Space Complexity

Frank

April 11, 2022

1 Space Complexity

1.1 Definitions

Def: Space complexity of (D/N) TM is a function $S : \mathbb{N} \to \mathbb{N}$ s.t. $S(n) =$ max number of cells scanned by TM in all computation paths on all inputs of size n (In the case of NTM, we count the maximum of any computation path, not all of them).

Def: $SPACE(S(n)) = \{L : \text{L is decided by a DTM in } O(S(n)) \text{ space}\}$

Def: $NSPACE(S(n)) = \{L : \text{L is decided by a NTM in } O(S(n)) \text{ space}\}$

1.2 What is space complexity?

Consider SAT, it is known that SAT $\in SPACE(n)$

Consider a SAT-Decider,

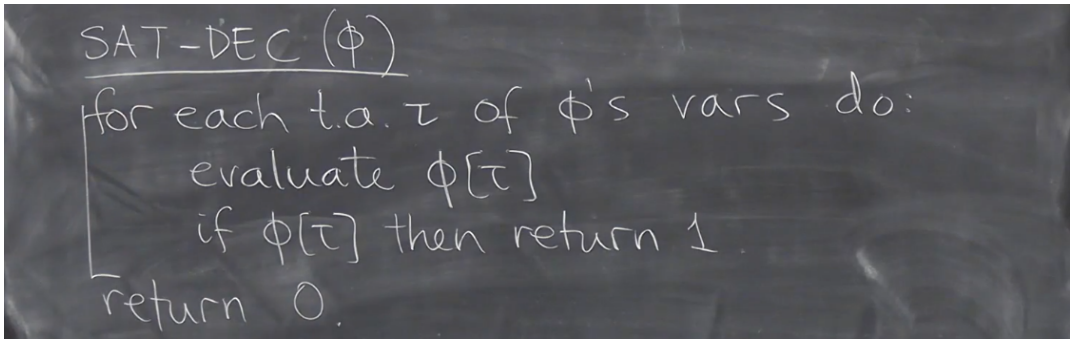


Figure 1: k1

This is obviously not a polytime algorithm, but this is a linear space complexity algorithm.

Space has an important property of being reusable that time does not have.

Note here we have 2 things we must have in memory, the formula, and the truth assignment. We can reuse the space for the truth assignment by simply modifying the encoding at each iteration.

For example of NSPACE, look at Ex 8.4 in Sipser.

1.3 Big Result

Definitions:

$PSPACE = \bigcup_{k \geq 0} SPACE(n^k)$

$NPSPACE = \bigcup_{k \geq 0} NSPACE(n^k)$

Thm: $PSPACE = NPSPACE$

Preliminaries:

Fix TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$

M's space complexity $S(n) \geq n$ (at least as big as its input, so we can store it)

Input x, $|x| = n$

Number of configs of M on $x = |Q| \cdot |\Gamma|^{S(n)} \cdot S(n) \leq 2^{c \cdot S(n)}$, where c is dependent on the TM

remember that a config is a string of the form xqy

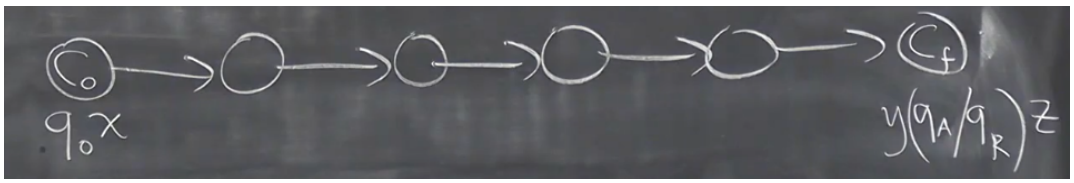
$\implies$ each config can be described using $\leq c \cdot S(n)$ bits,

