Synopsis.

- Multi-Valued Partial CFL Functions
- CFLMV Hierarchy
- State Complexity for LSH
- Function-Oracle Turing Machines
- Type-2 Computability
Course Schedule: 16 Weeks

Subject to Change

- **Week 1**: Basic Computation Models
- **Week 2**: NP-Completeness, Probabilistic and Counting Complexity Classes
- **Week 3**: Space Complexity and the Linear Space Hypothesis
- **Week 4**: Relativizations and Hierarchies
- **Week 5**: Structural Properties by Finite Automata
- **Week 6**: Stype-2 Computability, Multi-Valued Functions, and State Complexity
- **Week 7**: Cryptographic Concepts for Finite Automata
- **Week 8**: Constraint Satisfaction Problems
- **Week 9**: Combinatorial Optimization Problems
- **Week 10**: Average-Case Complexity
- **Week 11**: Basics of Quantum Information
- **Week 12**: BQP, NQP, Quantum NP, and Quantum Finite Automata
- **Week 13**: Quantum State Complexity and Advice
- **Week 14**: Quantum Cryptographic Systems
- **Week 15**: Quantum Interactive Proofs
- **Week 16**: Final Evaluation Day (no lecture)
YouTube Videos

• This lecture series is based on numerous papers of T. Yamakami. He gave conference talks (in English) and invited talks (in English), some of which were video-recorded and uploaded to YouTube.

• Use the following keywords to find a playlist of those videos.

• YouTube search keywords:
  Tomoyuki Yamakami  conference  invited talk playlist

Conference talk video
Main References by T. Yamakami I


(Continued to the next slide)
Main References by T. Yamakami Ⅱ


I. Multi-Valued Partial Functions

1. Multi-Valued Partial Functions
2. Write-Only Output Tapes
3. Valid (or Legitimate) Outputs
4. Multi-Valued Partial CFL Functions
5. $\text{CFL} \cap \text{co-CFL} \text{ vs. CFLSV}$
6. Functional Pumping Lemma
7. Function Class NFAMV
8. Boolean Operators
9. Basic Properties
Multi-Valued Partial Functions

• A (standard) function is designed to produce only one output value per each input.
• We can allow a function to output more than one value simultaneously, or even allow it to output no value at all.
• A total function is a standard function $f$ such that, for any input $x$, its output $f(x)$ always exists.
• By contrast, a partial function means that, the outputs of the function are not guaranteed to exist for all inputs.
• A multi-valued function is called single valued if, for any input $x$, the number of different output values in $f(x)$ is $\leq 1$.
• When a function produces no output value on a certain input $x$, we treat $f(x)$ to be undefined.
• Generally, we call such a function a multi-valued partial function (where “partial” is meant for undefined values).
Early Studies

• Firstly, we consider how to compute such a function using 1npda. Those functions are called **CFL functions**.

• CFL functions were first studied by **Evey** (1963) and **Fisher** (1963).
Write-Only Output Tapes

To compute a function, we need to equip a 1npda (also called a transducer) with an extra write-only output tape, along which its tape head moves rightward whenever it writes a non-blank symbol.
How to Produce Multi-Values

• We explain how a 1npda produces outcomes of a multi-valued partial function \( f \).

• We say that a 1npda \( M \) computes a multi-valued partial function \( f: \Sigma_1^* \rightarrow \wp(\Sigma_2^*) \) if \( M \) satisfies the following:
  1. for any \( x \in \text{dom}(f) \), \( M \) produces exactly all values in \( f(x) \) along accepting computation paths, and
  2. for any string \( x \in \Sigma_1^*-\text{dom}(f) \), \( M \) rejects the input \( x \) (in which all computation paths are rejecting).

• Namely, a 1npda \( M \) with a write-only output tape can compute a multi-valued partial function \( f: \Sigma_1^* \rightarrow \wp(\Sigma_2^*) \) defined by

\[
f(x) = \{ y \mid M(x) \text{ outputs } y \}.
\]
Valid (or Legitimate) Outputs

- A 1npda produces **valid** outcomes only along **accepting computation paths**.

\[ \text{input } x \quad \text{npda } M \quad \text{input } x \]

- \[ y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5 \]
  - **rejected**
  - **accepted**

- \[ \text{valid outputs} \]

\[ \text{M}(x) \text{ outputs } \{ y_4, y_5 \} \]

- \[ y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5 \]
  - **all rejected**

\[ \text{M}(x) \text{ outputs } \emptyset \]
Formal Definition

A 1npda \( M = (Q, \Sigma, \{ \emptyset, \$$ \}, \Theta, \Gamma, \delta, q_0, Z_0, Q_{\text{acc}}, Q_{\text{rej}}) \) with a write-only output tape is a standard 1npda plus a write-only output tape and a special transition function \( \delta \) of the form:

\[
\delta : (Q - Q_{\text{halt}}) \times (\bar{\Sigma} \cup \{ \lambda \}) \times \Theta \rightarrow P(Q \times \Gamma^* \times (\Gamma \cup \{ \lambda \}))
\]

- Termination condition of \( M \):
  - All computation paths (both accepting and rejecting) should terminate (reaching halting states) within \( O(n) \) time.
  - \( \text{ACC}_M(x) = \) set of accepting computation paths of \( M \) on \( x \)

This is because all context-free languages are recognized by \( O(n) \)-time npda’s.
Multi-Valued Partial CFL Functions

♦ Function Classes

- **CFLMV** = class of all *multi-valued* partial functions computed by 1npda’s
- **CFLSV** = class of all *single-valued* partial functions in CFLMV
- **CFLSV_t** = class of all *total* functions in CFLSV
- **CFLMV(2)** = class of all functions $g$ defined as $g(x) = f_1(x) \cap f_2(x)$ for $f_1, f_2 \in$ CFLMV

- CFLMV, CFLSV, and CFLSV_t are analogues of NPMV, NPSV, and NPSV_t, respectively.
Examples: PAL

• Here, we take a look at two simple examples.

