PSPACE, NPSPACE, Savitch's Theorem

True Quantified Boolean Formulas (TQBF)

$\forall w\ \exists x\ \forall y\ \exists z\ (w \vee x \vee \neg y) \wedge (\neg w \vee \neg x) \wedge (x \vee y \vee \neg z) \wedge z$

- **Instance:** Given a quantified Boolean formula (QBF) ϕ

  $Q_1 x_1\ Q_2 x_2 \ldots Q_n x_n\ \phi(x_1, x_2, \ldots, x_n)$

- **Problem:** Is ϕ true?

  Q_i is in $\{\exists, \forall\}$

True Quantified Boolean Formulas (TQBF)

$$\exists x_1 \forall x_2 \exists x_3 \exists x_n \ \phi(x_1, x_2, \ldots, x_n)$$

- Describe a simple algorithm for TQBF.
- What is its time/memory (space) complexity?

procedure TQBF $(Q_1 x_1 \ldots Q_n x_n \ \phi(x_1, \ldots, x_n))$

if $n = 0$ // ϕ is either T or F

if $\phi = T$ output Yes, else output No

else if $Q_1 = \exists$

return TQBF $(Q_2 x_2 \ldots Q_n x_n \ \phi|_{x_1=T}(x_2 \ldots x_n))$

$\vee$ TQBF $(\ldots \ldots \ \phi|_{x_1=F}(x_2 \ldots x_n))$

else // $Q_1 = \forall$

return TQBF $(Q_2 x_2 \ldots Q_n x_n \ \phi|_{x_1=T}(x_2 \ldots x_n))$

$\wedge$ TQBF $(\ldots \ldots \ \phi|_{x_1=F}(x_2 \ldots x_n))$

True Quantified Boolean Formulas (TQBF)

$$\exists x_1 \forall x_2 \exists x_3 \exists x_n \ \phi(x_1, x_2, \ldots, x_n)$$

- Describe a simple algorithm for TQBF.
- What is its time/memory (space) complexity?

Worst-case runtime:

time to plug x_i's value into ϕ

$$T(n) = \begin{cases} 2T(n-1) + c|\phi|, & n \geq 1 \\ c, & n = 0 \end{cases}$$

Exponential runtime

```
procedure TQBF(Q1 x1 ... Qn xn φ(x1, ..., xn))
    if n = 0   // φ is either T or F
        if φ = T output Yes, else output No
    else if Q1 = ∃
        return TQBF(Q2 x2 ... Qn xn φ|x1=T (x2...xn))
             ∨ TQBF( ...... φ|x1=F (x2...xn))
    else  // Q1 = ∀
        return TQBF(Q2 x2 ... Qn xn φ|x1=T (x2...xn))
             ∧ TQBF( ...... φ|x1=F (x2...xn))
```

TQBF is in EXPTIME $= \bigcup_{c \in \mathbb{N}} \mathrm{DTIME}(2^{n^c})$

True Quantified Boolean Formulas (TQBF)

$$\exists x_1 \forall x_2 \exists x_3 \exists x_n \ \phi(x_1, x_2, \ldots, x_n)$$

