形式语言与计算复杂性

第 4 章 Complexity Theory

4.2 Space Complexity

黄文超

https://faculty.ustc.edu.cn/huangwenchao

→ 教学课程 → 形式语言与计算复杂性

1. Introduction
2. Savitch’ s theorem
3. The class PSPACE
4. PSPACE-Completeness
5. The classes L and NL
6. NL-Completeness
7. Conclusions
8. Homework
9. Appendix

1 Introduction
2 Savitch' s theorem
3 The class PSPACE
4 PSPACE-Completeness
5 The classes L and NL
6 NL-Completeness
7 Conclusions
8 Homework
9 Appendix

问: Is a problem *solvable* in *practice*, when the problem is *decidable*?
答: No, consider the case that the solution requires an inordinate amount of *time* (Time Complexity) or *memory*

- Time Complexity
- *Space Complexity*

问: *Overview* of *Space* Complexity?
答:

- *Shares* many of the *features* of time complexity
- Serves as a *further way* of *classifying* problems

问: Is a problem *solvable* in *practice*, when the problem is *decidable*?
答: No, consider the case that the solution requires an inordinate amount of *time* (Time Complexity) or *memory*

- Time Complexity
- *Space Complexity*

问: *Overview* of *Space* Complexity?
答:

- *Shares* many of the *features* of time complexity
- Serves as a *further way* of *classifying* problems

定义: Time complexity

回顾

定义: Space complexity

Let M be a *deterministic* Turing machine that halts on all inputs.

- The *space complexity* of M is the function $f: \mathcal{N} \to \mathcal{N}$, where $f(n)$ is the ***maximum*** *number of tape cells* that M scans on ***any*** *input of length* n.
- If the space complexity of M is $f(n)$, we also say that M *runs in space* $f(n)$.

If M is a *nondeterministic* Turing machine wherein all branches halt on all inputs

- Space complexity $f(n)$: the *maximum number of tape cells* that M scans on *any branch* of its computation for any input of length n.

定义: Time complexity

回顾

定义: Space complexity

Let M be a *deterministic* Turing machine that halts on all inputs.

- The *space complexity* of M is the function $f : \mathcal{N} \to \mathcal{N}$, where $f(n)$ is the ***maximum*** *number of tape cells* that M scans on ***any*** *input of length* n.
- If the space complexity of M is $f(n)$, we also say that M *runs in space* $f(n)$.

If M is a *nondeterministic* Turing machine wherein all branches halt on all inputs

- Space complexity $f(n)$: the *maximum number of tape cells* that M scans on *any branch* of its computation for any input of length n.

定义: Time complexity classes

回顾

定义: Space complexity classes

Let $f : \mathcal{N} \to \mathcal{R}^+$ be a function. Define the *space complexity classes*, $\mathrm{SPACE}(f(n))$, $\mathrm{NSPACE}(f(n))$:

SPACE$(f(n)) = \{L \mid L$ is a language decided by an $O(f(n))$ space *deterministic* Turing machine$\}$.

NSPACE$(f(n)) = \{L \mid L$ is a language decided by an $O(f(n))$ space *nondeterministic* Turing machine$\}$.

问： *Time* complexity of SAT problem?
答： NP-complete

问: *Space* complexity of SAT problem?
答: At least *Linear* Space !!!

问： *Time* complexity of SAT problem?
答： NP-complete

问: *Space* complexity of SAT problem?
答: At least *Linear* Space !!!

问： *Time* complexity of SAT problem?
答： NP-complete

问: *Space* complexity of SAT problem?
答: At least *Linear* Space !!!

$M_1 =$ "On input $\langle\phi\rangle$, where ϕ is a Boolean formula:

1. For each truth assignment to the variables $x_1, \ldots, x_m$ of ϕ:
2. Evaluate ϕ on that truth assignment.
3. If ϕ ever evaluated to 1, *accept*; if not, *reject*."

Reason: each iteration of the loop can *reuse the same portion* of the tape.

问： *Time* complexity of SAT problem?
答： NP-complete

问: *Space* complexity of SAT problem?
答: At least *Linear* Space !!!

问: Space complexity of the following problem?

例: Space Complexity of ALL_{NFA}

$$ALL_{\text{NFA}} = \{\langle A \rangle \mid A \text{ is an NFA and } L(A) = \Sigma^*\}$$

问： *Time* complexity of SAT problem?
答： NP-complete

问: *Space* complexity of SAT problem?
答: At least *Linear* Space !!!

问: Space complexity of the following problem?

例: Space Complexity of ALL_{NFA}

$$ALL_{\text{NFA}} = \{\langle A\rangle \mid A \text{ is an NFA and } L(A) = \Sigma^*\}$$

回顾: 定理: Decidability of E_{DFA}

The language E_{DFA} is decidable, where

$$E_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

思路： decides the complement of this language, $\overline{ALL_{\text{NFA}}}$

例: Space Complexity of ALL_{NFA} is at least linear

$$ALL_{\mathrm{NFA}} = \{\langle A\rangle \mid A \text{ is an NFA and } L(A) = \Sigma^*\}$$

$N =$ "On input $\langle M\rangle$, where M is an NFA:

1. Place a marker on the start state of the NFA.
2. Repeat 2^q times, where q is the number of states of M:
3. Nondeterministically select an input symbol and change the positions of the markers on M's states to simulate reading that symbol.
4. *Accept* if stages 2 and 3 reveal some string that M rejects; that is, if at some point none of the markers lie on accept states of M. Otherwise, *reject*."

Hence N decides $\overline{ALL_{\mathrm{NFA}}}$, and runs in *nondeterministic* space $O(n)$

问：How about the *deterministic* space complexity of ALL_{NFA}?
答：See Savitch' s theorem.

例: Space Complexity of ALL_{NFA} is at least linear

$$ALL_{\text{NFA}} = \{\langle A\rangle \mid A \text{ is an NFA and } L(A) = \Sigma^*\}$$

N = "On input $\langle M\rangle$, where M is an NFA:

1. Place a marker on the start state of the NFA.
2. Repeat 2^q times, where q is the number of states of M:
3. Nondeterministically select an input symbol and change the positions of the markers on M's states to simulate reading that symbol.
4. *Accept* if stages 2 and 3 reveal some string that M rejects; that is, if at some point none of the markers lie on accept states of M. Otherwise, *reject*."

Hence N decides $\overline{ALL_{\text{NFA}}}$, and runs in *nondeterministic* space $O(n)$

问：How about the *deterministic* space complexity of ALL_{NFA}?
答：See Savitch' s theorem.

例: Space Complexity of ALL_{NFA} is at least linear

$$ALL_{\text{NFA}} = \{\langle A\rangle \mid A \text{ is an NFA and } L(A) = \Sigma^*\}$$

N = "On input $\langle M\rangle$, where M is an NFA:

1. Place a marker on the start state of the NFA.
2. Repeat 2^q times, where q is the number of states of M:
3. Nondeterministically select an input symbol and change the positions of the markers on M's states to simulate reading that symbol.
4. *Accept* if stages 2 and 3 reveal some string that M rejects; that is, if at some point none of the markers lie on accept states of M. Otherwise, *reject*."

Hence N decides $\overline{ALL_{\text{NFA}}}$, and runs in *nondeterministic* space $O(n)$

问: How about the *deterministic* space complexity of ALL_{NFA}?

答: See Savitch' s theorem.

例: Space Complexity of ALL_{NFA} is at least linear

$$ALL_{\text{NFA}} = \{\langle A\rangle \mid A \text{ is an NFA and } L(A) = \Sigma^*\}$$

$N =$ "On input $\langle M\rangle$, where M is an NFA:

1. Place a marker on the start state of the NFA.
2. Repeat 2^q times, where q is the number of states of M:
3. Nondeterministically select an input symbol and change the positions of the markers on M's states to simulate reading that symbol.
4. *Accept* if stages 2 and 3 reveal some string that M rejects; that is, if at some point none of the markers lie on accept states of M. Otherwise, *reject*."

Hence N decides $\overline{ALL_{\text{NFA}}}$, and runs in *nondeterministic* space $O(n)$

问: How about the *deterministic* space complexity of ALL_{NFA}?
答: See Savitch' s theorem.

1 Introduction

2 Savitch' s theorem

3 The class PSPACE

4 PSPACE-Completeness

5 The classes L and NL

6 NL-Completeness

7 Conclusions

8 Homework

9 Appendix

定理：Savitch' s theorem

For any function $f : \mathcal{N} \to \mathcal{R}^+$, where $f(n) \geq n$,

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$$

定理：Savitch’ s theorem

For any function $f : \mathcal{N} \rightarrow \mathcal{R}^+$, where $f(n) \geq n$,

NSPACE$(f(n))$ ⊆SPACE$(f^2(n))$

证明思路 1: A naive approach

Try *all* the *branches* of the NTM’ s computation, *one by one*:

- *Requirement*: *keep track* of *which branch* it is currently trying so that it is able to go on to the next one
- *Problem*: *A branch* that uses $f(n)$ space may run for $2^{O(f(n))}$ steps

定理：Savitch’ s theorem

For any function $f : \mathcal{N} \to \mathcal{R}^+$, where $f(n) \geq n$,

NSPACE$(f(n))$ $\subseteq$SPACE$(f^2(n))$

证明思路 1: A naive approach

Try *all* the *branches* of the NTM’ s computation, *one by one*:

- *Requirement*: *keep track* of *which branch* it is currently trying so that it is able to go on to the next one
- *Problem*: *A branch* that uses $f(n)$ space may run for $2^{O(f(n))}$ steps

定理：Savitch' s theorem

For any function $f : \mathcal{N} \to \mathcal{R}^+$, where $f(n) \geq n$,

NSPACE$(f(n))$ $\subseteq$SPACE$(f^2(n))$

证明思路 1: A naive approach

Try *all* the *branches* of the NTM' s computation, *one by one*:

- *Requirement*: *keep track* of *which branch* it is currently trying so that it is able to go on to the next one
- *Problem*: *A branch* that uses $f(n)$ space may run for $2^{O(f(n))}$ steps

定理：Savitch' s theorem

For any function $f : \mathcal{N} \to \mathcal{R}^+$, where $f(n) \geq n$,

NSPACE$(f(n))$ ⊆SPACE$(f^2(n))$

证明思路 1: A naive approach

Try *all* the *branches* of the NTM' s computation, *one by one*:

- *Requirement*: *keep track* of *which branch* it is currently trying so that it is able to go on to the next one
- *Problem*: *A branch* that uses $f(n)$ space may run for $2^{O(f(n))}$ steps
  - Each step may be a *nondeterministic* choice
  - Exploring the branches *sequentially* would *require recording all the choices* used on *a* particular *branch* in order to be able to find the next branch.
  - This approach may use $2^{O(f(n))}$ space

定理：Savitch' s theorem

For any function $f : \mathcal{N} \to \mathcal{R}^+$, where $f(n) \geq n$,

NSPACE$(f(n))$ $\subseteq$SPACE$(f^2(n))$

证明思路 1: A naive approach

Try *all* the *branches* of the NTM' s computation, *one by one*:

- *Requirement*: *keep track* of *which branch* it is currently trying so that it is able to go on to the next one
- *Problem*: *A branch* that uses $f(n)$ space may run for $2^{O(f(n))}$ steps
  - Each step may be a *nondeterministic* choice
  - Exploring the branches *sequentially* would *require recording all the choices* used on *a* particular *branch* in order to be able to find the next branch.
  - This approach may use $2^{O(f(n))}$ space

定理：Savitch' s theorem

For any function $f : \mathcal{N} \rightarrow \mathcal{R}^+$, where $f(n) \geq n$,

NSPACE$(f(n))$ $\subseteq$SPACE$(f^2(n))$

证明思路 1: A naive approach

Try *all* the *branches* of the NTM' s computation, *one by one*:

- *Requirement*: *keep track* of *which branch* it is currently trying so that it is able to go on to the next one
- *Problem*: *A branch* that uses $f(n)$ space may run for $2^{O(f(n))}$ steps
  - Each step may be a *nondeterministic* choice
  - Exploring the branches *sequentially* would *require recording all the choices* used on *a* particular *branch* in order to be able to find the next branch.
  - This approach may use $2^{O(f(n))}$ space

定理：Savitch' s theorem

For any function $f : \mathcal{N} \to \mathcal{R}^+$, where $f(n) \geq n$,

NSPACE($f(n)$) $\subseteq$SPACE($f^2(n)$)

证明思路 2: Solving yieldability problem

Given two configurations of the NTM, c_1 and c_2, together with a number t

- *Yieldability problem*: using *only* $f(n)$ space, *test* whether the NTM can get from c_1 to c_2 within t steps ?
- *Goal*: solving the yieldability problem,

定理：Savitch' s theorem

For any function $f : \mathcal{N} \rightarrow \mathcal{R}^+$, where $f(n) \geq n$,

NSPACE($f(n)$) $\subseteq$SPACE($f^2(n)$)

证明思路 2: Solving yieldability problem

Given two configurations of the NTM, c_1 and c_2, together with a number t

- *Yieldability problem*: using *only* $f(n)$ space, *test* whether the NTM can get from c_1 to c_2 within t steps ?
- *Goal*: solving the yieldability problem,

定理：Savitch' s theorem

For any function $f : \mathcal{N} \rightarrow \mathcal{R}^+$, where $f(n) \geq n$,
NSPACE($f(n)$) ⊆SPACE($f^2(n)$)

证明思路 2: Solving yieldability problem

Given two configurations of the NTM, c_1 and c_2, together with a number t

- *Yieldability problem*: using *only* $f(n)$ space, *test* whether the NTM can get from c_1 to c_2 within t steps ?
- *Goal*: solving the yieldability problem, where
  - c_1 is the *start* configuration
  - c_2 is the *accept* configuration
  - t is the *maximum number* of steps that the nondeterministic machine can use

定理：Savitch’s theorem

For any function $f : \mathcal{N} \rightarrow \mathcal{R}^+$, where $f(n) \geq n$,

NSPACE($f(n)$) $\subseteq$SPACE($f^2(n)$)

证明思路 2: Solving yieldability problem

Given two configurations of the NTM, c_1 and c_2, together with a number t

- *Yieldability problem*: using *only* $f(n)$ space, *test* whether the NTM can get from c_1 to c_2 within t steps ?
- *Goal*: solving the yieldability problem,
- *How* to solve?

定理：Savitch’ s theorem

For any function $f : \mathcal{N} \rightarrow \mathcal{R}^+$, where $f(n) \geq n$,

NSPACE$(f(n))$ $\subseteq$SPACE$(f^2(n))$

证明思路 2: Solving yieldability problem

Given two configurations of the NTM, c_1 and c_2, together with a number t

- *Yieldability problem*: using *only* $f(n)$ space, *test* whether the NTM can get from c_1 to c_2 within t steps ?
- *Goal*: solving the yieldability problem,
- *How* to solve? A *deterministic*, *recursive* algorithm

定理：Savitch’s theorem

For any function $f : \mathcal{N} \rightarrow \mathcal{R}^+$, where $f(n) \geq n$,

NSPACE($f(n)$) $\subseteq$SPACE($f^2(n)$)

证明思路 2: Solving yieldability problem

Given two configurations of the NTM, c_1 and c_2, together with a number t

- *Yieldability problem*: using *only* $f(n)$ space, *test* whether the NTM can get from c_1 to c_2 within t steps ?
- *Goal*: solving the yieldability problem,
- *How* to solve? A *deterministic*, *recursive* algorithm ：searching for an intermediate configuration c_m, and recursively *testing whether*
  1. c_1 can get to c_m within $t/2$ steps
  2. c_m can get to c_2 within $t/2$ steps

问题 (Yieldability Problem): Design deterministic algorithm testing:

Whether the NTM with $f(n)$ space can get from c_1 to c_2 within t steps?

解: (Proof by Construction)

Let N be an NTM deciding a language A in space $f(n)$

问题 (Yieldability Problem): Design deterministic algorithm testing:

Whether the NTM with $f(n)$ space can get from c_1 to c_2 within t steps?

解: (Proof by Construction)

Let N be an NTM deciding a language A in space $f(n)$

Design the procedure $\text{CANYIELD}(c_1, c_2, t)$

- Input: configurations c_1 and c_2 of N, and integer t
- Output:
  - *accept*, if N can go from configuration c_1 to configuration c_2 in t or fewer steps along some nondeterministic path.
  - *reject*, otherwise.

问题 (Yieldability Problem): Design deterministic algorithm testing:

Whether the NTM with $f(n)$ space can get from c_1 to c_2 within t steps?

解: (Proof by Construction)

Let N be an NTM deciding a language A in space $f(n)$
Implement $\mathrm{CANYIELD}(c_1, c_2, t)$

CANYIELD = "On input c_1, c_2, and t:

1. If $t = 1$, then test directly whether $c_1 = c_2$ or whether c_1 yields c_2 in one step according to the rules of N. *Accept* if either test succeeds; *reject* if both fail.
2. If $t > 1$, then for each configuration c_m of N using space $f(n)$:
3. Run $\mathrm{CANYIELD}(c_1, c_m, \frac{t}{2})$.
4. Run $\mathrm{CANYIELD}(c_m, c_2, \frac{t}{2})$.
5. If steps 3 and 4 both accept, then *accept*.
6. If haven't yet accepted, *reject*."

定理：Savitch' s theorem

For $f:\mathcal{N}\rightarrow\mathcal{R}^{+}$, where $f(n)\geq n$, NSPACE$(f(n))\subseteq$SPACE$(f^{2}(n))$

问题 (Yieldability Problem): Design deterministic algorithm testing:

Whether the NTM with $f(n)$ space can get from c_1 to c_2 within t steps?

定理: Savitch' s theorem

For $f : \mathcal{N} \to \mathcal{R}^+$, where $f(n) \geq n$, NSPACE$(f(n)) \subseteq$SPACE$(f^2(n))$

问题 (Yieldability Problem): Design deterministic algorithm testing:

Whether the NTM with $f(n)$ space can get from c_1 to c_2 within t steps?

问: Then how to prove *Savitch's theorem*?

答: (Proof by *Construction*): use the algorithm in *yieldability problem*:

$M =$ "On input w:

1. Output the result of CANYIELD$(c_{\text{start}}, c_{\text{accept}}, 2^{df(n)})$."

定理：Savitch' s theorem

For $f: \mathcal{N} \rightarrow \mathcal{R}^+$, where $f(n) \geq n$, NSPACE$(f(n)) \subseteq$SPACE$(f^2(n))$

问题 (Yieldability Problem): Design deterministic algorithm testing:

Whether the NTM with $f(n)$ space can get from c_1 to c_2 within t steps?

问: Then how to prove *Savitch's theorem*?

答: (Proof by *Construction*): use the algorithm in *yieldability problem*:

$M =$ "On input w:

1. Output the result of CANYIELD$(c_{\text{start}}, c_{\text{accept}}, 2^{df(n)})$."

问: Space complexity? (for storing the recursion *stack*)

答: The deterministic simulation uses $O(f^2(n))$ *space*

- *Each level* of *recursion* uses $O(f(n))$ space to store a configuration
- The depth of the recursion is $\log t$, where $t = 2^{d(f(n))}$

定理：Savitch' s theorem

For $f: \mathcal{N} \to \mathcal{R}^+$, where $f(n) \geq n$, NSPACE$(f(n))$ $\subseteq$SPACE$(f^2(n))$

问题 (Yieldability Problem): Design deterministic algorithm testing:

Whether the NTM with $f(n)$ space can get from c_1 to c_2 within t steps?

问: Then how to prove *Savitch's theorem*?

答: (Proof by *Construction*): use the algorithm in *yieldability problem*:

$M =$ "On input w:

1. Output the result of CANYIELD$(c_{\text{start}}, c_{\text{accept}}, 2^{df(n)})$."

问: Space complexity? (for storing the recursion *stack*)

答: The deterministic simulation uses $O(f^2(n))$ *space*

- *Each level* of *recursion* uses $O(f(n))$ space to store a configuration
- The depth of the recursion is $\log t$, where $t = 2^{d(f(n))}$
  - What is d? and why $t = 2^{d(f(n))}$?

定理：Savitch' s theorem

For $f: \mathcal{N} \to \mathcal{R}^+$, where $f(n) \geq n$, NSPACE$(f(n)) \subseteq$SPACE$(f^2(n))$

问题 (Yieldability Problem): Design deterministic algorithm testing:

Whether the NTM with $f(n)$ space can get from c_1 to c_2 within t steps?

问: Then how to prove *Savitch's theorem*?
答: (Proof by *Construction*): use the algorithm in *yieldability problem*:
$M =$ "On input w:

1. Output the result of CANYIELD$(c_{\text{start}}, c_{\text{accept}}, 2^{df(n)})$."

问: What is d? and why $t = 2^{d(f(n))}$?
答: select a constant d so that N has no more than $2^{df(n)}$ configurations using $f(n)$ tape

- So, $2^{df(n)}$ provides an *upper bound* on the *running time* of *any branch* of N

定理: Savitch' s theorem

For $f : \mathcal{N} \to \mathcal{R}^+$, where $f(n) \geq n$, NSPACE$(f(n)) \subseteq$SPACE$(f^2(n))$

问题 (Yieldability Problem): Design deterministic algorithm testing:

Whether the NTM with $f(n)$ space can get from c_1 to c_2 within t steps?

问: Then how to prove *Savitch's theorem*?

答: (Proof by *Construction*): use the algorithm in *yieldability problem*:

M = "On input w:

1. Output the result of CANYIELD$(c_{\text{start}}, c_{\text{accept}}, 2^{df(n)})$."

问: How to construct M if the value of $f(n)$ is unknown?

答: Modifying M so that it tries $f(n) = 1, 2, 3, \ldots$

1. Introduction
2. Savitch’s theorem
3. The class PSPACE
4. PSPACE-Completeness
5. The classes L and NL
6. NL-Completeness
7. Conclusions
8. Homework
9. Appendix

回顾: Class P

P is the class of languages that are decidable in *polynomial time* on *a deterministic single-tape* Turing machine. In other words,

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k)$$

定义: Class PSPACE

PSPACE is the class of languages that are decidable in *polynomial space* on a *deterministic* Turing machine. In other words,

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k)$$

问: How to define NSPACE?

答: $\mathrm{NPSPACE} = \bigcup_k \mathrm{NSPACE}(n^k)$

回顾: Class P

P is the class of languages that are decidable in *polynomial time* on *a deterministic single-tape* Turing machine. In other words,

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k)$$

定义: Class PSPACE

PSPACE is the class of languages that are decidable in *polynomial space* on a *deterministic* Turing machine. In other words,

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k)$$

问: How to define NSPACE?

