Lecture 6
Recap

• Two lectures ago: Our first application of the sum-check protocol.
  • An IP for #SAT with a polynomial-time verifier.
  • $P$ ran in time exponential in the input size.
  • But the fastest known algorithm for this problem requires exponential time.
    • So can’t really hope for a faster prover for this problem.

• Last lecture we saw some doubly-efficient IPs.
  • $V$ runs in linear time.
  • $P$ runs in polynomial time.
    • In fact, we achieved “super-efficiency”.
    • meaning $P$ ran the fastest known algorithm for the problem, and then did a low-order amount of additional work to prove correctness.
    • Counting triangles, matrix multiplication.
Today: A General-Purpose Doubly-Efficient Interactive Proof
General-Purpose Interactive Proof and Argument Implementations

• Start with a computer program written in high-level programming language (C, Java, etc.)

• Step 1: Turn the program into an equivalent model amenable to probabilistic checking.
  • Typically some type of arithmetic circuit.
  • Called the Front End of the system.

• Step 2: Run an interactive proof or argument on the circuit.
  • Called the Back End of the system.
The GKR Protocol: Overview

Front End

P and V run interactive proof or argument system (back end) on circuit
### Sources of Prover Overhead in VC Systems

<table>
<thead>
<tr>
<th>Source of Overhead</th>
<th>( P ) Overhead vs. Native (Crude Estimate)</th>
<th>Slowdown Depends On...</th>
</tr>
</thead>
</table>
| Front End          | (ratio of circuit size to number of machine steps of original program) \(1x-10,000x\) | • How amenable is the high-level computer program is to representation via circuits?  
• What type of circuits can the back-end handle? |
| Back-End           | (ratio of \( P \) time to evaluating circuit gate-by-gate) \(10x-1,000x\) | • Varies by back-end and computation structure (e.g., data parallel?) |
The GKR Protocol

A General-Purpose Doubly-Efficient Interactive Proof
The GKR Protocol: Overview

Layer 1

Layer 2

Layer 3

Layer 4

F_2 circuit
The GKR Protocol: Overview

Layer 1

Layer 2

Layer 3

Layer 4

P starts the conversation with an answer (output).

F₂ circuit
The GKR Protocol: Overview

Layer 1

Layer 2

Layer 3  V sends series of challenges. P responds with info about next circuit level.

Layer 4  a₁  a₂  a₃  a₄

F₂ circuit
The GKR Protocol: Overview

Challenges continue, layer by layer down to the input.
The GKR Protocol: Overview

Layer 1

Layer 2

Layer 3

Layer 4

Finally, $P$ says something about the (multilinear extension of the) input.
Notation

• Assume layers $i$ and $i + 1$ of $C$ have $S$ gates each.
  • Assign each gate a binary label ($\log S$ bits).
• Let $W_i(\alpha) : \{0,1\}^{\log S} \rightarrow F$ output the value of gate $\alpha$ at layer $i$. 
Notation

• Assume layers $i$ and $i + 1$ of $C$ have $S$ gates each.
  • Assign each gate a binary label ($\log S$ bits).
• Let $W_i(a): \{0,1\}^{\log S} \rightarrow F$ output the value of gate $a$ at layer $i$.
• Let $\text{add}_i(a, b, c): \{0,1\}^{3 \log S} \rightarrow F$ output 1 iff
  $(b, c) = (\text{in}_1(a), \text{in}_2(a))$ and gate $a$ at layer $i$ is an addition gate.
Notation

• Assume layers $i$ and $i + 1$ of $C$ have $S$ gates each.
  • Assign each gate a binary label ($\log S$ bits).
• Let $W_i(a): \{0,1\}^{\log S} \rightarrow F$ output the value of gate $a$ at layer $i$.
• Let $\text{add}_i(a, b, c): \{0,1\}^{3 \log S} \rightarrow F$ output 1 iff
  $(b, c) = (\text{in}_1(a), \text{in}_2(a))$ and gate $a$ at layer $i$ is an addition gate.
• Let $\text{mult}_i(a, b, c): \{0,1\}^{3 \log S} \rightarrow F$ output 1 iff
  $(b, c) = (\text{in}_1(a), \text{in}_2(a))$ and gate $a$ at layer $i$ is a multiplication gate.
The GKR Protocol: Overview

Layer 1

Layer 2

Layer 3

Layer 4

F_2 circuit

add_2(0, (0, 0), (0, 1)) = 1
add_2(1, (1, 0), (1, 1)) = 1
The GKR Protocol: Overview

Layer 1

Layer 2

Layer 3

Layer 4

F₂ circuit

\[
\text{mult}_3((0,0), (0, 0), (0, 0)) = 1 \\
\text{mult}_3((0,1), (0, 1), (0, 1)) = 1 \\
\text{mult}_3((1,0), (1, 0), (1, 0)) = 1 \\
\text{mult}_3((1,1), (1, 1), (1, 1)) = 1
\]
GKR Protocol: Goal of Iteration i

• Iteration $i$ starts with a claim from $P$ about $\tilde{W}_i(r_1)$ for a random point $r_1 \in F^{\log S}$.