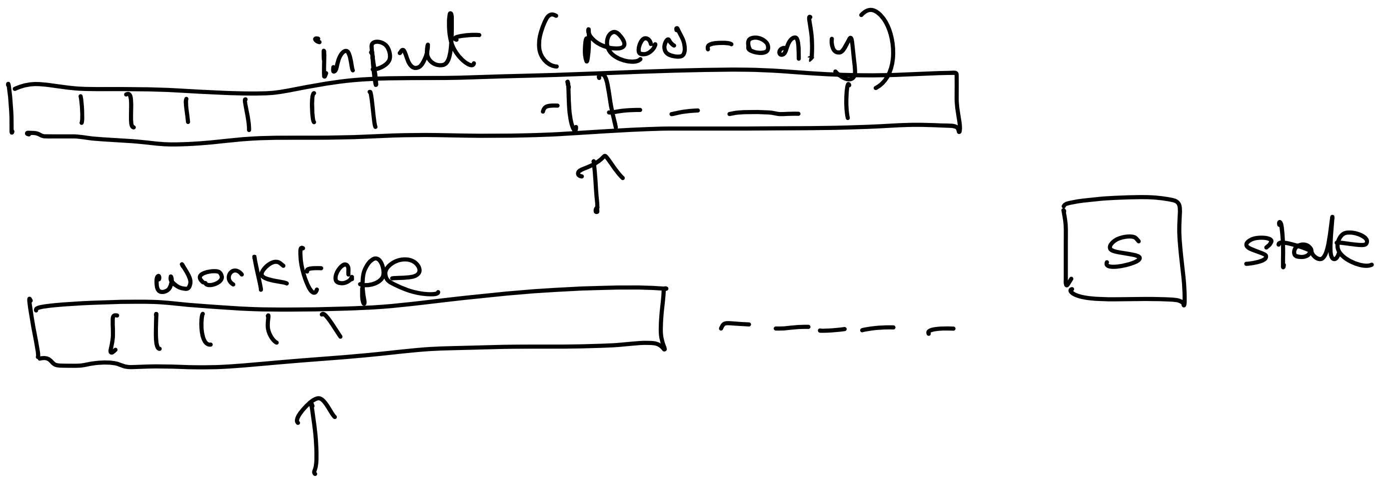
- Describe a simple algorithm for TQBF.
- What is its time/memory (space) complexity?

```
procedure TQBF ( Q1 x1 ... Qn xn φ(x1, ..., xn))
    if n = 0   // φ is either T or F
        if φ = T output Yes, else output No
    else if Q1 = ∃
        return TQBF (Q2 x2 ... Qn xn φ|x1=T (x2 ... xn))
             ∨ TQBF( ...... φ|x1=F (x2 ... xn))
    else // Q1 = ∀
        return TQBF (Q2 x2 ... Qn xn φ|x1=T (x2 ... xn))
             ∧ TQBF( ...... φ|x1=F (x2 ... xn))
```

Memory: n levels of recursion, $O(|\phi| + n)$ storage per level
$\Rightarrow$ polynomial space

Space Bounded Complexity Classes

- Distinguish between a *read-only input tape* and *work tapes* of a Turing Machine (TM).



Configuration exclude input, but do account for position of input tape head

Space Bounded Complexity Classes

- SPACE($s(n)$) is the set of languages accepted by deterministic TMs that always halt and use $O(s(n))$ *work tape* cells on inputs of length n.
- NSPACE($s(n)$): replace "deterministic" by "nondeterministic". (Regardless of nondeterministic choices made, the TM halts.)

Space Bounded Complexity Classes

- PSPACE = $\bigcup_{c>0}$ SPACE(n^c)
- NPSPACE = $\bigcup_{c>0}$ NSPACE(n^c)

- L = SPACE(log n)
- NL = NSPACE(log n)

How do these classes relate to each other, and to the time-bounded classes P, NP, EXPTIME, NEXPTIME?

What classes are contained in PSPACE?

P $\subseteq$ PSPACE, since a poly-time computation uses poly space.

NP $\subseteq$ PSPACE

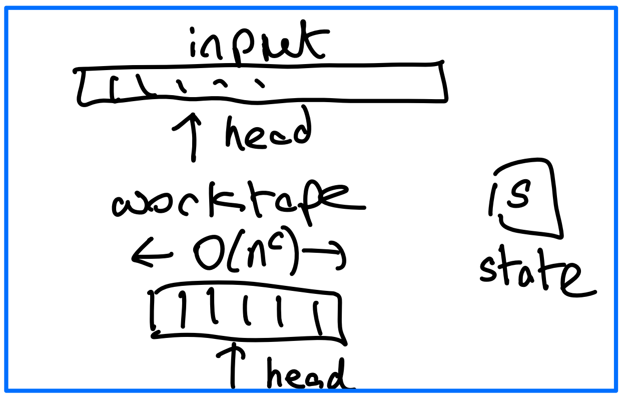
E.g. SAT: try all truth assignments until we find a satisfying assignment or we conclude that there is none. We can delete previously tried assignments as we progress.

$\therefore$ polynomial space.

What classes contain PSPACE?

PSPACE ⊆ EXPTIME

- If a TM uses $O(n^c)$ space, how many different *configurations* can it be in?