• \( \text{PAL}(w) = \{ x \mid \exists u,v \ [ w = uxv ] \land x = x^R \} \) for all \( w \in \{0,1\}^* \).

• I.e., \( \text{PAL}(w) \) outputs all possible palindrome blocks in \( w \).

• The right-hand side illustration shows how to compute \( \text{PAL} \).

• Thus, \( \text{PAL} \) is in \( \text{CFLMV}_t \). (total function)
Examples: $IP_2$

- Let $\odot$ be the binary inner product.

- $IP_2(x) = \{ z \mid |x|=|z|, x \odot z \equiv 1 \pmod{2} \}$ for all $x \in \{0,1\}^*$.

- This is different from the language $IP_2(x) = \{ xz \mid |x|=|z|, x^R \odot z \equiv 1 \pmod{2} \}$.

- The right-hand side illustration shows how to compute $IP_2$.

- Thus, $IP_2$ is in CFLMV (actually, in NFAMV).

- See a later slide for NFAMV.
CFL∩co-CFL vs. CFLSV

• CFLSV is closely related to the language family CFL∩co-CFL.

• Recall that $\chi_A$ is the characteristic function of a language A.

• **Lemma:** [Yamakami (2016)]
  Let A be any language. 
  \[ A \in \text{CFL} \cap \text{co-CFL} \iff \chi_A \in \text{CFLSV} \]

• We can replace CFLSV by CFLSV$_t$ and CFLMV.
Functional Pumping Lemma for CFLMV

- **Pumping Lemma for CFLMV:** [Yamakami (2014)]
  Let $\Sigma$ and $\Gamma$ be any alphabets and let $f: \Sigma^* \rightarrow \wp(\Gamma^*)$ be any function in CFLMV. There exist 3 numbers $m \in \mathbb{N}^+$ and $c,d \in \mathbb{N}$ s.t. any $w \in \Sigma^*$ with $|w| \geq m$ and any $s \in f(w)$ are decomposed into $w = uvxyz$ and $s = abpqr$ s.t.
  
  (1) $|uxy| \leq m$
  (2) $|vybq| \geq 1$
  (3) $|bq| \leq cm+d$ and
  (4) $ab^ipq^ir \in f(uv^i xy^i z)$.

  If $f$ is further length-preserving, then
  (5) $|v| = |b|$ and $|y| = |q|$.

  Moreover, (1)-(2) can be replaced by
  (1') $|bq| \geq 1$. 
Function Class NFAMV

• Similarly to CFLMV, we define the function class NFAMV as follows.

• Let f be any multi-valued partial function.

• f is in NFAMV ⇔ there is a 1nfa M equipped with a write-only output tape such that
  1. for every $x \in \text{dom}(f)$, M produces all elements in $f(x)$ along accepting computation paths, and
  2. for every $x \notin \text{dom}(f)$, M rejects the input x.

• (Claim) 1-FLIN $\subseteq$ NFAMV $\subseteq$ CFLMV.
Conjunction/Disjunction of Functions

• We define conjunction/disjunction of function classes.

- Conjunction of F and G
  • $f_1 \in F \land G$
    $$\iff \exists g_1 \in F \exists g_2 \in G \text{ s.t. } \forall x \ [ f_1(x) = g_1(x) \cap g_2(x) ]$$

- Disjunction of F and G
  • $f_2 \in F \lor G$
    $$\iff \exists g_1 \in F \exists g_2 \in G \text{ s.t. } \forall x \ [ f_2(x) = g_1(x) \cup g_2(x) ]$$
Simple Examples of $f \lor g$ and $f \land g$

• Here, we present two simple examples.

• Consider the following $f$ and $g$.
  $\triangleright$  $f(x) = \{ a^n b^n | n=|x| \}$
  $\triangleright$  $g(x) = \{ a^n b^{2n} | n=|x| \}$
  $\triangleright$  $(f \lor g)(x) = \{ a^n b^n, a^n b^{2n} | n=|x| \}$

• Consider the following $f$ and $g$.
  $\triangleright$  $f(x) = \{ a^n b^n c^m | n=|x|, m \geq 0 \}$
  $\triangleright$  $g(x) = \{ a^m b^n c^n | n=|x|, m \geq 0 \}$
  $\triangleright$  $(f \land g)(x) = \{ a^n b^n c^n | n=|x| \}$
Function Classes CFLMV(k)

- We extend CFLMV using “conjunction” operator.
  1. CFLMV(1) = CFLMV
  2. CFLMV(k+1) = CFLMV(k) ∧ CFLMV
  3. CFLSV(k) = \{ f \in CFLMV(k) \mid f \text{ is single-valued} \}

- **Lemma**: [Yamakami (2014)]
  1) CFLMV(max\{k,m\}) ⪯ CFLMV(k) ∨ CFLMV(m) ⪯ CFLMV(km).
  2) CFLMV(max\{k,m\}) ⪯ CFLMV(k) ∧ CFLMV(m) ⪯ CFLMV(k+m).
  3) CFLSV(k) ≠ CFLSV(k+1) for any k ≥ 1.
Difference/Complement of Functions

- We define the difference/complement of function classes.

**Difference between F and G**
- \( f \in F \ominus G \iff \exists g_1 \in F \exists g_2 \in G \text{ s.t. } \forall x [ f(x) = g_1(x) - g_2(x) ] \)

**Complement of F**
- \( f \in \text{co-F} \iff \exists g \in F \exists p: \text{linear polynomial} \exists n_0: \text{constant} \text{ s.t. } \forall (x,y) \text{ with } |x| \geq n_0 [ y \in f(x) \leftrightarrow |y| \leq p(|x|) \land y \in g(x) ] \)

\[ f(x) = \sum_{\leq p(|x|)} w - g(x) \]
Boolean Operations

• Using two operators $\Theta$ and co-, we define the following function classes.
  
