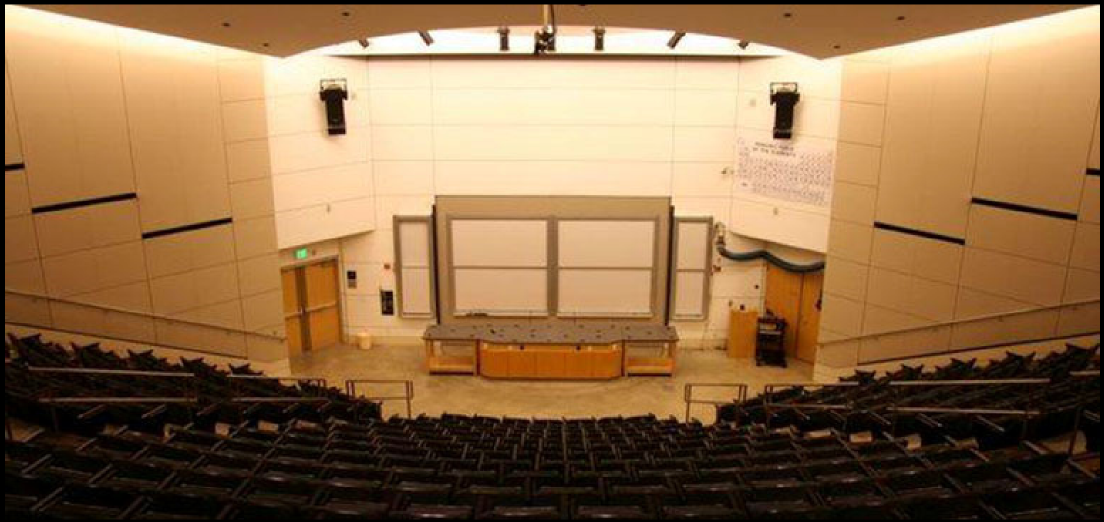
CS 154

Lecture 18: PSPACE

Final Exam Info

Hewlett 200



Thursday, March 19

7:00PM – 10:00PM

One sheet (double-sided) of notes are allowed

Final Exam Info

The final will be comprehensive

Everything discussed in lecture is fair game except foundations of math

Emphasis on concepts and proofs

Practice final and their solutions: coming out soon!

Let M be a deterministic TM.

Definition: The **space complexity** of M is the function $f : N \rightarrow N$, where $f(n)$ is the furthest tape cell reached by M on any input of length n.

Definition: SPACE(s(n)) =

{ L | L is decided by a Turing machine with $O(s(n))$ space complexity}

Corollary:
Space S(n) computations can be simulated in at most $2^{O(S(n))}$ time steps

$$\mathrm{SPACE}(s(n)) \subseteq \bigcup_{c \in \mathbf{N}} \mathrm{TIME}(2^{c \cdot s(n)})$$

Idea: After $2^{O(s(n))}$ time steps, a s(n)-space bounded computation must have repeated a configuration, so then it will never halt…

$$\text{PSPACE} = \bigcup_{k \in N} \text{SPACE}(n^k)$$

$$\text{EXPTIME} = \bigcup_{k \in N} \text{TIME}(2^{n^k})$$

PSPACE $\subseteq$ EXPTIME

Is P $\subseteq$ PSPACE?