possibilities for worktape is

$$O\left(w^{O(n^c)}\right)$$

where w = # worktape symbols

⇒ Exponential # of configurations

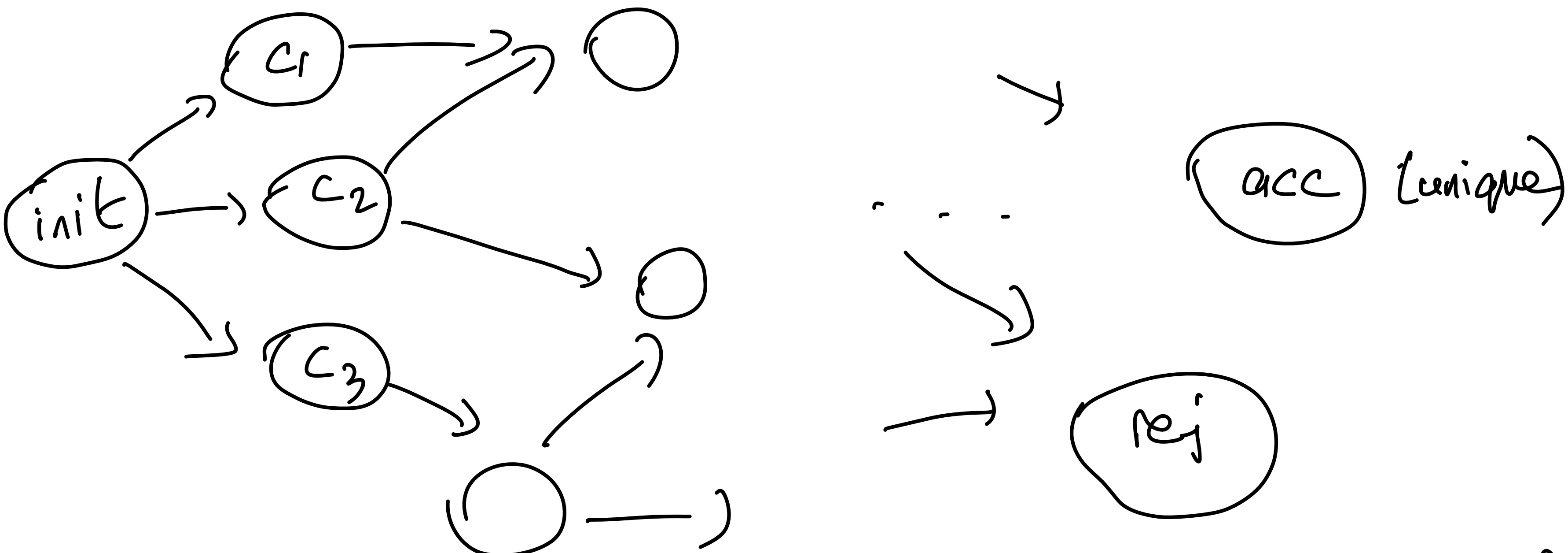
$$O\left(2^{n^{c'}}\right), \quad \text{for some } c' > 0$$

A machine that always halts never enters the same configuration more than once on a given input.

⇒ A pspace-bound machine runs in exponential time.

What classes contain NPSPACE?

- We can represent a computation on a fixed input as a *configuration graph* (acyclic)