  • co-CFLMV (complement)
  
  • CFLMV $\Theta$ CFLMV (difference)
  
  • CFLMV $\wedge$ co-CFLMV (conjunction with complement)

• Recall $IP_2(x) = \{ z \mid |x| = |z|, x \odot z \equiv 1 \pmod{2} \}$ for all $x \in \{0,1\}^*$.

• Define $IP^c(x) = \{ z \mid |x| \geq |z|, x \odot z 0^{|x|-|z|} \equiv 0 \pmod{2} \}$ for any $x$.

• It follows that $IP^c \in$ co-CFLMV$_t$ since $IP_2 \in$ CFLMV$_t$ and $IP^c(x) = \sum_{|z| \leq |x|} IP_2(x)$ for any $x$. 
Basic Properties

- The following basic properties hold.

- **Proposition: [Yamakami (2014)]**
  1. $\text{co-(co-CFLMV)} = \text{CFLMV}$
  2. $\text{co-CFLMV} = \text{NFAMV} \ominus \text{CFLMV}$
  3. $\text{CFLMV} \ominus \text{CFLMV} = \text{CFLMV} \land \text{co-CFLMV}$
  4. $\text{CFLMV} \neq \text{co-CFLMV}$
  5. $\text{CFLMV}_t \neq \text{co-CFLMV}_t$
II. Refinement of Functions

1. Refinement of Functions
2. Refinement Separation: CFLMV
3. 1-FLIN(partial), 1_NLINMV, and 1-NLINSV
4. Refinement of 1-NLINMV
Refinement of Functions

• The notion of refinement is more useful than a standard set inclusion, because, e.g., CFLSV_t ≠ CFLSV ≠ CFLMV holds.

• Let f, g be any two functions from $\Sigma^*$ to $\wp(\Gamma^*)$. g is a refinement of f (notationally, $f \sqsubseteq_{\text{ref}} g$)

\[ \iff \forall x \in \Sigma^* \]
\[ 1. f(x) \neq \emptyset \iff g(x) \neq \emptyset \]
\[ 2. g(x) \subseteq f(x) \quad (\text{as set inclusion}). \]

• For two function classes F and G,

\[ F \sqsubseteq_{\text{ref}} G \iff \forall f \in F \exists g \in G \ [ f \sqsubseteq_{\text{ref}} g ] \]

• NOTE: $F \subseteq G \Rightarrow F \sqsubseteq_{\text{ref}} G$. 

Refinement is also known as uniformization.
Example: maxPAL

- Let us see an example of refinement.
- Recall \( \text{PAL}(w) = \{ x \mid \exists u, v \ [ w = uxv ] \land x = x^R \} \).
- For each \( w \in \{0,1\}^* \), we define
  \[
  \text{maxPAL}(w) = \text{maximum element in } \text{PAL}(w),
  \]
  where “maximum” is according to a dictionary order.
- \( \text{maxPAL} \) is a single-valued total function.
- (Claim) \( \text{PAL} \sqsubseteq_{\text{ref}} \text{maxPAL} \) (\( \text{PAL} \) is refined by \( \text{maxPAL} \))

Proof: This is because \( \text{dom}(\text{PAL}) = \text{dom}(\text{maxPAL}) \) and \( \text{maxPAL}(x) \subseteq \text{PAL}(x) \) for all \( x \).
Refinement Separation: CFLMV Ⅰ

- Let us consider the refinement separation between CFLMV and CFLSV.
- Actually, we can show a much stronger separation as explained below.

- **CFL2V** is the collection of all partial functions $f$ in CFLMV such that the number of $f$’s output values on each input must be at most 2 (called 2-valued functions).
- The machine 1npda $M$ is called **unambiguous** if, for any input $x$ and any output value $y$, $M$ has exactly one accepting computation path producing $y$ from $x$.
- **UCFL2V** is the collection of all 2-valued partial functions computed by unambiguous 1npda’s.
- *(Claim) UCFL2V $\subseteq$ CFL2V $\subseteq$ CFLMV.*
Here, we claim the desired separation result.

**Theorem:** [Yamakami (2014)]

\[ \text{UCFL2V} \not\equiv_{\text{ref}} \text{CFLSV}. \]

The above theorem implies that \( \text{CFLMV} \not\equiv_{\text{ref}} \text{CFLSV}. \)

**Proof Sketch:**

It suffices to define an example function, say, \( h_3 \) as in the next slide and prove the following 2 claims.

1. \( h_3 \in \text{UCFL2V}. \)
2. \( h_3 \) has no refinement in \( \text{CFLSV} \).
Refinement Separation: CFLMV III

• The desired function $h_3$ is defined as follows.

\[
L = \left\{ x_1 \# x_2 \# x_3 \mid x_1, x_2, x_3 \in \{0,1\}^* \right\}
\]

\[
I_3 = \left\{ (i, j) \mid i, j \in N^+, 1 \leq i < j \leq 3 \right\}
\]

\[
L_3 = \left\{ w \mid \exists x_1, x_2, x_3[w = x_1 \# x_2 \# x_3 \in L], \exists (i, j) \in I_3[x_i^R = x_j] \right\}
\]

\[
h_3(w) = \begin{cases} 
0^i1^j \mid (i, j) \in I_3, x_i^R = x_j & \text{if } w = x_1 \# x_2 \# x_3 \in L, \\
\emptyset & \text{if } w \notin L.
\end{cases}
\]

• For example,

✓ $h_3(001\#100\#000) = \{0^11^2\}$

✓ $h_3(001\#100\#001) = \{0^11^2, 0^21^3\}$

✓ $h_3(111\#011\#101) = \emptyset$

\begin{align*}
001^R &= 100 & 001^R &= 100, 100^R &= 001
\end{align*}
1-FLIN(partial), 1-NLINMV, and 1-NLINSV

- Recall 1-FLIN from Week 1.
- Here, we relax the function condition of 1-FLIN to obtain 1-FLIN(partial), which is composed of all partial functions computable by 1DTM in linear time with no extra output.
- In other words, if we restrict all partial functions in 1-FLIN(partial) to be total, we immediately obtain 1-FLIN.
- Next, we define 1-NLINMV and 1-NLINSV.

