# Computation with Absolutely No Space Overhead

Lane Hemaspaandra[1] Proshanto Mukherji[1] Till Tantau[2]

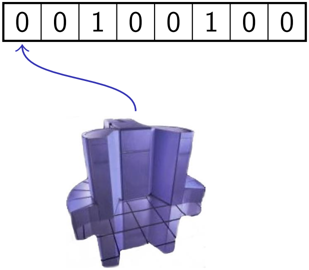
[1]Department of Computer Science
University of Rochester

[2]Fakultät für Elektrotechnik und Informatik
Technical University of Berlin

Developments in Language Theory Conference, 2003

# 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

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $ | 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

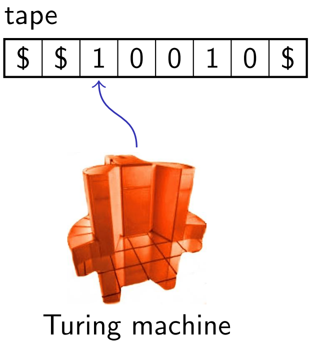
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $ | 0 | 1 | 0 | 0 | 1 | 0 | $ |

Turing machine

## Characteristics

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

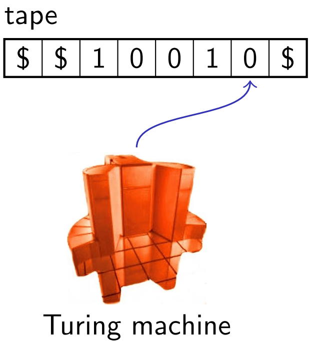
# The Standard Model of Linear Space




## Characteristics

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

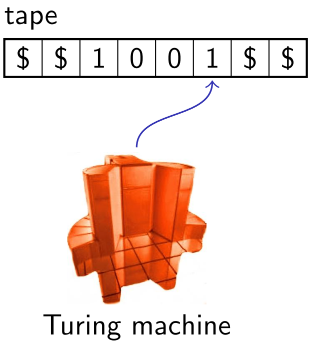
# The Standard Model of Linear Space




## Characteristics

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

# The Standard Model of Linear Space

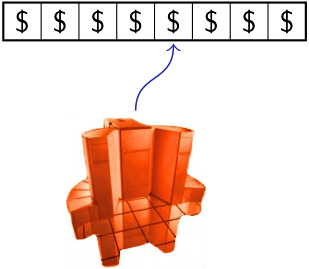



## Characteristics

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

# The Standard Model of Linear Space

tape



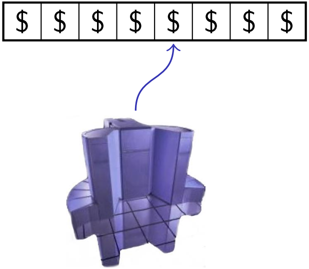
Turing machine

## Characteristics

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

# The Standard Model of Linear Space

tape

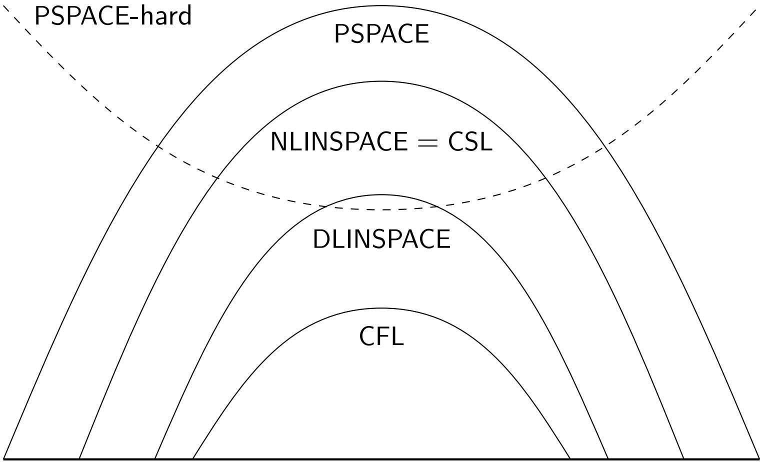



Turing machine