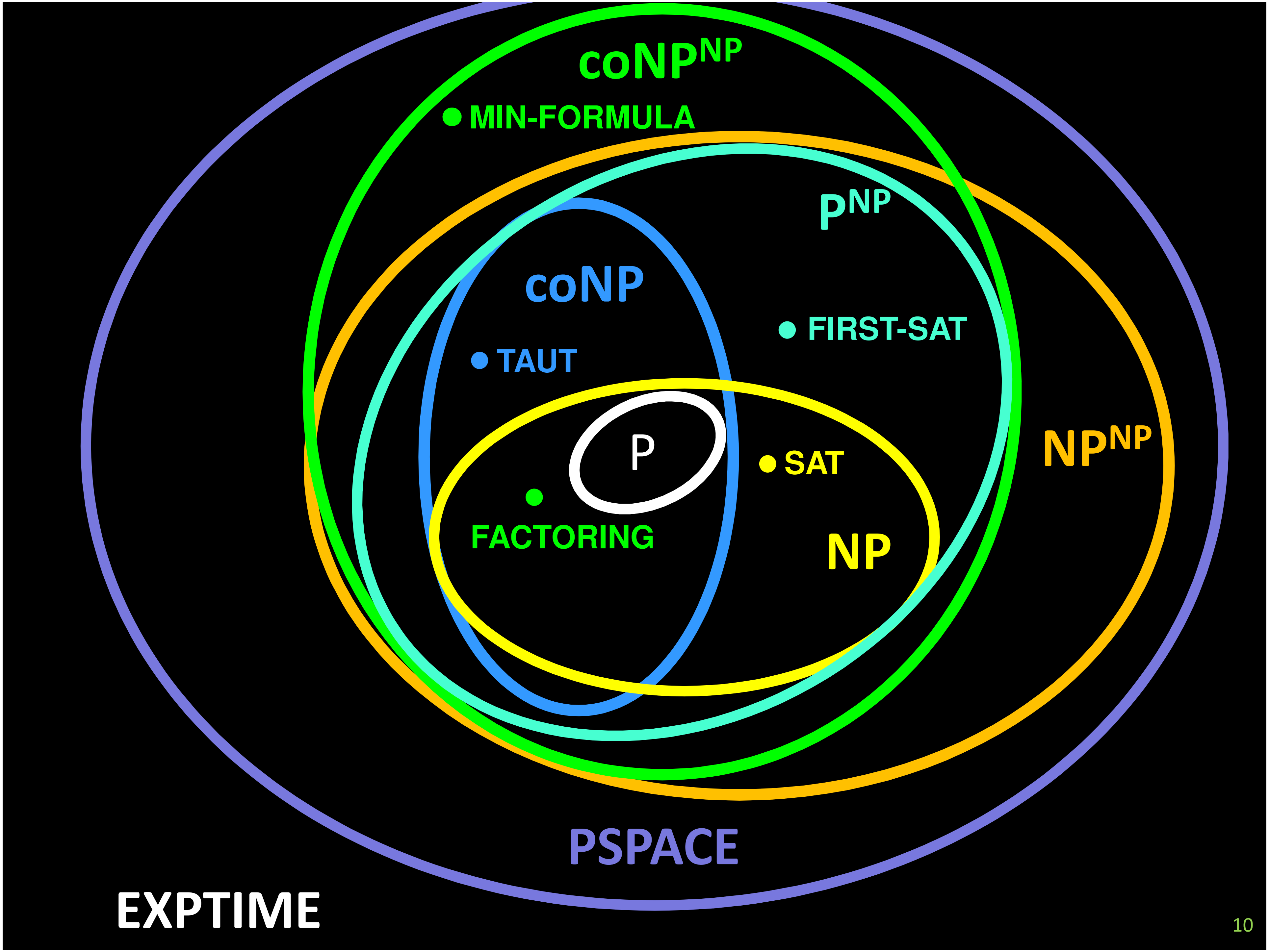
YES

Is NP $\subseteq$ PSPACE?

YES

Is $NP^{NP} \subseteq PSPACE$?

YES



coNP^NP
MIN-FORMULA
P^NP
coNP
FIRST-SAT
TAUT
P
SAT
NP^NP
FACTORING
NP
PSPACE
EXPTIME

$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME}$

Theorem: P ≠ EXPTIME

Why? The Time Hierarchy Theorem!

$$\mathbf{TIME}(2^n) \not\subset \mathbf{P}$$

Therefore P ≠ EXPTIME

PSPACE-complete problems

Definition: Language B is **PSPACE-complete** if:

1. B $\in$ PSPACE
2. Every A in PSPACE is poly-time reducible to B (i.e. B is PSPACE-hard)

Definition:

A **fully quantified Boolean formula** is a Boolean formula where every variable is quantified

These formulas are either **true or false**

$$\exists x \exists y\ [\ x \lor \neg y\ ]$$

$$\forall x\ [\ x \lor \neg x\ ]$$

$$\forall x\ [\ x\ ]$$

$$\forall x \exists y\ [\ (x \lor y) \land (\neg x \lor \neg y)\ ]$$

TQBF = { ϕ | ϕ is a true fully quantified Boolean formula}

Theorem (Meyer-Stockmeyer):
TQBF is PSPACE-complete

TQBF is in PSPACE

QBF-SOLVER(ϕ):

1. If ϕ has no quantifiers, then it is an expression with only constants. Evaluate ϕ. Accept iff ϕ evaluates to 1.

2. If $\phi = \exists x\ \psi$, call QBF-SOLVER on ψ twice: first with x set to 0, then with x set to 1. Accept iff *at least* one call accepts.

3. If $\phi = \forall x\ \psi$, call QBF-SOLVER on ψ twice: first with x set to 0, then with x set to 1. Accept iff *both* calls accept.

