills **fixed-size** tape
- Input may be **modified**
- Tape alphabet **is larger than** input alphabet

# The Standard Model of Linear Space

tape

\$	0	1	0	0	1	0	\$
----	---	---	---	---	---	---	----



Turing machine

## Characteristics

- Input fills **fixed-size** tape
- Input may be **modified**
- Tape alphabet **is larger than** input alphabet

# The Standard Model of Linear Space

tape

\$	0	1	0	0	1	0	\$
----	---	---	---	---	---	---	----



Turing machine

## Characteristics

- Input fills **fixed-size** tape
- Input may be **modified**
- Tape alphabet **is larger than** input alphabet



# The Standard Model of Linear Space

tape

\$	\$	1	0	0	1	0	\$
----	----	---	---	---	---	---	----



Turing machine

## Characteristics

- Input fills **fixed-size** tape
- Input may be **modified**
- Tape alphabet **is larger than** input alphabet

# The Standard Model of Linear Space

tape

\$	\$	1	0	0	1	0	\$
----	----	---	---	---	---	---	----



Turing machine

## Characteristics

- Input fills **fixed-size** tape
- Input may be **modified**
- Tape alphabet **is larger than** input alphabet

# The Standard Model of Linear Space

tape

\$	\$	1	0	0	1	\$	\$
----	----	---	---	---	---	----	----



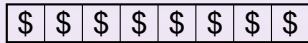
Turing machine

## Characteristics

- Input fills **fixed-size** tape
- Input may be **modified**
- Tape alphabet **is larger than** input alphabet

# The Standard Model of Linear Space

tape



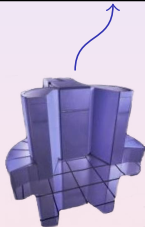
Turing machine

## Characteristics

- Input fills **fixed-size** tape
- Input may be **modified**
- Tape alphabet **is larger than** input alphabet

# The Standard Model of Linear Space

tape

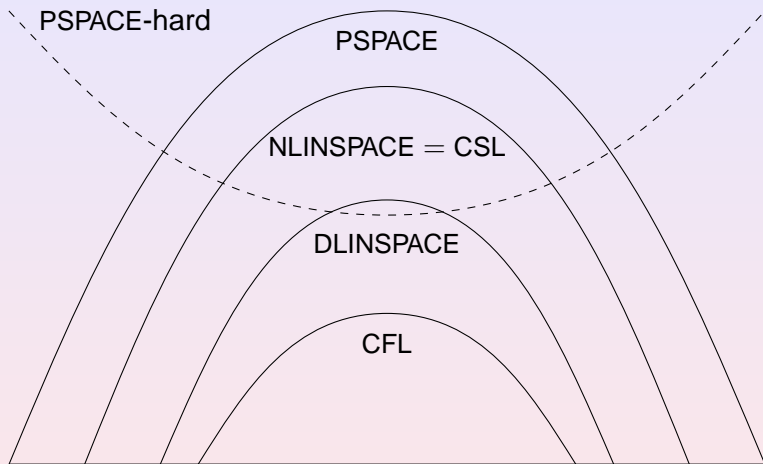


Turing machine

## Characteristics

- Input fills **fixed-size** tape
- Input may be **modified**
- Tape alphabet **is larger than** input alphabet

# Linear Space is a Powerful Model



# Outline



# Our Model of “Absolutely No Space Overhead”

tape

0	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---



Turing machine

## Characteristics

- Input fills **fixed-size tape**
- Input may be **modified**
- Tape alphabet **equals** input alphabet





# Our Model of “Absolutely No Space Overhead”

tape

1	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---



Turing machine

## Characteristics

- Input fills **fixed-size** tape
- Input may be **modified**
- Tape alphabet **equals** input alphabet



# Our Model of “Absolutely No Space Overhead”

tape

1	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---



Turing machine

## Characteristics

- Input fills **fixed-size** tape
- Input may be **modified**
- Tape alphabet **equals** input alphabet



# Our Model of “Absolutely No Space Overhead”

tape

1	0	1	0	0	1	0	1
---	---	---	---	---	---	---	---



Turing machine

## Characteristics

- Input fills **fixed-size** tape
- Input may be **modified**
- Tape alphabet **equals** input alphabet

# Our Model of “Absolutely No Space Overhead”

tape

1	0	1	0	0	1	0	1
---	---	---	---	---	---	---	---



Turing machine

## Characteristics

- Input fills **fixed-size** tape
- Input may be **modified**
- Tape alphabet **equals** input alphabet



# Our Model of “Absolutely No Space Overhead”

tape

1	1	1	0	0	1	0	1
---	---	---	---	---	---	---	---



Turing machine

## Characteristics

- Input fills **fixed-size** tape
- Input may be **modified**
- Tape alphabet **equals** input alphabet



# Our Model of “Absolutely No Space Overhead”



Turing machine

## Intuition

- Tape is used like a RAM module.