remembering that we can represent a number x using $\log x$ bits

Def: Config graph of M on x, $G_{M,x} = (V, E)$

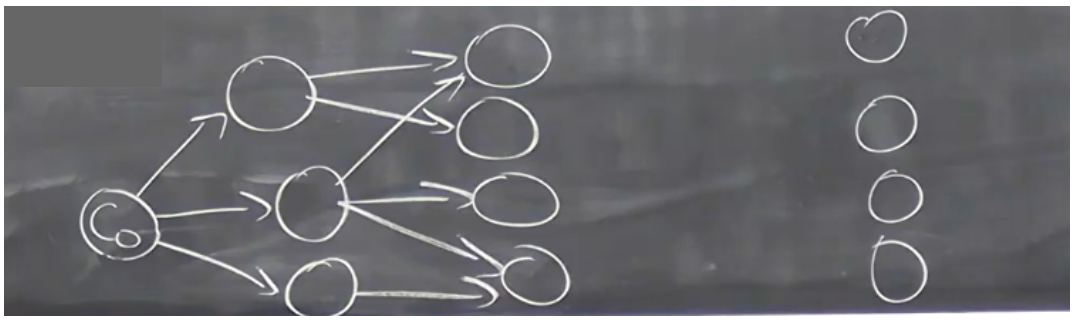
* $V = \{C : \ C \text{ is config of } M \text{ on } x\}$
* $E = \{(C, C') : C \vdash_M C'\}$

Now for DTM M, this graph looks like:



With the length bounded by $2^{c \cdot S(n)}$

For the NTM M, this graph is a DAG:



With the length bounded by $2^{c \cdot S(n)}$ as well.

WLOG, assume single accepting config, namely q_A (when we reach the accepting state, we can simply clear the tape without affecting the asymptotic time or space complexity)

Now the question transforms into:

$x \in L(M) \iff \exists q_0 x \to q_A$ path in $G_{M,x}$ of length $\leq 2^{c \cdot S(|x|)}$

Thm 12.1: If $S(n) \geq n$ and $S(n)$ is space-constructible then $NSPACE(S(n)) \subseteq SPACE(S^2(n))$

$S(n)$ is space-constructible if $\exists$ TM that given O^n as input it computes $O^{S(n)}$ using $S(n)$ space.

$n^k, n^k \log n, 2^n$ are all space constructible

Cor 12.2: PSPACE = NPSPACE

Space Management in programming languages when we have procedural calls:

P calls Q

$\to$ push record (state of P when it calls Q) to call stack

Q finishes

$\to$ pop record from call stack and resume P's computation

($\star$) Then space = (height of call stack) $\times$ (size of record)

Proof:

Note that we cannot simulate the NTM using a BFS or a DFS, which would require exponential memory.

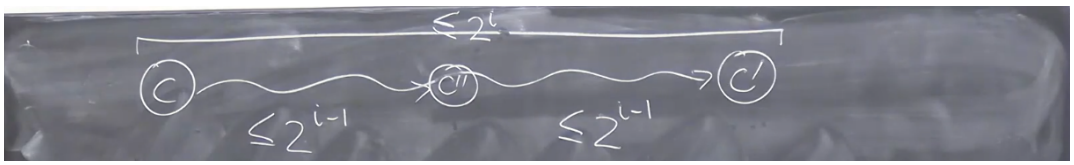
Instead, we want to ask if $\exists C \to C'$ path of length $\leq 2^i$

Define

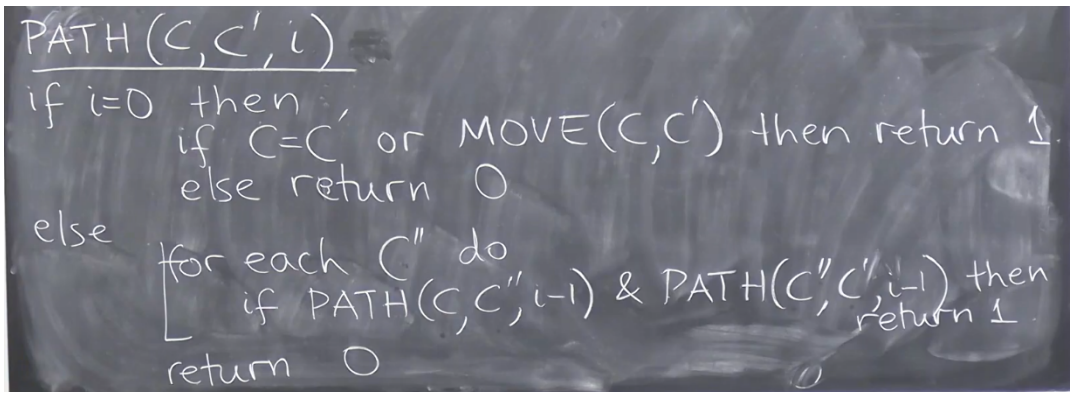
$$MOVE(C, C') = \begin{cases} 1 & \text{if } C \vdash_M C' \\ 0 & \text{otherwise} \end{cases}$$