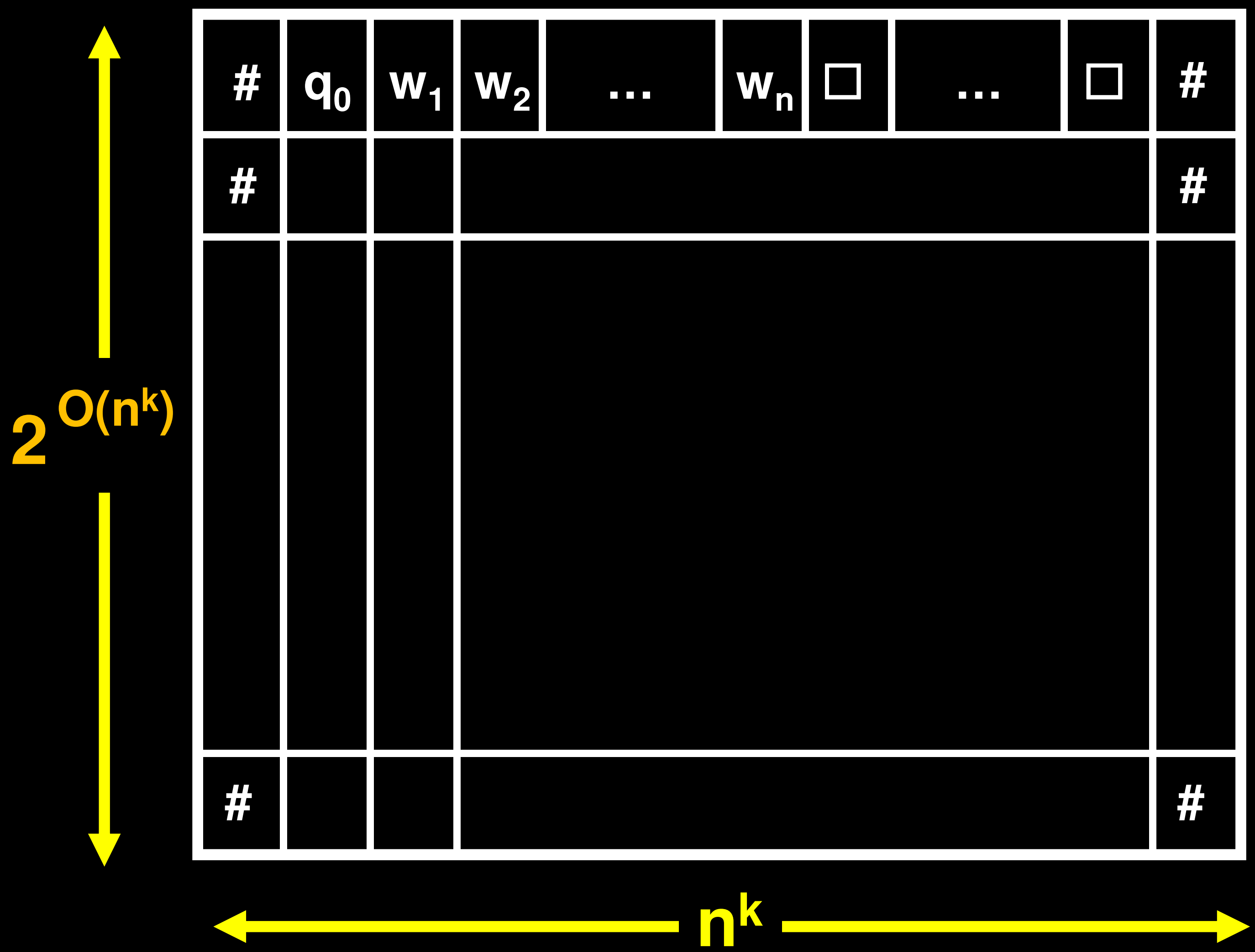
TQBF is PSPACE-hard: Every language A in PSPACE is polynomial time reducible to TQBF

Our poly-time reduction will map every string w to a **fully quantified Boolean formula ϕ** that simulates a **PSPACE machine** for A on w

Let **M** be a deterministic TM that decides A in space n^k for some k

How do we know M exists?

A **tableau for M on w** is an table whose rows are the configurations of M on input w



We design a QBF ϕ that is true
if and only if **M** accepts **w**

Let $s(n) = n^k$. Let $b \geq 0$ be an integer.

Using two collections of $b\, s(n)$ Boolean variables denoted **c** and **d** representing two configurations, and integer $t \geq 0$, we construct a QBF $\phi_{c,d,t}$

$\phi_{c,d,t}$ is true if and only if
M starting in config **c** reaches config **d** in $\leq t$ steps

Then we set $\phi = \phi_{c_{start},\, c_{acc},\, h}$, where

$h = 2^{b\, s(n)}$ upper bounds the total number of configurations of **M** on all inputs of length **n**

c_{start} = initial configuration of M on w,
c_{acc} = (unique) accepting configuration of M

IDEA:

Guess the configuration in the "middle" of the computation, and use recursion!

$\phi_{c,d,t}$ will say:
"there exists a configuration m such that $\phi_{c,m,t/2}$ is true and $\phi_{m,d,t/2}$ is true"

If M uses n^k space on inputs of length n, then the QBF ϕ will have size $O(n^{2k})$

If $t = 1$, then $\phi_{c,d,t}$ should look like:

$\phi_{c,d,t}$ = "c equals d" OR
"d follows from c in a single step of M"

How do we express "c equals d"?
Write a Boolean formula saying that each of the $b\ s(n)$ variables representing c is equal to the corresponding one in d

"d follows from c in a single step of M"?
Use 2 x 3 windows as in the Cook-Levin theorem, and write a CNF formula

For $t > 1$, let's try to construct $\phi_{c,d,t}$ recursively:

$$\phi_{c,d,t} = \exists m\ [\phi_{c,m,t/2} \wedge \phi_{m,d,t/2}\ ]$$

$\exists x_1 \exists x_2 \ldots \exists x_S$ where $S = b\ n^k$

But how long is this formula??