答: $\mathrm{NPSPACE} = \bigcup_k \mathrm{NSPACE}(n^k)$

回顾: Class P

P is the class of languages that are decidable in *polynomial time* on *a deterministic single-tape* Turing machine. In other words,

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k)$$

定义: Class PSPACE

PSPACE is the class of languages that are decidable in *polynomial space* on a *deterministic* Turing machine. In other words,

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k)$$

问: How to define NSPACE?
答: $\mathrm{NPSPACE} = \bigcup_k \mathrm{NSPACE}(n^k)$

定义: Class PSPACE

PSPACE is the class of languages that are decidable in *polynomial space* on a *deterministic* Turing machine. In other words,

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k)$$

问: How to define NSPACE?
答: $\mathrm{NPSPACE} = \bigcup_k \mathrm{NSPACE}(n^k)$

问: PSPACE v.s. NPSPACE?
答: PSPACE=NPSPACE

- Because of Savitch's Theorem: $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$

定义: Class PSPACE

PSPACE is the class of languages that are decidable in *polynomial space* on a *deterministic* Turing machine. In other words,

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k)$$

问: How to define NSPACE?
答: $\mathrm{NPSPACE} = \bigcup_k \mathrm{NSPACE}(n^k)$

问: PSPACE v.s. $\mathrm{NPSPACE}$?
答: $\mathrm{PSPACE}{=}\mathrm{NPSPACE}$

- Because of Savitch's Theorem: NSPACE$(f(n)) \subseteq$SPACE$(f^2(n))$

问: P v.s. PSPACE
答: $\mathrm{P} \subseteq \mathrm{PSPACE}$

- For $t(n) \geq n$, any machine that operates in *time* $t(n)$ can use *at most* $t(n)$ *space* because a machine can explore *at most one new cell* at each step of its computation

问: NP v.s. PSPACE
答: $\mathrm{NP} \subseteq \mathrm{PSPACE}$

- Because, $\mathrm{NP} \subseteq \mathrm{NPSPACE}$, similarly.

问: PSPACE v.s. $\mathrm{EXPTIME}(= \bigcup_k \mathrm{TIME}(2^{n^k}))$
答: $\mathrm{PSPACE} \subseteq \mathrm{EXPTIME}= \bigcup_k \mathrm{TIME}(2^{n^k})$

- For $f(n) \geq n$, a TM that uses $f(n)$ space can have at most $f(n)2^{O(f(n))}$ different configurations
- Therefore, a TM that uses space $f(n)$ must run in time $f(n)2^{O(f(n))}$

问: P v.s. PSPACE
答: $\mathrm{P} \subseteq \mathrm{PSPACE}$

- For $t(n) \geq n$, any machine that operates in *time $t(n)$* can use *at most $t(n)$ space* because a machine can explore *at most one new cell* at each step of its computation

问: NP v.s. PSPACE
答: $\mathrm{NP} \subseteq \mathrm{PSPACE}$

- Because, $\mathrm{NP} \subseteq \mathrm{NPSPACE}$, similarly.

问: PSPACE v.s. $\mathrm{EXPTIME}(= \bigcup_k \mathrm{TIME}(2^{n^k}))$
答: $\mathrm{PSPACE} \subseteq \mathrm{EXPTIME} = \bigcup_k \mathrm{TIME}(2^{n^k})$

- For $f(n) \geq n$, a TM that uses $f(n)$ space can have at most $f(n)2^{O(f(n))}$ different configurations
- Therefore, a TM that uses space $f(n)$ must run in time $f(n)2^{O(f(n))}$

问: P v.s. PSPACE
答: $\mathrm{P} \subseteq \mathrm{PSPACE}$

- For $t(n) \geq n$, any machine that operates in *time $t(n)$* can use *at most $t(n)$ space* because a machine can explore *at most one new cell* at each step of its computation

问: NP v.s. PSPACE
答: $\mathrm{NP} \subseteq \mathrm{PSPACE}$

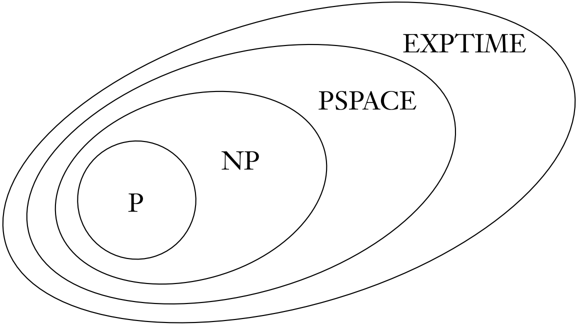
- Because, $\mathrm{NP} \subseteq \mathrm{NPSPACE}$, similarly.

问: PSPACE v.s. $\mathrm{EXPTIME}(= \bigcup_k \mathrm{TIME}(2^{n^k}))$
答: $\mathrm{PSPACE} \subseteq \mathrm{EXPTIME} = \bigcup_k \mathrm{TIME}(2^{n^k})$

- For $f(n) \geq n$, a TM that uses $f(n)$ space can have at most $f(n)2^{O(f(n))}$ different configurations
- Therefore, a TM that uses space $f(n)$ must run in time $f(n)2^{O(f(n))}$

In summary:

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXPTIME}$$



1 Introduction
2 Savitch’s theorem
3 The class PSPACE
4 PSPACE-Completeness
5 The classes L and NL
6 NL-Completeness
7 Conclusions
8 Homework
9 Appendix

定义: PSPACE-completeness

A language B is PSPACE-complete if it satisfies two conditions:

1. B is in PSPACE, and
2. every A in PSPACE is *polynomial **time** reducible* to B.

If B merely satisfies condition 2, we say that it is *PSPACE-hard*

回顾定义: NP-Completeness

A language B is NP-complete if it satisfies two conditions:

1. B is in NP, and
2. every A in NP is *polynomial time reducible* to B.

If B merely satisfies condition 2, we say that it is *NP-hard*

定义: PSPACE-completeness

A language B is PSPACE-complete if it satisfies two conditions:

1. B is in PSPACE, and
2. every A in PSPACE is *polynomial* ***time*** *reducible* to B.

If B merely satisfies condition 2, we say that it is *PSPACE-hard*

回顾定义: NP-Completeness

A language B is NP-complete if it satisfies two conditions:

1. B is in NP, and
2. every A in NP is *polynomial time reducible* to B.

If B merely satisfies condition 2, we say that it is *NP-hard*

问: Which language is PSPACE-complete?
答: The TQBF problem.

问: What is *TQBF* problem?
答: Determine whether a *fully quantified Boolean formula* is true or false:

- *Boolean formula*: an expression that contains Boolean variables, the constants 0 and 1, and the Boolean operations $\wedge$, $\vee$, and $\neg$.
- *Quantifiers*: $\forall$ (for all) and $\exists$ (there exists)
- *Quantified Boolean formulas*: Boolean formulas with quantifiers, e.g.,

$$\phi = \forall x \exists y [(x \vee y) \wedge (\overline{x} \vee \overline{y})]$$

  Here, ϕ is *true*; but *false* if the quantifiers $\exists x$ and $\exists y$ were reversed
- *Fully quantified* formula: *each* variable of the formula appears within the scope of some quantifier

问: Which language is PSPACE-complete?
答: The TQBF problem.

问: What is *TQBF* problem?
答: Determine whether a *fully quantified Boolean formula* is true or false:

- *Boolean formula*: an expression that contains Boolean variables, the constants 0 and 1, and the Boolean operations $\wedge$, $\vee$, and $\neg$.
- *Quantifiers*: $\forall$ (for all) and $\exists$ (there exists)
- *Quantified Boolean formulas*: Boolean formulas with quantifiers, e.g.,

$$\phi = \forall x \exists y [(x \vee y) \wedge (\overline{x} \vee \overline{y})]$$

  Here, ϕ is *true*; but *false* if the quantifiers $\exists x$ and $\exists y$ were reversed
- *Fully quantified* formula: *each* variable of the formula appears within the scope of some quantifier

定理: TQBF is PSPACE-complete

TQBF is PSPACE-complete, where

$$\text{TQBF} = \{\langle\phi\rangle \mid \phi \text{ is a true fully quantified Boolean formula}\}$$

证明思路: Imitate the proof of the Cook–Levin theorem

定理: TQBF is PSPACE-complete

TQBF is PSPACE-complete, where

$$\mathrm{TQBF} = \{\langle\phi\rangle \mid \phi \text{ is a true fully quantified Boolean formula}\}$$

证明思路: Imitate the proof of the Cook–Levin theorem

思路 1: Construct a formula ϕ that simulates M on an input w by expressing the requirements for an accepting tableau

- A tableau for M on w has *width $O(n^k)$*, the space used by M, but its *height* is ***exponential*** *in n^k* because M can *run for exponential time*.

思路 2: Use a technique related to the *proof* of Savitch' s theorem to construct the formula

- *Divide* the tableau into ***halves***, and employs the *universal* quantifier to represent *each half* with the same part of the formula

定理: TQBF is PSPACE-complete

TQBF is PSPACE-complete, where

$$\text{TQBF} = \{\langle\phi\rangle \mid \phi \text{ is a true fully quantified Boolean formula}\}$$

证明: (1) Prove TQBF is in PSPACE

T = "On input $\langle\phi\rangle$, a fully quantified Boolean formula:

1. If ϕ contains no quantifiers, then it is an expression with only constants, so evaluate ϕ and *accept* if it is true; otherwise, *reject*.
2. If ϕ equals $\exists x\ \psi$, recursively call T on ψ, first with 0 substituted for x and then with 1 substituted for x. If either result is accept, then *accept*; otherwise, *reject*.
3. If ϕ equals $\forall x\ \psi$, recursively call T on ψ, first with 0 substituted for x and then with 1 substituted for x. If both results are accept, then *accept*; otherwise, *reject*."

Complexity: the depth of the recursion is at most the number of variables

定理: TQBF is PSPACE-complete

TQBF is PSPACE-complete, where

$$\text{TQBF} = \{\langle\phi\rangle \mid \phi \text{ is a true fully quantified Boolean formula}\}$$

证明: (2) Prove TQBF is PSPACE-hard

Preparation:

- Give a *polynomial time reduction* from A to TQBF.
  - Let language A decided by a TM M in space n^k for some constant k.

定理: TQBF is PSPACE-complete

TQBF is PSPACE-complete, where

$$\text{TQBF} = \{\langle\phi\rangle \mid \phi \text{ is a true fully quantified Boolean formula}\}$$

证明: (2) Prove TQBF is PSPACE-hard

Preparation:

- Give a *polynomial time reduction* from A to TQBF.
- Construct a formula $\phi_{c_1,c_2,t}$
  - collections of variables c_1 and c_2 representing two configurations
  - number $t > 0$
  - If we assign c_1 and c_2 to actual configurations, the formula is true *iff* M can go from c_1 to c_2 in at most t steps.

定理: TQBF is PSPACE-complete

TQBF is PSPACE-complete, where

$$\mathrm{TQBF} = \{\langle\phi\rangle \mid \phi \text{ is a true fully quantified Boolean formula}\}$$

证明: (2) Prove TQBF is PSPACE-hard

Preparation:

- Give a *polynomial time reduction* from A to TQBF.
- Construct a formula $\phi_{c_1,c_2,t}$
- Let ϕ be the formula $\phi_{c_{\text{start}},c_{\text{accept}},h}$
  - $h = 2^{df(n)}$ for a constant d, chosen so that M has no more than $2^{df(n)}$ possible configurations on an input of length n
  - Let $f(n) = n^k$

定理: TQBF is PSPACE-complete

TQBF is PSPACE-complete, where

$$\text{TQBF} = \{\langle\phi\rangle \mid \phi \text{ is a true fully quantified Boolean formula}\}$$

证明: (2) Prove TQBF is PSPACE-hard

Preparation:

- Give a *polynomial time reduction* from A to TQBF.
- Construct a formula $\phi_{c_1,c_2,t}$
- Let ϕ be the formula $\phi_{c_{\text{start}},c_{\text{accept}},h}$
- The formula *encodes* the contents of configuration cells as in the proof of the Cook–Levin theorem

定理: TQBF is PSPACE-complete

TQBF is PSPACE-complete, where

$$\text{TQBF} = \{\langle\phi\rangle \mid \phi \text{ is a true fully quantified Boolean formula}\}$$

证明: (2) Prove TQBF is PSPACE-hard

If $t = 1$, construct $\phi_{c_1,c_2,t}$ using the technique presented in the proof of the Cook–Levin theorem

- c_1 yields c_2 in a single step of M by writing Boolean expressions stating that
  - the contents of each triple of c_1' s cells *correctly yields* the contents of the corresponding triple of c_2' s cells

定理: TQBF is PSPACE-complete

TQBF is PSPACE-complete, where

$$\text{TQBF} = \{\langle\phi\rangle \mid \phi \text{ is a true fully quantified Boolean formula}\}$$

证明: (2) Prove TQBF is PSPACE-hard

If $t > 1$, we construct $\phi_{c_1,c_2,t}$ recursively.

