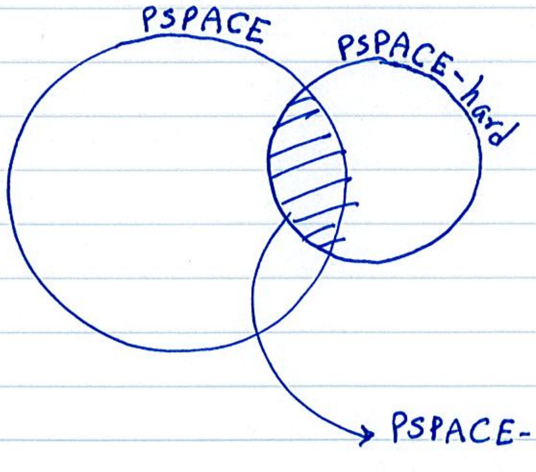
## PSPACE completeness

Definition: A language $L'$ is PSPACE-hard if $\forall L \in PSPACE$, $L \leq_p L'$.

Note here the use of "$\leq_p$" - polynomial time Karp reductions. Thus if a language $L' \in$ PSPACE-hard turned out to be in P then P = PSPACE. The motivation for using "$\leq_p$" is that we are interested in identifying languages in PSPACE that are hard w.r.t. our notion of efficient computation (i.e., poly-time computation). It is easy to define other notions of resource-bounded computations, e.g., $<_L$, which is a reduction that uses logarithmic space.

Definition: A language in PSPACE is PSPACE-complete if it is PSPACE-hard.

Thus:



As before (when we defined NP-completeness), the main motivation for defining PSPACE-complete & identify problems in this class is to see if we can understand the "essence" of what makes problems in PSPACE hard.

## TQBF

For the class NP, the first hard problem we identified was SAT. For PSPACE the first hard problem also turns out to involve boolean formulae, but now with quantifiers.

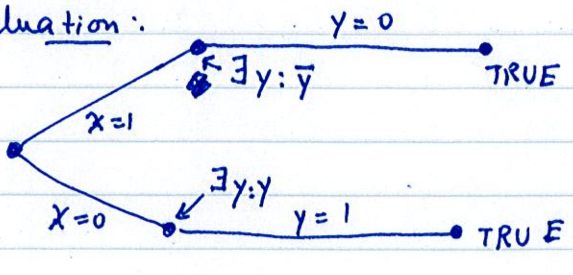
Example: $\phi := \underset{\substack{\uparrow \\ \text{universal} \\ \text{quantifier}}}{\forall x}\ \underset{\substack{\uparrow \\ \text{existential} \\ \text{quantifier}}}{\exists y}\ [(x \vee y) \wedge (\bar{x} \vee \bar{y})]$

- This is an example of a fully quantified boolean formula. In other words, all variables appearing in the formula are quantified.

- This formula is in prenex normal form. This indicates that all quantifiers appear at the beginning of the formula and the scope of each quantifier includes the entire formula.

- Any fully quantified formula is either true or false, Unlike in the case of SAT where the formula could be true or false depending on the assignment of truth values to variables.

- Evaluation:

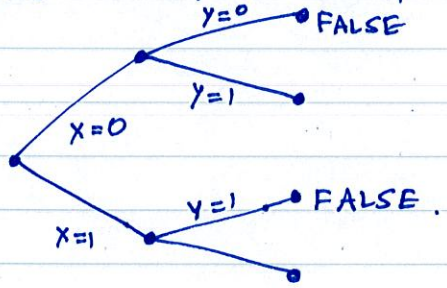


So yes, for each value of $x$, there is a value of $y$ that makes the formula true.

What about if we switch the quantifiers?

$$\phi' = \exists x \forall y \, [(x \vee y) \wedge (\bar{x} \vee \bar{y})]$$

Does there exist a value of $x$ that will make the formula true, no matter what value of $y$ is chosen?