# of nodes, on input of length n? $O(2^{poly(n)})$

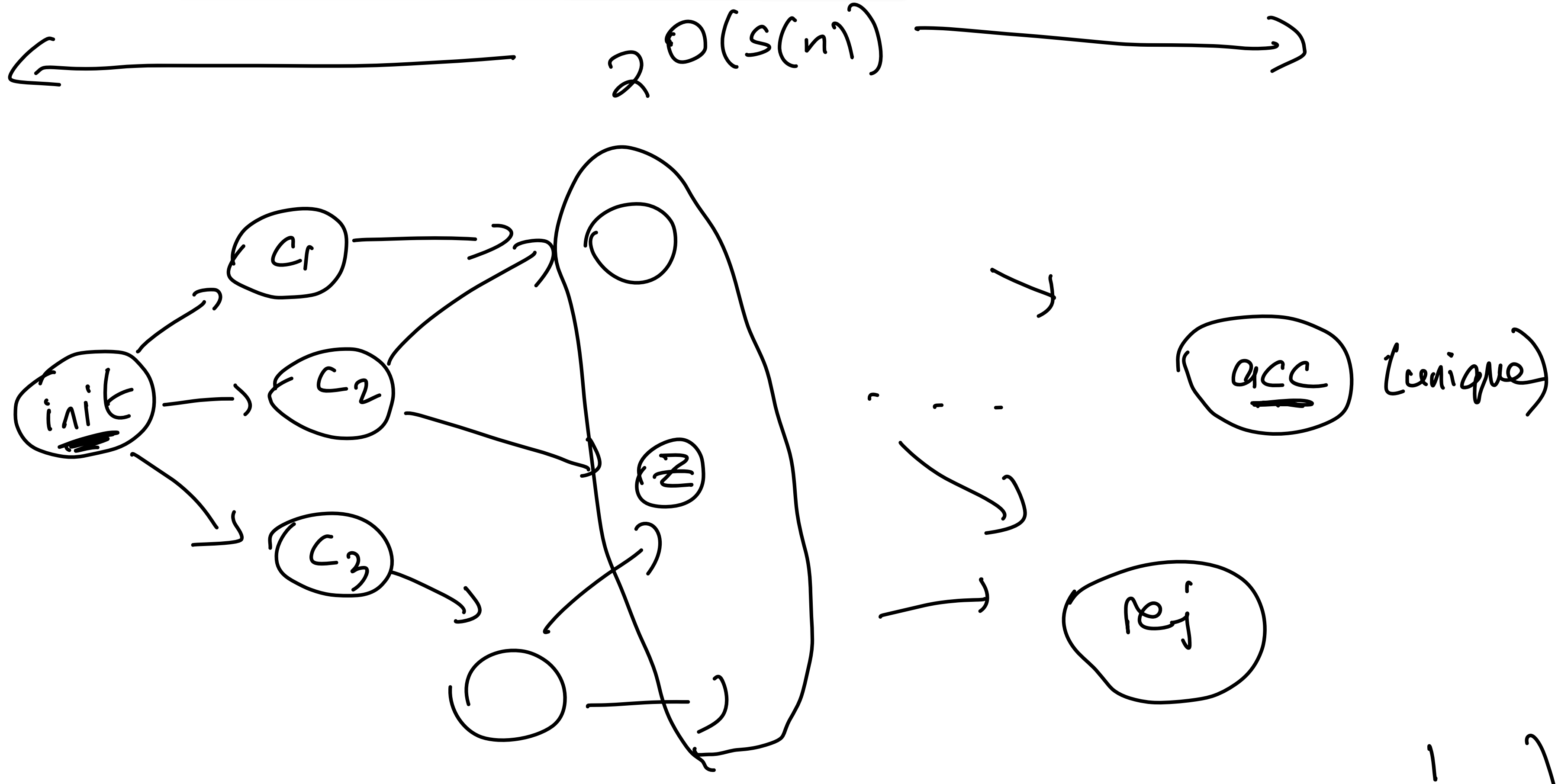
Input is accepted iff there is a path from init to acc.

NPSPACE ⊆ EXP

- Let M be a NTM using O(n^c) space .
- Exp-time algorithm for L(M): On input w:
  - Write down the configuration graph of M on w; the size of the graph is $2^{O(|w|^c)}$
  - Check if the accepting configuration can be reached from the initial configuration (use depth first search or breadth first search)

Also exponential space

Savitch's Theorem: NSPACE($s(n)$) ⊆ DSPACE($s(n)^2$)



BFS works to determine if acc is reachable from init, but takes $2^{O(s(n))}$ space. :(

Savitch's Theorem: NSPACE($s(n)$) ⊆ DSPACE($s(n)^2$)

- Proof idea: Let L be accepted by NTM M within $c.s(n)$) space and $2^{c.s(n)}$ time. We'll describe a deterministic algorithm that accepts L in O($s(n)^2$) space.
- Fix input *w* of length *n,* and let G be the configuration graph of M on *w.*

Savitch's Theorem: NSPACE($s(n)$) ⊆ DSPACE($s(n)^2$)

- Let Reach(x,y,i) be true if there is a path of length $\leq 2^i$ from node x to node y in configuration graph G, and false otherwise.
- On input w, the DTM computes

  Reach($init$,acc,$c.s$(n))

  and accepts if and only if the function returns true

Savitch's Theorem: NSPACE($s(n)$) ⊆ DSPACE($s(n)^2$)

Reach(x,y,i) // does G have a path of length ≤ 2^i from x to y?

if i == 0 then
  if (x == y) or (x,y) is edge of G
    return Yes
  else return No

else // i > 0
  for each child z of x
    if Reach(z, y . . . ← no way to choose new i :(

Savitch's Theorem: NSPACE($s(n)$) ⊆ DSPACE($s(n)^2$)

Reach(x,y,i) // does G have a path of length ≤ 2^i from x to y?

if i == 0 then
  if (x == y) or (x, y) is edge of G
    return Yes
  else return No

else // i > 0
  for each ~~descendant~~ node z of G
  • check Reach(z, y, i-1)
  • " Reach(x, z, i-1)

Technical detail: to enumerate all nodes z we'll mark off c · s(n) cells on worktape. We need to assume that s(n) is "space constructible" from n.