定理: TQBF is PSPACE-complete

TQBF is PSPACE-complete, where

$$\mathrm{TQBF} = \{\langle\phi\rangle \mid \phi \text{ is a true fully quantified Boolean formula}\}$$

证明: (2) Prove TQBF is PSPACE-hard

If $t > 1$, we construct $\phi_{c_1,c_2,t}$ recursively.

First Try (*naive* solution): Construct the following formula *recursively*

$$\phi_{c_1,c_2,t} = \exists m_1[\phi_{c_1,m_1,\frac{t}{2}} \wedge \phi_{m_1,c_2,\frac{t}{2}}]$$

- m_1 represents a configuration of M
- $\exists m_1$ is shorthand for $\exists x_1,\ldots,x_l$, where $l = O(n^k)$ and $x_1,\ldots,x_l$ are the variables that encode m_1.

定理: TQBF is PSPACE-complete

TQBF is PSPACE-complete, where

$$\text{TQBF} = \{\langle\phi\rangle \mid \phi \text{ is a true fully quantified Boolean formula}\}$$

证明: (2) Prove TQBF is PSPACE-hard

If $t > 1$, we construct $\phi_{c_1,c_2,t}$ recursively.
First Try (*naive* solution): Construct the following formula *recursively*

$$\phi_{c_1,c_2,t} = \exists m_1[\phi_{c_1,m_1,\frac{t}{2}} \wedge \phi_{m_1,c_2,\frac{t}{2}}]$$

- m_1 represents a configuration of M
- $\exists m_1$ is shorthand for $\exists x_1, \ldots, x_l$, where $l = O(n^k)$ and $x_1, \ldots, x_l$ are the variables that encode m_1.

Analysis: *Too big* formula: every level of the *recursion* involved in the construction cuts t in *half* but roughly *doubles* the size of the formula

定理: TQBF is PSPACE-complete

TQBF is PSPACE-complete, where

$$\text{TQBF} = \{\langle\phi\rangle \mid \phi \text{ is a true fully quantified Boolean formula}\}$$

证明: (2) Prove TQBF is PSPACE-hard

If $t > 1$, we construct $\phi_{c_1,c_2,t}$ recursively.
First Try (*naive* solution): Construct the following formula *recursively*

$$\phi_{c_1,c_2,t} = \exists m_1[\phi_{c_1,m_1,\frac{t}{2}} \wedge \phi_{m_1,c_2,\frac{t}{2}}]$$

定理: TQBF is PSPACE-complete

TQBF is PSPACE-complete, where

$$\text{TQBF} = \{\langle\phi\rangle \mid \phi \text{ is a true fully quantified Boolean formula}\}$$

证明: (2) Prove TQBF is PSPACE-hard

If $t > 1$, we construct $\phi_{c_1,c_2,t}$ recursively.

First Try (*naive* solution): Construct the following formula *recursively*

$$\phi_{c_1,c_2,t} = \exists m_1[\phi_{c_1,m_1,\frac{t}{2}} \wedge \phi_{m_1,c_2,\frac{t}{2}}]$$

Second Try (*good* solution): Construct the following formula *recursively*

$$\phi_{c_1,c_2,t} = \exists m_1 \forall (c_3, c_4) \in \{(c_1, m_1), (m_1, c_2)\}[\phi_{c_3,c_4,\frac{t}{2}}]$$

定理: TQBF is PSPACE-complete

TQBF is PSPACE-complete, where

$$\text{TQBF} = \{\langle\phi\rangle \mid \phi \text{ is a true fully quantified Boolean formula}\}$$

证明: (2) Prove TQBF is PSPACE-hard

If $t > 1$, we construct $\phi_{c_1,c_2,t}$ recursively.

Second Try (*good* solution): Construct the following formula *recursively*

$$\phi_{c_1,c_2,t} = \exists m_1 \forall (c_3, c_4) \in \{(c_1, m_1), (m_1, c_2)\}[\phi_{c_3,c_4,\frac{t}{2}}]$$

Analysis: the size of the formula $\phi_{c_{\text{start}},c_{\text{accept}},h}$

- $h = 2^{df(n)}$, so a portion of the formula is linear in the size of the configurations and is thus of size $O(f(n))$
- The number of levels of the recursion is $\log(2^{df(n)})$, or $O(f(n))$.

Hence the size of the resulting formula is $O(f^2(n))$

问: Which language is also PSPACE-complete, besides TQBF?
答: FORMULA-GAME for the formula game.

问: What is the formula game?
答: Rules:

- Let $\phi = \exists x_1 \forall x_2 \exists x_3 \ldots Qx_k[\psi]$ be a quantified Boolean formula
  - Q represents either a $\forall$ or an $\exists$ quantifier
- Two players, A and E, *take turns* selecting the values of $x_1, \ldots, x_k$
  - Player *A* selects values for the variables that are bound to $\forall$ *quantifiers*
  - Player *E* selects values for the variables that are bound to $\exists$ *quantifiers*
- The *order* of play is the *same* as that of the *quantifiers* at the beginning of the formula
- At the end of play
  - Player *E* has *won* the game, if ψ is *TRUE*
  - Player *A* has *won* the game, if ψ is *FALSE*

问: Which language is also PSPACE-complete, besides TQBF?
答: FORMULA-GAME for the formula game.

问: What is the formula game?
答: Rules:

- Let $\phi = \exists x_1 \forall x_2 \exists x_3 \ldots Q x_k [\psi]$ be a quantified Boolean formula
  - Q represents either a $\forall$ or an $\exists$ quantifier
- Two players, A and E, *take turns* selecting the values of $x_1, \ldots, x_k$
  - Player *A* selects values for the variables that are bound to $\forall$ *quantifiers*
  - Player *E* selects values for the variables that are bound to $\exists$ *quantifiers*
- The *order* of play is the *same* as that of the *quantifiers* at the beginning of the formula
- At the end of play
  - Player *E* has *won* the game, if ψ is *TRUE*
  - Player *A* has *won* the game, if ψ is *FALSE*

例 1: The formula game

Given the formula $\phi_1 = \exists x_1 \forall x_2 \exists x_3[(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})]$

- Round 1: E picks $x_1 = 1$, A picks $x_2 = 0$, E picks $x_3 = 1$, So E win
- Any Round: E may *always win* this game by selecting $x_1 = 1$ and then selecting x_3 to be the negation of whatever Player A selects for x_2
- So *E* has a *winning strategy* for this game
  - A player has a *winning strategy* for a game if *that player wins* when *both* sides play *optimally*

例 2: The formula game

Given the formula $\phi_1 = \exists x_1 \forall x_2 \exists x_3[(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3})]$

- *A* now has a winning strategy
  - no matter what Player E selects for x_1, Player A may select $x_2 = 0$

例 1: The formula game

Given the formula $\phi_1 = \exists x_1 \forall x_2 \exists x_3 [(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})]$

- Round 1: E picks $x_1 = 1$, A picks $x_2 = 0$, E picks $x_3 = 1$, So E win
- Any Round: E may *always win* this game by selecting $x_1 = 1$ and then selecting x_3 to be the negation of whatever Player A selects for x_2
- So *E* has a *winning strategy* for this game
  - A player has a *winning strategy* for a game if *that player wins* when *both* sides play *optimally*

例 2: The formula game

Given the formula $\phi_1 = \exists x_1 \forall x_2 \exists x_3 [(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3})]$

- *A* now has a winning strategy
  - no matter what Player E selects for x_1, Player A may select $x_2 = 0$

例 1: The formula game

Given the formula $\phi_1 = \exists x_1 \forall x_2 \exists x_3 [(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})]$

- Round 1: E picks $x_1 = 1$, A picks $x_2 = 0$, E picks $x_3 = 1$, So E win
- Any Round: E may *always win* this game by selecting $x_1 = 1$ and then selecting x_3 to be the negation of whatever Player A selects for x_2
- So *E* has a *winning strategy* for this game
  - A player has a *winning strategy* for a game if *that player wins* when *both* sides play *optimally*

例 2: The formula game

Given the formula $\phi_1 = \exists x_1 \forall x_2 \exists x_3 [(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3})]$

- *A* now has a winning strategy
  - no matter what Player E selects for x_1, Player A may select $x_2 = 0$

例 1: The formula game

Given the formula $\phi_1 = \exists x_1 \forall x_2 \exists x_3 [(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})]$

- Round 1: E picks $x_1 = 1$, A picks $x_2 = 0$, E picks $x_3 = 1$, So E win
- Any Round: E may *always win* this game by selecting $x_1 = 1$ and then selecting x_3 to be the negation of whatever Player A selects for x_2
- So *E* has a *winning strategy* for this game
    - A player has a *winning strategy* for a game if *that player wins* when *both* sides play *optimally*

例 2: The formula game

Given the formula $\phi_1 = \exists x_1 \forall x_2 \exists x_3 [(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3})]$

- *A* now has a winning strategy
    - no matter what Player E selects for x_1, Player A may select $x_2 = 0$

定理: FORMULA-GAME is PSPACE-complete

FORMULA-GAME is PSPACE-complete, where FORMULA-GAME=

$\{\langle\phi\rangle \mid$ E has a *winning strategy* in the formula game associated with $\phi\}$

定理: FORMULA-GAME is PSPACE-complete

FORMULA-GAME is PSPACE-complete, where FORMULA-GAME=

$\{\langle\phi\rangle \mid$ E has a *winning strategy* in the formula game associated with $\phi\}$

证明思路

Prove FORMULA-GAME = TQBF

- i.e., $\phi \in$ TQBF exactly when $\phi \in$ FORMULA-GAME
  - case 1: $\phi = \exists x_1 \forall x_2 \exists x_3 \cdots [\psi]$
  - case 2: $\phi = \forall x_1, x_2, x_3 \exists x_4, x_5 \forall x_6 [\psi]$

问: So easy, then is there other PSPACE-complete problem?

答: Yes, e.g., GG for *generalized geography*

定理: FORMULA-GAME is PSPACE-complete

FORMULA-GAME is PSPACE-complete, where FORMULA-GAME=

$\{\langle\phi\rangle \mid$ E has a *winning strategy* in the formula game associated with $\phi\}$

证明思路

Prove FORMULA-GAME = TQBF

- i.e., $\phi \in$ TQBF exactly when $\phi \in$ FORMULA-GAME
  - case 1: $\phi = \exists x_1 \forall x_2 \exists x_3 \cdots [\psi]$
  - case 2: $\phi = \forall x_1, x_2, x_3 \exists x_4, x_5 \forall x_6 [\psi]$

问: So easy, then is there other PSPACE-complete problem?

答: Yes, e.g., GG for *generalized geography*

定理: FORMULA-GAME is PSPACE-complete

FORMULA-GAME is PSPACE-complete, where FORMULA-GAME=

$\{\langle\phi\rangle \mid$ E has a *winning strategy* in the formula game associated with $\phi\}$

证明思路

Prove FORMULA-GAME = TQBF

- i.e., $\phi \in$ TQBF exactly when $\phi \in$ FORMULA-GAME
  - case 1: $\phi = \exists x_1 \forall x_2 \exists x_3 \cdots [\psi]$
  - case 2: $\phi = \forall x_1, x_2, x_3 \exists x_4, x_5 \forall x_6 [\psi]$

问: So easy, then is there other PSPACE-complete problem?