This algorithm takes $O(S(n))$ space, since we are merely scanning the 2 configurations and making sure that it is a valid transition. We do not need extra space for that.

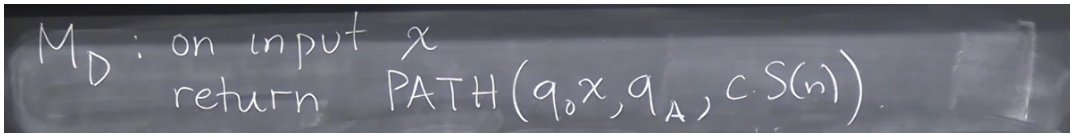
Note now that to determine the above question, we can divide and conquer by splitting the path into 2 piece, which now has half the length.



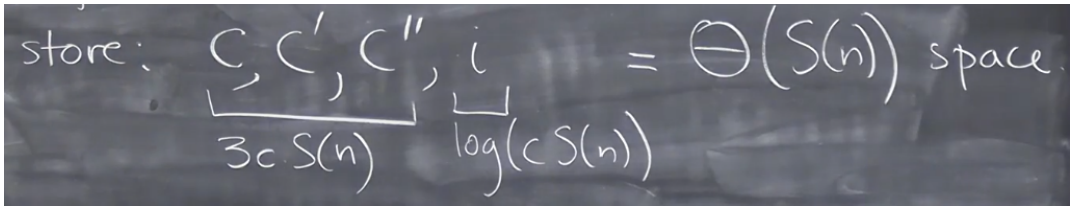
Now we define the algorithm PATH:



Then M_D that simulates the NTM is



Then the size of our call stack record is



And the maximum height of the call stack is $c \cdot S(n)$, since we start at $c \cdot S(n)$ and reduce by 1 every call

$\implies$ space needed for M_D is $O(S^2(n))$

2 Relationship between Space and Time

So far, we have

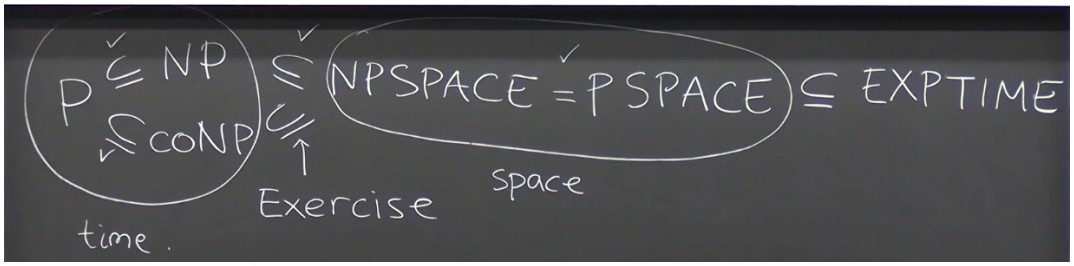


Figure 2: u1

As an exercise, prove that $coNP \subseteq NPSPACE$.

We will prove that $PSPACE \subseteq EXPTIME$

Note at somewhere, we need a proper containment, since we have proved early on that theres a language $EXP \in EXPTIME$ but $EXP \notin P$

Thus, all of these cannot be equal

2.1 PSPACE contained in EXPTIME

Def: $EXPTIME = \bigcup_{k \geq 0} TIME(2^{n^k})$

Thm 12.3: If $L \subseteq \Sigma^\star$ is decided by a TM M in $S(n)$ space, then it is decided by a DTM M_D in $O(2^{d \cdot s(n)})$ time, where d is a const. dependent on M

Proof:

Number of configs of $M \leq 2^{c \cdot S(n)}$

$G_{M,x}$ = config graph of M on x

Thus, $x \in L \iff \exists q_0 x \to q_A$ in $G_{M,x}$?

