VAS-Reachability is hard for EXPSPACE
VAS-Reachability is hard for EXPSPACE

\[
\text{EXPSPACE} = \bigcup_{c>0} \text{SPACE}(2^{n^c})
\]
• Vector Addition System (VAS): a tuple
  \((S, V_1, V_2, \ldots, V_n)\)
  where
  – \(S\) is \(k\)-dimensional vectors of natural numbers
  – the \(V_i\) are \(k\)-dimensional vectors of integers

• VAS-Reachability: Given a VAS and \(k\)-tuple \(F\) of natural numbers, do there exist vectors \(W_1, \ldots, W_m\) of natural numbers such that
  – \(S = W_1, F = W_m\) and
  – for each \(i\), \(W_{i+1} = W_i + V_j\) for some \(j\)
Outline for Today

• Build-up to EXPSPACE-hardness of VAS-Reachability:
  – Turing Machines (TM) vs Counter Machines (CMS)
  – Vector Addition Systems (VAS’s) vs Parallel Programs (PP)
  – A Parallel Program Counter Data Structure

• We’ll prove Lipton’s theorem by showing that CM-Acceptance-in-space-$2^n \leq_m$ PP-Acceptance (here, $\leq_m$ means “is poly-time reducible to”)


Counter Machines (CMs)

Adapted from: The Nature of Computation by Chris Moore
Counter Machines (CMs)

• CM counters have three operations:
  – increment
  – decrement
  – test for 0
TM vs CM

• Let L be decided by a TM M. Then there is a 3-counter TM C that decides L.

• If M uses space $s(n) = \Omega(n)$ then C's counters have value at most $2^{s(n)}$.

• (Roughly, the contents of M’s tapes, which are in binary, can be stored in unary in C’s counters)

• (Could we simulate counters using vector additions?)
Outline

• Build-up to EXPSPACE-hardness of VAS-Reachability:
  – Turing Machines (TMs) vs Counter Machines (CMS)
  – Vector Addition Systems (VAS’s) vs Parallel Programs (PPs)
  – A Parallel Program Counter Data Structure
VASSs and Parallel Programs

• A VAS is a primitive type of nondeterministic parallel program
• A nicer programming language: Parallel Programs
Parallel Programs

• Collection of flowcharts with four types of statements
• Initially, vector $x$ is 0 and all flowcharts are at Start

\[ \begin{align*}
  x &\leftarrow x_1 + c_1 \\
  \cdots \\
  x_n &\leftarrow x_n + c_n \\
\end{align*} \]

where each $c_i$ is 0, 1, or -1.
Parallel Programs

• At each step, follow any applicable transition from the current state of any flowchart
  – Transitions from Guess states are always applicable
  – Transitions from Assignment states are applicable iff \( x_i + c_i \geq 0 \)

• The program ends when some flowchart reaches Accept
• *Parallel Program size* is the total number of statements in all of the flowcharts

• PP-Acceptance: Given a parallel program, can it reach an Accept state?

• Lipton’s Lemma: PP-Acceptance $\leq_m$ VAS-Reachability
Outline

• Build-up to EXPSPACE-hardness of VAS-Reachability:
  – Turing Machines (TMs) vs Counter Machines (CMS)
  – Vector Addition Systems (VAS’s) vs Parallel Programs (PPs)
  – A Parallel Program Counter Data Structure
Counter Machines vs Parallel Programs

ASSIGNMENT

\[
\begin{align*}
\forall i & : x_i \leftarrow x_i + c_i \\
& \text{where each } c_i \text{ is } 0, 1, \text{ or } -1.
\end{align*}
\]

No test for 0
Simulating a counter with PP statements

• Increment and decrement can be simulated using assignment statements:
  • increment $x_i$: $x_i \leftarrow x_i + 1$; $x_j \leftarrow x_j + 0$ for $j <> i$
  • decrement $x_i$: $x_i \leftarrow x_i - 1$; $x_j \leftarrow x_j + 0$ for $j <> i$

• We’ll show how to simulate test for 0 using guess and assignment statements
• To do this, we’ll assume a bound, say $m$, on the max value that a counter can have
PP simulation of m-bounded counter r

- Use both r and its complement $r'$ to represent the counter
- Maintain invariant: $r + r' = m$
- Initialize: $r = 0$, $r' = m$