   - A multi-valued partial function \( f : \Sigma_1^* \rightarrow \wp(\Sigma_2^*) \) is in 1-NLINMV if there exists a 1NTM \( M \) such that
     1. for any string \( x \in \text{dom}(f) \), \( M \) produces exactly all values in \( f(x) \) along accepting computation paths, and
     2. for any string \( x \in \Sigma_1^* \text{-} \text{dom}(f) \), \( M \) rejects the input \( x \).
     3. for any input \( x \in \Sigma_1^* \), \( M \) halts within \( O(|x|) \) time in the strong sense.
Refinements of 1-NLINMV

• **1-NLINSV** is the collection of all single-valued partial functions in 1-NLINMV.

• **1-NLINSV\_t** consists of all total functions in 1-NLINSV.

• A single-valued function $f: \Sigma_1^* \rightarrow \Sigma_2^*$ is **length-preserving** if, for any input $x \in \Sigma_1^*$, $|f(x)| = |x|$ holds.

• **Theorem:** [Tadaki-Yamakami-Lin (2010)]

  Every length-preserving 1-NLINMV function has a 1-FLIN(partial) refinement.

• (*) This will be used for **one-way functions** in Week 7.
III. The CFLMV Hierarchy

1. The CFL Hierarchy
2. The CFLMV Hierarchy
3. Refinement Separations and Collapses
4. The //Advice Operator
5. Basic Properties
6. Functional Composition
7. Separations
The CFL Hierarchy (revisited)

- **AC^0(CFL)** = LOGCFL = SAC^1
- **AC^0(REG)** = NC^1
- **NL** = **L**
- **REG**
- **REG/n**
- **CFL(1)**
- **CFL(2)**
- **CFL(3)**
- **CFL(ω)**
- **CFL_m**
- **CFL_m[1]**
- **CFL_m[2]**
- **CFL(ω)**
- **CFL(2)**
- **CFL(3)**
- **CFL_m**
- **CFL_m[1]**
- **CFL_m[2]**
- **CFL/n**
- **REG/n**

- **co-CFL** = Π^{CFL}_1
- **Σ^{CFL}_1** = CFL
- **Σ^{CFL}_2**
- **Σ^{CFL}_3**
- **Π^{CFL}_2**
- **Π^{CFL}_3**

- **DSPACE(O(n))**
- **CSL**
- **REG**

- **proper inclusion**
- **inclusion**
- **no inclusion**
The CFLMV Hierarchy

• Similarly to $\text{CFL}^A$ (relative to $A$), we can relativize CFLMV to oracle $A$ and obtain $\text{CFLMV}^A$ by attaching query tapes to underlying 1npda’s with output tapes.

• We then define the CFLMV hierarchy as follows.

\[ \Sigma^1_{CFL} MV = \text{CFLMV}^A; \quad \Sigma^k_{CFL} MV = \text{CFLMV}^{\Sigma^k_{CFL}} \]

• Similarly, we define the CFLSV hierarchy by setting:

\[ \Sigma^k_{CFL} SV = \{ f \in \Sigma^k_{CFL} MV | f \text{ is single-valued} \} \]

• **Theorem:** [Yamakami (2014)] $(k \geq 1)$

1. $\Sigma^k_{CFL} SV \sqsubseteq \text{ref} \Sigma^k_{CFL} MV$.
2. $\Sigma^k_{CFL} SV = \Sigma^k_{CFL} SV \Rightarrow \Sigma^k_{CFL} = \Sigma^{k+1}_{CFL}$
3. $\Sigma^k_{CFL} = \Sigma^{k+1}_{CFL} \Rightarrow \Sigma^k_{CFL} \text{ SV} = \Sigma^{k+1}_{CFL} \text{ SV}$
Refinement Separations and Collapses

• We have seen $\text{CFLMV} \not\subseteq_{\text{ref}} \text{CFLSV}$. This is equivalent to $\Sigma_{\text{CFL}}^{1}\text{MV} \not\subseteq_{\text{ref}} \Sigma_{\text{CFL}}^{0}\text{SV}$.

• (Open Problem) Is $\Sigma_{\text{CFL}}^{k}\text{MV} \not\subseteq_{\text{ref}} \Sigma_{\text{CFL}}^{k}\text{SV}$ for each $k \geq 2$?

• Related to this question, we obtain the following.

• Lemma: [Yamakami (2014)] ($k \geq 1$)
  - $\Sigma_{\text{CFL}}^{k}\text{MV} \subseteq_{\text{ref}} \Sigma_{\text{CFL}}^{k+1}\text{SV}$

• Theorem: [Yamakami (2014)] ($k \geq 2$)
  - $\Sigma_{\text{CFL}}^{k} = \Sigma_{\text{CFL}}^{k+1} \Rightarrow \Sigma_{\text{CFL}}^{k+1}\text{MV} \subseteq_{\text{ref}} \Sigma_{\text{CFL}}^{k+1}\text{SV}$.

• Corollary: [Yamakami (2014)] ($k \geq 2$)
  - $\Sigma_{\text{CFL}}^{k}\text{MV} \subseteq_{\text{ref}} \Sigma_{\text{CFL}}^{k}\text{SV} \Rightarrow \text{PH} = \Sigma_{p}^{k}$. 
The //Advice Operator

- Köbler and Thierauf (1994) introduced the //advice operator, which is a natural extension of the /advice operator (used to define P/poly).
- We adapt this operator to apply to automata.

Let F be a class of multi-valued functions.