Note that number of nodes in $G_{M,x} \leq 2^{c \cdot S(n)}$ and number of edges in $G_{M,x} = O(2^{c \cdot S(n)})$

To see if there exists such path, we can simply DFS or BFS, which is linear in number of nodes when we use random access machine, and thus quadratic in TM.

$\implies$ can determine if $G_{M,x}$ has $q_0 x \to q_A$ path in $O(2^{2c \cdot S(n)})$, which is still exponential time

Cor 12.4: $PSPACE \subseteq EXPTIME$

3 PSPACE-Completeness

We want a class of the hardest problems in the language,

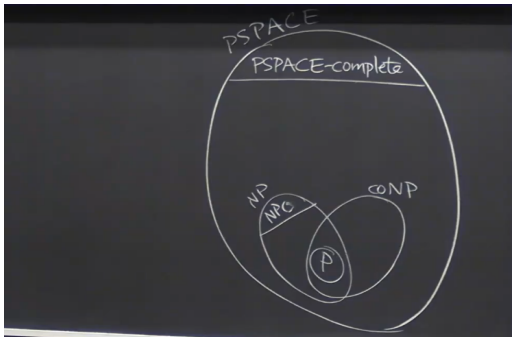


Figure 3: u2

Def: L is PSPACE-Complete if:

1. $L \in PSPACE$
2. $\forall L' \in PSPACE, L' \leq_m^P L$

Note that we are still using polytime reduction.

> We do this because we want the possibility of associating PSPACE with P, and if we do polyspace reduction for a language to PSPACECOMP, then the reduction could take exponential time, instead of polytime, thus we want to entertain the possibility of $PSPACE = PTIME$ by using polytime reduction.

Now lets find the "bootstrap" for PSPACECOMP

3.1 Quantified Boolean Formulas (QBF)

Consider $(x \vee \bar{y}) \wedge (\bar{x} \vee y)$ (boolean formula)

$\forall x, \exists y, (x \vee \bar{y}) \wedge (\bar{x} \vee y)$ can be satisfied by simply choosing y to be true when x is true and vice versa. (true QBF)

$\exists x, \forall y, (x \vee \bar{y}) \wedge (\bar{x} \vee y)$ cannot be satisfied, since we cannot choose boolean values that can satisfy the formula (false QBF)

Fully quantified BF: formula with no free variables

> either true or false

3.2 True-Fully-Quantified-Boolean-Formulas

DEF: $TQBF = \{\langle \phi \rangle : \phi \text{ is a true fully QBF}\}$

Thm 12.5: TQBF is PSPACECOMP

Proof:

1. $TQBF \in PSPACE$
2. $\forall L \in PSPACE, L \leq_m^P TQBF$

Part1:

Given $\phi = Q_1 x_1 Q_2 x_2 \ldots Q_k x_k \psi$ where ψ is quantifier-free

Create a tree as follows:

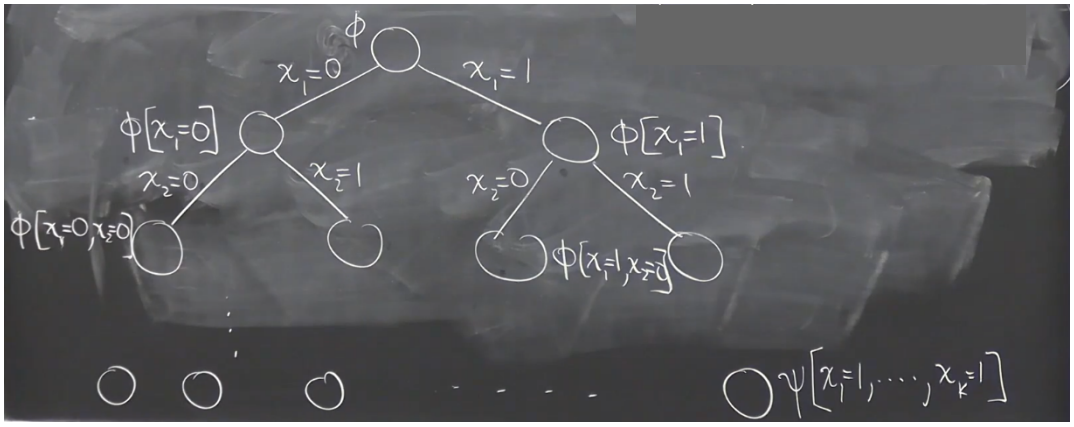


Figure 4: u3

Where we are able to evaluate ψ in the end (fully quantified - so each variable after a quantifier is in ψ). This in turn will allow us to evaluate each node using the following technique.

* If the node is associated with $\forall$, then we need all child nodes to be true.
* If the node is associated with $\exists$, then we only need 1 child node to be true.

This recursive algorithm will allow us to evaluate the entire formula.

Algorithm:

1. Traverse Tree in post-order (left right root)
2. At leaf: evaluate $\psi[x_1 = .., x_2 = .., \ldots, x_k = ..]$
3. At $\forall$ node: if both children are 1, then 1, else 0
4. At $\exists$ node: if ≥ 1 child is 1 then 1, else 0

Then $\phi \in TQBF \iff$ root is 1

Space analysis:

* max ht of call stack $= k \leq |\phi|$
* Each record must store
  - formula $\phi[..]$ corresponding to node
  - 1 bit

  Space is $O(|\phi|^2)$

Part2:

Let $L \in PSPACE$, M be DTM that decides L in space $p(n) \leq n^k$

Given $x \in \Sigma^\star$, construct ϕ QBF s.t. $x \in L \iff M$ accepts $x \iff \phi$ is a true QBF

We want the formula ϕ to say: "M on input x has an accepting computation"

Note that M on x has $\leq 2^{c \cdot p(n)}$ configurations

Each config C is represented by $c \cdot p(n)$ bits (take the logarithm), where each bit is a propositional variable

Specially, for C we have variables:

* S_C^q = in C the state of M is q, $\forall q \in Q$
* H_C^i = in C, the tape head is on i'th cell, $1 \leq i \leq p(n)$
* T_C^{ia} = in C, the i'th cell contains a, $1 \leq i \leq p(n)$, $a \in \Gamma$

Since Q and Γ has constant size, we have $O(p(n))$ variables for each config

We also have formulas:

* Coherent(C):
  - no two states in C
  - head not in two places
  - each cell not two symbols

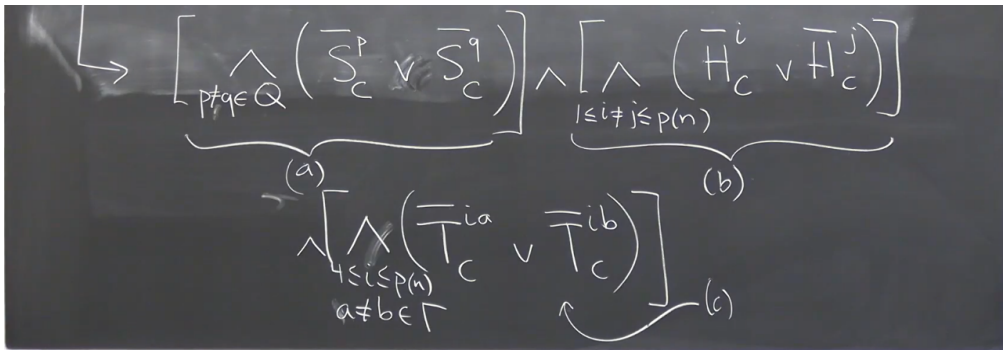


Figure 5: u4

With size $\leq O(p^2(n))$

Use abbreviation $\exists C$ and $\forall C$ to represent a long chain of quantified formulas

