Recall that \(\text{PSPACE} = \bigcup_{k=1}^{\infty} \text{SPACE}(n^k)\). We will see that a relationship between time and space complexity is given by:

\[
P \subseteq \text{NP} \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXPTIME}
\]

We know that \(P \neq \text{EXPTIME}\) because there are languages that can be decided in exponential but not polynomial time. Thus, at least one of the containments in the above expression must be a strict containment. Many computer scientists believe that each of the containments is strict, but there is not yet proof either way.

**Topics Covered**

1. Proof of Time and Space Containments
2. Languages in \(\text{PSPACE}\)
3. \(\text{PSPACE}\)-Completeness

## 1 Proof of Time and Space Containments

**Proof that \(\text{NP} \subseteq \text{PSPACE}\)** Let \(L\) be a language in \(\text{NP}\). To decide \(L\) in polynomial space, use the following algorithm:

**On input** \(w\):

1. Compute \(\phi\) such that \(\phi \in \text{SAT}\) if and only if \(w \in L\).
2. Try every possible truth assignment to the variables \(x_1, \ldots, x_n\). Accept if one satisfies \(\phi\).
3. Reject if none of the assignments satisfy \(\phi\).

Note that we can perform step (1) in polynomial space because \(L \leq_p \text{SAT}\). That is, there is a polynomial-time mapping reduction from \(L\) to \(\text{SAT}\), which also uses polynomial space. While trying every possible assignment in step (2), we only need to store one assignment at a time, as well as \(\phi\). Thus, each step requires only a polynomial amount of space. ■
Proof that \( \text{PSPACE} = \text{NPSPACE} \) In the last class, we proved Savitch’s Theorem, which states that \( \text{NSPACE}(f(n)) \subseteq \text{SPACE}(f(n)^2) \) for \( f(n) \geq n \). If \( f(n) \) is a polynomial, then \( f(n)^2 \) is also a polynomial. It follows that \( \text{NPSPACE} \subseteq \text{PSPACE} \). Moreover, \( \text{PSPACE} \subseteq \text{NPSPACE} \) because any language deterministically decided in polynomial space is also nondeterministically decided in polynomial space. ■

Lemma If \( M \) is an \( f(n) \)-space TM, then \( M \) runs in \( 2^{O(f(n))} \) time.

Proof of Lemma Let \( M \) be an \( f(n) \)-space TM. Then \( M \) has \( 2^{O(f(n))} \) configurations, each of which is specified by:

1. A state, using constant space \( A \)
2. A location of the tape head, using \( f(n) \) space
3. The contents of the tape, using \( |\Gamma|^{f(n)} \) space

The amount of space needed for \( c \) configurations is thus \( c \cdot (A + f(n) + |\Gamma|^{f(n)}) \), which is in the family \( 2^{O(f(n))} \). Since \( M \) is a decider, it never enters the same configuration twice. Thus, it runs in \( 2^{O(f(n))} \) time. ■

Proof that \( \text{PSPACE} \subseteq \text{EXPTIME} \) Recall that \( \text{EXPTIME} = \bigcup_{k=1}^{\infty} \text{TIME}(2^{O(n^k)}) \). It follows from the previous lemma that a \( p(n) \)-space TM for a polynomial \( p \) runs in \( 2^{O(p(n))} \) time, which is in \( \text{EXPTIME} \). Thus, any polynomial-space TM has at most an exponential runtime. ■

2 Languages in PSPACE

One language in PSPACE is based on the “Geography” game, in which players take turns naming cities such that the starting letter of the current city is the last letter of the previous city. A collection of possible moves between cities can be represented as a directed graph whose vertices represent cities, and whose edges connect a city to its potential successors in the game. For example, a set of edges might be: \{ (Providence, Edinburgh), (Providence, Edmonton), (Edinburgh, Hilton Head), (Hilton Head, Denver), (Edmonton, Newport) \}. More formally, we define the language:

- **GENERALIZEDGEOGRAPHY** = \{ \( (G,b) \mid G = (V,E) \) is a directed graph; the game begins at \( b \in V \); Player 1 starts; and no matter how Player 2 plays, Player 1 can always win \}

We also say that Player 1 has a “winning strategy”, meaning that there exists a way for Player 1 to win regardless of Player 2’s moves.
**Proof that GeneralizedGeography is in PSPACE**  
Consider the following algorithm $GG$, which decides `GENERALIZED_GEOGRAPHY`:

\[
GG, \text{ on input } \langle G, b \rangle:
\]

1. If $V = \{b\}$, accept.
2. For every neighbor $u$ of $b$:
   
   (a) Let $G' = G$ with $b$ removed.
   (b) If $GG$ accepts on input $\langle G', u \rangle$, reject.
3. Accept.

First, note that $GG$ terminates because $G'$ is smaller than $G$. Each time we make a recursive call, the graph shrinks, so the depth of recursion is bounded by the number of vertices in $G$. We can prove correctness by induction. Without going into detail, we can say that $GG$ is correct in the base case, in which $G$ has only one vertex. In the inductive case, $GG$ relies on its correctness when the input is a smaller graph $G'$.