- A language L is in \( \text{REG} // F \) if there are a language \( B \in \text{REG} \) and a function \( h \in F \) such that, for any \( x \),

\[
  x \in L \iff \exists y \in h(x) \text{ s.t. } \begin{bmatrix} x \\ y \end{bmatrix} \in B
\]

- Analogously, \( \text{CFL} // F \) is defined using CFL instead of REG.
Basic Properties

• We list basic properties of the // advice operator.

• Proposition: [Yamakami (2014)]
  1. $\text{REG//NFASV}_t \not\subseteq \text{CFL}$ and $\text{CFL} \not\subset \text{REG//NFAMV}$.
  2. $\text{REG//NFASV}_t = \text{co-(REG//NFASV}_t)$
  3. $\text{REG//NFAMV} \neq \text{co-(REG//NFAMV)}$
  4. $\text{CFL} \cap \text{co-CFL} \neq \text{REG//CFLSV}_t$

• (*)& The last claim is compared to $\text{NP} \cap \text{co-NP} = \text{P//NPSV}_t$.
  [Köbler-Thierauf (1994)]

• Proposition: [Yamakami (2014)]
  $\sum_{k=3}^{\infty} \text{CFL}_k \cap \Pi_{k=3}^{\infty} \text{CFL}_k = \text{REG//}\sum_{k=3}^{\infty} \text{CFL}_k \text{ SV}_t$, for any $k \geq 3$. 

Functional Composition

• Let $f, g$ be any multi-valued partial functions.
• The functional composition $f \circ g$ of $f$ and $g$ is defined as

$$ (f \circ g)(x) = \bigcup_{y \in g(x)} f(y) $$

for every $x$.
• For two function classes $F$ and $G$, a new function class $F \circ G$ is defined as

$$ F \circ G = \{ f \circ g \mid f \in F, g \in G \} $$

• Let
  - $\text{CFLSV}^{(1)} = \text{CFLSV}$.
  - $\text{CFLSV}^{(k+1)} = \text{CFLSV} \circ \text{CFLSV}^{(k)}$ for each $k \geq 1$. 
Separations

• We show a simple separation result.

• Proposition: [Yamakami (2014)]
  1. CFLSV\(_t\) \(\neq\) CFLSV\(^{(2)}\)\(_t\)
  2. The same holds for CFLSV and CFLMV.

Proof Sketch:
• Define \(f_{dup#}(x) = \{ x#x \} \) for any \(x \in \{0,1\}^*\).
• Clearly, \(f_{dup#}(x) \in\) CFLSV\(^{(2)}\)\(_t\).
• However, if \(f_{dup#}(x) \in\) CFLSV\(_t\), then the language DUP\(_#\) = \{ x#x | x \in \{0,1\}^* \} \) must belong to CFL.
• Since DUP\(_#\) \(\not\in\) CFL, we conclude \(f_{dup#}(x) \not\in\) CFLSV\(_t\).

QED
OptCFL

- Krentel (1988) introduced a function class $\text{OptP}$, which consists of the optimal cost functions of NP optimization problems.
- Similarly, Yamakami (2014) considered its pushdown-automaton version, which is called $\text{OptCFL}$.
- We assume the standard lexicographic order on $\Sigma^*$.

A function $f: \Sigma^* \to \Sigma^*$ is in $\text{OptCFL} \iff$ there exists a 1npda $M$ with a write-only output tape s.t.

$$f(x) = \text{opt} \{ y \in \Sigma^* \mid M(x) \text{ produces } y \},$$

where $\text{opt} \in \{ \text{max, min} \}$. 
Open Problems

1. Prove that $\Sigma_{CFL}^{k+1}MV \neq \Sigma_{CFL}^{k+2}MV$ for all $k \geq 1$.
   
   • Note that proving that $\Sigma_{CFL}^{k+1}MV = \Sigma_{CFL}^{k+2}MV$ is much more difficult because this implies $\Sigma^P_k = \Sigma^P_{k+1}$, as discussed in Week 4.

2. Prove that $\Sigma_{CFL}^{k}SV \not\preceq_{ref} \Sigma_{CFL}^{k}MV$ for all $k \geq 2$.

3. Prove that OptCFL $\not\in \Sigma_{CFL}^{2}SV_t$ or OptCFL $\not\in \Sigma_{CFL}^{3}SV_t$. 
IV. State Complexity Characterizations

1. State Complexity of Automata Families
2. L-Uniform Families of Finite Automata
3. State Complexity of Transformation
4. Characterization of \( \text{NL} \subseteq \text{L/poly} \)
5. Constant-Branching Simple 2nfa’s
6. Characterizing PsubLIN by Narrow 2afa’s
7. Non-Uniform Linear Space Hypothesis
8. Characterization of LSH
State Complexity of Automata Families

- Let $M = (Q, \Sigma, \delta, q_0, Q_{\text{acc}}, Q_{\text{rej}})$ be any finite automaton.
- The state complexity of $M$ is $\text{st}(M) = |Q|$ (the number of inner states).

We consider a family $\{M_n\}_{n \in \mathbb{N}}$ of finite automata, each $M_n$ of which is of the form $(Q_n, \Sigma_n, \delta_n, q_{0n}, Q_{\text{acc},n}, Q_{\text{rej},n})$.

- We often take the same input alphabet $\Sigma_n = \Sigma$ for all $n$.
- Note that the state complexity of this family $\{M_n\}_{n \in \mathbb{N}}$ becomes a function $\text{st}(n) = |Q_n|$ in length $n$. 
L-Uniform Families of Finite Automata

• We consider a family of finite automata, each of which can be constructed by a single production algorithm.

• Let \( \{M_n\}_{n \in \mathbb{N}} \) be any family of finite automata, each \( M_n \) of which is of the form \((Q_n, \Sigma_n, \delta_n, q_{0n}, Q_{\text{acc},n}, Q_{\text{rej},n})\).

