PSPACE-complete Problems

Definition: A language B is **PSPACE-complete** if

- $B \in$ PSPACE, and
- any $A \in$ PSPACE is polynomial-time reducible to B.

If B satisfies only the second condition, then B is **PSPACE-hard**.

PSPACE-complete Problems

Question: Why polynomial-time reducible instead of polynomial-space reducible?

Remark: Papadimitriou uses “logspace-reducible” which means essentially the same thing (but not exactly).

Completeness

Let $\mathcal{C}$ be a complexity class.

Definition: A language A is **$\mathcal{C}$-complete** if

- $A \in \mathcal{C}$, and
- any $B \in \mathcal{C}$ is "easily" reducible to B.

What should "easily" mean?

Completeness

Motivation

- A complete problem is the hardest in its class,
- and any other problem is "easily" reduced to it.

Goal: An "easy" solution to a complete problem yields an "easy" solution for any problem in the class.

Remark: A "hard" reduction to a complete problem would not yield an "easy" solution to the remaining problems in the class.

Completeness

Remark: Polynomial-space reduction would have undesirable consequences.

- Almost all problems in PSPACE would be PSPACE-complete,
- because almost every pair of languages in PSPACE are poly-space reducible.

Notion of reducibility must be more limited than complexity class itself.

The TQBF problem

Recall that a *boolean formula* is an expression involving

- Boolean variables
- Constants 0 and 1
- Boolean operators $\wedge$, $\vee$, and $\neg$.

We now introduce a more general type of Boolean formula.

Quantifiers

$$\forall x \; \phi$$

Means that for *every* value of x, ϕ is true.

- $\forall$ is called the **universal quantifier**.
- variable x is **bound** to the quantifier.

Quantifiers

$$\exists x \ \phi$$

Means that for *some* value of x, ϕ is true.

- $\exists$ is called the **existential quantifier**.
- variable x is **bound** to the quantifier.

Example

For the natural numbers,

$$\forall x\ [x + 1 > x]$$

means that the successor of every number is greater than that number.

This is *true*.

Example

For the natural numbers,

$$\exists y\ [y + y = 3]$$

means that some natural number is half of 3.

This is

- *false* for the natural numbers.
- *true* for the real numbers.

The **universe** from which values are drawn is important!

Example

A formula can have more than one quantifier:

$$\forall x \; \exists y \; [y > x]$$

Every natural number has another number larger than it.

This is *true*.

Example

Order of quantifiers matters:

$$\forall x\ \exists y\ [y > x]$$

Every natural number has another number larger than it.

$$\exists y\ \forall x\ [y > x]$$

There is some number such that every number is greater than it.

This is *false*.

Quantifiers

Quantifiers

- can appear anywhere.
- apply to fragment within next parenthesis or brackets
- called the quantifier's **scope**

Quantifiers

A formula is in **prenex normal form** if

- every quantifier has the entire formula as its scope.
- assumed unless stated otherwise

Quantified Boolean Formulas

Boolean formulas with quantifiers are

- called **Quantified Boolean Formulas**
- universe is $\{0, 1\}$.

Quantified Boolean Formulas

Example:

$$\phi = \forall x \; \exists y \; [(x \vee y) \wedge (\overline{x} \vee \overline{y})]$$

- This formula is true, but
- it would be false if the quantifiers were reversed.

Quantified Boolean Formulas

Definition: A quantified Boolean formula is **fully quantified** if every variable appears in the scope of a quantifier.

Quantified Boolean Formulas

Remark: A full quantified Boolean formula is always true or false.

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

is fully quantified, and is always true.

If we remove the first quantifier:

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

- ϕ is no longer fully quantified
- ϕ is no longer true nor false

TQBF

Definition:

$$\text{TQBF} = \{\langle\phi\rangle | \phi \text{ is a true, fully quantified Boolean formula}\}$$

TQBF

TQBF $=$
$\{\langle\phi\rangle|\phi$ is a true, fully quantified Boolean formula$\}$

Theorem: TQBF is PSPACE-complete.

TQBF: idea

To show that TQBF $\in$ PSPACE we give an algorithm:

- assigns values to variables
- evaluates truth of formula

TQBF: idea

Must show that any $A \in$ PSPACE reduces to TQBF in polynomial time.

- Take a polynomial-space bounded TM for A,
- produce a polynomial-time reduction
- carrying a string to a quantified Boolean formula ϕ
- ϕ encodes a simulation
- ϕ is true of the machine accepts

TQBF: bad idea

Imitate proof of Cook-Levin theorem

- Construct ϕ that simulates M on w.
- The tableau for M on w has
  - width $O(n^k)$, space used by M
  - height exponential in n^k ...
  - because M can run for exponential time in $O(n^k)$ space.
- direct simulation yields
  - exponential-size circuit
  - exponential-size formula
  - no poly-time reduction
- instead, use something like Savitch's theorem.

TQBF

T: on input $\langle \phi \rangle$, a fully quantified Boolean formula

- If ϕ contains no quantifiers, then
  - it contains only constants,
  - we can evaluate it, so
  - if it's true *accept*, if false *reject*.
- if $\phi = \exists x \psi$, recursively call T
  - with 0 substituted for x
  - with 1 substituted for x
  - if either accepts, *accept*, otherwise *reject*.
- if $\phi = \forall x \psi$, recursively call T
  - with 0 substituted for x
  - with 1 substituted for x
  - if both accept, *accept*, otherwise *reject*.

