Lecture 8: Space Complexity I

Arijit Bishnu

18.03.2010

Outline

Outline

Space Bounded Computation

- Our goal is to consider the computational complexity of problems in terms of the amount of space/memory they require.

Space Bounded Computation

- Our goal is to consider the computational complexity of problems in terms of the amount of space/memory they require.
- A key difference between space and time is that space can be reused, whereas time cannot be reused.

Space Bounded Computation

- Our goal is to consider the computational complexity of problems in terms of the amount of space/memory they require.
- A key difference between space and time is that space can be reused, whereas time cannot be reused.
- We look at space bounded computation where a TM peforms its tasks using a restricted number of tape cells. The number of tape cells is a function of the input size.

Space Bounded Computation

- Our goal is to consider the computational complexity of problems in terms of the amount of space/memory they require.
- A key difference between space and time is that space can be reused, whereas time cannot be reused.
- We look at space bounded computation where a TM peforms its tasks using a restricted number of tape cells. The number of tape cells is a function of the input size.
- Only cells used in the read/write tapes count towards the space bound.

Space Bounded Computation

Definition (Space Bounded Computation): The Class SPACE

Let $S : \mathbb{N} \to \mathbb{N}$ and $L \subseteq \{0,1\}^*$. We define $L \in \mathrm{SPACE}(S(n))$ if there is a constant c and a TM M deciding L such that on every input $x \in \{0,1\}^*$, the total number of locations on the read/write tape that are at some point non-blank during M's execution on x is at most $c \cdot S(|x|)$.

Space Bounded Computation

Definition (Space Bounded Computation): The Class SPACE

Let $S : \mathbb{N} \to \mathbb{N}$ and $L \subseteq \{0,1\}^*$. We define $L \in \mathrm{SPACE}(S(n))$ if there is a constant c and a TM M deciding L such that on every input $x \in \{0,1\}^*$, the total number of locations on the read/write tape that are at some point non-blank during M's execution on x is at most $c \cdot S(|x|)$.

Definition (Space Bounded Computation): The Class NSPACE

In the above definition, replace SPACE with NSPACE and the TM with NDTM.

Space Bounded Computation

Definition (Space Bounded Computation): The Class SPACE

Let $S : \mathbb{N} \rightarrow \mathbb{N}$ and $L \subseteq \{0,1\}^*$. We define $L \in \mathrm{SPACE}(S(n))$ if there is a constant c and a TM M deciding L such that on every input $x \in \{0,1\}^*$, the total number of locations on the read/write tape that are at some point non-blank during M's execution on x is at most $c \cdot S(|x|)$.

Definition (Space Bounded Computation): The Class NSPACE

In the above definition, replace SPACE with NSPACE and the TM with NDTM.

Remark

We will restrict our attention to space bounds $S : \mathbb{N} \rightarrow \mathbb{N}$ that are space constructible functions. Intuitively, if S is space constructible, then the machine knows the space bound it is operating under.

Remarks

- It makes sense to consider space bounded machines with $S(n) < n$ but not $\mathrm{DTIME}(T(n))$ for $T(n) < n$.

Remarks

- It makes sense to consider space bounded machines with $S(n) < n$ but not $\mathrm{DTIME}(T(n))$ for $T(n) < n$.
- We will assume $S(n) > \log n$ since the machine needs to remember the address of the cell currently being read.

Remarks

- It makes sense to consider space bounded machines with $S(n) < n$ but not DTIME($T(n)$) for $T(n) < n$.
- We will assume $S(n) > \log n$ since the machine needs to remember the address of the cell currently being read.
- DTIME($S(n)$) $\subseteq$ SPACE($S(n)$) since a TM can access only one tape cell per step and space can be reused.

Outline

Configuration Graph of a Turing Machine

Graph Specification

- A configuration of a TM M consists of the contents of all non-blank entries of M's tapes, state and head position at a particular point of its execution.