• Goal: Reduce this to a claim about $\tilde{W}_{i+1}(r_2)$ for a random point $r_2 \in F^{\log S}$.

• Observation: $W_i(a) =$

$$\sum_{b,c \in \{0,1\}^{\log S}} \left[ \text{add}_i(a, b, c)(W_{i+1}(b) + W_{i+1}(c)) + \text{mult}_i(a, b, c)(W_{i+1}(b) \cdot W_{i+1}(c)) \right]$$

• Hence, the following equality holds as formal polynomials:

$$\tilde{W}_i(a) =$$

$$\sum_{b,c \in \{0,1\}^{\log S}} \left[ \text{add}_i(a, b, c)(\tilde{W}_{i+1}(b) + \tilde{W}_{i+1}(c)) + \text{mult}_i(a, b, c)(\tilde{W}_{i+1}(b) \cdot \tilde{W}_{i+1}(c)) \right]$$
GKR Protocol: Goal of Iteration i

• So $V$ applies sum-check protocol to compute

• $\tilde{W}_i(r_1) = \sum_{b,c \in \{0,1\}^\log s} g(b, c)$, where:

\[
g(b, c) = \text{add}_i(r_1, b, c)(\tilde{W}_{i+1}(b) + \tilde{W}_{i+1}(c)) + \text{mult}_i(r_1, b, c)(\tilde{W}_{i+1}(b) \cdot \tilde{W}_{i+1}(c))
\]
GKR Protocol: Goal of Iteration $i$

- So $V$ applies sum-check protocol to compute
- $\tilde{W}_i(r_1) = \sum_{b, c \in \{0, 1\}^{\log s}} g(b, c)$, where:

$$g(b, c) = \tilde{a}_{di}(r_1, b, c)(\tilde{W}_{i+1}(b) + \tilde{W}_{i+1}(c))$$
$$+ \tilde{m}_{ult}(r_1, b, c)(\tilde{W}_{i+1}(b) \cdot \tilde{W}_{i+1}(c))$$

- At end of sum-check protocol, $V$ must evaluate $g(r_2, r_3)$. 
GKR Protocol: Goal of Iteration $i$

- So $V$ applies sum-check protocol to compute
- $\tilde{W}_i(r_1) = \sum_{b,c \in \{0,1\}^\text{log } s} g(b, c)$, where:
  \[
  g(b, c) = \text{add}_i(r_1, b, c)(\tilde{W}_{i+1}(b) + \tilde{W}_{i+1}(c)) \\
  + \text{mult}_i(r_1, b, c)(\tilde{W}_{i+1}(b) \cdot \tilde{W}_{i+1}(c))
  \]
- At end of sum-check protocol, $V$ must evaluate $g(r_2, r_3)$.
- Let us assume $V$ can compute $\text{add}_i(r_1, r_2, r_3)$ and $\text{mult}_i(r_1, r_2, r_3)$ unaided in time $\text{polylog}(n)$.
- Then $V$ only needs to know $\tilde{W}_{i+1}(r_2)$ and $\tilde{W}_{i+1}(r_3)$ to complete this check.
GKR Protocol: Goal of Iteration $i$

- So $V$ applies sum-check protocol to compute
- $\widetilde{W}_i(r_1) = \sum_{b,c \in \{0,1\}^\log s} g(b, c)$, where:
  $$g(b, c) = \text{add}_i(r_1, b, c)(\widetilde{W}_{i+1}(b) + \widetilde{W}_{i+1}(c))$$
  $$+ \text{mult}_i(r_1, b, c)(\widetilde{W}_{i+1}(b) \cdot \widetilde{W}_{i+1}(c))$$
- At end of sum-check protocol, $V$ must evaluate $g(r_2, r_3)$.
- Let us assume $V$ can compute $\text{add}_i(r_1, r_2, r_3)$ and $\text{mult}_i(r_1, r_2, r_3)$ unaided in time $\text{polylog}(n)$.
- Then $V$ only needs to know $\widetilde{W}_{i+1}(r_2)$ and $\widetilde{W}_{i+1}(r_3)$ to complete this check.
- Iteration $i + 1$ is devoted to computing these values.
Remaining Issue: Reducing to Verification of a Single Point

• There is one remaining problem: we don’t want to have to separately verify both $\tilde{W}_{i+1}(r_2)$ and $\tilde{W}_{i+1}(r_3)$ in iteration $i + 1$.