答: Yes, e.g., GG for *generalized geography*

定理: FORMULA-GAME is PSPACE-complete

FORMULA-GAME is PSPACE-complete, where FORMULA-GAME=

$\{\langle\phi\rangle \mid$ E has a *winning strategy* in the formula game associated with $\phi\}$

证明思路

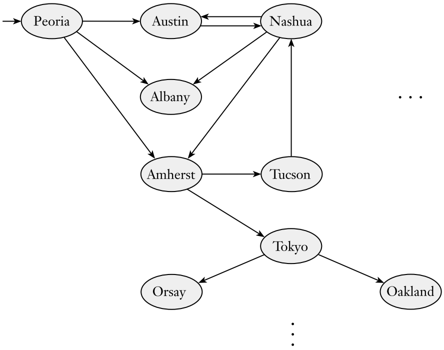
Prove FORMULA-GAME = TQBF

- i.e., $\phi \in$ TQBF exactly when $\phi \in$ FORMULA-GAME
  - case 1: $\phi = \exists x_1 \forall x_2 \exists x_3 \cdots [\psi]$
  - case 2: $\phi = \forall x_1, x_2, x_3 \exists x_4, x_5 \forall x_6 [\psi]$

问: So easy, then is there other PSPACE-complete problem?
答: Yes, e.g., GG for *generalized geography*

问: What is *generalized geography*?
答: *Firstly* introduce the game *geography* with the rules:

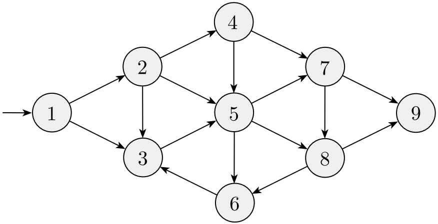
- Players take turns *naming cities* from anywhere in the world
- Each city chosen must *begin* with the *same letter* that *ended* the *previous* city’ s name
- Repetition isn’ t permitted
- The game starts with some designated starting city and ends when some player *loses* because he or she is *unable to continue*.



问: What is *generalized geography*?

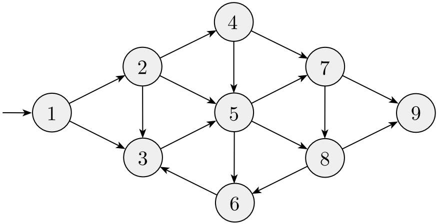
答: *Then* introduce the game *generalized geography*:

- Take an *arbitrary directed graph* with a designated start node
- Player *I* is the one who moves *first* and Player *II second*.



问: What is *generalized geography*?
答: *Then* introduce the game *generalized geography*:

- Take an *arbitrary directed graph* with a designated start node
- Player *I* is the one who moves *first* and Player *II second*.

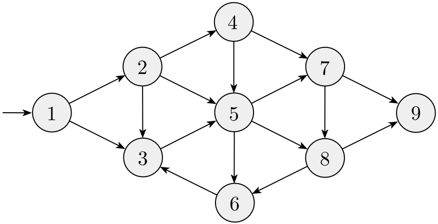


In this example, Player I has a ***winning strategy***:

- Player I starts at node 1, the designated start node
- Player I chooses 3
- Player II chooses 5
- Player I selects 6

问: What is *generalized geography*?
答: *Then* introduce the game *generalized geography*:

- Take an *arbitrary directed graph* with a designated start node
- Player *I* is the one who moves *first* and Player *II second*.

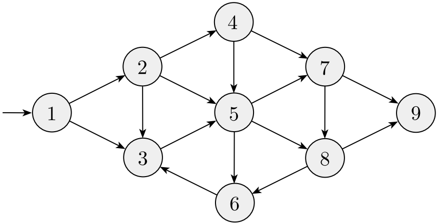


If *reversing* the *direction* of $6 \to 3$, Player II has a ***winning strategy***:

- case 1: Player I 3, Player II 6
- case 2: Player I 2, Player II 4
  - case 2.1: Player I 5, Player II 6
  - case 2.2: Player I 7, Player II 9

问: What is *generalized geography*?
答: *Then* introduce the game *generalized geography*:

- Take an *arbitrary directed graph* with a designated start node
- Player *I* is the one who moves *first* and Player *II second*.

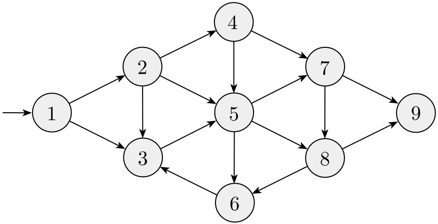


If *reversing* the *direction* of $6 \to 3$, Player II has a ***winning strategy***:

- case 1: Player I 3, Player II 6
- case 2: Player I 2, Player II 4
  - case 2.1: Player I 5, Player II 6
  - case 2.2: Player I 7, Player II 9

问: What is *generalized geography*?
答: *Then* introduce the game *generalized geography*:

- Take an *arbitrary directed graph* with a designated start node
- Player *I* is the one who moves *first* and Player *II second*.



If *reversing* the *direction* of $6 \to 3$, Player II has a ***winning strategy***:

- case 1: Player I 3, Player II 6
- case 2: Player I 2, Player II 4
  - case 2.1: Player I 5, Player II 6
  - case 2.2: Player I 7, Player II 9

定理: GG is PSPACE-complete

GG is PSPACE-complete, where

GG$= \{\langle G, b\rangle \mid$ Player I has a winning strategy for the generalized geography game played on graph G starting at node b $\}$

证明思路

- PSPACE: A recursive algorithm similar to the one used for TQBF
- PSPACE-hard: a *polynomial time reduction* from FORMULA-GAME to *GG*

定理: GG is PSPACE-complete

GG is PSPACE-complete, where

GG= $\{\langle G, b\rangle$ | Player I has a winning strategy for the generalized geography game played on graph G starting at node b $\}$

证明思路

- PSPACE: A recursive algorithm similar to the one used for TQBF
- PSPACE-hard: a *polynomial time reduction* from FORMULA-GAME to *GG*

定理: GG is PSPACE-complete

GG is PSPACE-complete, where

GG$= \{\langle G, b\rangle \mid$ Player I has a winning strategy for the generalized geography game played on graph G starting at node b $\}$

证明: (1) Prove GG is in PSPACE

$M =$ "On input $\langle G, b\rangle$, where G is a directed graph and b is a node of G:

1. If b has outdegree 0, *reject* because Player I loses immediately.
2. Remove node b and all connected arrows to get a new graph G'.
3. For each of the nodes $b_1, b_2, \ldots, b_k$ that b originally pointed at, recursively call M on $\langle G', b_i\rangle$.
4. If all of these accept, Player II has a winning strategy in the original game, so *reject*. Otherwise, Player II doesn't have a winning strategy, so Player I must; therefore, *accept*."

定理: GG is PSPACE-complete

GG is PSPACE-complete, where

GG= $\{\langle G, b\rangle$ | Player I has a winning strategy for the generalized geography game played on graph G starting at node b $\}$

证明: (1) Prove GG is in PSPACE

The only *space* required by this algorithm is for *storing* the recursion *stack*

- Each level of the recursion adds *a single node* to the stack
- at most m *levels* occur, where m is the number of nodes in G

Hence the algorithm runs in *linear space*.

定理: GG is PSPACE-complete

GG is PSPACE-complete, where

GG$= \{\langle G, b\rangle \mid$ Player I has a winning strategy for the generalized geography game played on graph G starting at node b $\}$

证明: (2) Prove GG is PSPACE-hard

Map the formula $\phi = \exists x_1 \forall x_2 \exists x_3 \ldots Q x_k[\psi]$ to an instance of $\langle G, b\rangle$

Assumption for *simplicity*:

- ϕ' s quantifiers begin and end with $\exists$
- They strictly alternate between $\exists$ and $\forall$

Discussion: A formula that doesn' t conform to this assumption may be converted to a slightly larger one

- by adding *extra quantifiers* binding otherwise *unused* or "dummy" variables

定理: GG is PSPACE-complete

GG is PSPACE-complete, where

GG$= \{\langle G, b\rangle \mid$ Player I has a winning strategy for the generalized geography game played on graph G starting at node b $\}$

证明: (2) Prove GG is PSPACE-hard

Map the formula $\phi = \exists x_1 \forall x_2 \exists x_3 \ldots Q x_k [\psi]$ to an instance of $\langle G, b\rangle$ *i.e.*, the reduction constructs a geography game on a *graph G* where optimal play mimics optimal play of the formula game on ϕ

定理: GG is PSPACE-complete

GG is PSPACE-complete, where

GG$= \{\langle G, b\rangle \mid$ Player I has a winning strategy for the generalized geography game played on graph G starting at node b $\}$

证明: (2) Prove GG is PSPACE-hard

Map the formula $\phi = \exists x_1 \forall x_2 \exists x_3 \ldots Qx_k[\psi]$ to an instance of $\langle G, b\rangle$ *i.e.*, the reduction constructs a geography game on a *graph G* where optimal play mimics optimal play of the formula game on ϕ

- Player I in the geography game takes the role of Player E in the formula game
- Player II takes the role of Player A

定理: GG is PSPACE-complete

GG is PSPACE-complete, where

GG$= \{\langle G, b\rangle$ | Player I has a winning strategy for the generalized geography game played on graph G starting at node b $\}$

证明: (2) Prove GG is PSPACE-hard

Map the formula $\phi = \exists x_1 \forall x_2 \exists x_3 \ldots Q x_k [\psi]$ to an instance of $\langle G, b\rangle$ *i.e.*, the reduction constructs a geography game on a *graph G* where optimal play mimics optimal play of the formula game on ϕ

- Play starts from the *left-hand side* to the *right-hand side*

定理: GG is PSPACE-complete

GG is PSPACE-complete, where

GG= $\{\langle G, b\rangle$ | Player I has a winning strategy for the generalized geography game played on graph G starting at node b $\}$

证明: (2) Prove GG is PSPACE-hard

Map the formula $\phi = \exists x_1 \forall x_2 \exists x_3 \ldots Q x_k[\psi]$ to an instance of $\langle G, b\rangle$ *i.e.*, the reduction constructs a geography game on a *graph G* where optimal play mimics optimal play of the formula game on ϕ

- Play starts from the *left-hand side* to the *right-hand side*
- At c, Player II may choose a node corresponding to one of ψ' s *clauses*
  - If ψ is FALSE, Player II *may win* by selecting the unsatisfied clause

定理: GG is PSPACE-complete

GG is PSPACE-complete, where

GG= $\{\langle G, b\rangle$ | Player I has a winning strategy for the generalized geography game played on graph G starting at node b $\}$

证明: (2) Prove GG is PSPACE-hard

Map the formula $\phi = \exists x_1 \forall x_2 \exists x_3 \ldots Q x_k[\psi]$ to an instance of $\langle G, b\rangle$ *i.e.*, the reduction constructs a geography game on a *graph G* where optimal play mimics optimal play of the formula game on ϕ

- Play starts from the *left-hand side* to the *right-hand side*
- At c, Player II may choose a node corresponding to one of ψ' s *clauses*
  - If ψ is FALSE, Player II *may win* by selecting the unsatisfied clause
- Player I may choose a node corresponding to a *literal* in that clause
  - If ψ is TRUE, any clause that Player II picks contains a TRUE literal

问: What can be inferred from the theorem?

答:

- *No polynomial time algorithm* exists for optimal play in generalized geography *unless* $P = PSPACE$
- Such generalizations of *chess*, *checkers*, and *GO* have been shown to be *PSPACE-hard* or hard for even larger complexity classes, depending on the details of the generalization

Outline

1 Introduction
2 Savitch' s theorem
3 The class PSPACE
4 PSPACE-Completeness
5 The classes L and NL
6 NL-Completeness
7 Conclusions
8 Homework
9 Appendix

问: Is it possible that the time and space complexity bounds that are less than linear—that is, bounds where $f(n)$ is less than n

答:

- For *time* complexity, *no*;
- For *space* complexity, *yes* – *sublinear* space bounds
  - The machine is *able* to *read* the entire *input*
  - but it *doesn' t* have enough *space* to *store* the input
  - i.e., we must *modify* our *computational model*

问: How to modify the computational model?