# Definition of Overhead-Free Computations

## Definition

A Turing machine is **overhead-free** if

- 1 it has only a single tape,
- 2 writes only on input cells,
- 3 writes only symbols drawn from the input alphabet.



# Overhead-Free Computation Complexity Classes

## Definition

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

**DOF** if  $L$  is accepted by a deterministic overhead-free machine with input alphabet  $\Sigma$ ,

$\text{DOF}_{\text{poly}}$  if  $L$  is accepted by a deterministic overhead-free machine with input alphabet  $\Sigma$  in polynomial time.

$\text{NOF}$  is the nondeterministic version of DOF,

$\text{NOF}_{\text{poly}}$  is the nondeterministic version of  $\text{DOF}_{\text{poly}}$ .





# Overhead-Free Computation Complexity Classes

## Definition

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

**DOF** if  $L$  is accepted by a deterministic overhead-free machine with input alphabet  $\Sigma$ ,

**DOF<sub>poly</sub>** if  $L$  is accepted by a deterministic overhead-free machine with input alphabet  $\Sigma$  in polynomial time.

**NOF** is the nondeterministic version of DOF,

**NOF<sub>poly</sub>** is the nondeterministic version of DOF<sub>poly</sub>.



# Overhead-Free Computation Complexity Classes

## Definition

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

**DOF** if  $L$  is accepted by a deterministic overhead-free machine with input alphabet  $\Sigma$ ,

**DOF<sub>poly</sub>** if  $L$  is accepted by a deterministic overhead-free machine with input alphabet  $\Sigma$  in polynomial time.

**NOF** is the nondeterministic version of DOF,

**NOF<sub>poly</sub>** is the nondeterministic version of DOF<sub>poly</sub>.



# Overhead-Free Computation Complexity Classes

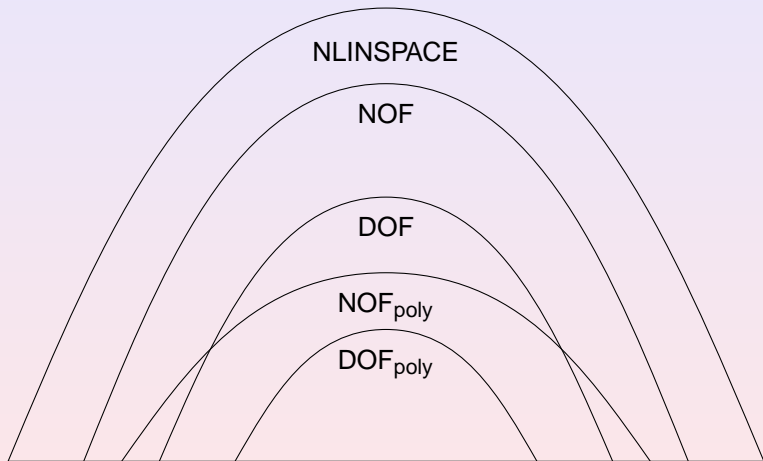
## Definition

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

- DOF** if  $L$  is accepted by a deterministic overhead-free machine with input alphabet  $\Sigma$ ,
- DOF<sub>poly</sub>** if  $L$  is accepted by a deterministic overhead-free machine with input alphabet  $\Sigma$  in polynomial time.
- NOF** is the nondeterministic version of DOF,
- NOF<sub>poly</sub>** is the nondeterministic version of DOF<sub>poly</sub>.