Configuration Graph of a Turing Machine

Graph Specification

- A configuration of a TM M consists of the contents of all non-blank entries of M's tapes, state and head position at a particular point of its execution.
- For every TM M and input $x \in \{0, 1\}^*$, the configuration graph of M on x, denoted as $G_{M,x}$ is a directed graph whose nodes represent the possible configurations that M can reach from C_s^x, the start configuration.

Configuration Graph of a Turing Machine

Graph Specification

- A configuration of a TM M consists of the contents of all non-blank entries of M's tapes, state and head position at a particular point of its execution.
- For every TM M and input $x \in \{0, 1\}^*$, the configuration graph of M on x, denoted as $G_{M,x}$ is a directed graph whose nodes represent the possible configurations that M can reach from C_s^x, the start configuration.
- There is an edge from a configuration C to C' if C' can be reached from C in one step according to δ of M.

Configuration Graph of a Turing Machine

Graph Specification

- A configuration of a TM M consists of the contents of all non-blank entries of M's tapes, state and head position at a particular point of its execution.
- For every TM M and input $x \in \{0,1\}^*$, the configuration graph of M on x, denoted as $G_{M,x}$ is a directed graph whose nodes represent the possible configurations that M can reach from C_s^x, the start configuration.
- There is an edge from a configuration C to C' if C' can be reached from C in one step according to δ of M.
- For a DTM, the outdegree of a node is 1 and for a NDTM, it depends on the number of branches.

Configuration Graph of a Turing Machine

Graph Specification continued....

- $G_{M,x}$ is a DAG.

Configuration Graph of a Turing Machine

Graph Specification continued....

- $G_{M,x}$ is a DAG.
- We ensure a single accepting configuration C_{acc} by suitably modifying M to erase all its work tapes before halting.

Configuration Graph of a Turing Machine

Graph Specification continued....

- $G_{M,x}$ is a DAG.
- We ensure a single accepting configuration C_{acc} by suitably modifying M to erase all its work tapes before halting.
- In terms of reachability in graphs, M accepts x iff $\exists$ a directed path in $G_{M,x}$ from C_s^x to C_{acc}.

Claim about $G_{M,x}$

Claim about $G_{M,x}$

Let $G_{M,x}$ be the configuration graph of a space-$S(n)$ machine M on some input x of length n. Then,

Claim about $G_{M,x}$

Claim about $G_{M,x}$

Let $G_{M,x}$ be the configuration graph of a space-$S(n)$ machine M on some input x of length n. Then,

- Every vertex in $G_{M,x}$ can be described using $c \cdot S(n)$ bits where c is a constant depending on M. $G_{M,x}$ has at most $2^{cS(n)}$ nodes.

Claim about $G_{M,x}$

Claim about $G_{M,x}$

Let $G_{M,x}$ be the configuration graph of a space-$S(n)$ machine M on some input x of length n. Then,

- Every vertex in $G_{M,x}$ can be described using $c \cdot S(n)$ bits where c is a constant depending on M. $G_{M,x}$ has at most $2^{cS(n)}$ nodes.
- There is an $O(S(n))$-size CNF formula $\varphi_{M,x}$ such that for every two strings C and C', $\varphi(M,x)(C,C') = 1$ if and only if C, C' encode two neighboring configurations in $G_{M,x}$

Claim about $G_{M,x}$

Claim about $G_{M,x}$

Let $G_{M,x}$ be the configuration graph of a space-$S(n)$ machine M on some input x of length n. Then,

- Every vertex in $G_{M,x}$ can be described using $c \cdot S(n)$ bits where c is a constant depending on M. $G_{M,x}$ has at most $2^{cS(n)}$ nodes.
- There is an $O(S(n))$-size CNF formula $\varphi_{M,x}$ such that for every two strings C and C', $\varphi(M,x)(C,C') = 1$ if and only if C, C' encode two neighboring configurations in $G_{M,x}$

Proof

The first part is just about the encoding of the TM. The second part follows a similar pattern as the proof of Cook Levin.