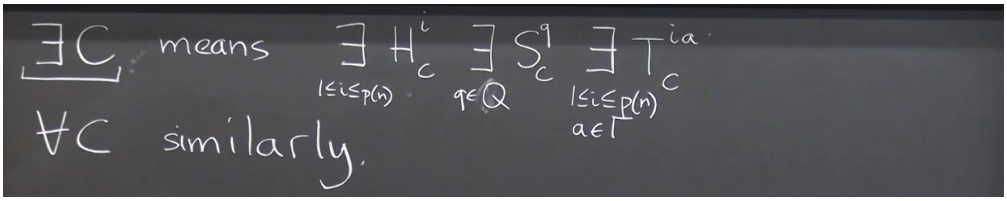


Figure 6: u5

Then the overall form of ϕ is:

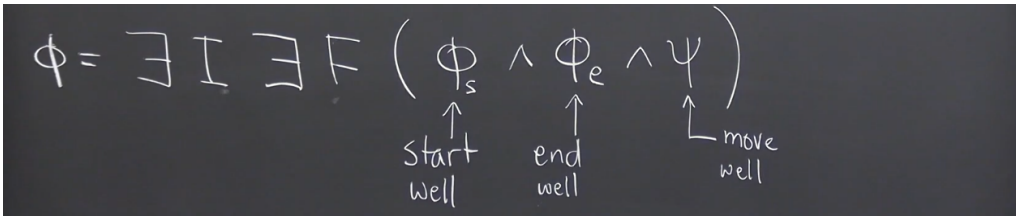


Figure 7: u6

START WELL:

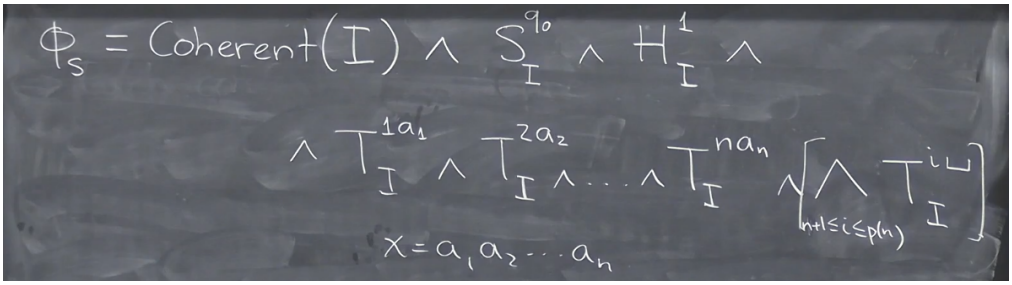


Figure 8: u7

Where I is initial config, F is finishing config

ENDS WELL: ..

MOVE WELL:

Declare a quantified formula $\psi_i(C, C')$ with 2 free vars

$\psi_i(C, C') = C \vdash_M^\star C'$ in $\leq 2^i$ moves

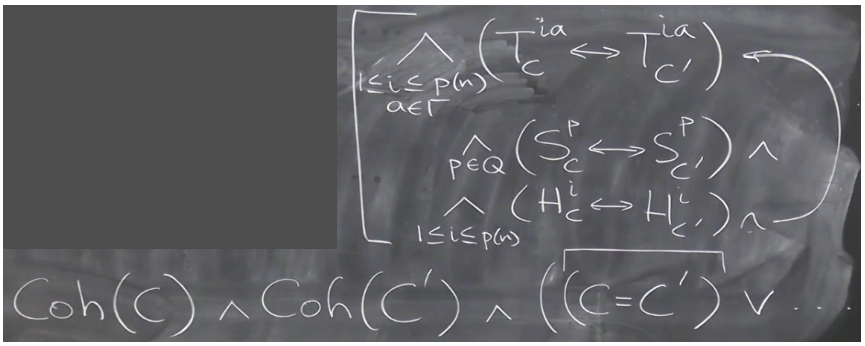
Inductively on i, $0 \leq i \leq c \cdot p(n)$ (upper bound, since if not, the TM will repeat config and therefore loop)

$\psi = \psi_{c \cdot p(n)}(I, F)$

Induction:

- Base: $i = 0$

  $\psi_0(C, C') =$



Which is the part if we do not change config

And if we do change configs (move 1 step), here is the formula:

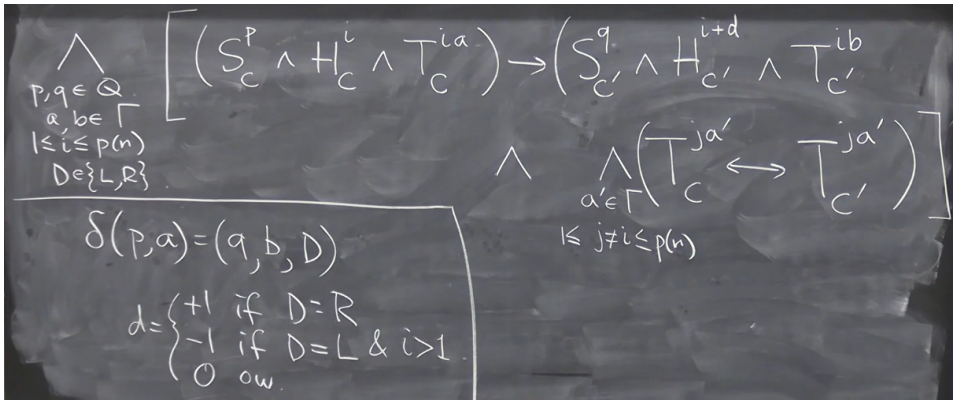


Figure 9: u9

Where $|\psi_0| = O(p(n))$

- Induction Step: $i > 0$. Define ψ_i based on ψ_{i-1}

  Here we know if that C goes to C' in some moves, it can go to a middle point in half the number of moves.

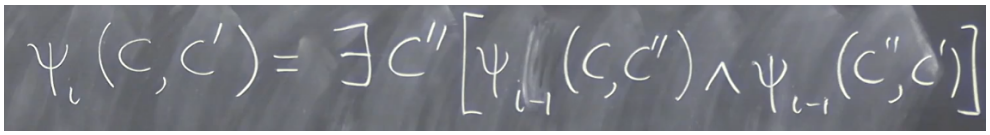


Figure 10: u10

However, the problem with this formula is that on every iteration of i, the formula doubles in size. We will end up with a formula that is $2^{c \cdot p(n)}$ in size. So we need to write this formula with one copy of the sub-formula

Here is a way to use one copy to write the formula:

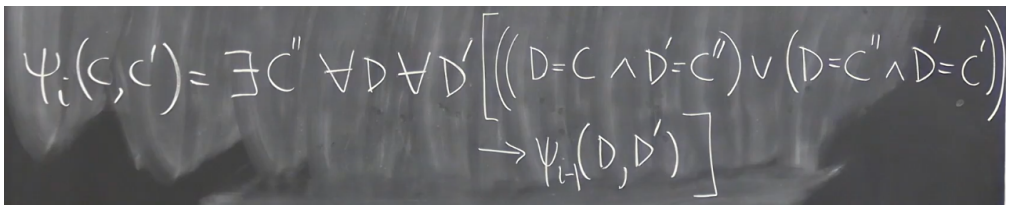


Figure 11: u11

So now we have $|\psi_i| = |\psi_{i-1}| + O(p(n))$

$\implies |\psi| = |\psi_{c \cdot p(n)}| = O(p^2(n))$

So $|\phi| = O(p^2(n))$

Thus, ϕ is true QBF $\iff x \in L$

And we have the PSPACECOMP of $TQBF$

3.3 Quantifiers and Two-person Games

Player 1 has a winning strategy:

