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

Outline

Outline

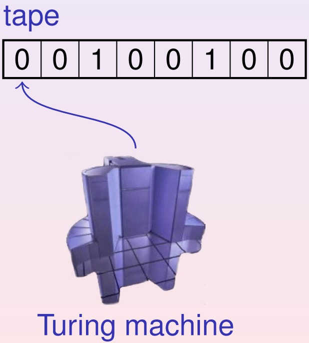
Outline

1. The Model of Overhead-Free Computation
   - The Standard Model of Linear Space
   - Our Model of Absolutely No Space Overhead
2. The Power of Overhead-Free Computation
   - Palindromes
   - Linear Languages
   - Context-Free Languages with a Forbidden Subword
   - Languages Complete for Polynomial Space
3. Limitations of Overhead-Free Computation
   - Linear Space is Strictly More Powerful

Outline

1. The Model of Overhead-Free Computation
   - The Standard Model of Linear Space
   - Our Model of Absolutely No Space Overhead
2. The Power of Overhead-Free Computation
   - Palindromes
   - Linear Languages
   - Context-Free Languages with a Forbidden Subword
   - Languages Complete for Polynomial Space
3. Limitations of Overhead-Free Computation
   - Linear Space is Strictly More Powerful

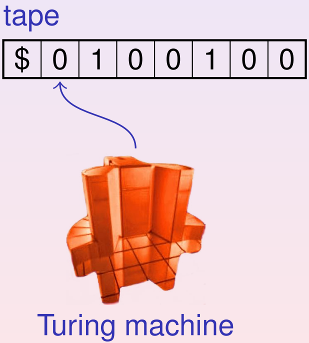
The Standard Model of Linear Space



Characteristics

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

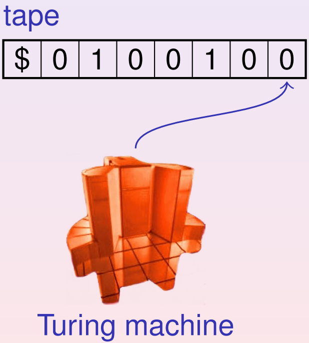
The Standard Model of Linear Space



Characteristics

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

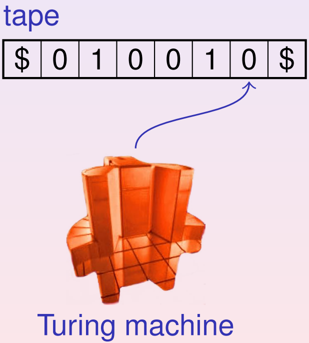
The Standard Model of Linear Space



Characteristics

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

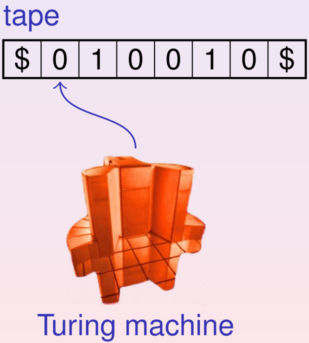
The Standard Model of Linear Space



Characteristics

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

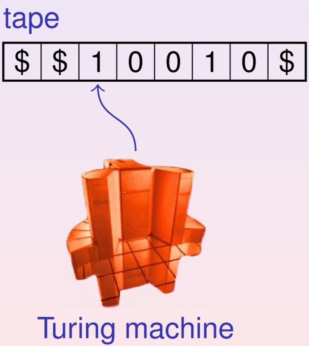
