PSPACE COMPLETENESS TBQF

THURSDAY April 17

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)

QUANTIFIED BOOLEAN FORMULAS

(in prenex normal form)

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

$$\forall x\, [\, x \vee \neg x \,]$$

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

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

Allow constants, 0 and 1, eg. $\forall x\, [\, 0 \vee \neg x \,]$

Wlog can assume we have = and => (why?)

Definition:

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

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

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

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

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

$$\forall x \exists y \, [\, (x \lor 0) \land (\neg x \lor \neg y) \,]$$

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

Theorem: TQBF is PSPACE-complete

TQBF $\in$ PSPACE

T(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 $\phi = \exists x\ \psi$, recursively call T on ψ, first with x = 0 and then with x = 1.
Accept iff either one of the calls accepts.

3. If $\phi = \forall x\ \psi$, recursively call T on ψ, first with x = 0 and then with x = 1.
Accept iff both of the calls accept.

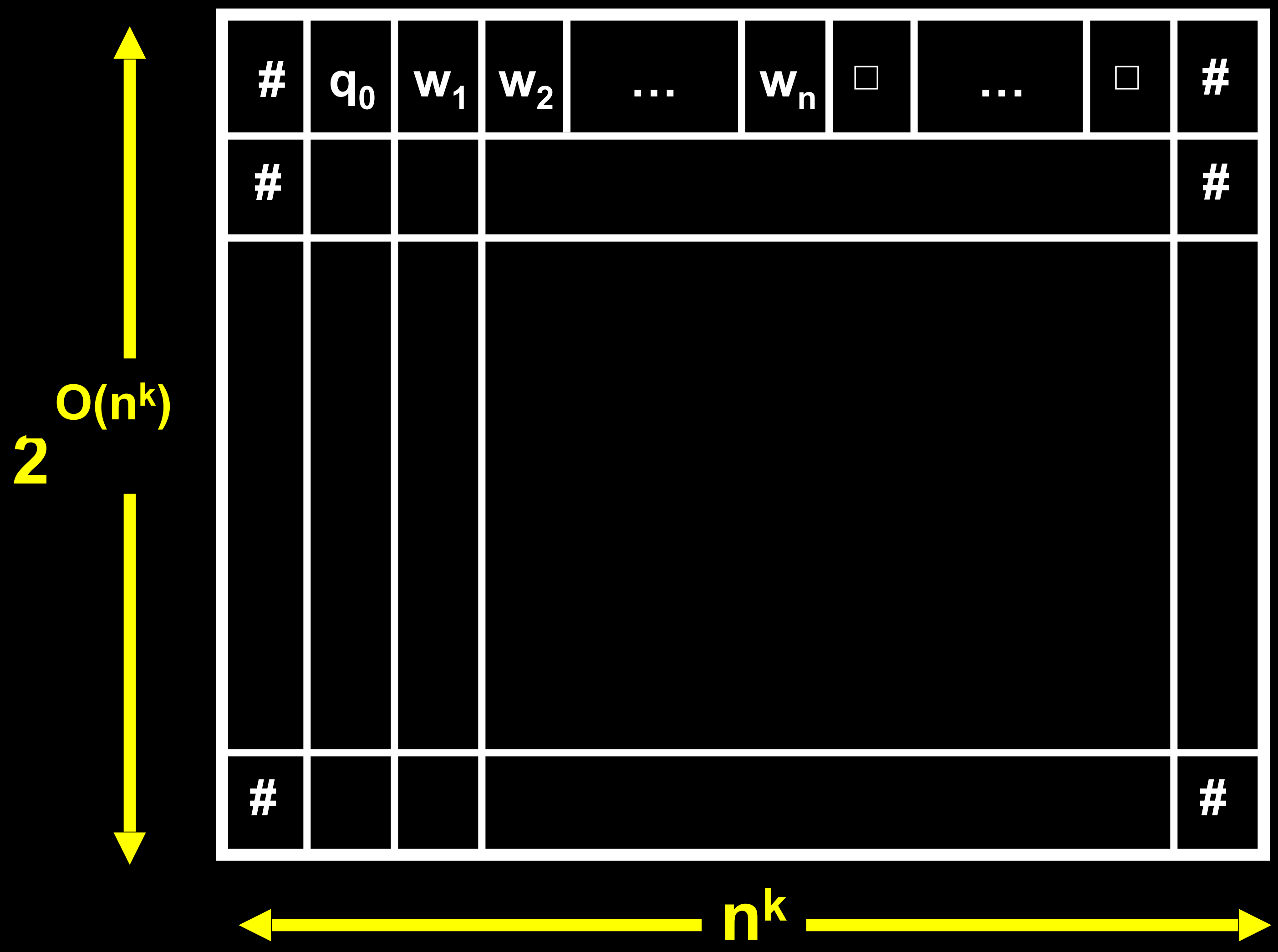
Claim: Every language A in PSPACE is polynomial time reducible to TQBF