答: a Turing machine with *two tapes*:

- a *read-only* input tape
  - e.g., a CD-ROM...
- a *read/write* work tape

问: Is it possible that the time and space complexity bounds that are less than linear—that is, bounds where $f(n)$ is less than n
答:

- For *time* complexity, *no*;
- For *space* complexity, *yes* – *sublinear* space bounds
  - The machine is *able* to *read* the entire *input*
  - but it *doesn' t* have enough *space* to *store* the input
  - i.e., we must *modify* our *computational model*

问: How to modify the computational model?
答: a Turing machine with *two tapes*:

- a *read-only* input tape
  - e.g., a CD-ROM...
- a *read/write* work tape

定义: L and NL

L is the class of languages that are decidable in *logarithmic* space on a *deterministic* Turing machine. In other words,

$$\mathrm{L} = \mathrm{SPACE}(\log n)$$

NL is the class of languages that are decidable in *logarithmic* space on a *nondeterministic* Turing machine. In other words,

$$\mathrm{NL} = \mathrm{NSPACE}(\log n)$$

例: L

The language $A = \{0^k 1^k \mid k \geq 0\}$ is a member of L

运行过程

The machine counts the number of 0s and, separately, the number of 1s in binary on the work tape.

分析:

- The only space required is that used to record the two counters.
- In binary, each counter uses only logarithmic space and hence the algorithm runs in O(log n) space.
- Therefore, $A \in L$

例: L

The language $A = \{0^k 1^k \mid k \geq 0\}$ is a member of L

运行过程

The machine counts the number of 0s and, separately, the number of 1s in binary on the work tape.

分析:

- The only space required is that used to record the two counters.
- In binary, each counter uses only logarithmic space and hence the algorithm runs in O(log n) space.
- Therefore, $A \in L$

例: L

The language $A = \{0^k 1^k \mid k \geq 0\}$ is a member of L

运行过程

The machine counts the number of 0s and, separately, the number of 1s in binary on the work tape.

分析:

- The only space required is that used to record the two counters.
- In binary, each counter uses only logarithmic space and hence the algorithm runs in O(log n) space.
- Therefore, $A \in L$

例: NL

The language $PATH = \{\langle G, s, t\rangle \mid G$ is a directed graph that has a directed path from s to $t\}$ is a member of NL

运行过程

The machine starts at node s and nondeterministically guesses the nodes of a path from s to t

分析:

- The machine records only the *position of the current node* at each step on the work tape, *not the entire path*
- *PATH* is in NL.

例: NL

The language $PATH = \{\langle G, s, t\rangle \mid G$ is a directed graph that has a directed path from s to $t\}$ is a member of NL

运行过程

The machine starts at node s and nondeterministically guesses the nodes of a path from s to t

分析:

- The machine records only the *position of the current node* at each step on the work tape, *not the entire path*
- *PATH* is in NL.

例: NL

The language $PATH = \{\langle G, s, t\rangle \mid G$ is a directed graph that has a directed path from s to $t\}$ is a member of NL

运行过程

The machine starts at node s and nondeterministically guesses the nodes of a path from s to t

分析:

- The machine records only the *position of the current node* at each step on the work tape, *not the entire path*
- *PATH* is in NL.

问: Does any $f(n)$ *space* bounded Turing machine run in *time* $2^{O(f(n))}$?
答: *Not* any longer, e.g., $O(1)$ space, n steps
问: Bound of *running* time if M runs in $f(n)$ *space*? $(f(n) \geq \log n)$
答: Redefine *configuration* first

重新定义: Configuration

If M is a Turing machine that has a *separate* read-only *input tape* and w is an input, a ***configuration*** of M on w is a setting of the *state*, the *work tape*, and the *positions* of the *two* tape *heads*. The input w is *not a part* of the configuration of M on w.

问: Again, time bound if M runs in $f(n)$ *space*? $(f(n) \geq \log n)$
答: At most *exponential*

- input of length$=n \Rightarrow$ number of configurations of $M= n2^{O(f(n))}$.
- Time bound is $n2^{O(f(n))} = 2^{O(f(n))}$, if $(f(n) \geq \log n)$

问: Does any $f(n)$ *space* bounded Turing machine run in *time* $2^{O(f(n))}$?
答: *Not* any longer, e.g., $O(1)$ space, n steps
问: Bound of *running* time if M runs in $f(n)$ *space*? $(f(n) \geq \log n)$
答: Redefine *configuration* first

重新定义: Configuration

If M is a Turing machine that has a *separate* read-only *input tape* and w is an input, a ***configuration*** of M on w is a setting of the *state*, the *work tape*, and the *positions* of the *two* tape *heads*. The input w is *not a part* of the configuration of M on w.

问: Again, time bound if M runs in $f(n)$ *space*? $(f(n) \geq \log n)$
答: At most *exponential*

- input of length$=n \Rightarrow$ number of configurations of $M = n2^{O(f(n))}$.
- Time bound is $n2^{O(f(n))} = 2^{O(f(n))}$, if $(f(n) \geq \log n)$

问: Does any $f(n)$ *space* bounded Turing machine run in *time* $2^{O(f(n))}$?
答: *Not* any longer, e.g., $O(1)$ space, n steps
问: Bound of *running* time if M runs in $f(n)$ *space*? ($f(n) \geq \log n$)
答: Redefine *configuration* first

重新定义: Configuration

If M is a Turing machine that has a *separate* read-only *input tape* and w is an input, a ***configuration*** of M on w is a setting of the *state*, the *work tape*, and the *positions* of the *two* tape *heads*. The input w is *not a part* of the configuration of M on w.

问: Again, time bound if M runs in $f(n)$ *space*? ($f(n) \geq \log n$)
答: At most *exponential*

- input of length$=n \Rightarrow$ number of configurations of $M = n2^{O(f(n))}$.
- Time bound is $n2^{O(f(n))} = 2^{O(f(n))}$, if ($f(n) \geq \log n$)

问: Does any $f(n)$ *space* bounded Turing machine run in *time* $2^{O(f(n))}$?
答: *Not* any longer, e.g., $O(1)$ space, n steps
问: Bound of *running* time if M runs in $f(n)$ *space*? ($f(n) \geq \log n$)
答: Redefine *configuration* first

重新定义: Configuration

If M is a Turing machine that has a *separate* read-only *input tape* and w is an input, a ***configuration*** of M on w is a setting of the *state*, the *work tape*, and the *positions* of the *two* tape *heads*. The input w is *not a part* of the configuration of M on w.

问: Again, time bound if M runs in $f(n)$ *space*? ($f(n) \geq \log n$)
答: At most *exponential*

- input of length$=n \Rightarrow$ number of configurations of $M = n2^{O(f(n))}$.
- Time bound is $n2^{O(f(n))} = 2^{O(f(n))}$, if ($f(n) \geq \log n$)

1 Introduction
2 Savitch’s theorem
3 The class PSPACE
4 PSPACE-Completeness
5 The classes L and NL
6 NL-Completeness
7 Conclusions
8 Homework
9 Appendix

问: Is the PATH problem in NL?
答: Yes.

问: Is the PATH problem in L?
答: Not known yet.

问: More generally, given
$\underline{\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXPTIME}}$, is L=NL?
答: Also unknown yet.

问: Any resolution for the questions?
答: Reducibility, but *not polynomial time* reducibility

- Because all problems in NL are solvable in polynomial time
    - which will also be proved.
- Instead, a new type of reducibility: *log space reducibility*

问: Is the PATH problem in NL?
答: Yes.

问: Is the PATH problem in L?
答: Not known yet.

问: More generally, given $\underline{\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXPTIME}}$, is L=NL?
答: Also unknown yet.

问: Any resolution for the questions?
答: Reducibility, but *not polynomial time* reducibility

- Because all problems in NL are solvable in polynomial time
  - which will also be proved.
- Instead, a new type of reducibility: *log space reducibility*

问: Is the PATH problem in NL?
答: Yes.

问: Is the PATH problem in L?
答: Not known yet.

问: More generally, given
$\underline{\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXPTIME}}$, is L=NL?
答: Also unknown yet.

问: Any resolution for the questions?
答: Reducibility, but *not polynomial time* reducibility

- Because all problems in NL are solvable in polynomial time
  - which will also be proved.
- Instead, a new type of reducibility: *log space reducibility*

问: Is the PATH problem in NL?
答: Yes.

问: Is the PATH problem in L?
答: Not known yet.

问: More generally, given
$\underline{\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXPTIME}}$, is L=NL?
答: Also unknown yet.

问: Any resolution for the questions?
答: Reducibility, but *not polynomial time* reducibility

- Because all problems in NL are solvable in polynomial time
    - which will also be proved.
- Instead, a new type of reducibility: *log space reducibility*

定义: Log Space Reducibility

A *log space transducer* is a Turing machine with a read-only input tape, a write-only output tape, and a read/write work tape.

- The head on the output tape *cannot move leftward*, so it *cannot read* what it has *written*. The *work tape* may contain $O(\log n)$ symbols.

A log space transducer M computes a function $f : \Sigma^* \to \Sigma^*$, where $f(w)$ is the string remaining on the output tape after M halts when it is started with w on its input tape. We call f a *log space computable function*. Language A is ***log space reducible*** to language B, written $A \leq_L B$, if A is mapping reducible to B by means of a log space computable function f.

回顾: 定义: Polynomial Time Reducibility

Language A is *polynomial time mapping reducible*, or simply ***polynomial time reducible***, to language B, written $A \leq_{\mathrm{P}} B$, if a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ exists, where for every w,

$$w \in A \Leftrightarrow f(w) \in B$$

The function f is called the *polynomial time reduction* of A to B

定义: NL-complete

A language B is NL-complete if

- $B \in$ NL, and
- every A in NL is *log space reducible* to B

回顾: 定义: PSPACE-completeness

A language B is PSPACE-complete if it satisfies two conditions:

1. B is in PSPACE, and
2. every A in PSPACE is *polynomial* ***time*** *reducible* to B.

If B merely satisfies condition 2, we say that it is *PSPACE-hard*

定理

If $A \leq_L B$ and $B \in \mathsf{L}$, then $A \in \mathsf{L}$.

定理

If $A \leq_L B$ and $B \in$ L, then $A \in$ L.

证明思路 1: (错误方法)

- A first maps its input w to $f(w)$, using the *log space reduction* f
- then applies the log space algorithm for B

定理

If $A \leq_L B$ and $B \in$ L, then $A \in$ L.

证明思路 1: (错误方法)

- A first maps its input w to $f(w)$, using the *log space reduction* f
- then applies the log space algorithm for B

问: What is the problem?
答: The *storage required for $f(w)$* (i.e., output tape) may be *too large* to fit within the *log space bound*

定理

If $A \leq_L B$ and $B \in$ L, then $A \in$ L.

证明思路 1: (错误方法)

- A first maps its input w to $f(w)$, using the *log space reduction* f
- then applies the log space algorithm for B

问: What is the problem?
答: The *storage required for $f(w)$* (i.e., output tape) may be *too large* to fit within the *log space bound*

问: How to solve the problem?
答: 证明思路 2

定理

If $A \leq_L B$ and $B \in$ L, then $A \in$ L.

证明思路 2: (正确方法)

A' s machine M_A computes *individual* symbols of $f(w)$ as requested by B' s machine M_B, i.e., every time M_B moves:

- M_A *restarts* the computation of f on w from the beginning, and
- *ignores* all the output *except* for the desired location of $f(w)$

定理

If $A \leq_L B$ and $B \in$ L, then $A \in$ L.