The Standard Model of Linear Space



Characteristics

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

The Standard Model of Linear Space

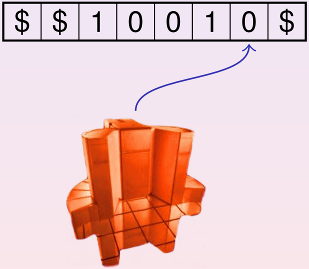


Characteristics

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

The Standard Model of Linear Space

tape

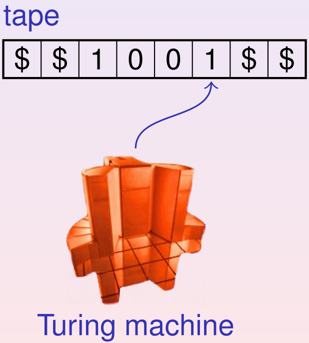


Turing machine

Characteristics

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

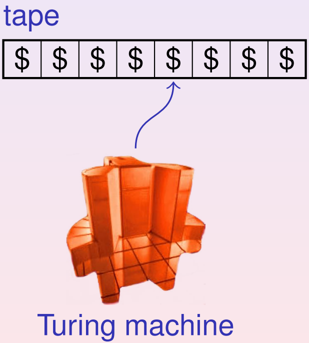
The Standard Model of Linear Space



Characteristics

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

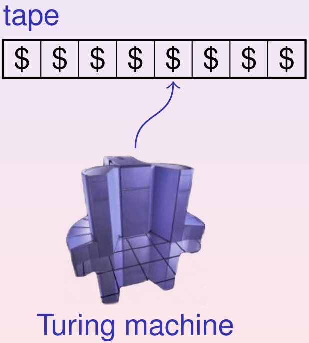
The Standard Model of Linear Space



Characteristics

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

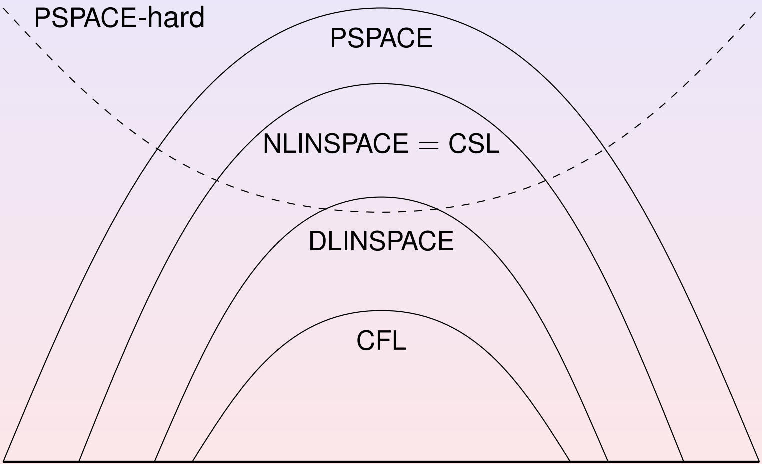
The Standard Model of Linear Space



Characteristics

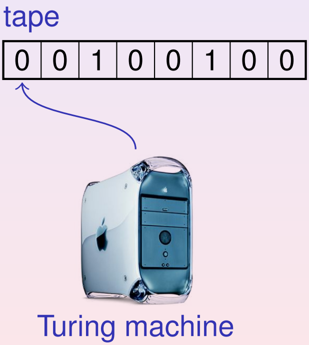
- Input fills fixed-size tape
- Input may be modified
- Tape alphabet is larger than input alphabet

Linear Space is a Powerful Model



Outline

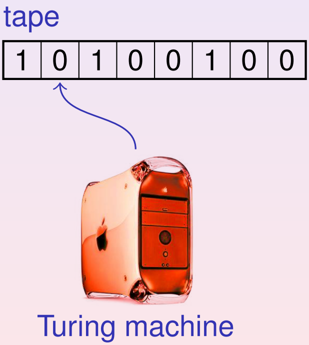
Our Model of “Absolutely No Space Overhead”



Characteristics

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

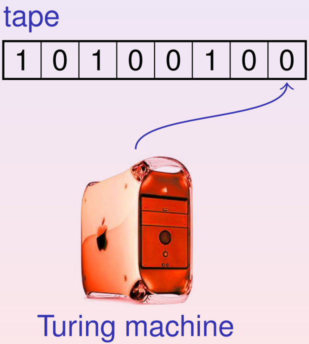
Our Model of “Absolutely No Space Overhead”



Characteristics

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

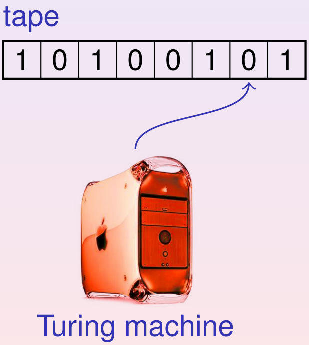
Our Model of “Absolutely No Space Overhead”



Characteristics

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

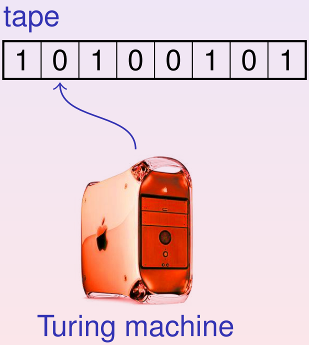
Our Model of “Absolutely No Space Overhead”



