Space Complexity

Computability and Complexity
Andrei Bulatov

Measuring Space

So far we have defined complexity measures based on the time used by the algorithm

Now we shall do a similar analysis based on the space used

In the Turing Machine model of computation this is easy to measure — just count the number of tape cells read during the computation

Definition 1 The space complexity of a Turing Machine T is the function Space_T such that $\mathrm{Space}_T(x)$ is the number of distinct tape cells visited during the computation $T(x)$

(Note that if $T(x)$ does not halt, then $\mathrm{Space}_T(x)$ is undefined.)

Palindromes

Consider the following 3-tape machine deciding if an input string is a palindrome

- The first tape contains the input string and is never overwritten
- The machine works in stages. On stage i it does the following
  - the second tape contains the binary representation of i
  - the machine finds and remembers the i-th symbol of the string by (1) initializing the 3rd string $j = 1$; (2) if $j < i$ then increment j; (3) if $j = i$ then remember the current symbol by switching to correspondent state
  - in a similar way the machine finds the i-th symbol from the end of the string
  - if the symbols are unequal, halt with "no"
- If the i-th symbol is the blank symbol, halt with "yes"

Another Definition

By Def.1, the space complexity of this machine is n

However, it seems to be fair to define its space complexity as $\log n$

Definition 2 Let T be a Turing Machine with 2 tapes such that the first tape contains the input and never overwritten. The space complexity of T is the function Space_T such that $\mathrm{Space}_T(x)$ is the number of distinct cells on the second tape visited during the computation $T(x)$

Blum's Axioms

Blum proposed that any useful dynamic complexity measure should satisfy the following properties:

- $\varphi_M(x)$ is defined exactly when $M(x)$ is defined
- The problem: for given M, x, r, does $\varphi_M(x) = r$ is decidable

Theorem Space_T is a proper complexity measure

Proof

1. is obvious.

2.

Note that if the machine has k states and l symbols in the alphabet, then there are $k \cdot r \cdot l^r$ possible configuration involving at most r visited cells on the second tape

If the same configuration occurs twice during the computation $M(x)$, then the machine goes into an infinite loop and $M(x)$ (as well as $\mathrm{Space}_M(x)$) is not defined

Therefore after performing $k \cdot r \cdot l^r$ steps $M(x)$ either
halts, or
visits more than r cells, or
goes into an infinite loop

Space Complexity of Problems

As with time complexity, we cannot define an exact space complexity for a language, but we can give an asymptotic form …

Definition

For any function f, we say that the space complexity of a decidable language L is in $O(f)$ if there exists a Turing Machine T which decides L, and constants n_0 and c such that for all inputs x with $|x| \geq n_0$

$$\text{Space}_T(x) \leq cf(|x|)$$

Space Complexity Classes

Now we are in a position to divide up the decidable languages into classes, according to their space complexity

Definition

The space complexity class SPACE[f] is defined to be the class of all languages with time complexity in $O(f)$

(Note, it is sometimes called DSPACE[f] — for Deterministic Space)

Polynomial Space

As with time, we can obtain a robust space complexity class by considering all polynomial space complexity classes

Definition

$$\boldsymbol{PSPACE} = \bigcup_{k \geq 0} \text{SPACE}[n^k]$$

Time vs. Space

- Visiting a cell takes at least one time step, so we have

Theorem

$$\text{TIME}[f] \subseteq \text{SPACE}[f]$$

- A more subtle analysis, based on the fact that there are only $|Q| \cdot f(n) \cdot |\Sigma|^{f(n)}$ different possible configurations with $f(n)$ non-blank tape cells, gives

Theorem

$$\text{SPACE}[f] \subseteq \text{TIME}[f \cdot k^f]$$

Reusing Space

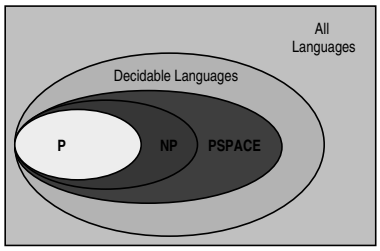
The previous result suggests that a given amount of space is more useful than a given amount of time

This is because space can be reused

For example, Satisfiability can be solved in linear space, by trying each possible assignment, one at a time, reusing the same space

Theorem

$$\boldsymbol{NP} \subseteq \boldsymbol{PSPACE}$$



Problems in PSPACE

Quantified Boolean Formula

Instance: A quantified Boolean formula
$\Phi = (Q_1 X_1)(Q_2 X_2)\ldots(Q_n X_n)\ \ B(X_1, X_2, \ldots, X_n)$
where each X_i is a Boolean variable, $B(X_1, X_2, \ldots, X_n)$ is a Boolean expression involving $X_1, X_2, \ldots, X_n$ and each Q_i is a quantifier ($\exists$ or $\forall$).