Every level of the recursion cuts t in half but roughly *doubles* the size of the formula…!

We can get around this. Modify the formula to be:

$$\phi_{c,d,t} = \exists m \forall x,y[\ ((x,y)=(c,m) \vee (x,y)=(m,d)) \Rightarrow \phi_{x,y,t/2}\ ]$$

This folds the two recursive sub-formulas into one!

$$\phi_{c,d,t} = \exists m \forall x,y[\ ((x,y)=(c,m) \vee (x,y)=(m,d)) \Rightarrow \phi_{x,y,t/2}\ ]$$

Set $\phi = \phi_{c_{start}\,,\, c_{acc}\,,\, h}$ where $h = 2^{b\, s(n)}$

Each recursive step adds a part that is linear in the size of the configurations, so has size $O(s(n))$

Number of levels of recursion is $\log h = O(s(n))$

Therefore the size of ϕ is $O(s(n)^2)$

PSPACE is a complexity class of
two-player games

For formalizations of
many popular two-player games,
it is PSPACE-complete to decide *who* has a winning strategy on a game board

TQBF as a Game

Played between two players, E and A

Given a fully quantified Boolean formula

$$\exists y \forall x \, [\, (x \vee y) \wedge (\neg x \vee \neg y) \,]$$

E chooses values for variables quantified by $\exists$

A chooses values for variables quantified by $\forall$

The game starts at the leftmost quantifier

E wins if the resulting formula evaluates to true

A wins otherwise

Examples: $\forall x \exists y \, [\, (x \vee y) \wedge (\neg x \vee \neg y) \,]$

$\exists x \forall y \, [\; x \vee \neg y \;]$

FG = { ϕ | Player E has a winning strategy in ϕ }

Theorem: FG is PSPACE-Complete

Proof:

FG = TQBF

The Geography Game

Two players take turns naming cities from anywhere in the world

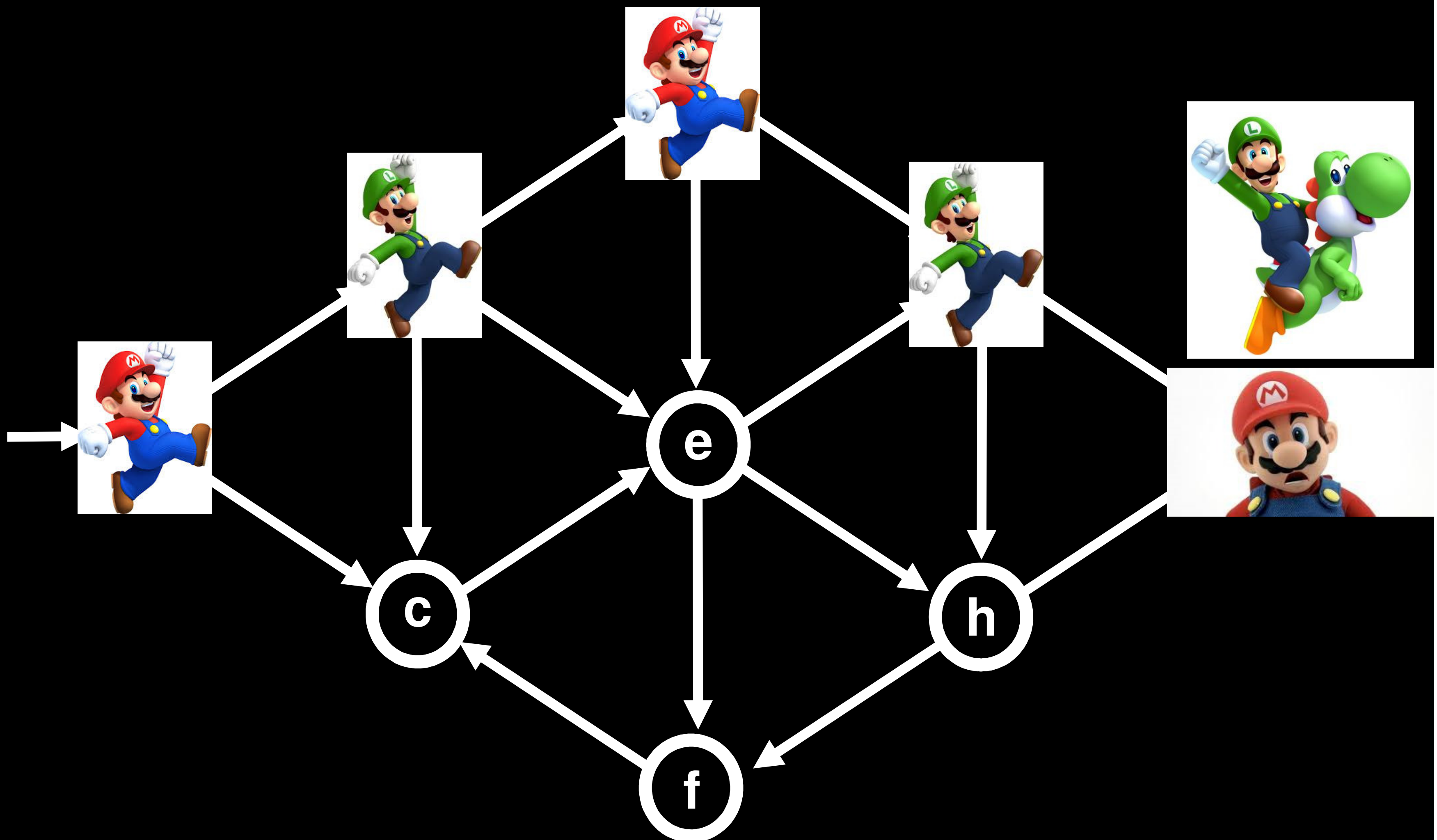
Each city chosen must begin with the same letter that the previous city ended with