A Theorem relating some classes

Theorem

For every space constructible $S : \mathbb{N} \to \mathbb{N}$,

$$\mathrm{DTIME}(S(n)) \subseteq \mathrm{SPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$$

A Theorem relating some classes

Theorem

For every space constructible $S : \mathbb{N} \to \mathbb{N}$,

$$\mathrm{DTIME}(S(n)) \subseteq \mathrm{SPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$$

Proof

A Theorem relating some classes

Theorem

For every space constructible $S : \mathbb{N} \to \mathbb{N}$,

$$\mathrm{DTIME}(S(n)) \subseteq \mathrm{SPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$$

Proof

- We have already seen $\mathrm{DTIME}(S(n)) \subseteq \mathrm{SPACE}(S(n))$.

A Theorem relating some classes

Theorem

For every space constructible $S : \mathbb{N} \to \mathbb{N}$,

$$\mathrm{DTIME}(S(n)) \subseteq \mathrm{SPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$$

Proof

- We have already seen $\mathrm{DTIME}(S(n)) \subseteq \mathrm{SPACE}(S(n))$.
- Also, it is obvious that $\mathrm{SPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.

A Theorem relating some classes

Theorem

For every space constructible $S : \mathbb{N} \to \mathbb{N}$,

$$\mathrm{DTIME}(S(n)) \subseteq \mathrm{SPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$$

Proof

- We have already seen $\mathrm{DTIME}(S(n)) \subseteq \mathrm{SPACE}(S(n))$.
- Also, it is obvious that $\mathrm{SPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.
- For proving $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$, use the concept of configuration graph. Enumerate over all possible configurations to construct $G_{M,x}$ in $2^{O(S(n))}$ time and check using BFS whether $\exists$ a directed path from C_s^x to C_{acc} in $G_{M,x}$.

Outline

Some Space Complexity Classes

Definition

PSPACE $= \bigcup_{c>0} \mathrm{SPACE}(n^c)$. The class PSPACE is an analog of the class P.

Some Space Complexity Classes

Definition

$\mathrm{PSPACE} = \bigcup_{c>0} \mathrm{SPACE}(n^c)$. The class PSPACE is an analog of the class P.

Definition

$\mathrm{NPSPACE} = \bigcup_{c>0} \mathrm{NSPACE}(n^c)$. The class NPSPACE is an analog of the class NP.

Some Space Complexity Classes

Definition

$\mathrm{PSPACE} = \bigcup_{c>0} \mathrm{SPACE}(n^c)$. The class PSPACE is an analog of the class P.

Definition

$\mathrm{NPSPACE} = \bigcup_{c>0} \mathrm{NSPACE}(n^c)$. The class NPSPACE is an analog of the class NP.

Definition

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

Some Space Complexity Classes

Definition

$\mathrm{PSPACE} = \bigcup_{c>0} \mathrm{SPACE}(n^c)$. The class PSPACE is an analog of the class P.

Definition

$\mathrm{NPSPACE} = \bigcup_{c>0} \mathrm{NSPACE}(n^c)$. The class NPSPACE is an analog of the class NP.

Definition

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

Definition

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

Some Examples

$3SAT \in PSPACE$

Some Examples

3SAT $\in$ PSPACE

- We can design a TM that decides 3SAT in linear space.

Some Examples

3SAT $\in$ PSPACE

- We can design a TM that decides 3SAT in linear space.
- The TM reuses linear space to evaluate all 2^k assignments where k is the number of variables.

Some Examples

3SAT $\in$ PSPACE

- We can design a TM that decides 3SAT in linear space.
- The TM reuses linear space to evaluate all 2^k assignments where k is the number of variables.

Claim

NP $\subseteq$ PSPACE

Some Examples

3SAT $\in$ PSPACE

- We can design a TM that decides 3SAT in linear space.
- The TM reuses linear space to evaluate all 2^k assignments where k is the number of variables.