Characteristics

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

Our Model of “Absolutely No Space Overhead”

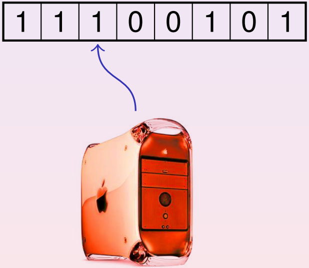


Characteristics

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

Our Model of “Absolutely No Space Overhead”

tape

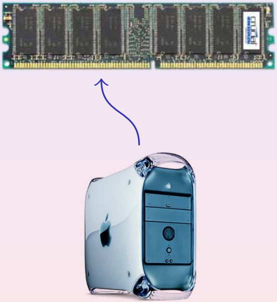


Turing machine

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

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

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 Σ,
- **DOF_{poly}** if L is accepted by a deterministic overhead-free machine with input alphabet Σ in polynomial time.
- **NOF** is the nondeterministic version of DOF,
- **NOF_{poly}** is the nondeterministic version of DOF_{poly}.

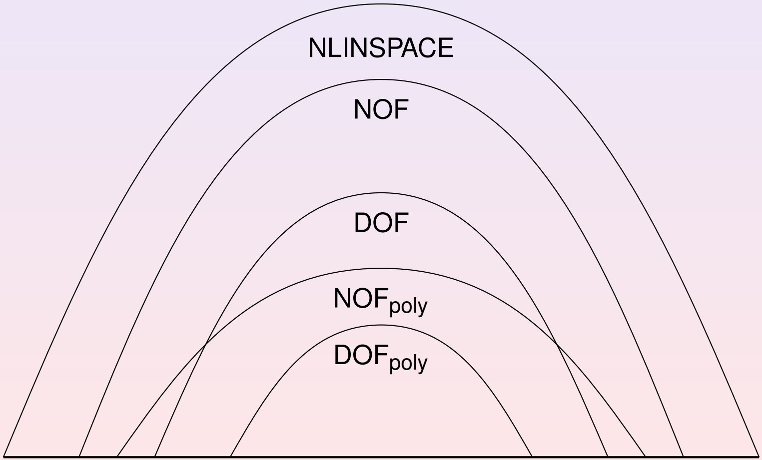
Overhead-Free Computation Complexity Classes

Definition

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

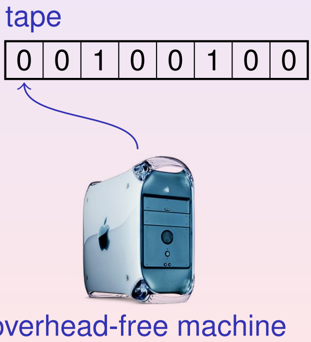
- DOF if L is accepted by a deterministic overhead-free machine with input alphabet Σ,
- DOF_{poly} if L is accepted by a deterministic overhead-free machine with input alphabet Σ in polynomial time.
- NOF is the nondeterministic version of DOF,
- NOF_{poly} is the nondeterministic version of DOF_{poly}.

Simple Relationships among Overhead-Free Computation Classes



Outline

Palindromes Can be Accepted in an Overhead-Free Way



Algorithm

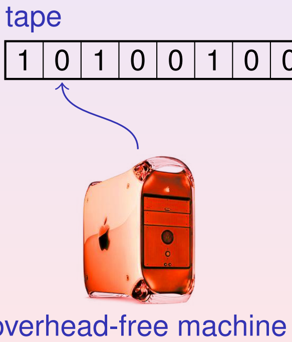
Phase 1:
Compare first and last bit

- 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

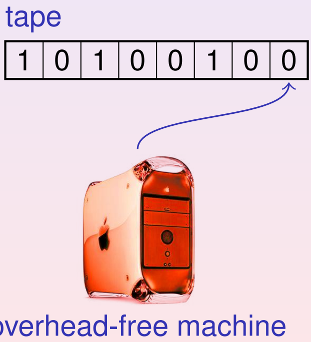
- Place left end marker
- Place right end marker