## Characteristics

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

# Linear Space is a Powerful Model



# Our Model of “Absolutely No Space Overhead”

tape

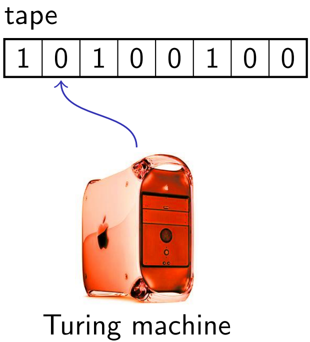
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

Turing machine

## Characteristics

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

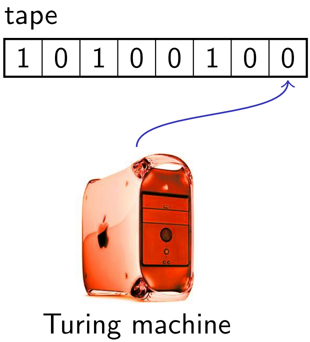
# Our Model of “Absolutely No Space Overhead”




## Characteristics

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

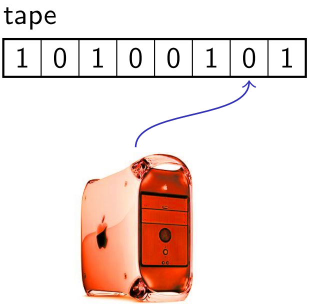
# Our Model of “Absolutely No Space Overhead”




## Characteristics

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

# Our Model of "Absolutely No Space Overhead"



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

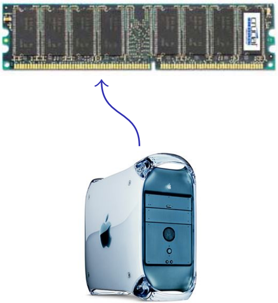
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |

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

- it has only a single tape,
- writes only on input cells,
- 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 $\Sigma$,

# 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.

# 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 $\Sigma$,
- $\mathrm{DOF}_{\mathrm{poly}}$ if $L$ is accepted by a deterministic overhead-free machine with input alphabet $\Sigma$ in polynomial time.
- NOF is the nondeterministic version of DOF,

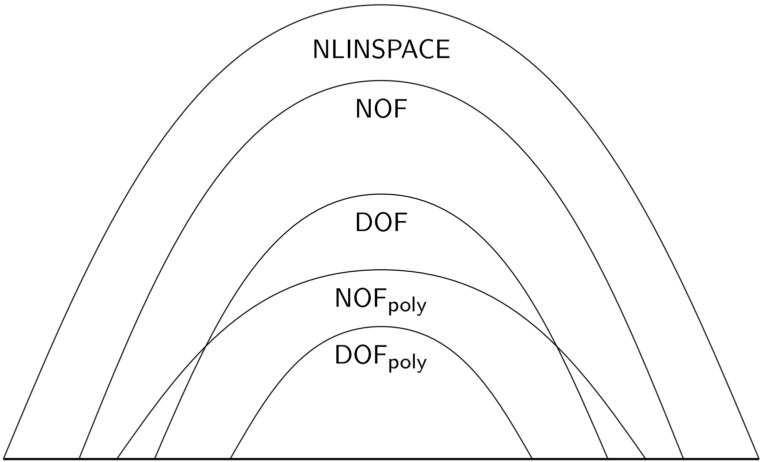
# Overhead-Free Computation Complexity Classes