Claim

NP $\subseteq$ PSPACE

Proof

Cycle through all possible certificates by reusing polynomial space.

Example

Is the following in L?

Check whether the following language is in L?

$$\text{EVEN} = \{x \mid x \text{ has an even number of 1s}\}.$$

Example

Is the following in NL?

Check whether the following language is in NL?

PATH $= \{< G, s, t > \mid G$ is a directed graph in which there is a path from s to $t\}$. G has n nodes.

Example

Is the following in NL?

Check whether the following language is in NL?
PATH $= \{ < G, s, t > \mid G$ is a directed graph in which there is a path from s to $t\}$. G has n nodes.

Solution: PATH $\in$ NL

Example

Is the following in NL?

Check whether the following language is in NL?
PATH $= \{< G, s, t > \mid G$ is a directed graph in which there is a path from s to $t\}$. G has n nodes.

Solution: PATH $\in$ NL

- A NDTM can make a non-deterministic walk starting at s.

Example

Is the following in NL?

Check whether the following language is in NL?
PATH $= \{< G, s, t > \mid G$ is a directed graph in which there is a path from s to $t\}$. G has n nodes.

Solution: PATH $\in$ NL

- A NDTM can make a non-deterministic walk starting at s.
- It maintains the index of the vertex the machine is at and nondeterministically picks the next vertex to visit.

Example

Is the following in NL?

Check whether the following language is in NL?
PATH $= \{< G, s, t > \mid G$ is a directed graph in which there is a path from s to $t\}$. G has n nodes.

Solution: PATH $\in$ NL

- A NDTM can make a non-deterministic walk starting at s.
- It maintains the index of the vertex the machine is at and nondeterministically picks the next vertex to visit.
- The machine accepts iff the walk ends at t in at most n steps.

Example

Is the following in NL?

Check whether the following language is in NL?
PATH $= \{< G, s, t > \mid G$ is a directed graph in which there is a path from s to $t\}$. G has n nodes.

Solution: PATH $\in$ NL

- A NDTM can make a non-deterministic walk starting at s.
- It maintains the index of the vertex the machine is at and nondeterministically picks the next vertex to visit.
- The machine accepts iff the walk ends at t in at most n steps.
- If the nondeterministic walk has run for n steps already and t has not been encountered, the machine rejects.

Example

Is the following in NL?

Check whether the following language is in NL?
PATH $= \{< G, s, t > \mid G$ is a directed graph in which there is a path from s to $t\}$. G has n nodes.

Solution: PATH $\in$ NL

- A NDTM can make a non-deterministic walk starting at s.
- It maintains the index of the vertex the machine is at and nondeterministically picks the next vertex to visit.
- The machine accepts iff the walk ends at t in at most n steps.
- If the nondeterministic walk has run for n steps already and t has not been encountered, the machine rejects.
- The work tape needs to hold only $O(\log n)$ bits for the number of steps that the walk has been made and the identity of the current vertex.

The role of PATH

Is PATH in L?

- This is an open problem and is equivalent to the question whether L = NL?
- PATH is to NL what 3SAT is to NP.

Outline

PSPACE completeness

Some Relations

We know P $\subseteq$ NP $\subseteq$ PSPACE. So, there is a strong belief that P $\neq$ PSPACE.

PSPACE completeness

Some Relations

We know P $\subseteq$ NP $\subseteq$ PSPACE. So, there is a strong belief that P $\neq$ PSPACE.

Claim

P $=$ PSPACE $\Rightarrow$ P $=$ NP.

PSPACE completeness

Some Relations

We know P $\subseteq$ NP $\subseteq$ PSPACE. So, there is a strong belief that P $\neq$ PSPACE.

Claim

P $=$ PSPACE $\Rightarrow$ P $=$ NP.