• Solution: Reduce verifying both of the above values to verifying $\tilde{W}_{i+1}(r_4)$ for a single point $r_4$. 
Remaining Issue: Reducing to Verification of a Single Point

• There is one remaining problem: we don’t want to have to separately verify both $\tilde{W}_{i+1}(r_2)$ and $\tilde{W}_{i+1}(r_3)$ in iteration $i + 1$.

• Solution: Reduce verifying both of the above values to verifying $\tilde{W}_{i+1}(r_4)$ for a single point $r_4 \in F^{\log S}$.

\[ \tilde{W}_{i+1} \]
Remaining Issue: Reducing to Verification of a Single Point

• There is one remaining problem: we don’t want to have to separately verify both \( \tilde{W}_{i+1}(r_2) \) and \( \tilde{W}_{i+1}(r_3) \) in iteration \( i + 1 \).

• Solution: Reduce verifying both of the above values to verifying \( \tilde{W}_{i+1}(r_4) \) for a single point \( r_4 \in F^{\log S} \).
Remaining Issue: Reducing to Verification of a Single Point

• There is one remaining problem: we don’t want to have to separately verify both $\tilde{W}_{i+1}(r_2)$ and $\tilde{W}_{i+1}(r_3)$ in iteration $i+1$.

• Solution: Reduce verifying both of the above values to verifying $\tilde{W}_{i+1}(r_4)$ for a single point $r_4 \in F^{\log S}$. 

\begin{itemize}
  \item Challenge line $\lambda$
  \item Extended Hypercube $F^{\log S}$
  \item Boolean Hypercube $\{0,1\}^{\log S}$
\end{itemize}
Remaining Issue: Reducing to Verification of a Single Point

- There is one remaining problem: we don’t want to have to separately verify both \( \tilde{W}_{i+1}(r_2) \) and \( \tilde{W}_{i+1}(r_3) \) in iteration \( i + 1 \).
- Solution: Reduce verifying both of the above values to verifying

\[ \tilde{W}_{i+1}(r_4) \]

for a single point \( r_4 \in F^{\log S} \).
Costs of the GKR protocol

• \( V \) time is \( O(n + D \log S) \) where \( n \) is input size, \( D \) is circuit depth, and \( S \) is circuit size.
  • Assumes \( V \) can compute \( \text{add}_i(r_1, r_2, r_3) \) and \( \text{mult}_i(r_1, r_2, r_3) \) unaided in time \( \text{polylog}(n) \)
• Communication cost is \( O(D \log S) \).
Costs of the GKR protocol

- \( V \) time is \( O(n + D \log S) \) where \( n \) is input size, \( D \) is circuit depth, and \( S \) is circuit size.
  - Assumes \( V \) can compute \( \text{add}_i(r_1, r_2, r_3) \) and \( \text{mult}_i(r_1, r_2, r_3) \) unaided in time \( \text{polylog}(n) \)

- Communication cost is \( O(D \log S) \).

- \( P \) time is \( O(S) \).
  - A naïve implementation of \( P \) takes \( \Omega(S^3) \) time, where \( S \) is circuit size.
  - A sequence of works has brought this down to \( O(S) \), for arbitrary circuits
    [CMT12, Thaler13, WJBSTWW17, XZZPS19]
GKR Prover Runtime: Details

Recall: Core of the GKR protocol is applying sum-check to compute \( \sum_{b,c \in \{0,1\}^{\log s}} g(b,c) \) where

\[
g(b,c) = \widetilde{\text{add}}_i(r_1, b, c)(\tilde{W}_{i+1}(b) + \tilde{W}_{i+1}(c))
+ \widetilde{\text{mult}}_i(r_1, b, c)(\tilde{W}_{i+1}(b) \cdot \tilde{W}_{i+1}(c))
\]
Recall: Core of the GKR protocol is applying sum-check to compute
\[ \sum_{b,c \in \{0,1\}^{\log S}} g(b,c) \]
where
\[ g(b,c) = \text{add}_i(r_1, b, c)(\widetilde{W}_{i+1}(b) + \widetilde{W}_{i+1}(c)) \\
+ \text{mult}_i(r_1, b, c)(\widetilde{W}_{i+1}(b) \cdot \widetilde{W}_{i+1}(c)) \]

- A naïve implementation of P takes \( \Omega(S^3) \) time, where \( S \) is circuit size.
  - Same idea as prover implementation for \#SAT protocol.
  - i.e., P evaluates \( g \) in each round of sum-check at all \( O(S^2/2^i) \) necessary points \( z \), taking \( O(S) \) time per point.
GKR Prover Runtime: Details