证明思路 2: (正确方法)

A' s machine M_A computes *individual* symbols of $f(w)$ as requested by B' s machine M_B, i.e., every time M_B moves:

- M_A *restarts* the computation of f on w from the beginning, and
- *ignores* all the output *except* for the desired location of $f(w)$

分析:

- *Inefficient* in its *time* complexity
  - require occasional recomputation of parts of $f(w)$
- *Advantage*: only a single symbol of $f(w)$ needs to be stored at any point

定义: NL-complete

A language B is NL-complete if

- $B \in$ NL, and
- every A in NL is *log space reducible* to B

定理

If $A \leq_L B$ and $B \in$ L, then $A \in$ L.

推论

If any NL-complete language is in L, then L $=$ NL.

定理: NL-completeness of PATH

PATH is NL-complete

证明思路

核心问题: Prove every language A in NL is log space reducible to PATH i.e., construct a *graph* that represents the computation of the *nondeterministic* log space Turing machine for A

- *One node* points to a *second node*,
  - if the corresponding *first configuration* can *yield* the *second configuration* in a *single step* of the NTM
- Hence, the machine accepts w whenever some *path* from the node corresponding to the *start configuration* leads to the node corresponding to the *accepting configuration*

定理: NL-completeness of PATH

PATH is NL-complete

证明思路

核心问题: Prove every language A in NL is log space reducible to PATH

i.e., construct a *graph* that represents the computation of the *nondeterministic* log space Turing machine for A

- *One node* points to a *second node*,
  - if the corresponding *first configuration* can *yield* the *second configuration* in a *single step* of the NTM
- Hence, the machine accepts w whenever some *path* from the node corresponding to the *start configuration* leads to the node corresponding to the *accepting configuration*

定理: NL-completeness of PATH

PATH is NL-complete

证明思路

核心问题: Prove every language A in NL is log space reducible to PATH i.e., construct a *graph* that represents the computation of the *nondeterministic* log space Turing machine for A

- *One node* points to a *second node*,
  - if the corresponding *first configuration* can *yield* the *second configuration* in a *single step* of the NTM
- Hence, the machine accepts w whenever some *path* from the node corresponding to the *start configuration* leads to the node corresponding to the *accepting configuration*

定理: NL-completeness of PATH

PATH is NL-complete

证明思路

核心问题: Prove every language A in NL is log space reducible to PATH

i.e., construct a *graph* that represents the computation of the *nondeterministic* log space Turing machine for A

- *One node* points to a *second node*,
  - if the corresponding *first configuration* can *yield* the *second configuration* in a *single step* of the NTM
- Hence, the machine accepts w whenever some *path* from the node corresponding to the *start configuration* leads to the node corresponding to the *accepting configuration*

定理: NL-completeness of PATH

PATH is NL-complete

证明思路

核心问题: Prove every language A in NL is log space reducible to PATH
i.e., construct a *graph* that represents the computation of the *nondeterministic* log space Turing machine for A

- *One node* points to a *second node*,
  - if the corresponding *first configuration* can *yield* the *second configuration* in a *single step* of the NTM
- Hence, the machine accepts w whenever some *path* from the node corresponding to the *start configuration* leads to the node corresponding to the *accepting configuration*

定理: NL-completeness of PATH

PATH is NL-complete

证明 (Proof by Construction)

Assume: NTM M decides A in $O(\log n)$ space.

Goal: Given an input w, we construct $\langle G, s, t\rangle$ in log space

- where G is a directed graph that contains a path from s to t if and only if M accepts w

Construction:

- nodes of G: the configurations of M on w
- (c_1, c_2) is an edge: if c_2 is one of the possible next configurations of M starting from c_1

定理: NL-completeness of PATH

PATH is NL-complete

证明 (Proof by Construction)

Assume: NTM M decides A in $O(\log n)$ space.
Goal: Given an input w, we construct $\langle G, s, t\rangle$ in log space
Construction:

- nodes of G: the configurations of M on w
- (c_1, c_2) is an edge: if c_2 is one of the possible next configurations of M starting from c_1

定理: NL-completeness of PATH

PATH is NL-complete

证明 (Proof by Construction)

Assume: NTM M decides A in $O(\log n)$ space.
Goal: Given an input w, we construct $\langle G, s, t\rangle$ in log space
Construction:

- nodes of G: the configurations of M on w
- (c_1, c_2) is an edge: if c_2 is one of the possible next configurations of M starting from c_1

Correctness of reduction:

- $(\Rightarrow)$ Whenever M *accepts* its input, some *branch* of its computation accepts, it corresponds to a *path* from the start configuration s to the accepting configuration t in G

定理: NL-completeness of PATH

PATH is NL-complete

证明 (Proof by Construction)

Assume: NTM M decides A in $O(\log n)$ space.
Goal: Given an input w, we construct $\langle G, s, t\rangle$ in log space
Construction:

- nodes of G: the configurations of M on w
- (c_1, c_2) is an edge: if c_2 is one of the possible next configurations of M starting from c_1

Correctness of reduction:

- ($\Leftarrow$) If some path exists from s to t in G, some computation branch accepts when M runs on input w, and M accepts w.

定理: NL-completeness of PATH

PATH is NL-complete

证明 (Proof by Construction)

Assume: NTM M decides A in $O(\log n)$ space.
Goal: Given an input w, we construct $\langle G, s, t\rangle$ in log space
Construction:

- nodes of G: the configurations of M on w
- (c_1, c_2) is an edge: if c_2 is one of the possible next configurations of M starting from c_1

Space complexity of reduction:

- *Listing the nodes*:
  - *Represent* a node: $c \log n$ space for some constant c
  - *List* nodes: sequentially goes through all possible strings of length $c \log n$, test the legality, and output the legal ones

定理: NL-completeness of PATH

PATH is NL-complete

证明 (Proof by Construction)

Assume: NTM M decides A in $O(\log n)$ space.
Goal: Given an input w, we construct $\langle G, s, t\rangle$ in log space
Construction:

- nodes of G: the configurations of M on w
- (c_1, c_2) is an edge: if c_2 is one of the possible next configurations of M starting from c_1

Space complexity of reduction:

- *Listing the edges*:
  - Tries all pairs (c_1, c_2) in turn to find which qualify as edges of G.
    - by examine M' s transition function
  - Those that do are added to the output tape.

推论

NL $\subseteq$ P

证明

- Any language in NL is *log space reducible* to PATH
  - A Turing machine that uses space $f(n)$ runs in time $n2^{O(f(n))}$
  - A reducer that runs in *log space* also runs in *polynomial time*
  - Any language in NL is *polynomial time reducible* to PATH
- PATH is in P (Proved in Sec.4.1)
- Every language that is polynomial time reducible to a language in P, is also in P
- Proved

1 Introduction
2 Savitch' s theorem
3 The class PSPACE
4 PSPACE-Completeness
5 The classes L and NL
6 NL-Completeness
7 Conclusions
8 Homework
9 Appendix

总结

定义:

- PSPACE, PSPACE-complete, PSPACE-hard
- TQBF, formula game, generalized geography
- L, NL, Log Space Reducibility, NL-complete

定理:

- Savitch' s Theorem: NSPACE$(f(n))$ $\subseteq$SPACE$(f^2(n))$
- PSPACE-complete: TQBF, FORMULA-GAME, GG
- NL: PATH, NL-complete: PATH,
- If $A \leq_L B$ and $B \in$ L, then $A \in$ L
- NL $=$ coNL (略)

推论:

- If any NL-complete language is in L, then L $=$ NL.
- NL $\subseteq$ P

$$\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXPTIME}$$

Outline

1. Introduction
2. Savitch’ s theorem
3. The class PSPACE
4. PSPACE-Completeness
5. The classes L and NL
6. NL-Completeness
7. Conclusions
8. Homework
9. Appendix

4.2 Space Complexity

Homework

8.2 Consider the following position in the standard tic-tac-toe game.

×		
	○	
○		×

Let's say that it is the × -player's turn to move next. Describe a winning strategy for this player. (Recall that a winning strategy isn't merely the best move to make in the current position. It also includes all the responses that this player must make in order to win, however the opponent moves.)

8.3 Consider the following generalized geography game wherein the start node is the one with the arrow pointing in from nowhere. Does Player I have a winning strategy? Does Player II? Give reasons for your answers.

8.9 A *ladder* is a sequence of strings $s_1, s_2, \ldots, s_k$, wherein every string differs from the preceding one by exactly one character. For example, the following is a ladder of English words, starting with "head" and ending with "free":

head, hear, near, fear, bear, beer, deer, deed, feed, feet, fret, free.

Let $LADDER_{\mathsf{DFA}} = \{\langle M, s, t\rangle | \ M$ is a DFA and $L(M)$ contains a ladder of strings, starting with s and ending with $t\}$. Show that $LADDER_{\mathsf{DFA}}$ is in PSPACE.

8.11 Show that if every NP-hard language is also PSPACE-hard, then PSPACE $=$ NP.

1 Introduction
2 Savitch’ s theorem
3 The class PSPACE
4 PSPACE-Completeness
5 The classes L and NL
6 NL-Completeness
7 Conclusions
8 Homework
9 Appendix

定义: Time complexity, Running time

Let M be a *deterministic* Turing machine that *halts* on all inputs.

- The *running time* or *time complexity* of M: is the function $f : \mathcal{N} \to \mathcal{N}$, where $f(n)$ is the ***maximum*** *number* of steps that M uses on ***any*** *input of length* n.
- If $f(n)$ is the running time of M, we say that M *runs in time* $f(n)$ and that M is an $f(n)$ *time Turing machine*.
- Customarily we use n to represent the *length of the input*.

定义: Running time of a nondeterministic TM

Let N be a *nondeterministic* Turing machine that is a decider. The *running time* of N is the function $f : \mathcal{N} \to \mathcal{N}$, where $f(n)$ is the *maximum number* of steps that N uses on *any branch* of its computation on *any input of length* n.

定义: Time complexity class

Let $t : \mathcal{N} \to \mathcal{R}^+$ be a function.
Define the *time complexity class*, $\mathrm{TIME}(t(n))$, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

定义: NTIME

NTIME$(t(n)) = \{L \mid L$ is a language decided by an $O(t(n))$ time *nondeterministic Turing machine*$\}$

定理: Decidability of E_{DFA}

The language E_{DFA} is decidable, where

$$E_{\mathrm{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

证明 (Proof by Construction)

Design a TM T

$T =$"On input $\langle A\rangle$, where A is a DFA:

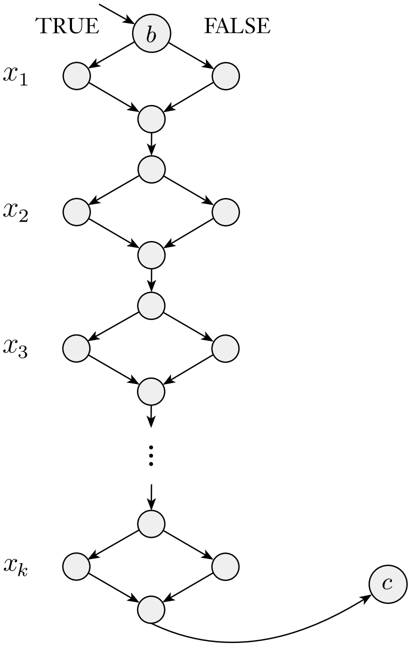
1. *Mark* the *start state* of A
2. Repeat until no new states get marked
3. *Mark any state* that has a *transition* coming into it *from any state* that is already marked.
4. If no accept state is marked, accept; otherwise, reject."