Definition

A language A is PSPACE-hard if for every $L \in$ PSPACE, $L \leq_P A$. If $A \in$ PSPACE, then A is PSPACE-complete.

PSPACE completeness

Some Relations

We know P $\subseteq$ NP $\subseteq$ PSPACE. So, there is a strong belief that P $\neq$ PSPACE.

Claim

P $=$ PSPACE $\Rightarrow$ P $=$ NP.

Definition

A language A is PSPACE-hard if for every $L \in$ PSPACE, $L \leq_P A$. If $A \in$ PSPACE, then A is PSPACE-complete.

Claim

If any PSPACE-complete language is in P, then so is every other language in PSPACE. The contrapositive says that if PSPACE $\neq$ P, then a PSPACE-complete language is not in P.

Quantified Boolean Formula

- A Quantified Boolean Formula (QBF) is a boolean formula in which variables are quantified using $\exists$ and $\forall$.

Quantified Boolean Formula

- A Quantified Boolean Formula (QBF) is a boolean formula in which variables are quantified using $\exists$ and $\forall$.
- We also specify the universe over which the quantifiers work; in our case it is $\{0, 1\}$.

Quantified Boolean Formula

- A Quantified Boolean Formula (QBF) is a boolean formula in which variables are quantified using $\exists$ and $\forall$.
- We also specify the universe over which the quantifiers work; in our case it is $\{0, 1\}$.
- Thus, a QBF has the form $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \varphi(x_1, x_2, \ldots, x_n)$ where each $Q_i \in \{\exists, \forall\}$ and φ is an unquantified boolean formula.

Quantified Boolean Formula

- A Quantified Boolean Formula (QBF) is a boolean formula in which variables are quantified using $\exists$ and $\forall$.
- We also specify the universe over which the quantifiers work; in our case it is $\{0, 1\}$.
- Thus, a QBF has the form $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \varphi(x_1, x_2, \ldots, x_n)$ where each $Q_i \in \{\exists, \forall\}$ and φ is an unquantified boolean formula.
- If there are no free variables, then that formula is either TRUE or FALSE; there is nothing in between.

Quantified Boolean Formula

- A Quantified Boolean Formula (QBF) is a boolean formula in which variables are quantified using $\exists$ and $\forall$.
- We also specify the universe over which the quantifiers work; in our case it is $\{0, 1\}$.
- Thus, a QBF has the form $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \varphi(x_1, x_2, \ldots, x_n)$ where each $Q_i \in \{\exists, \forall\}$ and φ is an unquantified boolean formula.
- If there are no free variables, then that formula is either TRUE or FALSE; there is nothing in between.

Examples

Quantified Boolean Formula

- A Quantified Boolean Formula (QBF) is a boolean formula in which variables are quantified using $\exists$ and $\forall$.
- We also specify the universe over which the quantifiers work; in our case it is $\{0, 1\}$.
- Thus, a QBF has the form $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \varphi(x_1, x_2, \ldots, x_n)$ where each $Q_i \in \{\exists, \forall\}$ and φ is an unquantified boolean formula.
- If there are no free variables, then that formula is either TRUE or FALSE; there is nothing in between.

Examples

- Consider the QBF $\forall x \exists y (x \wedge y) \vee (\overline{x} \wedge \overline{y})$ over $\{0, 1\}$.

Quantified Boolean Formula

- A Quantified Boolean Formula (QBF) is a boolean formula in which variables are quantified using $\exists$ and $\forall$.
- We also specify the universe over which the quantifiers work; in our case it is $\{0,1\}$.
- Thus, a QBF has the form $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \varphi(x_1, x_2, \ldots, x_n)$ where each $Q_i \in \{\exists, \forall\}$ and φ is an unquantified boolean formula.
- If there are no free variables, then that formula is either TRUE or FALSE; there is nothing in between.

Examples

- Consider the QBF $\forall x \exists y (x \wedge y) \vee (\overline{x} \wedge \overline{y})$ over $\{0,1\}$.
- The above QBF is TRUE.

