1 PSPACE-completeness

Definition. A language \( L \) is \textbf{PSPACE-hard} if for all \( L' \in \text{PSPACE} \), \( L' \leq_p L \). That is, \( L' \) can be reduced to \( L \) in polynomial time.

Why polynomial-time reduction, rather than considering a new kind of reduction, polynomial-space reductions? Suppose \( A \in \text{PSPACE} \). With polynomial space (no bound on time), we can already decide \( A \). Then we may as well do that instead of reducing to some other language \( B \). In fact, any language \( A \) in PSPACE can be reduced in polynomial space to any other language \( B \), except for when \( B = \emptyset \) or \( \Sigma^* \). The reduction from \( A \) to \( B \) is just: on input \( w \), first run the polynomial-space decider to decide if \( w \in A \). If yes, output some fixed “yes” instance \( x_{\text{yes}} \in B \); if no, output some fixed “no” instance \( x_{\text{no}} \not\in B \). All we need for this reduction to work is that some \( x_{\text{yes}} \) and \( x_{\text{no}} \) exist, i.e. that \( B \) is not \( \emptyset \) or \( \Sigma^* \).

Definition. A language \( L \) is \textbf{PSPACE-complete} if:

1. \( L \in \text{PSPACE} \).
2. \( L \) is PSPACE-hard.

2 Quantified boolean formulas and TQBF

Recall the following about Boolean formulas. A \textbf{Boolean formula} is, for example,

\[
(x_1 \lor x_2 \lor x_3) \land (\overline{x_2} \lor \overline{x_3})
\]

We can also have a \textbf{Boolean formula with constants}, which might look like

\[
(T \lor x_2 \lor x_3) \land (\overline{x_2} \lor \overline{x_3})
\]

We could also have a Boolean formula with just constants, and no variables:

\[
(T \lor F \lor F) \land (\overline{F} \lor \overline{F})
\]

Definition. A \textbf{quantified boolean formula} is a string of the form

\[
\psi = Q_1x_1Q_2x_2\ldots Q_nx_n\phi(x_1, x_2, \ldots, x_n),
\]

where \( Q_i \) are each quantifiers (either \( \forall \) or \( \exists \)), and \( \phi \) is a (regular) boolean formula of the variables \( x_1, \ldots, x_n \). In other words, \( \phi \) is the kind of formula we have seen before, e.g. in the language 3SAT.
For example, one quantified boolean formula is
\[ \exists x_1 \forall x_2 \exists x_3 (x_1 \lor x_2 \lor x_3) \land (\overline{x_2} \lor \overline{x_3}). \]

This definition also lets us redefine SAT and \textsc{Tautology}:
\[
\text{SAT} = \{ \langle \phi \rangle \mid \psi = \exists x_1 \exists x_2 \ldots \exists x_n \varphi(x_1, \ldots, x_n) \text{ is a true quantified boolean formula} \}.
\]
\[
= \{ \langle \phi \rangle \mid \psi = \forall x_1 \forall x_2 \ldots \forall x_n \varphi(x_1, \ldots, x_n) \text{ is a true quantified boolean formula} \}.
\]

A quantified boolean formula is defined to be true exactly when the quantified statement is true, where each \( x_i \) is taken to be \( T \) or \( F \). Consider the previous example,
\[
\exists x_1 \forall x_2 \exists x_3 (x_1 \lor x_2 \lor x_3) \land (\overline{x_2} \lor \overline{x_3}).
\]

Consider the case that \( x_1 = T \). Then the formula becomes
\[
\forall x_2 \exists x_3 (T \lor x_2 \lor x_3) \land (\overline{x_2} \lor \overline{x_3}).
\]

If \( x_2 = T \), then \( \exists x_3 (T \lor T \lor x_3) \land (\overline{T} \lor \overline{x_3}) \) is true, since we can take \( x_3 \) to be \( F \), and the formula becomes \( (T \lor T \lor F) \land (\overline{T} \lor \overline{F}) = T \). On the other hand if \( x_2 = F \), then \( \exists x_3 (T \lor F \lor x_3) \land (\overline{F} \lor \overline{x_3}) \) is true, no matter whether we take \( x_3 \) to be \( T \) or \( F \). So regardless of \( x_2 \), \( \exists x_3 (T \lor x_2 \lor x_3) \land (\overline{x_2} \lor \overline{x_3}) \) is true, which means \( \forall x_2 \exists x_3 (T \lor x_2 \lor x_3) \land (\overline{x_2} \lor \overline{x_3}) \) is true. So we have shown that if \( x_1 = T \), the remaining formula turns out to be true. Therefore,
\[
\exists x_1 \forall x_2 \exists x_3 (x_1 \lor x_2 \lor x_3) \land (\overline{x_2} \lor \overline{x_3}).
\]
is true. In particular, there exists a choice of \( x_1 \) (namely \( T \)) such that for all choices of \( x_2 \) (either \( T \) or \( F \)), there is a choice of \( x_3 \) that makes the regular boolean formula true (namely, \( x_3 = F \) when \( x_2 = T \), and say \( x_3 = T \) (or \( x_3 = F \), either works) when \( x_2 = F \)). So this is an example of a \textbf{true quantified boolean formula}.

