CMPUT 675: Computational Complexity Theory	**Winter 2019**
Lecture 6 (Jan 24): Space Complexity	
Lecturer: Zachary Friggstad	*Scribe: Laura Petrich*

6.1 Space-Bounded Computation

Space-bounded computation can be thought of as the memory requirements of computational tasks. Recall that only cells in read-write work tapes visited by the tape head (not the input tapes) count towards the space bound. In contrast with time-bounded computation, we are interested in computations that run in sublinear space, i.e, with a workspace smaller than the input length. We can define this as follows:

Definition 1 (Space-bounded computation) *For any function $S : \mathbb{N} \to \mathbb{N}$ and language $L \subseteq \{0,1\}^*$. We say that $L \in \mathbf{DSPACE}(S(n))$ if there is a* TM *M deciding L that uses at most $c \cdot S(n)$ nonblank tape locations on inputs of length n. For a non-deterministic Turing Machine, $L \in \mathbf{NSPACE}(S(n))$ can be defined similarly.*

Previously it was discussed that for any function $f(n) \in \Omega(\log n)$ (space-constructable function):

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

The first inclusion is clearly because the heads of a TM that uses time $O(S(n))$ can only visit cells that are $O(S(n))$ steps away from the initial cell. The second is for the usual reason that determinism is a special case of nondeterminism. The last is a bit more subtle and will be discussed momentarily.

6.2 Configuration Graph

A snapshot C of a TM with using space $O(f(n))$ when given an input with length n can be described with $O(f(n) + \log n)$ bits: $O(1)$ to describe the state, $O(f(n))$ to describe all contents of the work tapes, $O(\log f(n))$ to describe the positions of the heads on the work tapes, and $O(\log n)$ to describe the position of the head in the input tape. Our entire space complexity discussion concerns functions $f(n) \geq \log_2 n$, so we simply say that a configuration C can be described with at most $c \cdot f(n)$ bits where c is a constant depending only on the machine M (not on the input to the machine). That is, a snapshot C for a computation with input x is some $\{0,1\}^{c \cdot f(|x|)}$.

We define the **configuration graph** of M on an input x, denoted $G^{M,x}$, as a directed graph with nodes corresponding to valid snapshots of M on x. Each node in the configuration graph represents a snapshot C' that is accessible from snapshot C in a single step determined by M's transition function. It follows that each node C will transition to at most one other node if M is deterministic, and transition to at most two other nodes if M is non-deterministic. Note, with input length $|x|$, $G^{M,x}$ can have at most $2^{c \cdot f(|x|)}$ nodes. Whether or not snapshots C and C' share an edge can be represented by an $O(f(|x|))$-space constructible CNF instance $\varphi_{M,x}$ such that $\varphi(C, C') = \mathit{true}$ if and only if C and C' encode valid snapshots and $C \to C'$ is a transition. Notice we also include $C \in \{0,1\}^{c \cdot f(|x|)}$ in the vertices of $G^{M,x}$ that do not correspond to valid snapshots, such C will not have any incident edges.

Recall we let C_{start} denote the initial snapshot of M's computation on input x (M and x will always be clear from the context). We will assume there is only one accepting snapshot called C_{accept}: this can be done by having the TM clear all of its work tapes and resetting its heads to the initial positions before entering its accepting state.

Since M accepts an input x if and only if a path from C_{start} to C_{accept} exists, this means we are able to build the graph in $2^{c \cdot f(|x|)}$-time and check for existence using a breadth-first search. Notationally, given M and x:

$$G^{M,x} = (\{0,1\}^{c \cdot f(|x|)}, E)$$
$$(C, C') \in E \text{ iff } \varphi(C, C') = true.$$

A breadth-first search can determine, in time that is linear in the size of $G^{M,x}$, if there is a path from C_{start} to C_{accept}. This shows $\mathbf{NSPACE}(f(n)) \subseteq \mathbf{DTIME}(2^{O(f(n))})$.

6.3 PSPACE Completeness

We do not know whether $\mathbf{P} = \mathbf{PSPACE}$, although it is strongly believed not to be the case, since we know that $\mathbf{NP} \subseteq \mathbf{PSPACE}$. Thus, if it were true that $\mathbf{P} = \mathbf{PSPACE}$, it would imply that $\mathbf{P} = \mathbf{NP}$.

Definition 2 *A language L' is* **PSPACE**-*hard if for all $L \in$* **PSPACE**, *$L \leq_p L'$.*

Definition 3 *A language L' is* **PSPACE**-*complete if L' is* **PSPACE**-*hard and $L' \in$* **PSPACE**.