## Definition

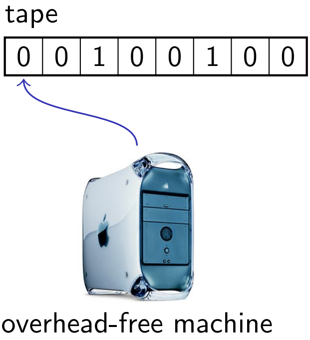
A language $L \subseteq \Sigma^*$ is in

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

# Simple Relationships among Overhead-Free Computation Classes



# Palindromes Can be Accepted in an Overhead-Free Way

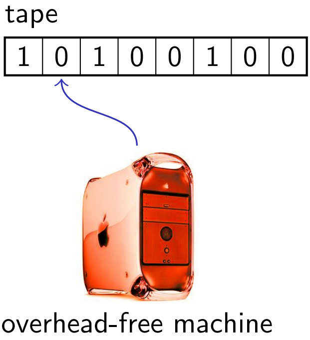



## Algorithm

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



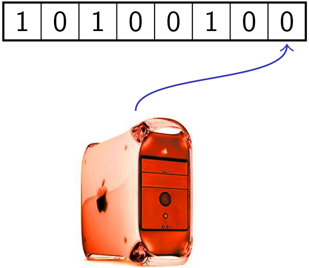

## Algorithm

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



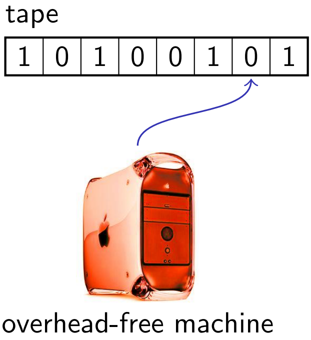

overhead-free machine

## 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




## 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 |

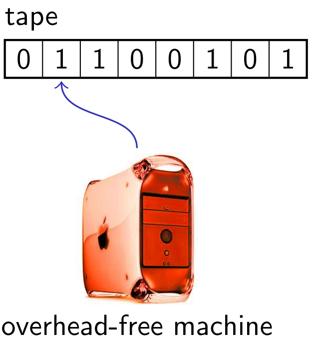
overhead-free machine

## 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

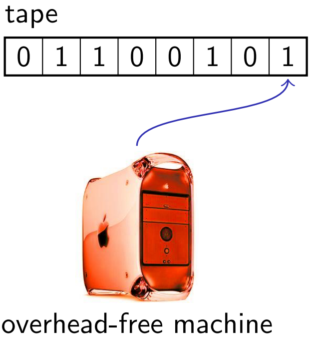



## Algorithm

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




## 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 |
|---|---|---|---|---|---|---|---|

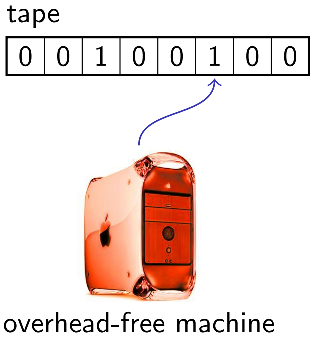
overhead-free machine

## 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

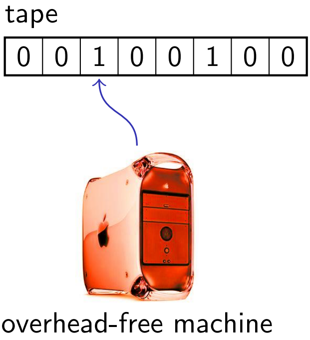



## Algorithm

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



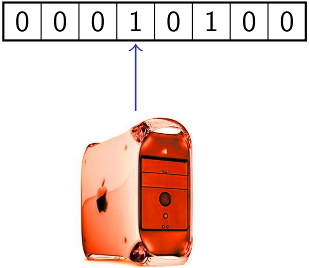

## Algorithm

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




overhead-free machine

## Algorithm

Phase 1:

Compare first and last bit

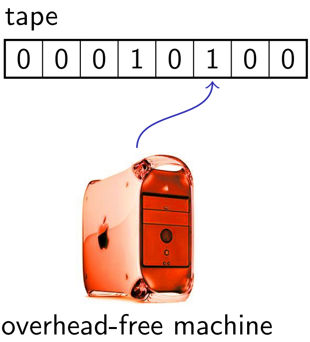
- Place left end marker
- Place right end marker