Recall: Core of the GKR protocol is applying sum-check to compute $\sum_{b,c \in \{0,1\}^{\log S}} g(b,c)$ where

$$g(b,c) = \overline{\text{add}_i(r_1, b, c)}(\overline{W_{i+1}(b)} + \overline{W_{i+1}(c)}) + \overline{\text{mult}_i(r_1, b, c)}(\overline{W_{i+1}(b)} \cdot \overline{W_{i+1}(c)})$$

- A naïve implementation of $P$ takes $\Omega(S^3)$ time, where $S$ is circuit size.
  - Same idea as prover implementation for #SAT protocol.
  - i.e., $P$ evaluates $g$ in each round of sum-check at all $O(S^2/2^i)$ necessary points $z$, taking $O(S)$ time per point.
- [CMT12]: $P$ time is $O(S \log S)$.
  - Exploit structure in multilinear extensions $\overline{\text{add}_i}$ and $\overline{\text{mult}_i}$. Ensures that each gate of $C$ contributes to $g(z)$ for $O(1)$ relevant points $z$ in each round.
GKR Prover Runtime: Details

Recall: Core of the GKR protocol is applying sum-check to compute \( \sum_{b,c \in \{0,1\}^{\log S}} g(b,c) \) where

\[
g(b,c) = \tilde{\text{add}}_i(r_1, b, c)(\tilde{W}_{i+1}(b) + \tilde{W}_{i+1}(c)) \\
+ \tilde{\text{mult}}_i(r_1, b, c)(\tilde{W}_{i+1}(b) \cdot \tilde{W}_{i+1}(c))
\]

- A naïve implementation of \( P \) takes \( \Omega(S^3) \) time, where \( S \) is circuit size.
  - Same idea as prover implementation for #SAT protocol.
  - i.e., \( P \) evaluates \( g \) in each round of sum-check at all \( O(S^2/2^i) \) necessary points \( z \), taking \( O(S) \) time per point.

- [CMT12]: \( P \) time is \( O(S \log S) \).
  - Exploit structure in multilinear extensions \( \tilde{\text{add}}_i \) and \( \tilde{\text{mult}}_i \). Ensures that each gate of \( C \) contributes to \( g(z) \) for \( O(1) \) relevant points \( z \) in each round.

- All subsequent works seek to bring “Approach 3” to bear on the GKR protocol, letting \( P \) reuse work across rounds.
GKR Prover Runtime: Details

• [Thaler13]:
  1. $\text{P time } O(S)$ for circuits with “nice” wiring patterns.
  2. $\text{P time } O(S \log S')$ for data parallel circuits
     i.e., that apply the same subcomputation (of size $S'$)
     independently to different pieces of data
GKR Prover Runtime: Details

• [Thaler13]:
  1. $P$ time $O(S)$ for circuits with “nice” wiring patterns.
  2. $P$ time $O(S \log S')$ for data parallel circuits
     i.e., that apply the same subcomputation (of size $S'$)
     independently to different pieces of data

• [WJBSTWW17] improved the data parallel time to
  $O(S + S' \log S')$. 
GKR Prover Runtime: Details

• [Thaler13]:
  1. $\mathcal{P}$ time $O(S)$ for circuits with “nice” wiring patterns.
  2. $\mathcal{P}$ time $O(S \log S')$ for data parallel circuits
     i.e., that apply the same subcomputation (of size $S'$)
     independently to different pieces of data

• [WJBSTWW17] improved the data parallel time to
  $O(S + S' \log S')$.

• [ZGKPP18] extends to data parallel computations
  where each subcomputation may not be the same.
GKR Prover Runtime: Details

• [Thaler13]:
  1. $\mathcal{P}$ time $O(S)$ for circuits with “nice” wiring patterns.
  2. $\mathcal{P}$ time $O(S \log S')$ for data parallel circuits
     i.e., that apply the same subcomputation (of size $S'$) independently to different pieces of data

• [WJBSTWW17] improved the data parallel time to $O(S + S' \log S')$.

• [ZGKPP18] extends to data parallel computations where each subcomputation may not be the same.

• [XZZPS19] achieved $O(S)$ time for general circuits.
Rumination on Generality Vs. Efficiency
Generality vs. Efficiency

• The GKR protocol for circuit evaluation has now been rendered optimally efficient for $P$ (up to constant factors).

• Any computation can be represented as a circuit evaluation (or satisfiability) problem.
  • But this can introduce tremendous overheads.
  • The GKR protocol forces the prover to compute the output in a prescribed manner, which may be far from optimal (gate-by-gate evaluation of a circuit).

• To achieve scalability, the gold standard is really super-efficiency.
  • i.e., $P$ computed the right answer directly using the fastest known algorithm, and did a low-order amount of extra work to prove correctness