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



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

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

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

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

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

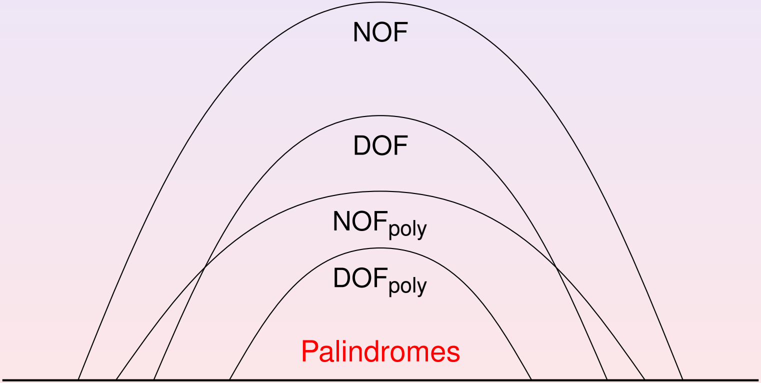
Palindromes Can be Accepted in an Overhead-Free Way

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



Outline

1. The Model of Overhead-Free Computation
   - The Standard Model of Linear Space
   - Our Model of Absolutely No Space Overhead
2. The Power of Overhead-Free Computation
   - Palindromes
   - Linear Languages
   - Context-Free Languages with a Forbidden Subword
   - Languages Complete for Polynomial Space
3. Limitations of Overhead-Free Computation
   - Linear Space is Strictly More Powerful

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

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

Definition

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

Example

$G_1: S \to 00S0 \mid 1$ is deterministic.
$G_2: S \to 0S10 \mid 0$ 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

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

Definition

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

Example

$G_1: S \to 00S0 \mid 1$ is deterministic.
$G_2: S \to 0S10 \mid 0$ is not deterministic.

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

Theorem

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

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 $\mathrm{NOF}_{\mathrm{poly}}$.

Metalinear Languages Can Be Accepted in an Overhead-Free Way

Definition

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

Example

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

Theorem

Every metalinear language is in NOF_{poly}.

Metalinear Languages Can Be Accepted in an Overhead-Free Way

Definition

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

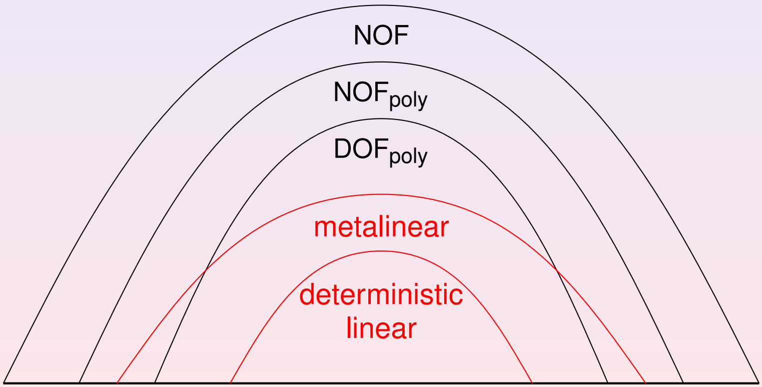
Example

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

Theorem

Every metalinear language is in NOF_{poly}.

Relationships among Overhead-Free Computation Classes



Outline

1. The Model of Overhead-Free Computation
   - The Standard Model of Linear Space
   - Our Model of Absolutely No Space Overhead
2. The Power of Overhead-Free Computation
   - Palindromes
   - Linear Languages
   - Context-Free Languages with a Forbidden Subword
   - Languages Complete for Polynomial Space
3. Limitations of Overhead-Free Computation
   - Linear Space is Strictly More Powerful