• This family \( \{M_n\}_{n \in \mathbb{N}} \) is called L-uniform if there exists a log-space DTM \( A \) with a write-only output tape such that, for any length \( n \in \mathbb{N} \), \( A \) takes input of the form \( 1^n \) and produces an encoding of \( M_n \) on the output tape.

• (*) In comparison, we will discuss uniform families of Boolean circuits in Week 8.
Equivalent Finite Automata

- We define the notion of equivalence between two finite automata.
- Let $M$ and $N$ be two finite automata (of possibly different types).
  - We say that $M$ is equivalent to $N$ if $L(M) = L(N)$.
  - That is, $M$ agrees with $N$ on all inputs; i.e., for every input string $x$,
    
    $M$ accepts $x \iff N$ accepts $x$.

- Two families $\{M_n\}_{n \in \mathbb{N}}$ and $\{N_n\}_{n \in \mathbb{N}}$ of finite automata are said to be equivalent if, for any $n \in \mathbb{N}$, $M_n$ and $N_n$ are equivalent.
State Complexity of Transformation

• Consider two different types of finite automata: type 1 and type 2.

• We say that the state complexity of transforming type-1 automata to type-2 automata is $t(n)$ if, for any $n$-state type-1 automaton $M$, there exists another type-2 automaton $N$ such that (i) $N$ has at most $t(n)$ states and (ii) $N$ is equivalent to $M$. 
Example of Transformation

• Consider the following example.

• Fig.1 is a 1nfa with 3 states, and Fig.2 is its equivalent 1dfa with 4 states.
Characterization of $\text{NL} \subseteq \text{L/poly}$

- Recall the non-uniform class $\text{L/poly}$ from Week 3.
- Note that we do not know whether or not $\text{NL} \subseteq \text{L/poly}$.
- Kapoutsis (2014) and Kapoutsis and Pighizzini (2015) gave a new characterization of $\text{NL} \subseteq \text{L/poly}$ in terms of state-complexity of transforming 2nfa’s to 2dfa’s.

- **(Claim)** The following statements are logically equivalent.
  1. $\text{NL} \subseteq \text{L/poly}$.
  2. There exists a polynomial $p$ such that, for any $n$-state 2nfa $N$, there is another 2dfa $M$ of at most $p(n)$ states such that $M$ agrees with $N$ on all inputs of length $\leq n$.

- Note that a straightforward textbook algorithm transforms an $n$-state 2nfa into an equivalent 2dfa of $2^{O(n)}$ states.
The Linear Space Hypothesis (LSH) (revisited)

• Recall the linear space hypothesis (LSH) from Week 3.

• LSH (or LSH for $2\text{SAT}_3$) states:
  There is no deterministic algorithm that solves $2\text{SAT}_3$ in time $p(|x|)$ using at most $m_{\text{vbl}}(x)^\varepsilon l(|x|)$ space on instance $x$ for a certain polynomial $p$, a certain polylog function $l$, and a certain constant $\varepsilon \in [0,1)$.

• We can replace $(2\text{SAT}_3,m_{\text{vbl}})$ by $(3\text{DSTCON},m_{\text{ver}})$, where $m_{\text{ver}}(\langle G,s,t \rangle) =$ the number of vertices in $G$.

• Here, we want to give a state complexity characterization of LSH.
Circular Tapes and Sweeping Moves

- When both ends of a tape are glued together, we call this tape a **circular tape**.
- A tape head is said to **sweep** a tape if the tape head moves to the right from $\exists$ to $\$$. In this case, the tape head is called **sweeping**.
Constant Branching

- Let $c \in \mathbb{N}^+$.  
- A 2nfa is \textit{c-branching} if it makes only at most $c$ nondeterministic choices at every step.  
- In particular, every 2dfa is 1-branching.  
- A family $\{M_n\}_{n \in \mathbb{N}}$ of 2nfa’s is called \textit{constant-branching} if there exists a constant $c \in \mathbb{N}^+$ such that every $M_n$ is $c$-branching.
Constant-Branching Simple 2nfa’s

• We place certain restrictions on 2nfa’s.
• We consider only 2nfa’s whose input tapes are circular.

• We say that a 2nfa is simple if
  1. its input tape is circular,
  2. its tape head sweeps the tape, and
  3. it makes nondeterministic choices only at the right endmarker ($).

• In what follows, we will consider only a family of constant-branching simple 2nfa’s.
Alternating Finite Automata (revisited)

- Recall the definition of 2afa’s from Week 1.
Narrow 2afa’s

• Instead of using computation trees, we use computation graphs.

• Here, we further consider additional restrictions on 2afa’s.

• Let $f: \mathbb{N} \to \mathbb{N}$ be a function.

• A family $\{M_n\}_{n \in \mathbb{N}}$ of 2afa’s is called $f(n)$-narrow if, for any $n \in \mathbb{N}$ and any input $x$ of length $n$, a $\{\forall, \exists\}$-leveled computation graph of $M_n$ on input $x$ has width at most $f(n)$ at every $\forall$-level.
t(n)-Time Space Constructibility

- We need a restricted notion of space constructibility.
- Let \( t, f: \mathbb{N} \to \mathbb{N} \) be functions.
- A function \( f \) is called \( t(n) \)-time space constructible \( \iff \) there exists a DTM \( M \) with a write-only output tape that, on each input \( 1^n \), \( M \) produces \( 1^{f(n)} \) on the output tape and halts within \( O(t(n)) \) steps.
Characterizing PsubLIN by Narrow 2afa’s

• Theorem: [Yamakami (2018)]

Let \( t, \ell: \mathbb{N} \rightarrow \mathbb{N}^+ \) be s.t. \( t \) is log-space computable and \( \ell \) is \( O(t(n)) \)-time space constructible. Let \( L \) and \( m \) be a language over alphabet \( \Sigma \) and a log-space size parameter.