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

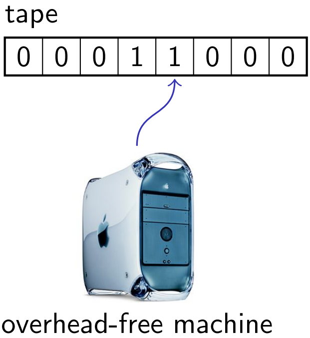



## Algorithm

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

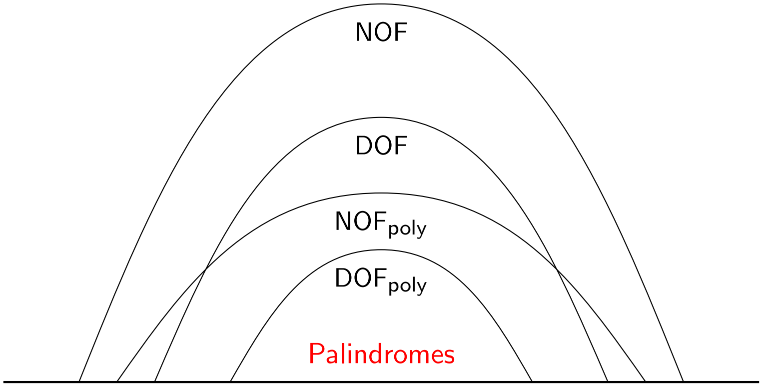



## Algorithm

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



# 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

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

# 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

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

### Definition

A grammar is deterministic if "there is always only one rule that can be applied."

### Example

$G_1$ is deterministic.
$G_2$ is not deterministic.

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

### Theorem

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

# Continued Review of Linear Grammars

### Definition

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

# Continued Review of Linear Grammars

### Definition

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

### Example

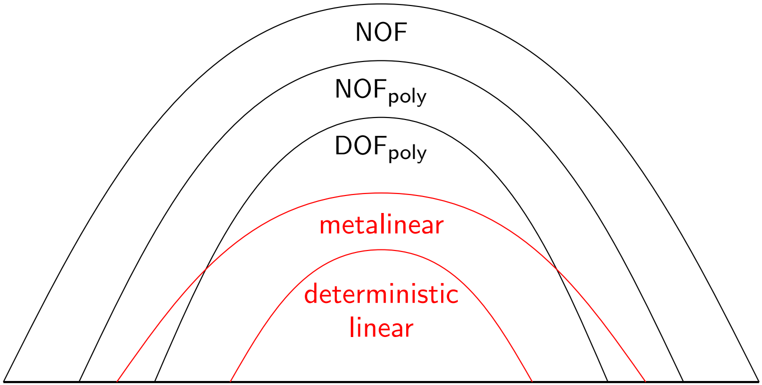
$\text{TRIPLE-PALINDROME} = \{uvw \mid u, v, \text{ and } w \text{ are palindromes}\}$.