We build a poly-time reduction from A to TQBF

The reduction turns a string w into a fully quantified Boolean formula ϕ that simulates the PSPACE machine for A on w

Let M be a deterministic TM that decides A in space n^k How do we know M exists?

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



We design ϕ to encode a simulation of M on w

ϕ will be true if and only if M accepts w

Given two collections of variables denoted c and d representing two configurations and t > 0, we construct a formula $\phi_{c,d,t}$

If we assign c and d to actual configurations,

$\phi_{c,d,t}$ will be true if and only if

M can go from c to d in t steps

We let $\phi = \phi_{c_{start}, c_{accept}, h}$, where $h = 2^{e\,s(n)}$ for a constant e chosen so that M has less than $2^{e\,s(n)}$ possible configurations on an input of length n

Here $s(n) = n^k$

We design ϕ to encode a simulation of M on w

ϕ will be true if and only if M accepts w

Given two collections of variables denoted c and d representing two configurations and t > 0, we construct a formula $\phi_{c,d,t}$

If we assign c and d to actual configurations, $\phi_{c,d,t}$ will say:

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

We let $\phi = \phi_{c_{start}, c_{accept}, h}$, where $h = 2^{e\, s(n)}$ for a constant e chosen so that M has less than $2^{e\, s(n)}$ possible configurations on an input of length n

Here $s(n) = n^k$

HIGH-LEVEL IDEA:

Encode the Algorithm of Savitch's Theorem with a Quantified Boolean Formula

If M uses n^k space,

then the QBF ϕ will have size $O(n^{2k})$

If we assign c and d to actual configurations, $\phi_{c,d,t}$ will say:

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

We let $\phi = \phi_{c_{start}, c_{accept}, h}$, where $h = 2^{e\,s(n)}$ for a constant e chosen so that M has less than $2^{e\,s(n)}$ possible configurations on an input of length n

Here $s(n) = n^k$

To construct $\phi_{c,d,t}$

use ideas of Cook-Levin plus Savitch:

Each cell in a configuration is associated with variables representing possible tape symbols and states.

Each config has n^k cells so and is encoded by $O(n^k)$ variables.

We will not have distinct variables for all cells

(Why?)

If t = 0 or 1, we can easily construct $\phi_{c,d,t}$:

$\phi_{c,d,t}$ = "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 variables representing c is equal to the corresponding one in d

"d follows from c in a single step of M"?

If t = 0 or 1, we can easily construct $\phi_{c,d,t}$:

$\phi_{c,d,t}$ = "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 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 that expresses that:
the contents of each triple of c's cells correctly yieds the contents of thr corresponding triple of d's cells.

If $t > 1$, we construct $\phi_{c,d,t}$ *recursively*:

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

/

$\exists x_1\ \exists x_2 \ldots \exists x_L$ $\quad L = O(n^k)$

But how long is this formula?

Every level of the recursion cuts t in half but roughly doubles the size of the formula (so back to length O(t))
So, we modify the formula to be:

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

This folds the 2 recursive sub-formulas into 1

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

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

- **Each recursive step adds a portion that is linear in the size of the configurations, so has size $O(s(n))$**
- **Number of levels of recursion is $\log h = O(s(n))$**
- **Hence, the size of ϕ is $O(s(n)^2)$**

PSPACE is often called the class of games

Formalizations of many popular games are PSPACE-Complete

THE FORMULA GAME (**FG**)

…is 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 ∃

A chooses values for variables quantified by ∀

Start at the leftmost quantifier

E wins if the resulting formula is true

A wins otherwise

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

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

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

Theorem: FG is PSPACE-Complete

Proof:

FG = TQBF

GEOGRAPHY

Two players take turns naming cities from anywhere in the world

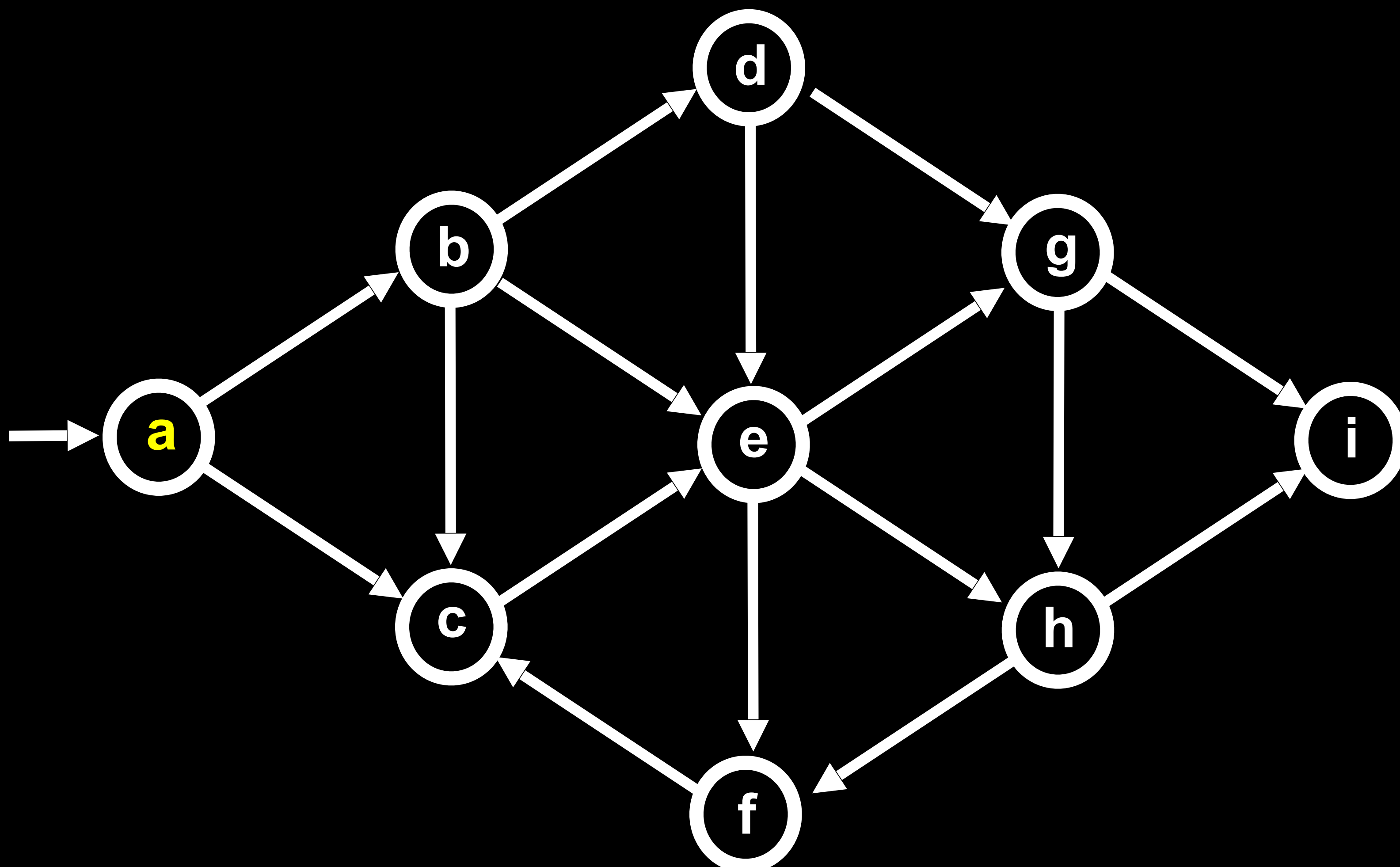
Each city chosen must begin with the same letter that the previous city ended with

Cities cannot be repeated

Austin → Nashua → Albany → York

Whoever cannot name any more cities loses

GENERALIZED GEOGRAPHY



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

Theorem: GG is PSPACE-Complete

GG ∈ PSPACE

WANT: Machine M that accepts (G,a)
⇔ Player 1 has a winning strategy on (G, a)