TQBF

Claim: Algorithm T decides TQBF in polynomial space.

- T obviously decides TQBF.
- depth of recursion is $\leq$ number of variables.
- each level store one variable
- total space is $O(m)$, where m is number of variables in ϕ
- therefore T runs in linear space.

TQBF

Claim: Algorithm T is PSPACE-hard.

Let A be a language

- decided by TM M
- in space $O(n^k)$ for constant k.

We give a polynomial-time reduction from A to TQBF.

TQBF

Claim: Algorithm T is PSPACE-hard.

- we map input string w
- to boolean formula ϕ
- that is true IFF M accepts w.

TQBF

Claim: Algorithm T is PSPACE-hard.

Convenient to solve a more general problem.

Given

- configurations c_1, c_2
- $t > 0$

We construct $\phi_{c_1,c_2,t}$ such that

- $\phi_{c_1,c_2,t}$ is true iff ...
- M can go from c_1 to c_2 in $\leq t$ steps.

We then look at $\phi_{c_s,c_a,h}$, where

- c_s is starting configuration
- c_a is accepting configuration
- $h = 2^{df(n)}$ for constant d is max number of configurations on input of length n.

TQBF

The formula (as in proof of Cook-Levin)

- encodes contents of tape cells with variables
- each cell has one variable for each tape symbol and state
- variable indicates cell contents (light)
- each cell requires constant number of variables
- each configuration has $O(n^k)$ cells
- *ergo*, each configuration is encoded by $O(n^k)$ variables.

TQBF

When $t = 1$, $\phi_{c_1,c_2,t}$ is easy.

The formula says:

- either $c_1 = c_2$, or
- c_1 yields c_2 in one step ...
- by wiring triples *à la* Cook-Levin.

TQBF

When $t > 1$, we construct $\phi_{c_1,c_2,t}$ recursively.

Simple (but not exactly correct) idea:

$$\phi_{c_1,c_2,t} = \exists m_1 \left[\phi_{c_1,m1,\lceil \frac{t}{2} \rceil}, \phi_{m_1,c_2,\lceil \frac{t}{2} \rceil} \right]$$

Where

- m_1 represents a configuration of M.
- $\exists m_1$ shorthand for $\exists x_1, \ldots, x_\ell$
- where $\ell = O(n^k)$, and
- $x_1, \ldots, x_\ell$ are the variables that encode m_1.

TQBF

$$\phi_{c_1,c_2,t} = \exists m_1 \left[\phi_{c_1,m1,\lceil\frac{t}{2}\rceil}, \phi_{m_1,c_2,\lceil\frac{t}{2}\rceil}\right]$$

This formula states:

- M can go from c_1 to c_2 in $\leq t$ steps if
- an intermediate configuration m_1 exists . . .
- such that M can go from c_1 to m_1 in $\leq \lceil t/2 \rceil$ steps . . .
- and from m_1 to c_2 in $\leq \lceil t/2 \rceil$ steps.

TQBF

One problem:

- $\phi_{c_1,c_2,t}$ has the correct value.
- Every level of recursion cuts t in half, but
- roughly doubles size of formula.
- Final size is roughly $t = 2^{df(n)}$.

That's too big.

TQBF

Fix the size problem by bringing in the $\forall$ symbol.

Let

$$\phi_{c_1,c_2,t} = \\ \exists m_1 \, \forall (c_1, c_2) \in \{(c_1, m_1), (m_1, c_2)\} \, \left[\phi_{c_3,c_4,\lceil \frac{t}{2} \rceil} \right].$$

- this formula says the same thing
- the new variables c_1 and c_2 "fold" the two recursive formulas into one

TQBF

The expression

$$\exists m_1 \, \forall (c_1, c_2) \in \{(c_1, m_1), (m_1, c_2)\}$$

is not really a quantified Boolean formula.

TQBF

Just rewrite

$$\phi_{c_1,c_2,t} = \exists m_1 \, \forall (c_1,c_2) \in \{(c_1,m_1),(m_1,c_2)\} \, \left[\phi_{c_3,c_4,\lceil \frac{t}{2} \rceil}\right].$$

as

$$\phi_{c_1,c_2,t} = \exists m_1 \, \forall (c_1,c_2) \\ \left[((c_3,c_4)=(c_1,m_1) \vee (c_3,c_4)=(c_2,m_2)) \to \phi_{c_3,c_4,\lceil \frac{t}{2} \rceil}\right]$$

Then replace $=, \to$ with $\wedge, \vee, \neg$.

TQBF

$$\phi_{c_1,c_2,t} = \exists m_1\ \forall(c_1,c_2)$$
$$\left[((c_3,c_4)=(c_1,m_1) \vee (c_3,c_4)=(c_2,m_2)) \rightarrow \phi_{c_3,c_4,\lceil \frac{t}{2} \rceil}\right]$$

How big is $\phi_{c_s,c_a,2^{df(n)}}$?

- recursion adds portion to formula linear in size of configuration
- and thus of size $O(f(n))$.
- number of recursions is $\log 2^{df(n)} = O(f(n))$.
- size of resulting formula is $O(f^2(n))$.