The algorithm takes polynomial space. Let $S(n)$ be the amount of space needed for $GG$ to run on an input graph with $n$ vertices. $S(1)$ is constant and $S(n) = O(n) + S(n - 1)$. Expanding this recurrence relation yields a total of $O(n^2)$ space. Thus, $GG$ correctly runs in polynomial space.

On the other hand, let $T(n)$ be the time needed for $GG$ to run on a graph with $n$ vertices. Then $T(1)$ is constant and $T(n) = (\deg G) \cdot T(n - 1)$. When we expand this relation, the total time is of the form $(\deg G)^n$, which is exponential.

**TQBF and Friends**  
The following related languages are in PSPACE:

- $\text{TQBF} = \{ \langle \psi \rangle \mid \psi \text{ is a true quantified Boolean formula} \}$
- $\text{TQBF-3CNF} = \{ \langle \psi \rangle \mid \psi \text{ is of the form } Q_1 x_1 \ldots Q_n x_n \phi(x_1, \ldots, x_n), \text{ where } \phi \text{ is in 3CNF, each } Q_i \in \{\exists, \forall\}, \text{ and } \psi \text{ is a true quantified Boolean formula} \}$
- $\text{FORMULAGAME} = \{ \langle \psi \rangle \mid \psi \text{ is of the form } Q_1 x_1 \ldots Q_n x_n \phi(x_1, \ldots, x_n), \text{ where } \phi \text{ is in 3CNF, each } Q_i \in \{\exists, \forall\}, Q_1 = \exists, \text{ and the } 3\text{-player wins} \}$

Recall that a Boolean formula $\phi$ over the variables $x_1, \ldots, x_n$ is a “recipe” or algorithm that evaluates to ‘true’ or ‘false’, depending on the variable assignment. A quantified
**Boolean formula** also includes quantifiers for variables. For example, \( \forall x_1 \exists x_2((x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3)) \) is a quantified Boolean formula. In this case, \( x_1 \) and \( x_2 \) are quantified variables, while \( x_3 \) is a free variable. A **true quantified Boolean formula** is a quantified Boolean formula which has no free variables and evaluates to ‘true’. For example, consider the formula: \( \forall x_1 \exists x_2 \forall x_3((x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3)) \). If \( x_1 \) is ‘true’, then we can make \( x_2 \) be ‘false’, and both clauses are satisfied. If \( x_1 \) is ‘false’, then we can set \( x_2 \) to ‘true’, and both clauses are satisfied. Note that \( \langle \phi \rangle \in \text{SAT} \) if and only if \( \langle \psi \rangle \) of the form \( \exists x_1 \ldots \exists x_n \phi(x_1, \ldots, x_n) \) is in TQBF.

**FORMULAGAME** represents a game between two players, the \( \exists \)-player and the \( \forall \)-player. The players take turns assigning values to variables, based on the quantifiers in the quantified Boolean formula. The \( \exists \)-player wins if they can make the formula true, while the \( \forall \)-player wins if they can make the formula false. In fact, **FORMULAGAME** is the same language as TQBF-3CNF. Without loss of generality, we can set the first quantifier to be \( \exists \), and we can also add existential quantifiers so that universal and existential quantifiers alternate.

### 3 PSPACE-Completeness

- A language \( L \) is **PSPACE-complete** if (1) \( L \) is in PSPACE, and (2) for all \( A \in \text{PSPACE} \), \( A \leq_p L \).

Note that \( A \leq_p L \) is still a polynomial-time reduction. If the reduction were instead polynomial-space, the reduction itself could decide any language in PSPACE.

**Theorem** TQBF, TQBF-3CNF, and **FORMULAGAME** are PSPACE-complete. We will see proof of this in the next class.

**Theorem** **GENERALIZEDGEOGRAPHY** is PSPACE-Complete

**Proof** We have already seen that **GENERALIZEDGEOGRAPHY** is in PSPACE. It remains to show that it is PSPACE-hard. We will prove this via a reduction from TQBF-3CNF. Consider the following function, which takes as input \( \langle \psi \rangle \) where \( \psi = Q_1x_1 \ldots Q_nx_n \phi(x_1, \ldots, x_n) \) and \( \phi = C_1 \land C_2 \land \cdots \land C_m \). Without loss of generality, assume \( Q_1 = \exists \) and the quantifiers alternate. Output a graph of the following form:
Each variable $x_i$ has a trapezoidal gadget, and each clause $C_i$ has an edge to the negation of variables in the clause. For example, in the figure above, $C_2$ might be $x_1 \lor x_4 \lor \neg x_n$. To analyze this reduction, consider $\psi$ as an instance of FORMULAGAME. The $\exists$-player goes first, and must select the node $b$. Then the $\forall$-player only has one subsequent node to choose. After this, the $\exists$-player can choose whether to set $x_1$ to ‘true’ or ‘false’ by selecting the corresponding node. The $\forall$-player again only has one choice. This continues for all $x_n$ variables, with the $\exists$-player selecting a truth assignment. Finally, the $\exists$-player selects the node $\phi$. The $\forall$-player selects a clause $C_i$, and the $\exists$-player must pick the negation of a variable in $C_i$. At this step, there exists an unvisited node for the $\exists$-player to visit only if some variable in the clause $C_i$ is satisfied by the selected truth assignment. If the $\exists$-player can make a move, they win. In other words, the $\exists$-player has a winning strategy if the formula is true.