# Simple Relationships among Overhead-Free Computation Classes



# Outline



# Palindromes Can be Accepted in an Overhead-Free Way

tape

0	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

Find left end marker

Advance left end marker

Find right end marker

Advance right end marker



# Palindromes Can be Accepted in an Overhead-Free Way

tape

1	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

Find left end marker

Advance left end marker

Find right end marker

Advance right end marker



# Palindromes Can be Accepted in an Overhead-Free Way

tape

1	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

Find left end marker

Advance left end marker

Find right end marker

Advance right end marker





# Palindromes Can be Accepted in an Overhead-Free Way

tape

1	0	1	0	0	1	0	1
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

Find left end marker

Advance left end marker

Find right end marker

Advance right end marker



# Palindromes Can be Accepted in an Overhead-Free Way

tape

1	0	1	0	0	1	0	1
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

**Find left end marker**

Advance left end marker

Find right end marker

Advance right end marker



# Palindromes Can be Accepted in an Overhead-Free Way

tape

0	1	1	0	0	1	0	1
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

Find left end marker

**Advance left end marker**

Find right end marker

Advance right end marker



# Palindromes Can be Accepted in an Overhead-Free Way

tape

0	1	1	0	0	1	0	1
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

Find left end marker

Advance left end marker

**Find right end marker**

Advance right end marker



# Palindromes Can be Accepted in an Overhead-Free Way

tape

0	1	1	0	0	1	1	0
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

Find left end marker

Advance left end marker

Find right end marker

Advance right end marker



# Palindromes Can be Accepted in an Overhead-Free Way

tape

0	1	1	0	0	1	1	0
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

**Find left end marker**

Advance left end marker

Find right end marker

Advance right end marker



# Palindromes Can be Accepted in an Overhead-Free Way

tape

0	0	1	0	0	1	1	0
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

Find left end marker

**Advance left end marker**

Find right end marker

Advance right end marker



# Palindromes Can be Accepted in an Overhead-Free Way

tape

0	0	1	0	0	1	1	0
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

Find left end marker

Advance left end marker

**Find right end marker**

Advance right end marker





# Palindromes Can be Accepted in an Overhead-Free Way

tape

0	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

Find left end marker

Advance left end marker

Find right end marker

Advance right end marker



# Palindromes Can be Accepted in an Overhead-Free Way

tape

0	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

**Find left end marker**

Advance left end marker

Find right end marker

Advance right end marker



# Palindromes Can be Accepted in an Overhead-Free Way

tape

0	0	0	1	0	1	0	0
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

Find left end marker

**Advance left end marker**

Find right end marker

Advance right end marker



# Palindromes Can be Accepted in an Overhead-Free Way

tape

0	0	0	1	0	1	0	0
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

Find left end marker

Advance left end marker

**Find right end marker**

Advance right end marker



# Palindromes Can be Accepted in an Overhead-Free Way

tape

0	0	0	1	1	0	0	0
---	---	---	---	---	---	---	---



overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

Place left end marker

Place right end marker

Phase 2:

Compare bits next to end markers

Find left end marker

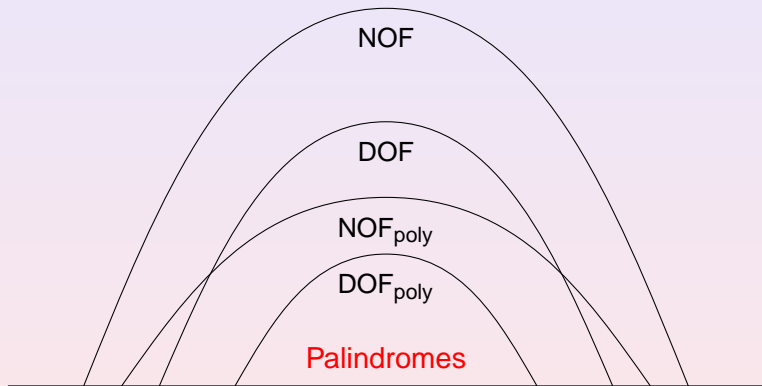
Advance left end marker

Find right end marker

**Advance right end marker**



# Relationships among Overhead-Free Computation Classes



# Outline



# A Review of Linear Grammars

## Definition

A grammar is **linear** if it is context-free and there is only one nonterminal per right-hand side.

## Example

$G_1: S \rightarrow 00S0 \mid 1$  and  $G_2: S \rightarrow 0S10 \mid 0$ .

## Definition

A grammar is **deterministic** if  
“there is always only one rule that can be applied.”

## Example

$G_1: S \rightarrow 00S0 \mid 1$  is deterministic.