Quantified Boolean Formula

- A Quantified Boolean Formula (QBF) is a boolean formula in which variables are quantified using $\exists$ and $\forall$.
- We also specify the universe over which the quantifiers work; in our case it is $\{0, 1\}$.
- Thus, a QBF has the form $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \varphi(x_1, x_2, \ldots, x_n)$ where each $Q_i \in \{\exists, \forall\}$ and φ is an unquantified boolean formula.
- If there are no free variables, then that formula is either TRUE or FALSE; there is nothing in between.

Examples

- Consider the QBF $\forall x \exists y (x \wedge y) \vee (\overline{x} \wedge \overline{y})$ over $\{0, 1\}$.
- The above QBF is TRUE.
- What about $\forall x \forall y (x \wedge y) \vee (\overline{x} \wedge \overline{y})$?

Example

Examples

- We can recast SAT in terms of QBF as $\psi = \exists\, x_1, \ldots, \exists x_n$ $\varphi(x_1, \ldots, x_n)$ is TRUE.

Example

Examples

- We can recast SAT in terms of QBF as $\psi = \exists\ x_1, \ldots, \exists x_n$ $\varphi(x_1, \ldots, x_n)$ is TRUE.
- What about negation of the above formula?

Example

Examples

- We can recast SAT in terms of QBF as $\psi = \exists\ x_1, \ldots, \exists x_n\ \varphi(x_1, \ldots, x_n)$ is TRUE.
- What about negation of the above formula?
- The switch of $\exists$ to $\forall$ in case of SAT gives instances of TAUTOLOGY.

A New Language

Definition: A New Language TQBF

The language TQBF is the set of QBFs that are TRUE.

A New Language

Definition: A New Language TQBF

The language TQBF is the set of QBFs that are TRUE.

Theorem

TQBF is PSPACE-complete.

A New Language

Definition: A New Language TQBF

The language TQBF is the set of QBFs that are TRUE.

Theorem

TQBF is PSPACE-complete.

Proof of TQBF $\in$ PSPACE

A New Language

Definition: A New Language TQBF

The language TQBF is the set of QBFs that are TRUE.

Theorem

TQBF is PSPACE-complete.

Proof of TQBF $\in$ PSPACE

- Let $\psi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \varphi(x_1, x_2, \ldots, x_n)$ be a QBF with n variables; and $|\varphi| = m$. We design a recursive algorithm A to decide the truth of ψ.

A New Language

Definition: A New Language TQBF

The language TQBF is the set of QBFs that are TRUE.

Theorem

TQBF is PSPACE-complete.

Proof of TQBF $\in$ PSPACE

- Let $\psi = Q_1 x_1 Q_2 x_2 \dots Q_n x_n \varphi(x_1, x_2, \dots, x_n)$ be a QBF with n variables; and $|\varphi| = m$. We design a recursive algorithm A to decide the truth of ψ.
- **Case I** ($n = 0$)**:** ψ contains only constants (TRUE and/or FALSE) and can be evaluated in $O(m)$ time and space.

Proof of TQBF $\in$ PSPACE

The Proof Continued ...

- **Case II** $(n > 0)$**:** For $b \in \{0, 1\}$, denote by $\psi_{|(x_1=b)}$, a modification of ψ where Q_1 is dropped and all occurrences of x_1 are replaced with the constant b.

Proof of TQBF $\in$ PSPACE

The Proof Continued ...

- **Case II** ($n > 0$)**:** For $b \in \{0, 1\}$, denote by $\psi_{|(x_1=b)}$, a modification of ψ where Q_1 is dropped and all occurrences of x_1 are replaced with the constant b.
- A works as follows:

Proof of TQBF $\in$ PSPACE

The Proof Continued ...