Definition 4 *A* ***quantified Boolean formula*** (QBF) *is a formula of the form $Q_1 x_1 Q_2 x_2 ... Q_n x_n \phi(x_1, x_2, ..., x_n)$ where $Q_i \in \{\forall, \exists\}$, $x_1, ... x_n \in \{0, 1\}$, and ϕ is an unquantified Boolean formula. A* QBF *is always either true or false.*

Note that when all quantifiers appear to the far left, this is known as **prenex normal form**. Any formula not in this form can be rewritten in prenex normal form using polynomial time through the use of identities (e.g. $\neg\forall x \phi(x) = \exists x \neg\phi(x)$, and $\psi \vee \exists x \varphi(x) = \exists x \psi \vee \varphi(x)$ where ψ does not contain x). For example, the formula $\forall x(\varphi(x) \vee \exists y \varphi'(x, y))$ can be rewritten as $\forall x \exists y(\varphi(x) \vee \varphi'(x, y))$.

We will now use configuration graphs and quantified Boolean formulae to showcase an interesting **PSPACE**-complete problem. We define the language TQBF to be the set of quantified Boolean formulae that are *true*.

$$\text{TQBF} = \{true\ Q_1 x_1 Q_2 x_2 ... Q_n x_n \phi(x_1, ... x_n) : Q_i \in \{\forall, \exists\} \forall i, \phi \text{ is a CNF formula}\}.$$

For example, $\forall x_1 \exists x_2 \exists x_3 (x_1 \vee x_3) \wedge (\bar{x}_1 \vee x_2) \wedge (x_3 \vee \bar{x}_2) \in$ TQBF. To see this, for $x_1 = true$ setting $x_2 = true$ and $x_3 = true$ suffices. For $x_1 = false$ setting $x_3 = true$ and x_2 to anything suffices.

Theorem 1 (Stockmeyer and Meyer, '73) TQBF *is* **PSPACE**-*complete with respect to polynomial-time Karp reductions.*

Proof. To show TQBF is **PSPACE**-complete with respect to polynomial time karp reductions, we will use a configuration graph to inductively construct a QBF of size $O(f(n)^2)$ that is true if and only if M accepts x.

Intuitively it is obvious that TQBF $\in$ **PSPACE**, so let a language L $\in$ **PSPACE** be decidable by a space-constructible TM M on an input x. First, we consider the configuration graph, $G^{M,x}$, of M on input x. Recall there is a polynomial-time constructible CNF instance ϕ such that there is a directed edge $C \to C'$ between two nodes C and C' in the configuration graph $G^{M,x}$ if and only if $\phi(C, C') = true$. Next, we use the configuration graph to inductively construct QBF instances ψ_i with two unquantified variable inputs C and C'.

For all $i \geq 1$, we construct a TQBF instance $\psi_i(C, C')$ where C, C' are unquantified (free) variables that is true if and only if there is a path in $G_{M,x}$ from C to C' with *length* $\leq 2^i$. This path will only exist if there exists a node C'' that splits the path between C and C' into two paths. These two new paths will clearly have *length* $\leq 2^{i-1}$ from C to C'' and *length* $\leq 2^{i-1}$ from C'' to C' (Figure 6.1). Notationally, we can define this as:

$$\psi_i(C, C') : \exists C'' \psi_{i-1}(C, C'') \wedge \psi_{i-1}(C'', C')$$

Therefore, by setting $\psi_0(C, C') = \phi(C, C')$, we can inductively build up ψ, and check if a path from C_{start} to C_{accept} exists by evaluating $\psi_{c \cdot f(x)}(C_{start}, C_{end})$. It's easy to see how this can lead to exponential blow up in space, thus, for $i \geq 1$, we can consider a different approach to forming $\psi_i(C, C')$ using additional quantified variables, D_1 and D_2, as follows:

$$\psi_i(C, C') : \exists C'' \forall D_1 D_2 [(C = D_1 \wedge C'' = D_2) \vee (C'' = D_1 \wedge C' = D_2) \Rightarrow \psi_{i-1}(D_1, D_2)]$$

Note that ψ_{i-1} has additional quantifiers, but we can rewrite ψ_i by moving all of the quantifiers in ψ_{i-1} to appear just after $\forall D_1 D_2$ using the reduction to prenex normal form mentioned above.

Observe all variables quantified over in the construction of each ψ_i have size $O(f(|x|))$ as they are all snapshots. So, the final form $\psi_{c \cdot f(|x|)}(C_{start}, C_{accept})$ with C_{start} and C_{accept} now "hard-coded" in has size,

$$|\psi_0| = O(f(|x|))$$
$$|\psi_{i+1}| \leq |\psi_i| + O(f(|x|)) = O(i \cdot f(|x|))$$
$$|\psi_{c \cdot f(|x|)}(C_{start}, C_{accept})| \leq O(f(|x|)^2)$$

Figure 6.1: A path from C to C' through snapshot C''.