We define the language of all such true quantified boolean formulas,
\[
\text{TQBF} = \{ \langle \psi \rangle \mid \psi \text{ is quantified boolean formula that is true} \}.
\]

### 3 TQBF is PSPACE-complete

**Theorem.** TQBF is PSPACE-complete.

**Proof.** We need to show (1) that \( \text{TQBF} \in \text{PSPACE} \), and (2) that \( \text{TQBF} \) is PSPACE-hard.

1. Consider the following algorithm:
   \begin{itemize}
   \item **Alg:** On input \( \psi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi(x_1, x_2, \ldots, x_n) \):
     \begin{itemize}
     \item If \( n = 0 \), evaluate the truth of \( \psi \), and accept if it is true, or reject if it is false.
     \item Otherwise, if \( Q_1 = \exists \), run:
       \begin{align*}
       \text{Alg}(Q_2 x_2 \ldots Q_n x_n \phi(T, x_2, \ldots, x_n)) \\
       \text{Alg}(Q_2 x_2 \ldots Q_n x_n \phi(F, x_2, \ldots, x_n)).
       \end{align*}
     \end{itemize}
   \item Else, i.e. \( Q_1 = \forall \), again run the same things:
     \begin{align*}
     \text{Alg}(Q_2 x_2 \ldots Q_n x_n \phi(T, x_2, \ldots, x_n)) \\
     \text{Alg}(Q_2 x_2 \ldots Q_n x_n \phi(F, x_2, \ldots, x_n)).
     \end{align*}
   \end{itemize}
   If either of the two accepts, accept; otherwise, reject.
   \begin{itemize}
   \end{itemize}

Accept only if \textit{both} accept, and otherwise reject.
We can see that this algorithm is correct recursively. If there are no quantifiers (the \( n = 0 \) case), then the formula \( \phi \) does not contain any variables, and the algorithm correctly accepts if and only if \( \phi \) evaluates to true. For \( \exists x_i \), the algorithm just checks whether there exists a value of \( x_i \) that makes the remaining formula true. For \( \forall x_i \), the algorithm checks whether the remaining formula is true for all possible \( x_i \).

How much space do we use? Let \( S(n, m) \) be the space Alg needs for inputs \( \psi \) with \( n \) variables, and with length \(|\phi|=m\). We have

\[
S(n, m) = \underbrace{n + m} + \underbrace{1} + S(n-1, m)
\]

Now, \( S(0, m) \) is polynomial space in \( m \); say it uses \( p(m) \) space for some polynomial \( p \). Then

\[
S(n, m) = \sum_{i=0}^{n} \left[ (n+1)+m+1 \right] + \sum_{i=2}^{n} \left[ (n+2)+m+1 \right] + \cdots + \left[ (n+1)+m+1 \right] + S(0, m)
= O(n^2) + nm + p(m),
\]

which is polynomial in the size of the input \( m \) (since the number of variables \( n \) is bounded by \( m \)).

(2) Let \( L \in \text{PSPACE} \). Let \( M \) be a TM deciding \( L \) using polynomial space, \( p(n) \), such that \( M \) has a unique starting and accepting configuration. In the last lecture, we saw that \( M \) must halt within \( 2^h \cdot p(n) \) steps, for some constant \( h \).