- **Case II** $(n > 0)$**:** For $b \in \{0,1\}$, denote by $\psi_{|(x_1=b)}$, a modification of ψ where Q_1 is dropped and all occurrences of x_1 are replaced with the constant b.
- A works as follows:
  - If $Q_1 = \exists$, then output 1 iff at least one of $A(\psi_{|(x_1=0)})$ and $A(\psi_{|(x_1=1)})$ returns 1.

Proof of TQBF $\in$ PSPACE

The Proof Continued ...

- **Case II** $(n > 0)$**:** For $b \in \{0, 1\}$, denote by $\psi_{|(x_1=b)}$, a modification of ψ where Q_1 is dropped and all occurrences of x_1 are replaced with the constant b.
- A works as follows:
  - If $Q_1 = \exists$, then output 1 iff at least one of $A(\psi_{|(x_1=0)})$ and $A(\psi_{|(x_1=1)})$ returns 1.
  - If $Q_1 = \forall$, then output 1 iff both $A(\psi_{|(x_1=0)})$ and $A(\psi_{|(x_1=1)})$ returns 1.

Proof of TQBF $\in$ PSPACE

The Proof Continued ...

- **Case II** $(n > 0)$**:** For $b \in \{0, 1\}$, denote by $\psi_{|(x_1=b)}$, a modification of ψ where Q_1 is dropped and all occurrences of x_1 are replaced with the constant b.
- A works as follows:
  - If $Q_1 = \exists$, then output 1 iff at least one of $A(\psi_{|(x_1=0)})$ and $A(\psi_{|(x_1=1)})$ returns 1.
  - If $Q_1 = \forall$, then output 1 iff both $A(\psi_{|(x_1=0)})$ and $A(\psi_{|(x_1=1)})$ returns 1.
  - A returns the correct answer as it just works on the definition of $\exists$ and $\forall$.

Proof of TQBF $\in$ PSPACE

The Proof Continued ...

- **Case II** $(n > 0)$**:** For $b \in \{0, 1\}$, denote by $\psi_{|(x_1=b)}$, a modification of ψ where Q_1 is dropped and all occurrences of x_1 are replaced with the constant b.
- A works as follows:
  - If $Q_1 = \exists$, then output 1 iff at least one of $A(\psi_{|(x_1=0)})$ and $A(\psi_{|(x_1=1)})$ returns 1.
  - If $Q_1 = \forall$, then output 1 iff both $A(\psi_{|(x_1=0)})$ and $A(\psi_{|(x_1=1)})$ returns 1.
  - A returns the correct answer as it just works on the definition of $\exists$ and $\forall$.
- As A is recursive, we need to find a recurrence for the space $S(n, m)$ used by A on ψ.

Proof of TQBF $\in$ PSPACE

The Proof Continued ...

- We now use the fact that space can be reused; i.e. $A(\psi_{|(x_1=0)})$ and $A(\psi_{|(x_1=1)})$ can use the same space.

Proof of TQBF $\in$ PSPACE

The Proof Continued ...

- We now use the fact that space can be reused; i.e. $A(\psi_{|(x_1=0)})$ and $A(\psi_{|(x_1=1)})$ can use the same space.
- After computing $A(\psi_{|(x_1=0)})$, A needs to retain the single bit output for operating with the output of $A(\psi_{|(x_1=1)})$ which can now reuse the rest of the space.

Proof of TQBF $\in$ PSPACE

The Proof Continued ...

- We now use the fact that space can be reused; i.e. $A(\psi_{|(x_1=0)})$ and $A(\psi_{|(x_1=1)})$ can use the same space.
- After computing $A(\psi_{|(x_1=0)})$, A needs to retain the single bit output for operating with the output of $A(\psi_{|(x_1=1)})$ which can now reuse the rest of the space.
- Assuming that A uses $O(m)$ space to write $\psi_{|x_1=b}$ for its recursive calls, we get $S(n,m) = S(n-1,m) + O(m)$, which solves to $S(n,m) = O(nm)$.