1. \((L,m) \in \text{TIME,SPACE}(t(|x|),\ell(m(x)))\), then there are two constants \( c_1, c_2 > 0 \) and an \( L \)-uniform family \( \{M_n,\ell\}_{n,\ell \in \mathbb{N}} \) of \( c_2 \ell(m(x)) \)-narrow 2afa’s such that each \( M_n,|x| \) has at most \( c_1 t(|x|) \ell(m(x)) \) states and computes \( L(x) \) on all inputs satisfying \( m(x) = n \).

2. If there are constants \( c_1, c_2 > 0 \) and an \( L \)-uniform family \( \{M_n,\ell\}_{n,\ell \in \mathbb{N}} \) of \( c_2 \ell(m(x)) \)-narrow 2afa’s such that each \( M_n,|x| \) has at most \( c_1 t(|x|) \ell(m(x)) \) states and computes \( L(x) \) on all inputs satisfying \( m(x) = n \), then \((L,m) \) belongs to \( \text{TIME,SPACE}(t(|x|)\ell(m(x)),\ell(m(x)) + \log(t(|x|)) + \log|x|) \).
State Complexity Bounds

• The following assertion is an easy adaptation of Barnes et al.’s (1998) algorithm for DSTCON on top of the previous theorem.

• **Proposition:** [Yamakami (2018)]
  Every L-uniform family of constant-branching $O(n\log(n))$-state simple 2nfa’s can be converted into another L-uniform family of equivalent $O(n^{1-c/\sqrt{\log(n)}})$-narrow 2afa’s with $n^{O(1)}$-states for a certain constant $c>0$.

• **(Open Problem)**
  Is it possible to reduce the factor $n^{1-c/\sqrt{\log(n)}}$ to $n^\varepsilon$ for a certain constant $\varepsilon$ with $0 \leq \varepsilon < 1$?
Characterization of LSH: Uniform Case

• **Theorem:** [Yamakami (2018)]

The following statements are logically equivalent.

1. LSH fails.

2. For any two constants $c > 0$ and $k \geq 1$, there exists a constant $\varepsilon \in [0,1)$ such that every $L$-uniform family of constant-branching simple 2nfa’s of state at most $cn\log^k(n)$ can be converted into another $L$-uniform family of equivalent $O(n^{\varepsilon})$-narrow 2afa’s with $n^{O(1)}$ states.

3. For any constant $c > 0$, there exists a constant $\varepsilon \in [0,1)$ and a function $f \in \mathcal{FL}$ such that, on inputs of an encoding of $c$-branching simple $n$-state 2nfa, $f$ produces another encoding of equivalent $O(n^{\varepsilon})$-narrow 2afa of $n^{O(1)}$ states.
Direct Implications

- The previous theorem implies the following.
- If we need to prove the validity of LSH, it suffices to show that the state complexity of transforming an L-uniform family of constant-branching simple 2afa’s of \( O(n \cdot \text{polylog}(n)) \) states to an L-uniform family of equivalent \( O(n^\varepsilon) \)-narrow 2afa’s is super-polynomial in \( n \) for any \( \varepsilon \in [0,1) \).
Next, we give a state-complexity characterization of a non-uniform version of the linear space hypothesis.

In 2018, Yamakami introduced the non-uniform version of LSH.

Similarly to P/poly and L/poly, we can define a non-uniform version of PsubLIN (denoted by PsubLIN/poly) by supplementing polynomial-size advice to underlying DTMs with a read-only advice tape.

The non-uniform LSH states that $\text{\textsc{2SAT}}_{3,m_{vbl}}$ does not belong to PsubLIN/poly.
Characterization of LSH: Non-Uniform Case

• We also obtain a non-uniform version of the previous characterization of LSH.

• **Theorem:** [Yamakami (2018)]
The following statements are logically equivalent.

1. The non-uniform LSH fails.
2. For any constant \( c > 0 \), there exists a constant \( \varepsilon \in [0,1) \) such that every \( c \)-branching simple \( n \)-state 2nfa can be converted into an equivalent \( O(n^\varepsilon) \)-narrow 2afa of \( n^{O(1)} \) states.
Open Problems

1. Is it possible to reduce the factor $n^{1-c/\sqrt{\log(n)}}$ to $n^\varepsilon$ for a certain constant $\varepsilon$ with $0 \leq \varepsilon < 1$?
2. Prove or disprove that LSH is true.
3. Find a different characterization of LSH.
4. Find natural applications of the characterization of LSH in terms of state complexity.
V. Type-2 Computability

1. Historical Account
2. Functionals and Relations
3. Function-Oracle Turing Machines
4. Type-2 Computation
5. Power of Generic Oracles
6. Close Connection to Generic Oracles
7. The Polynomial Hierarchy of Type 2
8. Hierarchy Theorem
9. Regular/Irregular Complexity Classes
Historical Account

- **Constable** (1973) and **Mehlhorn** (1973, 1976) initiated a functional approach to the study on the polynomial-time computability.
- **Buss** (1986) also considered polynomial-time computability of type-2 functionals.
- In a slightly different way, **Ko** (1985) considered complexity-bounded class of operators.
- **Yamakami** (1995) further developed a theory of type-2 functionals and also introduced a type-2 analogue of the polynomial-time hierarchy, extending Townsend’s framework.
Functionals and Relations

• $\omega \equiv N$ (the set of all non-negative integers)
• $\omega_\omega$ = the set of all total functions from $\omega$ to $\omega$
• $k, l_\omega = \omega^k \times (\omega_\omega)^l$  E.g., $3, 2_\omega = \omega \times \omega \times \omega \times \omega_\omega \times \omega_\omega$
• $(m, \alpha) \in k, l_\omega \iff m \in \omega^k$ and $\alpha \in (\omega_\omega)^l$
• A partial functional $F$ of rank $(k, l)$ satisfies that
  $\text{Dom}(F) \subseteq k, l_\omega$ and $\text{Im}(F) \subseteq \omega$.
• A total functional $F$ of rank $(k, l)$ satisfies that
  $\text{Dom}(F) = k, l_\omega$ and $\text{Im}(F) \subseteq \omega$.
• A relation $R$ of rank $(k, l)$ is a subset of $k, l_\omega$. (namely, $R \subseteq k, l_\omega$.)
Here, we use a function $f$ as an oracle, which returns values (not limited to YES or NO) of $f$ when a query is invoked, directly to a designated tape, called a query tape.