Savitch's Theorem: NSPACE($s(n)$) ⊆ DSPACE($s(n)^2$)

```
Reach(x,y,i)  // does G have a path of length ≤ 2^i from x to y?
    If i = 0 then
        If (x = y) or ( (x,y) is an edge of G) Return True
        Else Return False
    Else
        For each node z of G
            If (Reach (x, z, i-1) and Reach(z, y, i-1) )
                Return True
        Return False
```

Savitch's Theorem: NSPACE($s(n)$) ⊆ DSPACE($s(n)^2$)

```
Reach(x,y,i)  // does G have a path of length ≤ 2^i from x to y?
    If i = 0 then
        If (x = y) or ( (x,y) is an edge of G) Return True
        Else Return False
    Else
        For each node z of G
            If (Reach (x, z, i-1) and Reach(z, y, i-1) )
                Return True
        Return False
```

O(1) configurations stored per recursive level.

Reach(*init*,*acc*,*c.s*(n)) analysis:

- recursion depth: $c \cdot s(n)$
- space per recursive call: $O(1) \cdot$ length of a configuration $= O(s(n))$.

Total space: (recursion depth) * (space per recursion level)

$= O(s(n)^2)$.

Savitch's Theorem: NSPACE($s(n)$) ⊆ DSPACE($s(n)^2$)

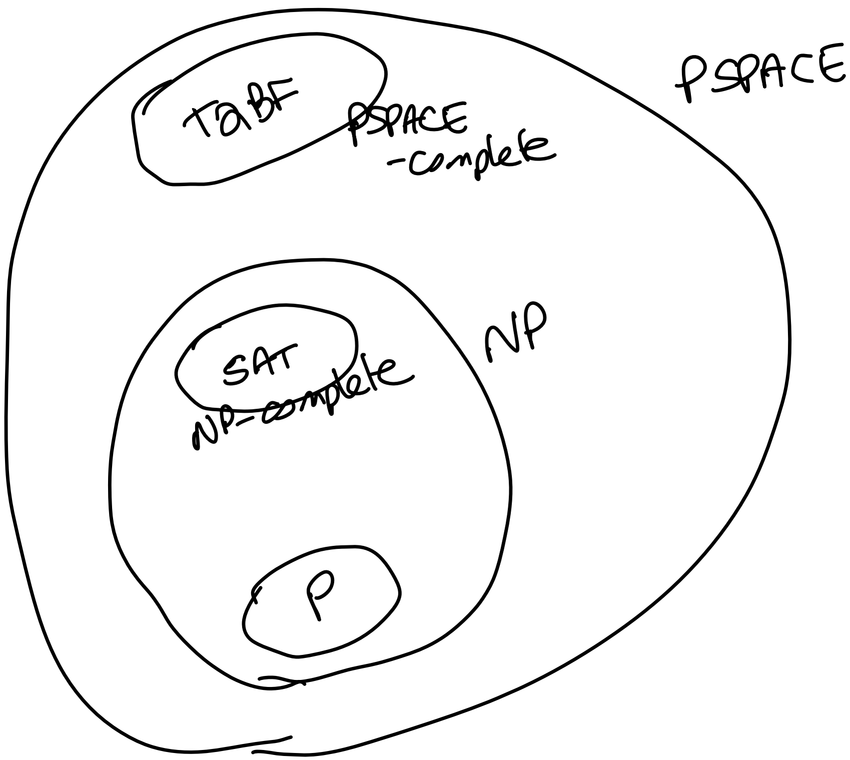
Reach(x,y,i) analysis:

- The space per recursion level is proportional to the space, s(n) used by M on w (where $|w| = n$)
- The recursion depth is i
- So, the recursion depth is c.s(n) on call

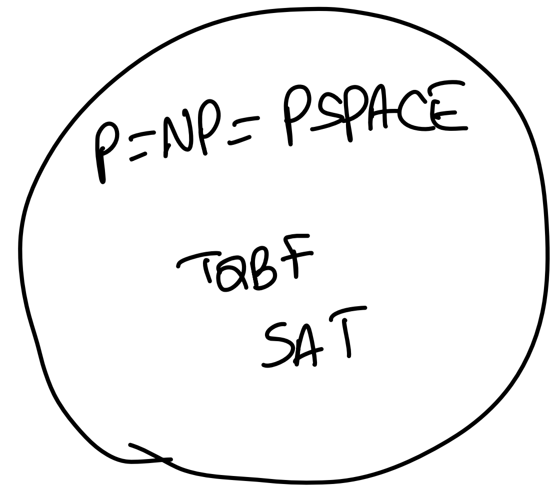
  Reach(*init*, *acc*, *c*.s(n)),

  and the total space used is $O(s(n)^2$.