Question: Is Φ logically valid ?

Generalized Geography

Instance: A game for 2 players consisting of a directed graph G, and a start vertex s. Players take turns to move along the edges from the current vertex. A player who cannot move without returning to a vertex already visited loses

Question: Is there a winning strategy for the first player?

PSPACE-Completeness

Definition
A language L is said to be **PSPACE**-complete if, for any
$A \in$ **PSPACE**, $A \leq L$

PSPACE-Complete Problem

Quantified Boolean Formula

Instance: A quantified Boolean formula
$\Phi = (Q_1 X_1)(Q_2 X_2)\ldots(Q_n X_n)\ \ B(X_1, X_2, \ldots, X_n)$
where each X_i is a Boolean variable, $B(X_1, X_2, \ldots, X_n)$ is a Boolean expression involving $X_1, X_2, \ldots, X_n$ and each Q_i is a quantifier ($\exists$ or $\forall$).

Question: Is Φ logically valid ?

Theorem
QBF is PSPACE-complete

Proof

First we show that QBF is in **PSPACE,** using the following recursive algorithm

For the input Boolean sentence Φ

- If Φ contains no quantifiers, then it contains no variables, only constants. Evaluate this expression; accept if it is **true**; otherwise reject
- If $\Phi = \exists X\ \Psi$ then Φ is **true** if and only if $\Phi|_{X=0} \vee \Phi|_{X=1}$ is **true**. Recursively call the algorithm on $\Phi|_{X=0}$ and $\Phi|_{X=1}$, and accept if one of them is accepted; otherwise reject
- If $\Phi = \forall X\ \Psi$ then Φ is **true** if and only if $\Phi|_{X=0} \wedge \Phi|_{X=1}$ is **true**. Recursively call the algorithm on $\Phi|_{X=0}$ and $\Phi|_{X=1}$, and accept if both of them is accepted; otherwise reject

This algorithm is polynomial space, because

- on every round of recursion it needs to store only a transformed copy of the input formula
- the depth of recursion is the number of variables, m
- Therefore, it uses $m \cdot |\Phi|$ space

- Let A be a language in **PSPACE**
- Let T be a deterministic Turing Machine that decides A with space complexity $f(n)$, where f is a polynomial
- Choose an encoding for the computation $T(x)$ that uses $k \cdot f(|x|)$ symbols for each configuration
- Let C_0 be the initial configuration, and C_a be the accepting configuration

When proving Cook's theorem we managed to define a Boolean CNF formula $\Phi_T(\overline{X})$ such that $\Phi_T(\overline{X})$ is true if and only if the values of variables $\overline{X}$ describe in a certain way a legal configuration of the Turing machine T

Then we constructed a formula $\Psi_T(\overline{X},\overline{Y})$ which is true if and only if $\overline{X}$ and $\overline{Y}$ represent legal configurations, say C_1 and C_2, and T can go from C_1 to C_2 in at most one step

The length of both formulas is polynomial in the amount of tape cells involved in the computation. Denote the length of Ψ_T by $g(|x|)$

We construct a formula $\Omega(\overline{X},\overline{Y},t)$ which is true if and only if $\overline{X}$ and $\overline{Y}$ represent legal configurations, say C_1 and C_2, and T can go from C_1 to C_2 in at most 2^t steps

Finally, T accepts x if and only if $\Omega(C_0, C_a, f(|x|))$ is true

The formula is built recursively

Straightforward approach:

$$\Omega(\overline{X},\overline{Y},0) = \Psi_T(\overline{X},\overline{Y})$$
$$\Omega(\overline{X},\overline{Y},t) = \exists \overline{Z}\ (\Omega(\overline{X},\overline{Z},t-1) \wedge \Omega(\overline{Z},\overline{Y},t-1))$$

It does not work because $\Omega(\overline{X},\overline{Y}, f(|x|))$ is build of $2^{f(|x|)}$ formulas $\Psi_T(\overline{X},\overline{Y})$

There are universal quantifiers to do the job!

$$\Omega(\overline{X},\overline{Y},t) = \exists \overline{Z}\ \forall(\overline{U},\overline{V})$$
$$((((\overline{U},\overline{V}) = (\overline{X},\overline{Z})) \vee ((\overline{U},\overline{V}) = (\overline{Z},\overline{Y}))) \wedge \Omega(\overline{U},\overline{V},t-1))$$

The length of this formula is $O(f(|x|)g(|x|))$