$G_2: S \rightarrow 0S10 \mid 0$  is **not** deterministic.



# A Review of Linear Grammars

## Definition

A grammar is **linear** if it is context-free and there is only one nonterminal per right-hand side.

## Example

$G_1: S \rightarrow 00S0 \mid 1$  and  $G_2: S \rightarrow 0S10 \mid 0$ .

## Definition

A grammar is **deterministic** if  
“there is always only one rule that can be applied.”

## Example

$G_1: S \rightarrow 00S0 \mid 1$  is deterministic.

$G_2: S \rightarrow 0S10 \mid 0$  is **not** deterministic.

# Deterministic Linear Languages Can Be Accepted in an Overhead-Free Way

## Theorem

*Every deterministic linear language is in  $\text{DOF}_{\text{poly}}$ .*



# Metalinear Languages

## Can Be Accepted in an Overhead-Free Way

### Definition

A language is **metalinear** if it is the concatenation of linear languages.

### Example

TRIPLE-PALINDROME =  $\{uvw \mid u, v, \text{ and } w \text{ are palindromes}\}$ .

### Theorem

*Every metalinear language is in  $\text{NOF}_{\text{poly}}$ .*



# Metalinear Languages

## Can Be Accepted in an Overhead-Free Way

### Definition

A language is **metalinear** if it is the concatenation of linear languages.

### Example

TRIPLE-PALINDROME =  $\{uvw \mid u, v, \text{ and } w \text{ are palindromes}\}$ .

### Theorem

*Every metalinear language is in  $\text{NOF}_{\text{poly}}$ .*



# Metalinear Languages

## Can Be Accepted in an Overhead-Free Way

### Definition

A language is **metalinear** if it is the concatenation of linear languages.

### Example

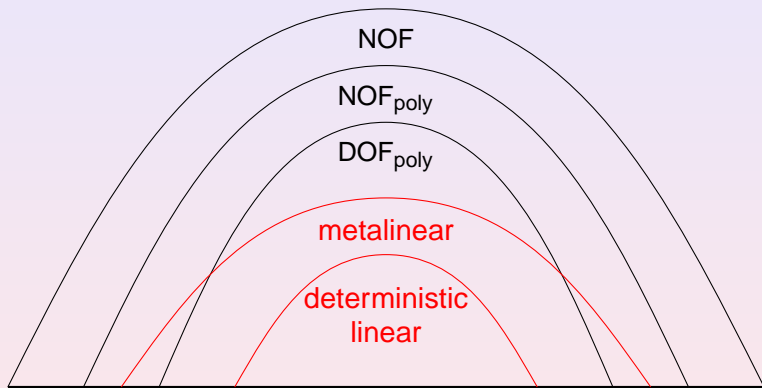
TRIPLE-PALINDROME =  $\{uvw \mid u, v, \text{ and } w \text{ are palindromes}\}$ .

### Theorem

*Every metalinear language is in  $\text{NOF}_{\text{poly}}$ .*



# Relationships among Overhead-Free Computation Classes



# Outline



# Definition of Almost-Overhead-Free Computations

## Definition

A Turing machine is **almost-overhead-free** if

- 1 it has only a single tape,
- 2 writes only on input cells,
- 3 writes only symbols drawn from the input alphabet plus one special symbol.





# Definition of Almost-Overhead-Free Computations

## Definition

A Turing machine is **almost-overhead-free** if

- 1 it has only a single tape,
- 2 **writes only on input cells,**
- 3 writes only symbols drawn from the input alphabet plus one special symbol.



# Definition of Almost-Overhead-Free Computations

## Definition

A Turing machine is **almost-overhead-free** if

- 1 it has only a single tape,
- 2 writes only on input cells,
- 3 **writes only symbols drawn from the input alphabet plus one special symbol.**



# Context-Free Languages with a Forbidden Subword Can Be Accepted in an Overhead-Free Way

## Theorem

*Let  $L$  be a context-free language with a forbidden word.  
Then  $L \in \text{NOF}_{\text{poly}}$ .*