TQBF is PSPACE-complete



what we think the world looks like



alternative

TQBF is PSPACE-complete

- Let L be a PSPACE language, accepted by TM M within space $c.s(n)$ and time $2^{c.s(n)}$.

- Goal: Poly-time reduction $w \rightarrow$ QBF(w) such that

  w is in L iff QBF(w) is true.

- Equivalently, if Reach(x,y,i) is as before, then

  Reach($init$,acc,c.s($|w|$)) iff QBF(w) is true.

TQBF is PSPACE-complete

Reach(x,y,i) // does G have a path of length $\leq 2^i$ from x to y?

If $i = 0$ [base case omitted]

Else

For each node z of G

If (Reach(x, z, i-1) and Reach(z, y, i-1))

Return True

Return False

First try at expressing this using logic:

$\exists$ config z: Reach(x,z,i-1) $\wedge$ Reach(z,y,i-1))

TQBF is PSPACE-complete

Reach(x,y,i) // does G have a path of length $\leq 2^i$ from x to y?

If $i = 0$ [base case omitted]

Else

```
For each node z of G
    If (Reach(x, z, i-1) and Reach(z, y, i-1) )
        Return True
Return False
```

First try at expressing this using logic:

$$\exists \text{ config } z\text{: Reach}(x,z,i\text{-}1) \land \text{Reach}(z,y,i\text{-}1))$$

Problem: formula size will blow up when expanding the Reach expressions, because of doubling

TQBF is PSPACE-complete

Reach(x,y,i) // does G have a path of length $\leq 2^i$ from x to y?

If $i = 0$ [base case omitted]

Else

```
For each node z of G
    If (Reach(x, z, i-1) and Reach(z, y, i-1) )
        Return True
Return False
```

Better way of expressing this using logic:

$\exists$ config z $\forall$ $v \in$ {True, False} $\exists$ configs z',z''

$(v \Rightarrow z',z''=x,z) \wedge (\neg v \Rightarrow z',z''=z,y) \wedge \text{Reach}(z',z'',i\text{-}1)$

TQBF is PSPACE-complete

Overall QBF:

$$\exists\, z_1\, \forall\, v_1\, \exists\, z_1',z_1''\ \exists\, z_2\, \forall\, v_2 \ldots \exists\, z_m',z_m''$$

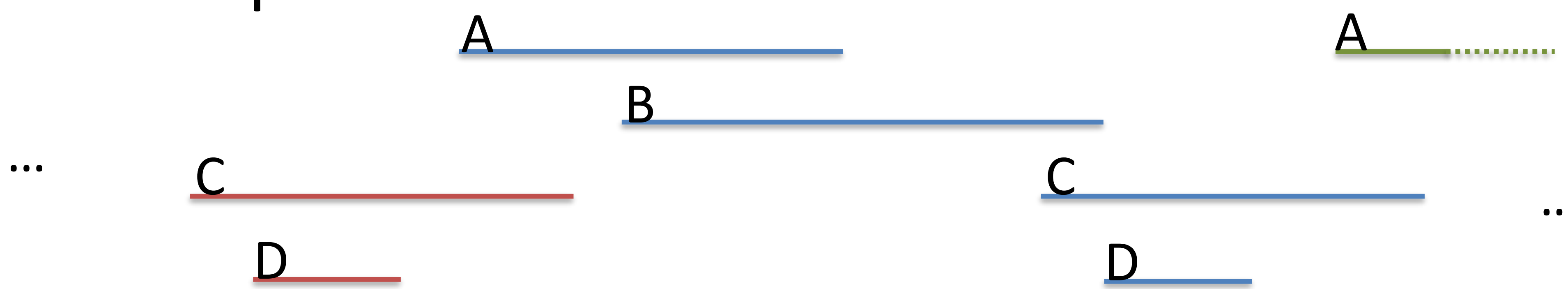
$$\phi(z_0',z_0'', z_1,v_1, z_1',z_1'', ,\ldots, z_m, v_m, z_m',z_m'')$$