■

Theorem 2 (Savitch, '70) *For any space-constructible $f(n)$ with $f(n) \geq \log n$,*

$$\textbf{NSPACE}(f(n)) \subseteq \textbf{DSPACE}(f(n)^2)$$

Corollary 1 PSPACE = NPSPACE

Proof. To prove this we will use concepts from the proof of Theorem 1 to define a recursive function that checks whether there is a directed path between snapshots C to C' with *length* $\leq 2^i$.

Let $L \in \textbf{NPSPACE}(f(n))$ be decided by a TM M on an input x. We know that the resulting configuration graph, $G_{M,x}$, has at most $V = 2^{c \cdot f(|x|)}$ nodes. We can define a boolean recursive function, R, that checks whether there is a path from C to C' with *length* $\leq 2^i$. For all $i \geq 1$, $R(C, C', i)$ returns *true* if a path exists, and otherwise returns $false$. As with all recursive algorithms, the amount of space used can be bounded by the depth of the recursion, which is $O(f(|x|))$, multiplied by the amount of space required in each recursive call, which is also $O(f(|x|))$ as the only memory requirements are to store the parameters in $O(f(|x|))$ space, to enumerate the vertices C'' in $O(f(|x|))$ space, and, at the base case $i = 0$, to check if $C \to C'$ is an edge in the configuration graph using $O(f(|x|))$ space.

So $R(C_{start}, C_{accept})$ can be computed deterministically in $O(f(|x|)^2)$ space and is true if and only if there is a sequence of transitions causing M to accept x.

■

6.4 NL Completeness

Definition 5 *A function $f : \{0,1\}^* \to \{0,1\}^*$ is implicitly **log-space computable** if there is a polynomial $p(n)$ such that $|f(x)| \leq p(|x|)\ \forall x$ and $L = \{(x,i) : f(x)_i = 1\} \in \mathbf{L}$.*

Definition 6 *For languages L, L', say $L \leq_\ell L'$ (i.e. L is **log-space reducible** to L') if there exists an implicitly log-space computable f such that $\forall x \in \{0,1\}^*$, $x \in L$ if and only if $f(x) \in L'$.*

Definition 7 *We say that L' is $\mathbf{NL}$-complete if it is in $\mathbf{NL}$ and for every $L \in \mathbf{NL}$, $L \leq_\ell L'$.*

Lemma 1 *$L \leq_\ell L'$ and $L' \leq_\ell L'' \Rightarrow L \leq_\ell L''$*

Proof. If f and g are the reductions, then $g \circ f$ is the reduction from L to L'', because whenever any TM deciding $L^b_{g \circ f}$ needs a bit i from $f(\lambda)$, it decides $L^b_f((x,i))$ in logarithmic space. We can intuitively imagine log-space reductions as having output tapes that can only either write a bit or move to the right, it is not allowed to read bits or move to the left.

■

$$\text{DIRPATH} = \{(G, s, t) : G \text{ is a directed path containing an } s\text{-}t \text{ path}\}$$

Theorem 3 DIRPATH *is* $\mathbf{NL}$*-complete with respect to implicitly log-space reductions.*

Proof. DIRPATH $\in \mathbf{NL}$: If there is a path from s to t, we know it has $\leq n$ because it does not repeat a vertex where n denotes the number of nodes in G. By starting at s, we can non-deterministically choose a step to take and verify, in logarithmic space, if the step follow an edge of G. There are two possible outcomes, either the machine has run for n steps and t has not been found or one of the steps is invalid, or the path ends at t (taking at most valid n steps in the process). In the latter case, the machine accepts the input, otherwise, it rejects. At any step, the work tape need only hold $O(\log n)$ bits of information to store the number of steps already taken, the current node index, and the guess for the next node. Therefore, DIRPATH is in $\mathbf{NL}$.

Now we must show that DIRPATH is $\mathbf{NL}$-complete with respect to implicitly log-space reductions. Let $L \in \mathbf{NL}$ be decidable by a NDTM M in $O(\log n)$ space, we need to show that there is an implicitly log-space computable function f that reduces L to DIRPATH. We can again use a configuration graph, setting $f(x) = G_{M,x}$, where $G^{M,x}$ can have at most $2^{O(\log n)}$ nodes. Since M accepts x if and only if there is a path from C_{start} to C_{accept}, we only need to show that we can compute whether there is a path in $G_{M,x}$ from any snapshot C to snapshot C' in logarithmic-space. This is clearly possible, as any deterministic machine can check if a given C' is a transition from C (which has out-degree at most 2) in space $O(|C| + |C'|) = O(\log |x|)$. ■

References

AB09 S. Arora and B. Barak, Computational Complexity: A Modern Approach, *Cambridge University Press*, 2009, pp. 78-89.