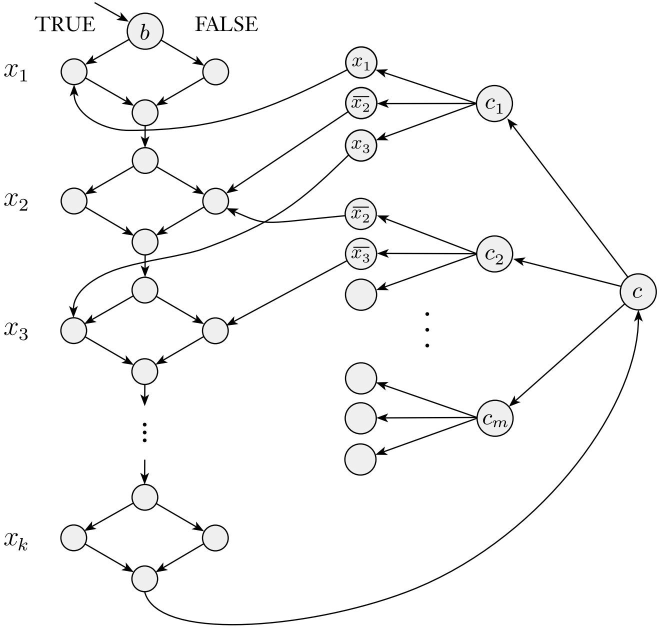
Basics on the *left-hand* side:

- Play starts at node b, which appears at the top *left-hand side* of G
- Underneath b, a sequence of *diamond* structures appears, *one* for each of the *variables* of ϕ

Play Process:

- Play starts at b
- Player I must select one of the two edges going from b
  - i.e., $x_1 = 1$ or $x_1 = 0$
- Forced move for II and then I
- Player II chooses $x_2 = 1$ or $x_2 = 0$
- ...
- Move c, i.e., the *right-hand* side





$$\phi = \exists x_1 \forall x_2 \cdots \exists x_k \left[(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3} \vee \cdots) \wedge \cdots \wedge (\qquad) \right]$$

定理: Cook–Levin theorem

SAT is NP-complete

定理: Cook–Levin theorem

SAT is NP-complete

证明

(1) Prove that SAT is in NP

- A *nondeterministic polynomial* time machine can guess an assignment to a given formula φ
- and *accept* if the assignment satisfies φ

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}} SAT$ 1 2 3

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}}$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}}$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
  - A tableau for N on w is an $n^k \times n^k$ table whose *rows* are the *configurations* of *a branch* of the computation of N on input w

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}}$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
  - A tableau for N on w is an $n^k \times n^k$ table whose *rows* are the *configurations* of *a branch* of the computation of N on input w
  - A tableau is *accepting* if *any row* of the tableau is an *accepting configuration*.

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_P$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
  - A tableau for N on w is an $n^k \times n^k$ table whose *rows* are the *configurations* of *a branch* of the computation of N on input w
  - A tableau is *accepting* if *any row* of the tableau is an *accepting configuration*.
  - the problem of determining whether N *accepts* w is *equivalent to* the problem of determining whether an *accepting* tableau for N on w exists.

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}}$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides* A *in* n^k *time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}}$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - Say that Q and Γ are the *state set* and *tape alphabet* of N, respectively
  - Let $C = Q \cup \Gamma \cup \{\#\}$
  - For each i and j between 1 and n_k and for each s in C, we have a variable, $x_{i,j,s}$.
  - Each of the $(n^k)^2$ entries of a tableau is called a *cell*
  - The cell in row i and column j is called $cell[i,j]$ and contains a symbol

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}}$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - Say that Q and Γ are the *state set* and *tape alphabet* of N, respectively
  - Let $C = Q \cup \Gamma \cup \{\#\}$
  - For each i and j between 1 and n_k and for each s in C, we have a variable, $x_{i,j,s}$.
  - Each of the $(n^k)^2$ entries of a tableau is called a *cell*
  - The cell in row i and column j is called $cell[i,j]$ and contains a symbol

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_P$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - Say that Q and Γ are the *state set* and *tape alphabet* of N, respectively
  - Let $C = Q \cup \Gamma \cup \{\#\}$
  - For each i and j between 1 and n_k and for each s in C, we have a variable, $x_{i,j,s}$.
  - Each of the $(n^k)^2$ entries of a tableau is called a *cell*
  - The cell in row i and column j is called $cell[i,j]$ and contains a symbol

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_P$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - Say that Q and Γ are the *state set* and *tape alphabet* of N, respectively
  - Let $C = Q \cup \Gamma \cup \{\#\}$
  - For each i and j between 1 and n_k and for each s in C, we have a variable, $x_{i,j,s}$.
  - Each of the $(n^k)^2$ entries of a tableau is called a *cell*
  - The cell in row i and column j is called $cell[i,j]$ and contains a symbol

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}}$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - Say that Q and Γ are the *state set* and *tape alphabet* of N, respectively
  - Let $C = Q \cup \Gamma \cup \{\#\}$
  - For each i and j between 1 and n_k and for each s in C, we have a variable, $x_{i,j,s}$.
  - Each of the $(n^k)^2$ entries of a tableau is called a *cell*
  - The cell in row i and column j is called $cell[i,j]$ and contains a symbol

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}}$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - Say that Q and Γ are the *state set* and *tape alphabet* of N, respectively
  - Let $C = Q \cup \Gamma \cup \{\#\}$
  - For each i and j between 1 and n_k and for each s in C, we have a variable, $x_{i,j,s}$.
  - Each of the $(n^k)^2$ entries of a tableau is called a *cell*
  - The cell in row i and column j is called $cell[i,j]$ and contains a symbol

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_P$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i, j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_P$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - $\phi = \underline{\phi_{\text{cell}}} \wedge \underline{\phi_{\text{start}}} \wedge \underline{\phi_{\text{accept}}} \wedge \underline{\phi_{\text{move}}}$

$$\phi_{\text{cell}} = \bigwedge_{1 \leq i,j \leq n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]$$

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \le_{\mathrm{P}}$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \ne t}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right]$$

1st part: *at least one* variable that is associated with *each cell* is on

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}}$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$

$$\phi_{\text{cell}} = \bigwedge_{1 \leq i,j \leq n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]$$

2nd part: *no more than one variable* is on for *each cell*

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_P SAT$ 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$

$$\begin{aligned}\phi_{\text{start}} = {} & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}\end{aligned}$$

the *first row* of the table is the *starting configuration* of N on w

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}}$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$

$$\phi_{\text{accept}} = \bigvee_{1 \leq i,j \leq n^k} x_{i,j,q_{\text{accept}}}$$

an *accepting configuration* occurs in the tableau

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}}$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$

$$\phi_{\text{move}} = \bigwedge_{1 \leq i < n^k,\ 1 < j < n^k} \big(\text{the } (i,j)\text{-window is legal}\big).$$

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}} SAT$ 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$

$$\bigvee_{\substack{a_1,\ldots,a_6 \\ \text{is a legal window}}} \left(x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6}\right)$$

$$\delta(q_1, \mathtt{a}) = \{(q_1,\mathtt{b},\mathrm{R})\} \text{ and } \delta(q_1, \mathtt{b}) = \{(q_2,\mathtt{c},\mathrm{L}), (q_2,\mathtt{a},\mathrm{R})\}$$

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}} SAT$ 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$

Examples of *legal* windows:

(a)

a	q_1	b
q_2	a	c

(b)

a	q_1	b
a	a	q_2

(c)

a	a	q_1
a	a	b

(d)

#	b	a
#	b	a

(e)

a	b	a
a	b	q_2

(f)

b	b	b
c	b	b

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_P SAT$ 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$

Examples of *illegal* windows:

(a)

a	b	a
a	a	a

(b)

a	q_1	b
q_2	a	a

(c)

b	q_1	b
q_2	b	q_2

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}}$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$
- Analyze the Complexity of Reduction 3.1 3.2 3.3

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}} SAT$ 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$
- Analyze the Complexity of Reduction 3.1 3.2 3.3
  - The total number of variables is $O(n^{2k})$
    - tableau is an $n^k \times n^k$ table, so it contains n^{2k} cells

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}} SAT$ 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$
- Analyze the Complexity of Reduction 3.1 3.2 3.3
  - The total number of variables is $O(n^{2k})$
    - tableau is an $n^k \times n^k$ table, so it contains n^{2k} cells

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}}$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - $\phi = \phi_{\mathrm{cell}} \wedge \phi_{\mathrm{start}} \wedge \phi_{\mathrm{accept}} \wedge \phi_{\mathrm{move}}$
- Analyze the Complexity of Reduction 3.1 3.2 3.3
  - ϕ' s total size is $O(n^{2k})$

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_P$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$
- Analyze the Complexity of Reduction 3.1 3.2 3.3
  - ϕ' s total size is $O(n^{2k})$
    - ϕ_{cell}: contains a fixed-size fragment of the formula for each cell of the tableau, so its size is $O(n^{2k})$

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_P SAT$ 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i, j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$
- Analyze the Complexity of Reduction 3.1 3.2 3.3
  - ϕ' s total size is $O(n^{2k})$
    - ϕ_{start} has a fragment for each cell in the top row, so its size is $O(n^k)$

定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}} SAT$ 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i,j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$
- Analyze the Complexity of Reduction 3.1 3.2 3.3
  - ϕ' s total size is $O(n^{2k})$
    - ϕ_{move} and ϕ_{accept} each contain a fixed-size fragment of the formula for each cell of the tableau, so their size is $O(n^{2k})$

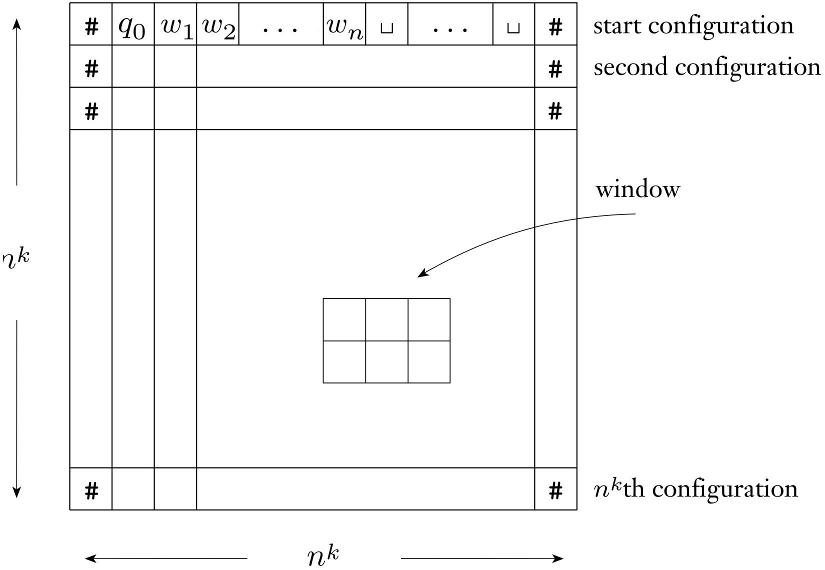
定理: Cook–Levin theorem

SAT is NP-complete

证明

(2) Take any language A in NP and show that $A \leq_{\mathrm{P}}$ *SAT* 1 2 3

- Let N be a nondeterministic Turing machine that *decides A in n^k time* for some constant k, and construct a tableau for N
- Reduction from w to a formula ϕ: 2.1 2.2
  - If $x_{i,j,s}$ takes on the value 1, it means that $cell[i, j]$ contains an s
  - $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$
- Analyze the Complexity of Reduction 3.1 3.2 3.3
  - We may easily construct a reduction that produces ϕ in polynomial time from the input w
    - Each component of the formula is composed of many nearly identical fragments



#
q_0
w_1
w_2
...
w_n
␣
...
␣
#
start configuration
second configuration
window
n^kth configuration
n^k
n^k