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

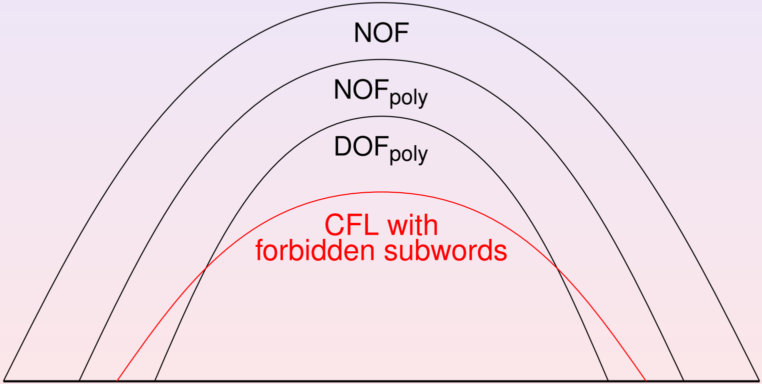
Theorem

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

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

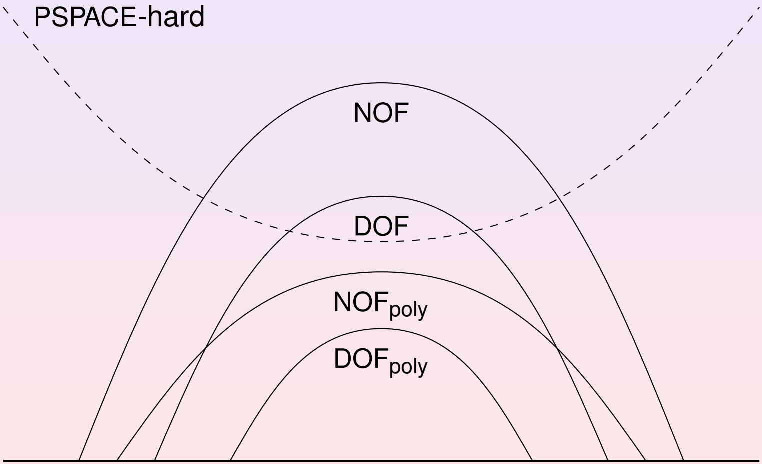
Overhead-Free Languages can be PSPACE-Complete

Theorem

DOF *contains languages that are complete for* PSPACE.

Proof details

Relationships among Overhead-Free Computation Classes



Outline

1. The Model of Overhead-Free Computation
   - The Standard Model of Linear Space
   - Our Model of Absolutely No Space Overhead
2. The Power of Overhead-Free Computation
   - Palindromes
   - Linear Languages
   - Context-Free Languages with a Forbidden Subword
   - Languages Complete for Polynomial Space
3. Limitations of Overhead-Free Computation
   - Linear Space is Strictly More Powerful

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

Theorem

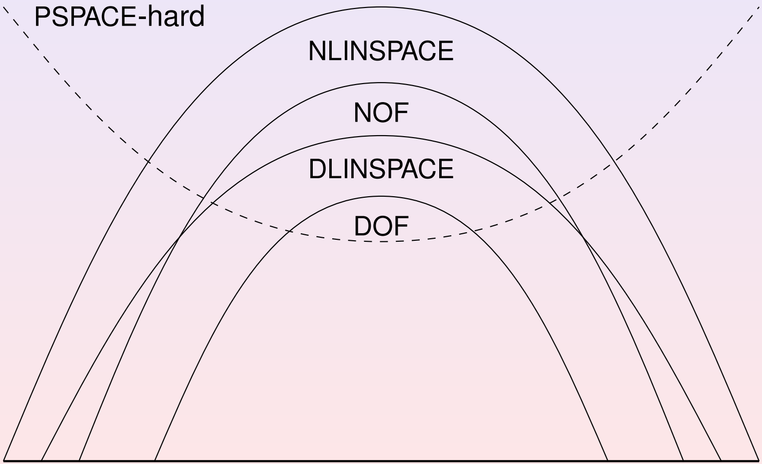
DOF $\subsetneq$ DLINSPACE.

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.

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

Theorem

DOUBLE-PALINDROMES $\in$ DOF.

Conjecture

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

4. Appendix
   - Complete Languages
   - Improvements for Context-Free Languages
   - Abbreviations

Overhead-Free Languages can be PSPACE-Complete

Theorem

DOF *contains languages that are complete for* PSPACE*.*

Proof.

1. Let $A \in$ 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 Γ.
3. Let $h\colon \Gamma \to \{0,1\}^*$ be an isometric, injective homomorphism.
4. Then $h(L)$ is in DOF and it is PSPACE-complete. □

Return

Improvements

Theorem

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

Explanation of Different Abbreviations

DOF	Deterministic Overhead-Free.
NOF	Nondeterministic Overhead-Free.
$\mathrm{DOF}_{\mathrm{poly}}$	Deterministic Overhead-Free, polynomial time.
$\mathrm{DOF}_{\mathrm{poly}}$	Nondeterministic Overhead-Free, polynomial time.

Table: Explanation of what different abbreviations mean.