Cities cannot be repeated

Austin → Newark → Kalamazoo → Opelika

Whenever someone can no longer name any more cities, they lose and other player wins

Generalized Geography



GG = { (G, a) | Player 1 has a winning strategy for geography on graph G starting at node a }

Theorem: GG is PSPACE-Complete

GG $\in$ PSPACE

Want: PSPACE machine M that accepts (G,a)
$\Leftrightarrow$ Player 1 has a winning strategy on (G,a)

M(G, a): If node a has no outgoing edges, *reject*

Remove node a and all adjacent edges, getting a smaller graph G_1

For all nodes a_1, a_2, …, a_k that node a pointed to, Recursively call $M(G_1, a_i)$

If all the calls accept, then *reject* else *accept*

Claim: All the calls accept
$\Leftrightarrow$ Player 2 has a winning strategy!

GG is PSPACE-hard

We show that **FG $\leq_P$ GG**

We transform a formula ϕ into **(G, a)** such that:

Player E has winning strategy in ϕ

if and only if

Player 1 has winning strategy in (G, a)

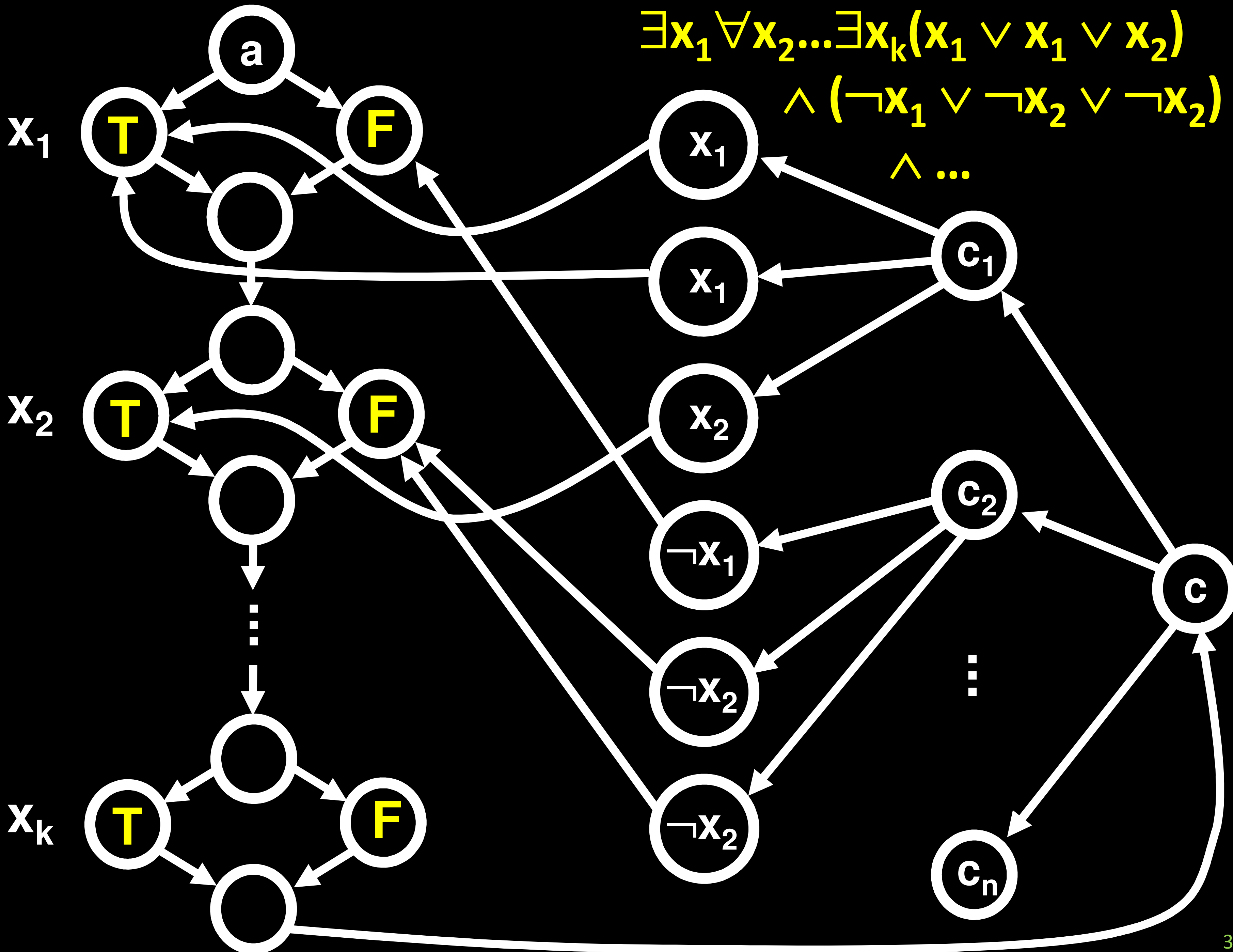
For simplicity we assume ϕ is of the form:

$$\phi = \exists x_1 \forall x_2 \exists x_3 \ldots \exists x_k \, [\psi]$$

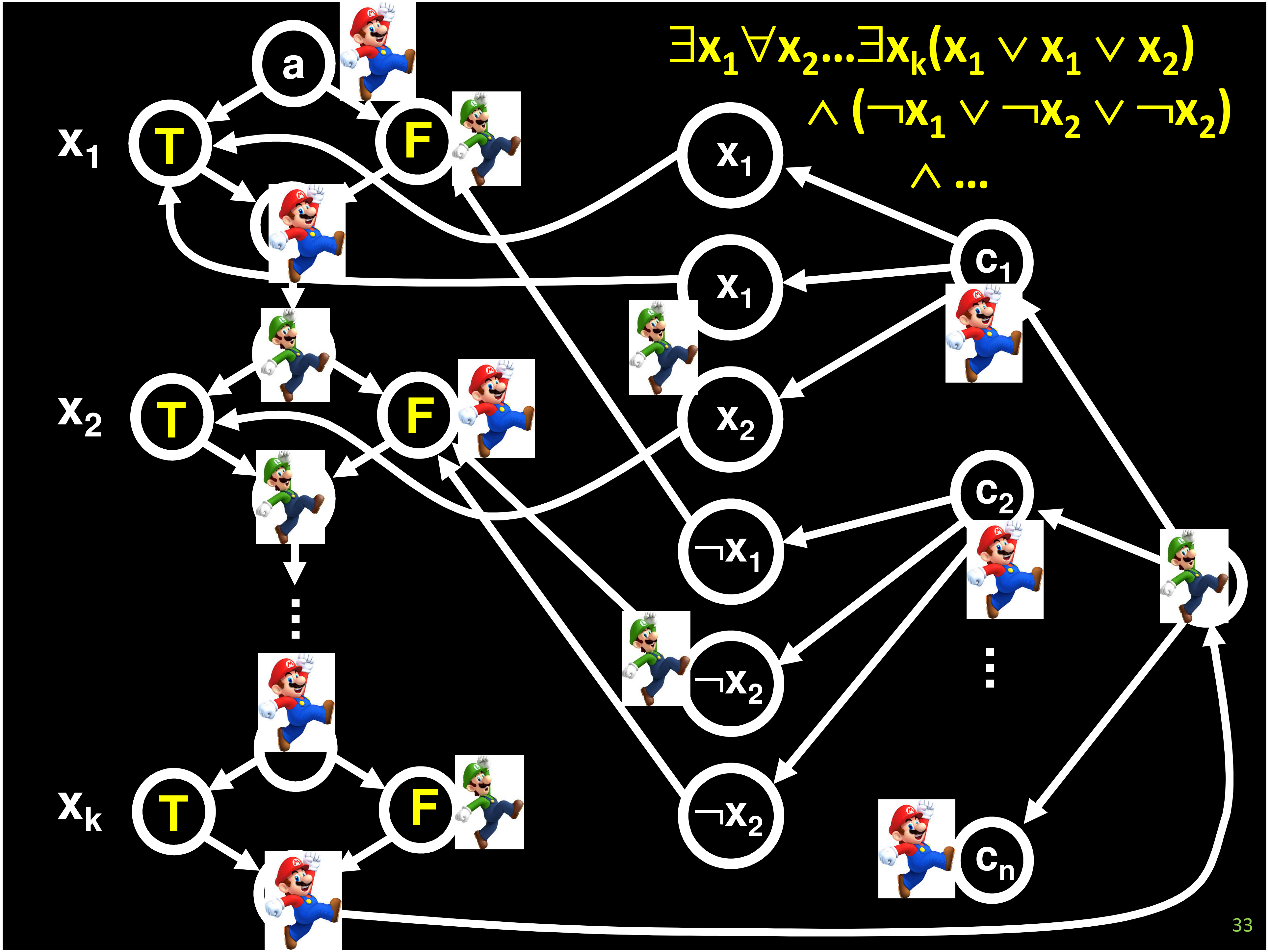
where ψ is in CNF: an AND of ORs of literals.