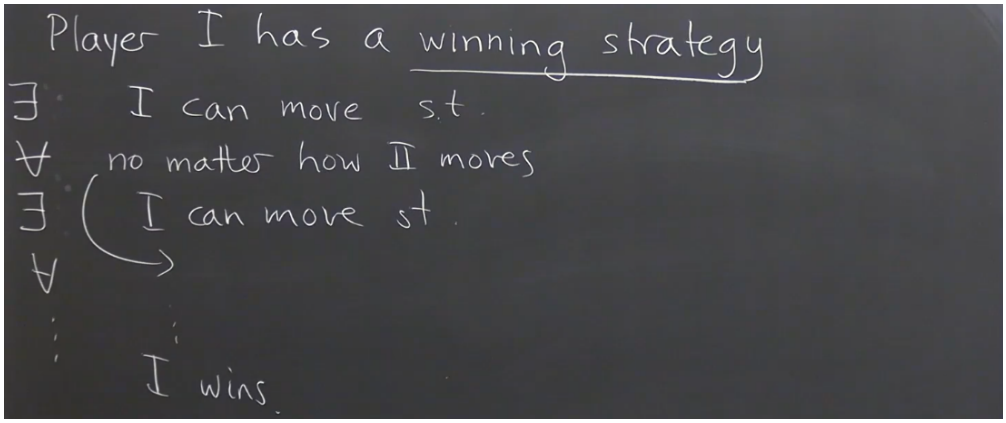


Figure 12: u12

We can represent a game like this as a tree:

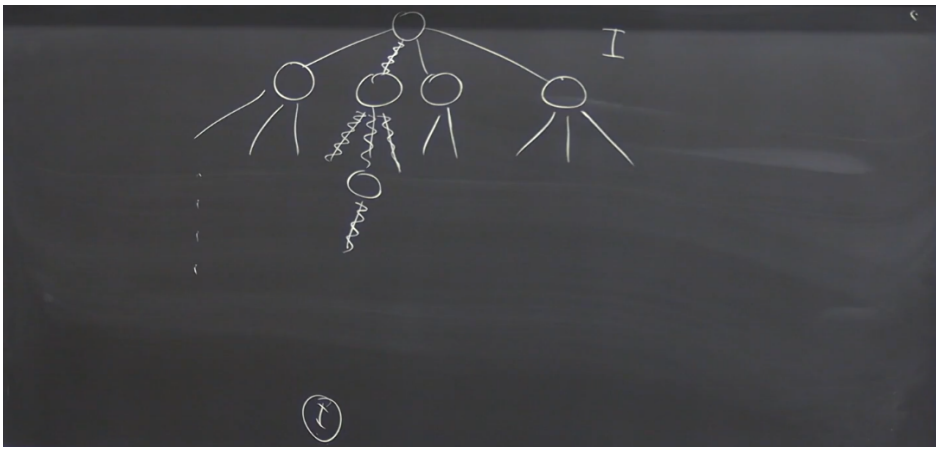


Figure 13: u13

Where the node is the starting config, then player 1 makes a move to end up with different configs where player 2 can make a move.

Then player 1 has a winning strategy if for that config, for all moves player 2 makes, there exists a move player 1 can make such that for all moves player 2 makes, ..., player 1 will win.

A QBF is proper if:

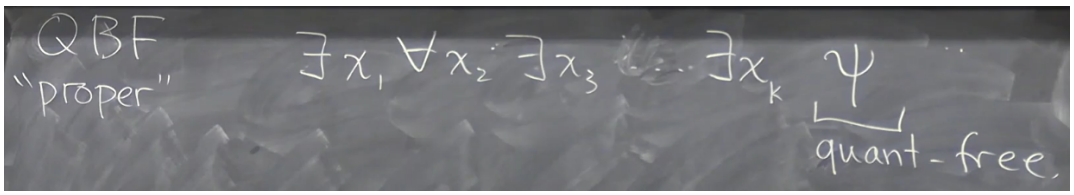


Figure 14: u14

Then we can proceed to view this as a game between 2 players, where for example:

- player 1 chooses $x_1 = 0$
- player 2 chooses $x_2 = 1$
- ...

And player 1 wins if the choice of values makes ψ true, and player 2 wins if the choice of values makes ψ false. (no draws)

Then the problem is does player 1 have a winning strategy (as defined above)?

FORMGAME = $\{\langle\phi\rangle : \phi$ is a proper QBF for which player 1 has a winning strategy$\}$

Thm 12.6: FORMGAME is PSPACECOMP

Proof: TQBF=FORMGAME

Thm 8.14: sipser, take a look.