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

1. Remove node a and all edges touching it to get to a new graph G_1

2. For each of the nodes a_1, a_2, …, a_k that a originally pointed at, recursively call M(G_1, a_i)

3. If all of these accept, Player 2 has a winning strategy, so *reject*.
Otherwise, *accept.*

GG IS PSPACE-HARD

We show that FG $\leq_P$ GG

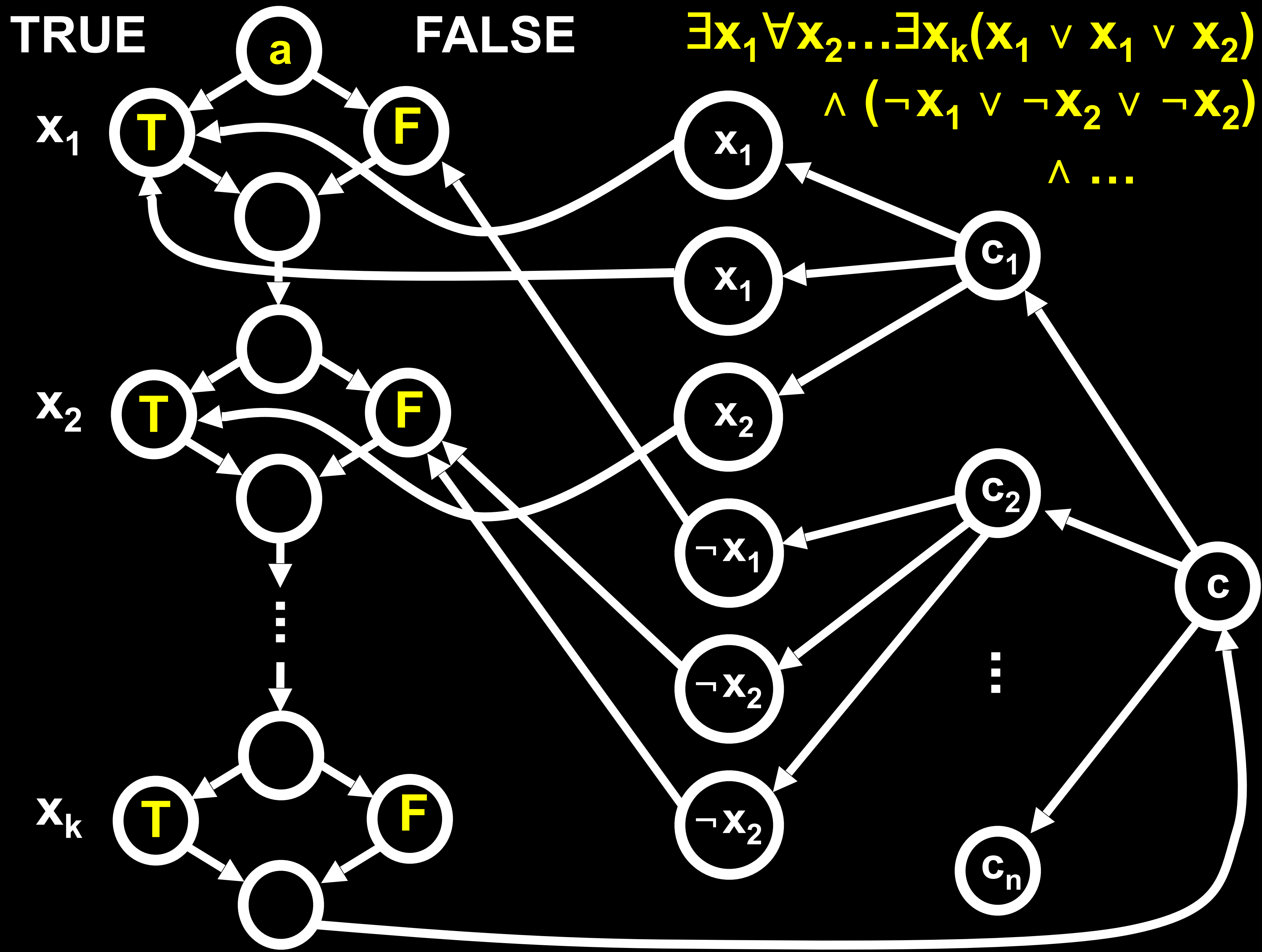
We convert a formula ϕ into (G, a) such that:

Player E has winning strategy in ϕ
if and only if
Player 1 has winning strategy in (G, a)