(Quantifiers alternate, and the last move is E's)

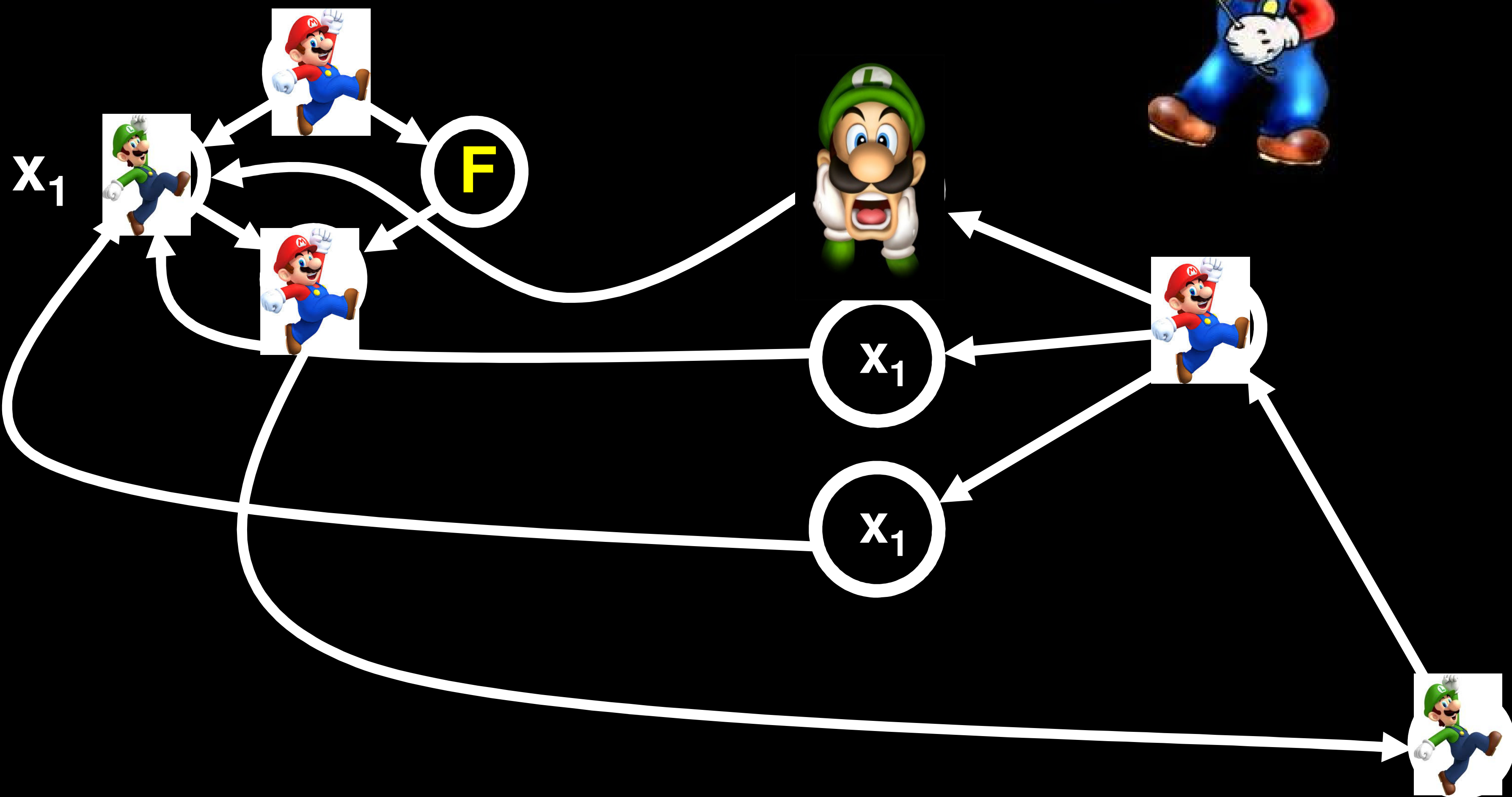
$$\exists x_1 \forall x_2 \ldots \exists x_k (x_1 \vee x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_2) \wedge \ldots$$



$$\exists x_1 \forall x_2 \ldots \exists x_k (x_1 \vee x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_2) \wedge \ldots$$