So the answer is no. In other words $\phi'$ is false. □

$$\text{TQBF} = \left\{ \llcorner \phi \lrcorner \;\middle|\; \phi \text{ is a fully quantified } \underline{\text{true}} \text{ boolean formula in prenex normal form} \right\}$$

$$\xrightarrow{} Q_1 x_1 \, Q_1 x_2 \cdots Q_n x_n : \underbrace{\psi(x_1, x_2, \ldots, x_n)}_{\text{need not be in CNF}}$$

where each $Q_i \in \{\exists, \forall\}$.

C<u>LAIM</u>: TQBF is in PSPACE

<u>PROOF</u>: We consider a wider class of formulas - those in which a literal is not just $x_i$ or $\bar{x}_i$, for some variable $x_i$, but also a constant - 0 or 1. Assuming that our input is a formula from this class, we design the following algorithm:

<u>Algo A($\llcorner \phi \lrcorner$)</u>: $\xleftarrow{\text{input}}$

```
    (1) If $n = 0$, then $\psi$ only contains constants and
        so it can be evaluated in $\llcorner \psi \lrcorner$ space.
    (2) If $n > 1$, then there are two cases depending on
        whether $Q_1 = \exists$ or $Q_1 = \forall$.
```

(a) If $Q_1 = \exists$, then return 1 (true) if either $A(\llcorner \phi|_{x_1=1} \lrcorner) = 1$ or $A(\llcorner \phi|_{x_1=0} \lrcorner) = 1$; return 0 otherwise.

(b) If $Q = \forall$ then return 1 (true) if $A(\llcorner \phi|_{x_1=1} \lrcorner) = 1$ and $A(\llcorner \phi|_{x_1=0} \lrcorner) = 1$; return 0 otherwise.

~~the fact that this~~ Suppose that $\phi$ has $n$ variables & size $m$. Then it is easy to see that the above recursive algo. runs in space $m \cdot n \leq m^2$.

$\square$

CLAIM: TQBF is PSPACE-hard.

PROOF: This shares some ideas from the Cook-Levin theorem, but is much more straightforward. It uses the configuration graph idea we used in the proof of $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$.

Suppose $L$ is an arbitrary language in PSPACE. We want to show that $L \leq_p$ TQBF. Let $M$ be the machine that decides $L$ in space $S(n)$ space (for a polynomial $S(n)$) and let $x \in \{0,1\}^n$. Let $m = O(S(n))$ denote the number of bits needed to encode configs of $M$ on length $n$ inputs.

~~For any two configs $C, C'$ of $M$ on length $n$ inputs whether there is an edge from $C$ to $C'$ can be encoded by a boolean formula $\varphi_M$ on $2m$ variables.~~ Using ideas similar to those in the Cook-Levin theorem it is easy to see that there is a boolean formula $\varphi_M$ on $2m$ variables such that $\varphi_M(C, C') = 1$

iff there is an edge from $C$ to $C'$ in the config. graph $G_{M,x}$. Furthermore, $|\llcorner \varphi_M \lrcorner| = O(m)$. This is the key aspect of the claim & follows from the "locality" ideas in the Cook-Levin Theorem.

We will use $\varphi_M(\cdot,\cdot)$ to come up with a poly-sized quantified boolean formula $\Phi$ that has polynomially many variables bounded by quantifiers & two unquantified variables such that for every $C, C' \in \{0,1\}^m$, $\Phi(C,C') = 1$ iff $C$ has a directed path to $C'$ in $G_{M,x}$.

We will define $\Phi$ inductively. Let $\Phi_i(C,C')$ be true if there is a $\overset{\text{dir.}}{}$path of length $\leq 2^i$ from $C$ to $C'$. Then $\Phi = \Phi_m$ and $\Phi_0 = \varphi_M$. It is now easy to see that

$$\Phi_i(C,C') = \exists C'' : \Phi_{i-1}(C,C'') \wedge \Phi_{i-1}(C'',C').$$

However, this does not quite work out since $\Phi_m$ will have size about $2^m$, whereas we are shooting for size $O(m)$. So we use a simple trick of using quantifiers & "folding" this formula.

$$\Phi_i(C,C') \equiv \exists C'' \forall D^1 \forall D^2 \Big( (D^1 = C \wedge D^2 = C'') \vee (D^1 = C'' \wedge D^2 = C') \Rightarrow \Phi_{i-1}(D^1, D^2) \Big)$$

$\square$