where $m = c.s(n)$ and ϕ encodes that

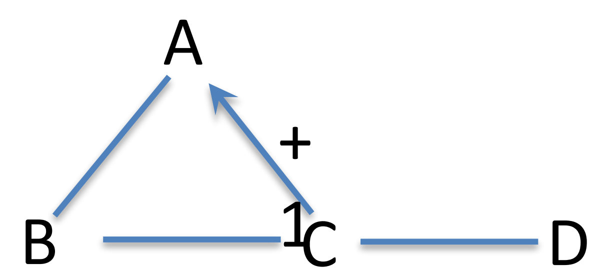
- z_0' and z_0'' are the initial and accepting configs of M on w
- for each i, if v_i = true then $z_i', z_i'' = z_{i-1}', z_i$
- for each i, if v_i = false then $z_i', z_i'' = z_i, z_{i-1}''$
- all of the z_i, z_i' and z_i'' encode valid configurations
- $z_m' = z_m''$ or (z_m', z_m'') is an edge of G (base case)

Periodic Graph Colouring

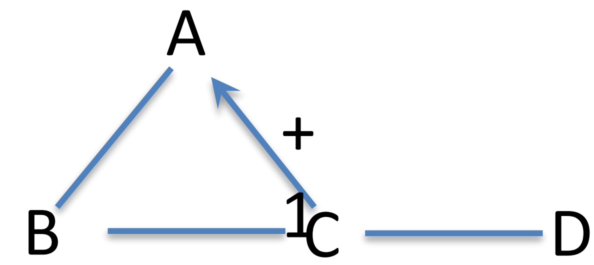
- Motivation: schedule jobs at the same time period each day; want to minimize processors
- Example:



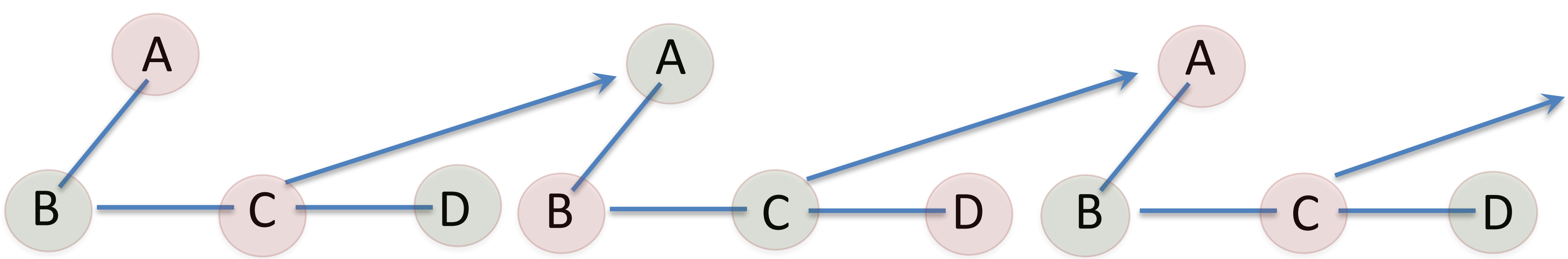
- Succinct representation:



Periodic Graph Colouring



- Succinct graph is 3-colourable, suggesting that we need 3 processors (since two jobs in overlapping time intervals cannot be scheduled on the same processor)
- But the infinite graph is actually 2-colourable, and so we can use just 2 processors!



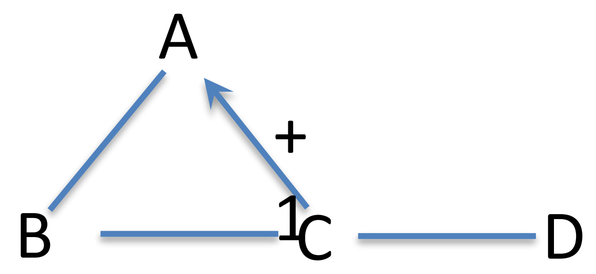
Periodic Graph Colouring

- A *periodic graph* G is an infinite undirected graph, specified by a triple (V,E,E')

- G's nodes:
  - $\cup$ V_i where V_i = {v_i | v in V}, for all i in $\mathbb{Z}$
- G's edges: ($\cup$ E_i) $\cup$ ($\cup$ E_i')
  - where E_i = {{u_i v_i} | {u,v} in E}
  - and E_i' = { {u_i v_{i+1}}| (u,v) in E'}

Periodic Graph Colouring

- **Instance**: A periodic graph G = (V,E,E') and a positive number *k*
- **Problem**: Is G *k*-colourable?