► Skip proof



# Context-Free Languages with a Forbidden Subword Can Be Accepted in an Overhead-Free Way

## Theorem

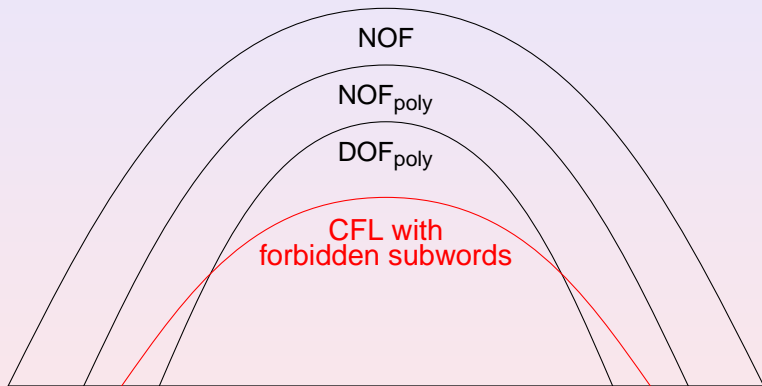
*Let  $L$  be a context-free language with a forbidden word.  
Then  $L \in \text{NOF}_{\text{poly}}$ .*

## Proof.

Every context-free language can be accepted by a nondeterministic almost-overhead-free machine in polynomial time. □



# Relationships among Overhead-Free Computation Classes



# Outline



# Overhead-Free Languages can be PSPACE-Complete

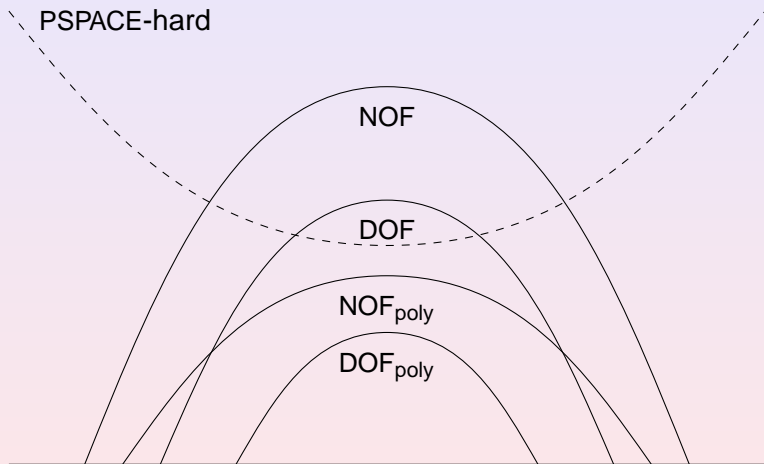
## Theorem

DOF *contains languages that are complete for PSPACE.*

► Proof details



# Relationships among Overhead-Free Computation Classes





# Outline



# Some Context-Sensitive Languages Cannot be Accepted in an Overhead-Free Way

## Theorem

$\text{DOF} \subsetneq \text{DLINSPACE}$ .

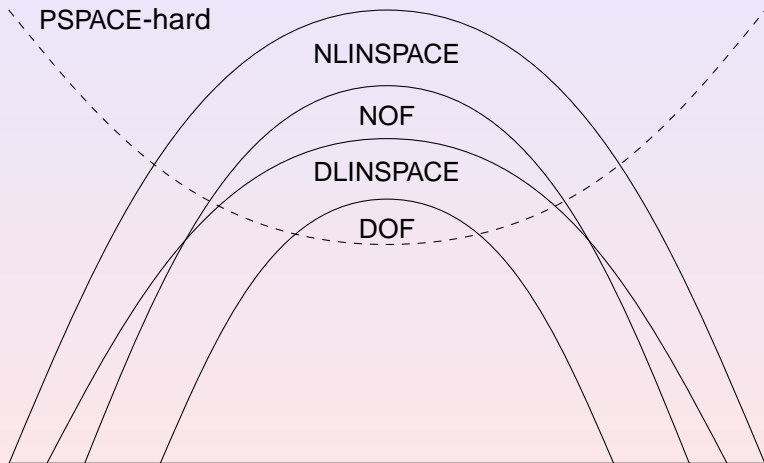
## Theorem

$\text{NOF} \subsetneq \text{NLINSPACE}$ .

The proofs are based on old diagonalisations due to Feldman, Owings, and Seiferas.



# Relationships among Overhead-Free Computation Classes



# Candidates for Languages that Cannot be Accepted in an Overhead-Free Way

## Conjecture

DOUBLE-PALINDROMES  $\notin$  DOF.