$$\exists x_1 \,[\, (x_1 \vee x_1 \vee x_1) \,]$$



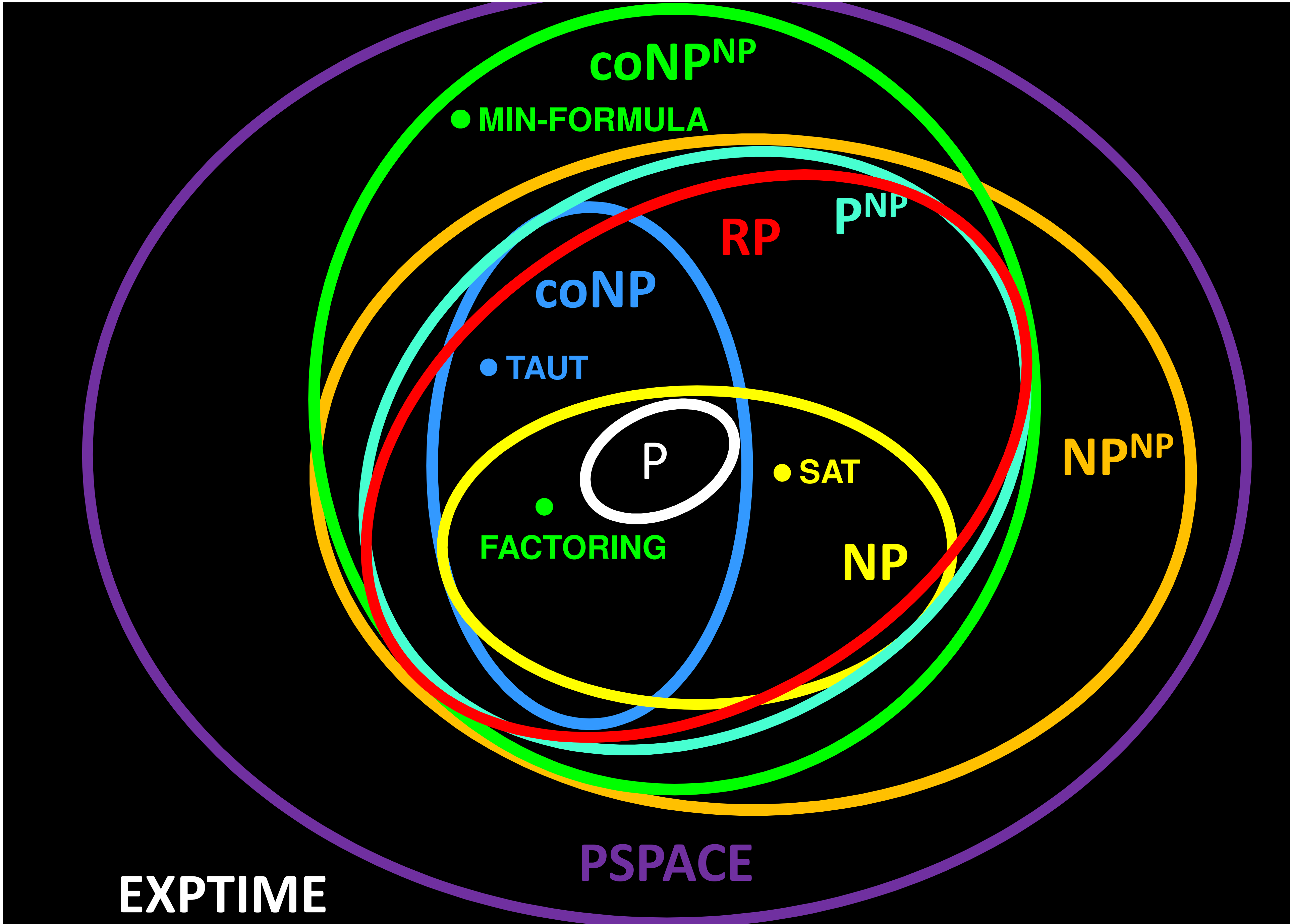
GG = { (G, a) | Player 1 has a winning strategy for geography on graph G starting at node a }

Theorem: GG is PSPACE-Complete

Question:
Is Chess a PSPACE complete problem?

No, because determining whether a player has a winning strategy takes CONSTANT time and space (OK, the constant is large…)

But generalized versions of Chess, GO, and Checkers (on n x n boards) can be shown to be PSPACE-hard


coNP^NP
MIN-FORMULA
P^NP
RP
coNP
TAUT
P
SAT
FACTORING
NP
NP^NP
PSPACE
EXPTIME

What's next?

A few possibilities…

CS161 – Design and Analysis of Algorithms

CS254 – Complexity Theory

CS354 – Topics in Circuit Complexity (next year)

CS266 – Parameterized Algorithms and Complexity (in 2016-17?)

Thank you!

For being a great class