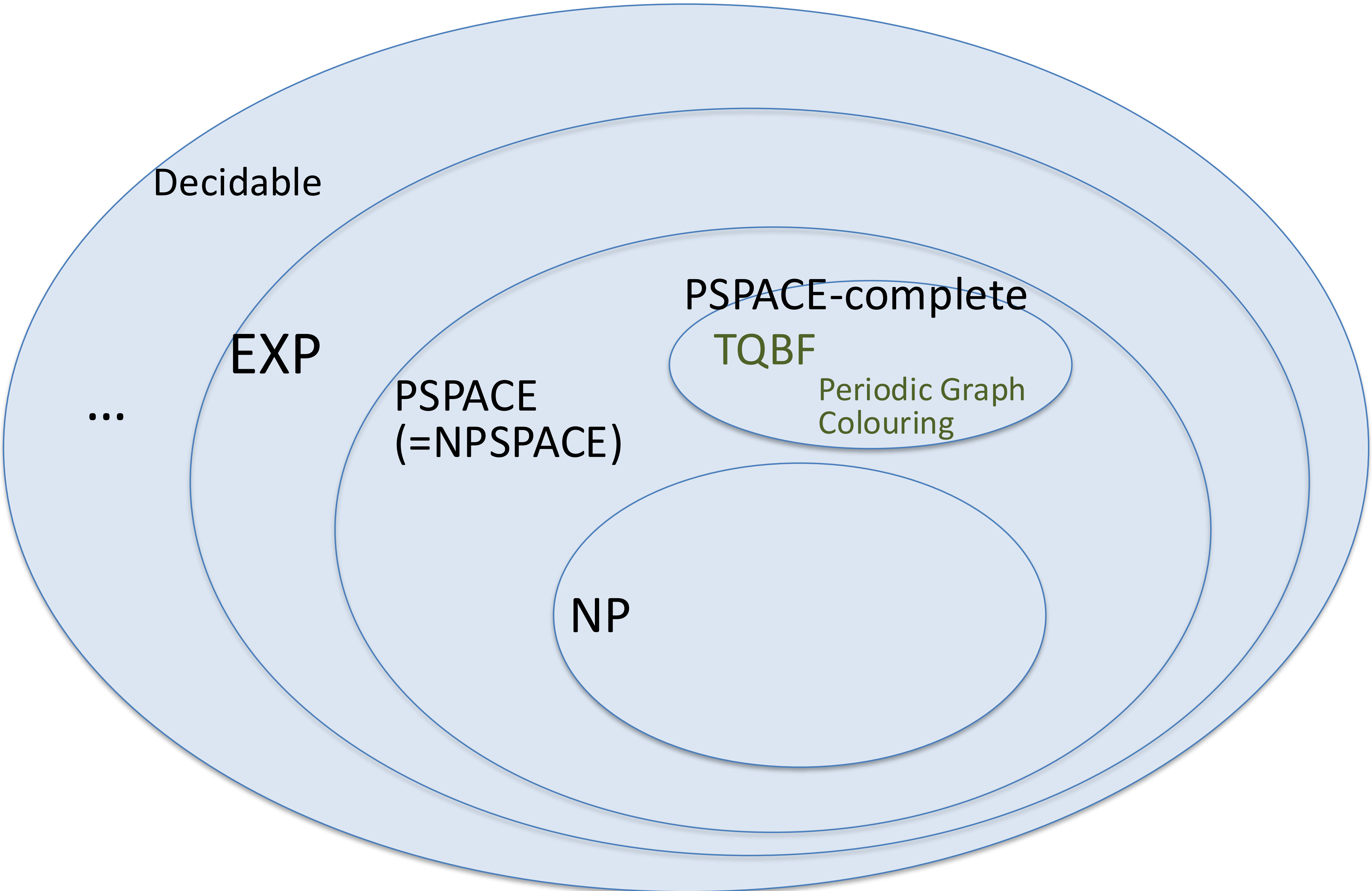


- Can you suggest a nondeterministic algorithm for Periodic Graph Colouring that runs in polynomial space?

Summary

- PSPACE refines the categorization of problems within EXP: those that can be solved with only polynomial space vs those that seem to need both exponential time *and* space
- NPSPACE = PSPACE! (Savitch's Theorem)
- We can leverage Savitch's Theorem to simplify proofs that some problems are in PSPACE (e.g., Periodic Graph Colouring, which also happens to be PSPACE-complete.

Summary


Decidable
EXP
...
PSPACE
(=NPSPACE)
PSPACE-complete
TQBF
Periodic Graph
Colouring
NP