- Increment: $r \rightarrow r + 1$, $r' \rightarrow r' - 1$
- Decrement: $r \rightarrow r - 1$, $r' \rightarrow r' + 1$
PP simulation of m-bounded counter r

• Test for 0 by testing that \( r' \geq m! \):

  - Guess
  - \( r \leftarrow r - 1 \)
  - \( r \leftarrow r + 1 \)
  - No

  - \( r \leftarrow r + 1 \)
  - \( r' \leftarrow r' - 1 \)
  - ... (Repeat m times)
  - Yes

• Correctness: From the Guess node, it's possible to reach the "No" exit iff \( r > 0 \), and it's possible to reach to reach the "yes" exit iff \( r = 0 \)
Problem: The Initialize and Test for 0 components have size proportional to m
We need to implement a $2^n$-bounded counter with a parallel program of size $O(n)$
PP simulation of $2^{2^{k+1}}$-bounded counter 

- Let $A_1 = 2$ and for $k > 1$ let $A_{k+1} = A_k^2$
- Then $A_i = 2^i$

Recursive construction:
- Assume that $x_k$ and $y_k$ simulate $A_k$-bounded counters (using also $x_k'$ and $y_k'$)
- Use $x_k$ and $y_k$ to build a $A_{k+1}$-bounded counter
Test for 0 of $A_{k+1}$-bounded counter $r$
Test for 0 of $A_{k+1}$-bounded counter $r$

Side effect:
- $r = A_{k+1}$ and $r' = 0$
Test for 0 of $A_{k+1}$-bounded counter $r$

Side effect:
- $r = A_{k+1}$ and $r' = 0$
Test for 0 of $A_{k+1}$-bounded counter $r$

Program size?

with side effect

with side effect

with side effect
Test for 0 of $A_{k+1}$-bounded counter $r$

Program size?

- $PP\text{-size}(k+1) = 2PP\text{-size}(k) + 4$
- Program size is exponential in $k$: better than our first try, but still not good enough
So far we have:
- recursive construction of test for 0, with side effect
- size of test for 0 is exponential in $k$
- size of counter initialization is double exponential in $k$

Remaining tasks:
- reduce the size of the test for 0 program to $O(k)$
- develop a test for 0 without side effect
- reduce the size of the initialization
Test for 0 without side effect

\[ \tilde{z} = 0 \]

\[ \tilde{z}' = 0 \] (Impossible)
Shaving Another Exponential Off the Test for 0

• Subroutine trick:
• Don’t have separate programs for testing whether \( x_k' \) is 0 and \( y_k' \) is 0
• Instead, have just one subroutine to check that some other variable, say \( S_k \), is 0
• Call the subroutine, passing \( y_k' \) as a parameter
• Call the subroutine, passing \( x_k' \) as a parameter
Outline

• Build-up to EXPSPACE-hardness of VAS-Reachability:
  – Turing Machines (TMs) vs Counter Machines (CMS)
  – Vector Addition Systems (VAS’s) vs Parallel Programs (PPs)
  – A Parallel Program Counter Data Structure

• We’ll prove Lipton’s theorem by showing that
  \[ \text{CM-Acceptance-in-space-}2^{2^n} \leq_m \text{PP-Acceptance} \]
CM-Acceptance-in-space-\(2^{2n}\) \(\leq_m\) PP-Acceptance

Proof structure:

• Let L accepted by counter machine C in \(2^{2n}\) space
• On input x, construct a Parallel Program P that accepts if and only if C accepts x
  – use the PP counter data structure to simulate C’s counters
  – initially set one of the counters to x
Summary

- Lipton uses several nice tricks to prove EXPSPACE hardness of VAS-Reachability
  - work with CM and PP programming models, rather than TMs and VASs
  - clever use of nondeterminism (Guess nodes) to test for 0
  - nice recursive construction to handle large numbers
  - use of “subroutines” to further reduce program size
Still, the complexity of VAS-Reachability is still wide open! It’s decidable, but no better upper bound is known.

Upper bounds are known for related problems: for a given $i$ with $S_i$ initially 0, does $S_i$ ever become greater than 0? (Is it possible to “produce” a given chemical species? Is it possible for a PP to accept?)

Next Class

• Introduction to Randomized Complexity Classes
• Reading: Arora-Barak 7.1-7.3