## Conjecture

$\{ww \mid w \in \{0, 1\}^*\} \notin$  NOF.

Proving the first conjecture would show  $\text{DOF} \subsetneq \text{NOF}$ .



# Candidates for Languages that Cannot be Accepted in an Overhead-Free Way

## Theorem

DOUBLE-PALINDROMES  $\in$  DOF.

## Conjecture

$\{ww \mid w \in \{0,1\}^*\} \notin \text{NOF}.$

Proving the first conjecture would show  $\text{DOF} \subsetneq \text{NOF}.$



# Summary

- Overhead-free computation is a more faithful **model of fixed-size memory**.
- Overhead-free computation is **less powerful** than linear space.
- **Many** context-free languages can be accepted by overhead-free machines.
- We conjecture that **all** context-free languages are in  $\text{NOF}_{\text{poly}}$ .
- Our results can be seen as new results on the power of **linear bounded automata with fixed alphabet** size.



# For Further Reading



A. Salomaa.  
*Formal Languages.*  
Academic Press, 1973.



E. Dijkstra.  
Smoothsort, an alternative for sorting in situ.  
*Science of Computer Programming*, 1(3):223–233, 1982.



E. Feldman and J. Owings, Jr.  
A class of universal linear bounded automata.  
*Information Sciences*, 6:187–190, 1973.



P. Jančar, F. Mráz, M. Plátek, and J. Vogel.  
Restarting automata.  
*FCT Conference 1995*, LNCS 985, pages 282–292. 1995.



# For Further Reading



A. Salomaa.

*Formal Languages.*

Academic Press, 1973.



E. Dijkstra.

Smoothsort, an alternative for sorting in situ.

*Science of Computer Programming*, 1(3):223–233, 1982.



E. Feldman and J. Owings, Jr.

A class of universal linear bounded automata.

*Information Sciences*, 6:187–190, 1973.



P. Jančar, F. Mráz, M. Plátek, and J. Vogel.

Restarting automata.

*FCT Conference 1995*, LNCS 985, pages 282–292. 1995.





# For Further Reading



A. Salomaa.

*Formal Languages.*

Academic Press, 1973.



E. Dijkstra.

Smoothsort, an alternative for sorting in situ.

*Science of Computer Programming*, 1(3):223–233, 1982.



E. Feldman and J. Owings, Jr.

A class of universal linear bounded automata.

*Information Sciences*, 6:187–190, 1973.



P. Jančar, F. Mráz, M. Plátek, and J. Vogel.

Restarting automata.

*FCT Conference 1995*, LNCS 985, pages 282–292. 1995.



# For Further Reading



A. Salomaa.

*Formal Languages.*

Academic Press, 1973.



E. Dijkstra.

Smoothsort, an alternative for sorting in situ.

*Science of Computer Programming*, 1(3):223–233, 1982.



E. Feldman and J. Owings, Jr.

A class of universal linear bounded automata.

*Information Sciences*, 6:187–190, 1973.



P. Jančar, F. Mráz, M. Plátek, and J. Vogel.

Restarting automata.

*FCT Conference 1995*, LNCS 985, pages 282–292. 1995.



# Appendix Outline



# Overhead-Free Languages can be PSPACE-Complete

## Theorem

DOF contains languages that are complete for PSPACE.

## Proof.

- 1 Let  $A \in \text{DLINSPACE}$  be PSPACE-complete.  
Such languages are known to exist.
- 2 Let  $M$  be a linear space machine that accepts  $A \subseteq \{0, 1\}^*$   
with tape alphabet  $\Gamma$ .
- 3 Let  $h: \Gamma \rightarrow \{0, 1\}^*$  be an isometric, injective  
homomorphism.
- 4 Then  $h(L)$  is in DOF and it is PSPACE-complete. □

◀ Return



## Theorem

- 1  $\text{DCFL} \subseteq \text{DOF}_{\text{poly}}$ .
- 2  $\text{CFL} \subseteq \text{NOF}_{\text{poly}}$ .

# Explanation of Different Abbreviations

DOF	Deterministic Overhead-Free.
NOF	Nondeterministic Overhead-Free.
DOF <sub>poly</sub>	Deterministic Overhead-Free, polynomial time.
DOF <sub>poly</sub>	Nondeterministic Overhead-Free, polynomial time.

Table: Explanation of what different abbreviations mean.