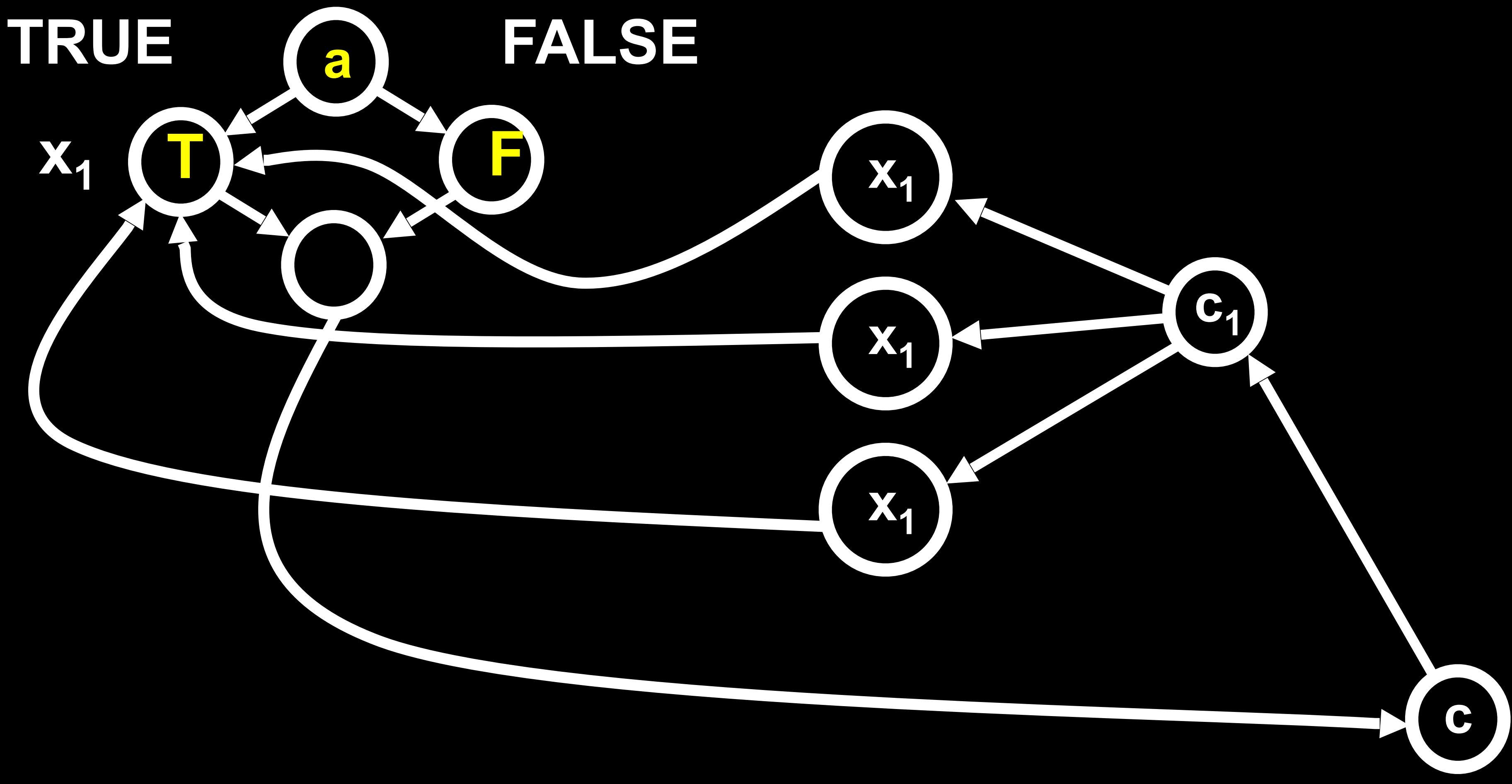
For simplicity we assume ϕ is of the form:

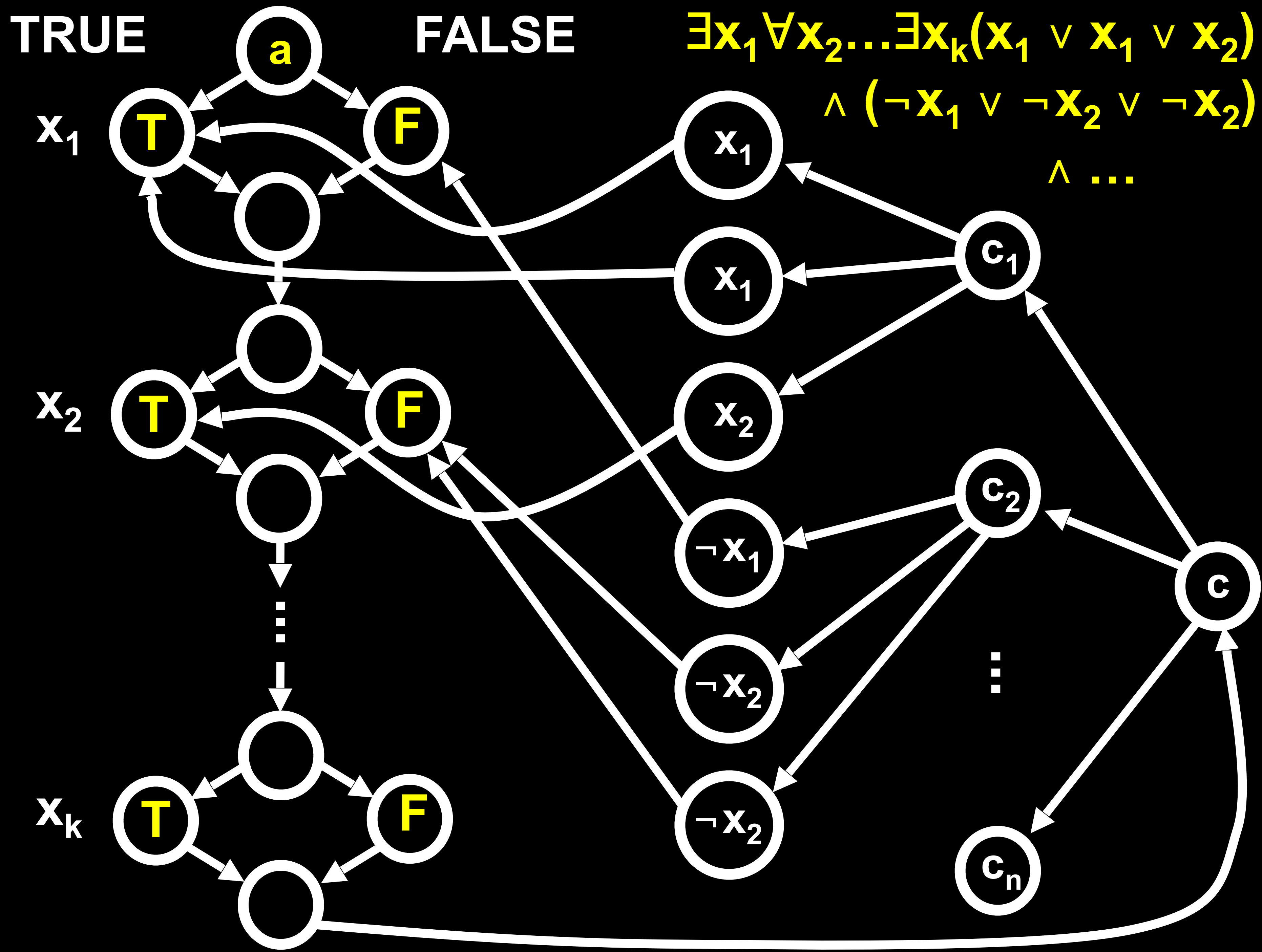
$$\phi = \exists x_1 \forall x_2 \exists x_3 \ldots \exists x_k [\psi]$$

where ψ is in cnf.
(Quantifiers alternate, and the last move is E's)


TRUE
FALSE
∃x1∀x2…∃xk(x1 ∨ x1 ∨ x2)
∧ (¬x1 ∨ ¬x2 ∨ ¬x2)
∧ …
a
x1
T
F
x2
T
F
xk
T
F
x1
x1
x2
¬x1
¬x2
¬x2
c1
c2
cn
c

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




TRUE
FALSE
a
$\exists x_1 \forall x_2 \ldots \exists x_k (x_1 \vee x_1 \vee x_2)$
$\wedge (\neg x_1 \vee \neg x_2 \vee \neg x_2)$
$\wedge \ldots$
x_1
T
F
x_2
T
F
x_k
T
F
x_1
x_1
x_2
$\neg x_1$
$\neg x_2$
$\neg x_2$
c_1
c_2
c_n
c

GG = { (G, a) | Player 1 has a winning strategy for generalized geography played 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 n x n GO, Chess and Checkers can be shown to be PSPACE-hard