Similar to in the Cook-Levin theorem, we prove by reducing from

\[
x \in L \iff M \text{ accepts } x
\]

\[
\iff M \text{ goes from the unique starting configuration for } x
\]

\[
to the unique accepting configuration in } 2^h \cdot p(n) \text{ steps}
\]

\[
\iff \exists \text{ an intermediate configuration } c_m \text{ such that }
\]

\[
M \text{ goes from } c_0(x) \text{ to } c_m \text{ in } \frac{1}{2} 2^h \cdot p(n) \text{ steps, and }
\]

\[
M \text{ goes from } c_m \text{ to } c_{\text{accept}} \text{ in } \frac{1}{2} 2^h \cdot p(n) \text{ steps}
\]

\[
\iff \exists c_m \text{ such that } \forall (c_1, c_2) \in \{ (c_0(x), c_m), (c_m, c_{\text{accept}}) \},
\]

\[
M \text{ goes from } c_1 \text{ to } c_2 \text{ in } \frac{1}{2} 2^h \cdot p(n) \text{ steps}
\]

\[
\iff \exists c_m^1 \text{ such that } \forall (c_1^1, c_2^1) \in \{ (c_0(x), c_m^1), (c_m^1, c_{\text{accept}}) \},
\]

\[
\iff \exists c_m^2 \text{ such that } \forall (c_1^2, c_2^2) \in \{ (c_1^1, c_m^2), (c_m^2, c_{\text{accept}}) \},
\]

\[
M \text{ goes from } c_1^1 \text{ to } c_2^2 \text{ in } \frac{1}{2} 2^h \cdot p(n) \text{ steps}
\]

\[
\cdots
\]

\[
\iff \exists c_m^h\cdot p(n) \text{ such that } \forall (c_1^h, c_2^h) \in \{ (c_1^{h-1}, c_m^h), (c_m^h, c_{\text{accept}}) \},
\]

\[
M \text{ goes from } c_1^h \text{ to } c_2^h \text{ in } 1 \text{ step.}
\]

The total number of \( \iff \) s above is \( h \cdot p(n) \).

A configuration \( C \) looks like

\[
\# y_1 \ y_2 \ \cdots \ y_m \ q_i \ y_{m+1} \ \cdots \ \# \underbrace{\cdots \cdots \cdots \ #}_{p(n)+3 \text{ cells}}
\]
Let $S$ be the set of states in $M$. To encode $C$ as Boolean variables, as in the proof of Cook-levin, we need

$$(p(n) + 3)(|\Gamma| + |S|)$$

variables. For $j$ a cell in $C$, and $\alpha \in \Gamma \cup S \cup \{\#\}$, we define the variables $x^C_{j,\alpha}$, which is true if $\alpha$ is found in position $j$ in $C$, and false otherwise.

Define the function $F$, which takes as input an assignment $\{a_{j,\alpha} : 1 \leq j \leq p(n) + 3, \alpha \in \Gamma \cup S \cup \{\#\}\}$ that encodes configuration $C_1$, and an assignment $\{b_{j,\alpha} : 1 \leq j \leq p(n) + 3, \alpha \in \Gamma \cup S \cup \{\#\}\}$ that encodes a configuration $c_2$. OR, just a variable. So each $a_{j,\alpha}$ is either $T$, $F$, or a variable. On input $(c_1, c_2, t)$:

- If $t = 1$, output $\phi_{\text{cell}} \land \phi_{\text{move}}$, for these, where $\phi_{\text{cell}} = \bigwedge_{j=1}^{p(n)+3}$ only one variable per cell is true both in $c_1$ and $c_2$. And,

$$\phi_{\text{move}} = \bigwedge_{j=1}^{p(n)+3} \bigvee \left[ \begin{array}{cccc} c & d & e \\ f & g & h \end{array} \right] \in \text{valid windows} a_{j,c} \land a_{j+1,d} \land a_{j+2,e} \land b_{j,f} \land b_{j+1,g} \land b_{j+2,h}$$

- If $t > 1$, output

$$\exists \{x_{j,\alpha}\} \land \exists \{y_{j,\alpha}\} \land \exists \{z_{j,\alpha}\} \left[ F\left( \{y_{j,\alpha}\}, \{z_{j,\alpha}\}, \frac{t}{2} \right) \right]$$

\begin{align*}
\exists \{x_{j,\alpha}\} & \land \{y_{j,\alpha}\} & \exists \{x_{j,\alpha}\} & \exists \{y_{j,\alpha}\} \\
\text{middle} & \text{new start, could} & \text{new end} & \text{be old start,}
\end{align*}

could be middle

\begin{align*}
\lor (\{y's\} = \{a's\}, \{z's\} = \{x's\}) \\
\lor (\{y's\} = \{x's\}, \{z's\} = \{b's\})
\end{align*}

We’re using shorthand: a logical $A \rightarrow B$ can be expressed as $\overline{A} \lor B$, and $A = B$ can be expressed as $(A \land B) \lor (\overline{A} \land \overline{B})$. So we can express the statement (e.g.) that all the $x$’s equal all the $y$’s.