# Metalinear Languages Can Be Accepted in an Overhead-Free Way

### Theorem

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

# Relationships among Overhead-Free Computation Classes



# Definition of Almost-Overhead-Free Computations

## Definition

A Turing machine is almost-overhead-free if

- it has only a single tape,

# Definition of Almost-Overhead-Free Computations

## Definition

A Turing machine is almost-overhead-free if

- it has only a single tape,
- writes only on input cells,

# Definition of Almost-Overhead-Free Computations

### Definition

A Turing machine is almost-overhead-free if

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

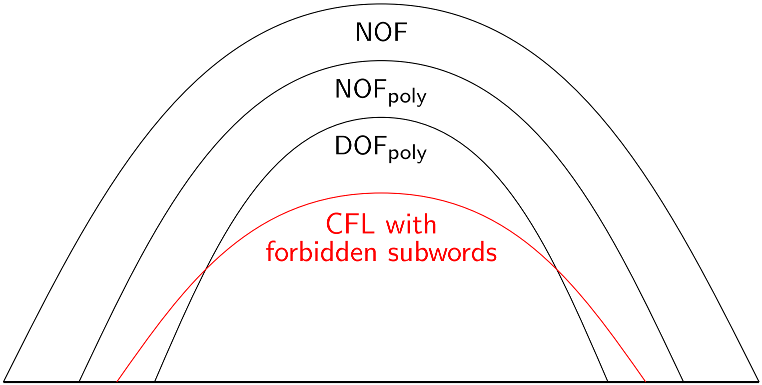
# Context-Free Languages with a Forbidden Subword Can Be Accepted in an Overhead-Free Way

### Theorem

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

The proof is based on the fact that every context-free language can be accepted by a nondeterministic almost-overhead-free machine in polynomial time.

# Relationships among Overhead-Free Computation Classes



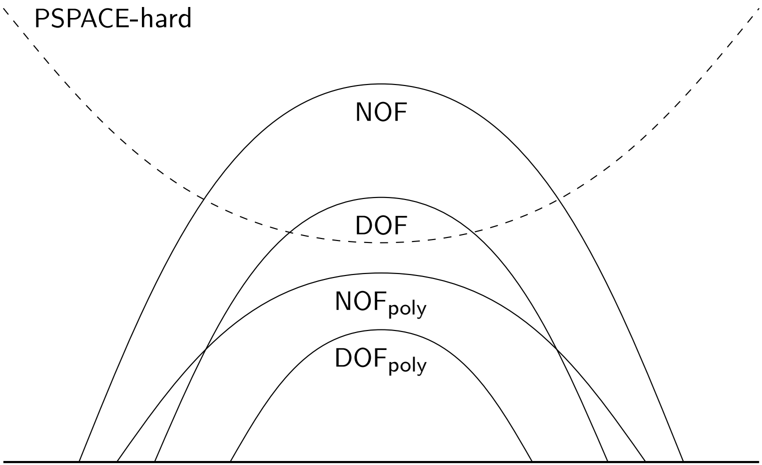
# Some PSPACE-complete Languages Can Be Accepted in an Overhead-Free Way

### Theorem

DOF contains languages that are complete for PSPACE.

The proof is based on the fact that for every $L \in$ DLINSPACE there exists an isometric homomorphism $h$ such that $h(L) \in$ DOF.

# Relationships among Overhead-Free Computation Classes



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

**Theorem**

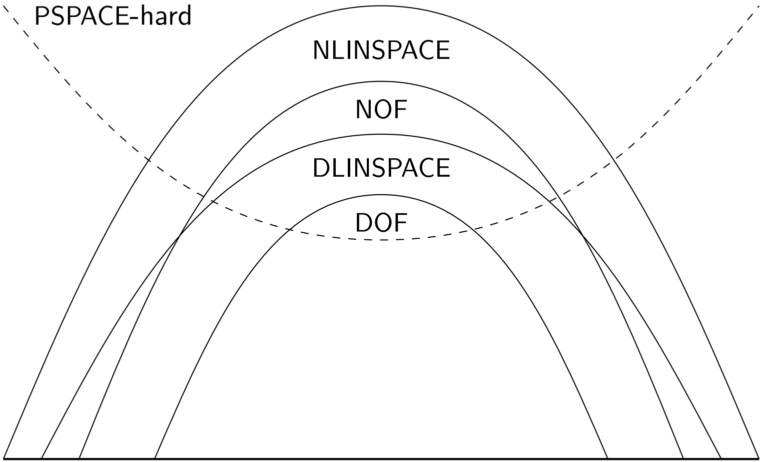
DOF $\subsetneq$ DLINSPACE.

**Theorem**

NOF $\subsetneq$ 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 DOF $\subsetneq$ 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 $\mathrm{NOF}_{\mathrm{poly}}$.
- 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.