1. An underlying oracle Turing machine $M$ wants to make a query to the function oracle $f$ by writing a query word $z$ on the query tape.
2. $M$ enters a query state $q_{\text{query}}$.
3. The query word $z$ is sent to the function oracle $f$, the tape automatically becomes empty (i.e., blank), and the tape head of this tape jumps to the start cell.
4. The function oracle $f$ returns $f(z)$ by writing it down onto the query tape and changes $M$’s inner state to $q_{\text{answer}}$.
5. $M$ can now read some symbols of $f(z)$ by moving its tape head back and forth.

See the next slide!
Query-and-Answer Mechanism

query tape

\[ z \]

\[ \text{blank} \]

\[ q_{\text{query}} \]

\[ q_{\text{answer}} \]

\[ f(z) \]

function oracle

\[ f \]
Type-2 Computation

• A partial functional $F$ is polynomial-time computable if it is computed by a certain function-oracle Turing machine with an output tape.

• (*) When a function oracle returns an extremely long bits of an answer to a query, a time-bounded machine may not read all bits of this answer.

• A relation $R$ is called polynomial-time computable if there exists a deterministic function-oracle Turing machine that recognizes $R$. 
Functional Classes  Ptf and Ptf(A)

• We define a functional class, called Ptf.

• $\text{Ptf} =$ class of all polynomial-time computable total functionals

• Let A be any language.

• $\text{Ptf}(A) =$ class of all functionals computed by polynomial-time function-oracle Turing machines with output tapes using oracle A

• Let C be any family of languages (or a complexity class).

• Let $\text{Ptf}(C) = \bigcup_{A \in C} \text{Ptf}(A)$. 
The Polynomial Hierarchy of Type 2

• We define the polynomial(-time) hierarchy of type 2. [Townsend (1982, 1990), Yamakami (1995)]

\[ \Delta_0^{0,p} = \Sigma_0^{0,p} = \Pi_0^{0,p} = \text{class of polynomial-time computable relations} \]

\[ \Box_0^{0,p} = Ptf \]

\[ \Sigma_{k+1}^{0,p} = \{(\exists x \leq F(m,\alpha)) R(x,m,\alpha) \mid R \in \Pi_k^{0,p}, F \in \Box_0^{0,p}\} \]

\[ \Pi_{k+1}^{0,p} = \{(\forall x \leq F(m,\alpha)) R(x,m,\alpha) \mid R \in \Sigma_k^{0,p}, F \in \Box_0^{0,p}\} \]

\[ \Box_{k+1}^{0,p} = Ptf \left( \Sigma_k^{0,p} \right) \]

\[ \Delta_{k+1}^{0,p} = \{R \mid \chi_R \in \Box_{k+1}^{0,p}\} \]
Hierarchy Theorem

• Townsend (1990) proved the following.

  Hierarchy Theorem: for all $n \geq 1$,
  \[
  \Delta_{n}^{0,p} \neq \Sigma_{n}^{0,p} \neq \Pi_{n}^{0,p} \\
  \Box_{n}^{0,p} \neq \Box_{n+1}^{0,p}
  \]

  \[
  \Delta_{k+1}^{NP} = \left\{ R \mid \chi_{R} \in Ptf \left( \Sigma_{k-1}^{0,p} \cup \Sigma_{k}^{p} \right) \right\}
  \]

  This is compared to $\Delta_{k+1}^{0,p} = \{ R \mid \chi_{R} \in Ptf(\Sigma_{k}^{0,p}) \}$.

  Proposition: [Yamakami (1995)] for all $n \geq 1$,

  \[
  \Delta_{n}^{p} = \Sigma_{n}^{p} \iff \Delta_{n}^{0,p} = \Delta_{n+1}^{NP} \\
  \Sigma_{n}^{p} = \Pi_{n}^{p} \iff \Delta_{n+1}^{NP} \subseteq \Sigma_{n}^{0,p} \cap \Pi_{n}^{0,p}
  \]
Relativization and Type-2 Computation

• Let \( C \) be any “typical” type-1 complexity class.

• Let \( \overline{C} \) be any “natural” type-2 counterpart, based on the same resource-bounds used to define \( C \), and for each oracle \( A \), a natural relativized version \( C^A \).

• For example, we can take the following classes as \( C \): \( \text{P, NP, BPP, NP} \cap \text{co-NP} \), etc.

• Given a type-2 relation \( R \) and an oracle \( A \), we define the type-1 relation \( R[A] \) as

\[
R[A] (x) = R(x,A)
\]

for every type-1 object \( x \).

• For a class \( \overline{C} \) of type-2 relations, let \( \overline{C[A]} = \{ R[A] | R \in C \} \)
Regular Complexity Classes

- Let $C$ be any “typical” type-1 complexity class and let $\overline{C}$ be any “natural” type-2 counterpart, based on the same resource-bounds used to define $C$, and for each oracle $A$, a natural relativized version $C^A$.

- We say that $C$ is regular if, for all $A$, $C^A = \overline{C} [A]$

- (Claim)
  
  $P$ and $NP$ are regular.

  Namely, for any oracle $A$, it follows that

  $$P^A = \overline{P} [A]$$

  $$NP^A = \overline{NP} [A]$$
Irregular Complexity Classes

• A complexity class C that is not regular is called **irregular**.

• **Question:** is there any irregular complexity class?

• **Proposition:** [Cook-Impagliazzo-Yamakami (1997)]
  
  $NP \cap co-NP$ and $BPP$ are irregular. That is, there exist oracles $A$, $B$ such that

  $$NP^A \cap co-NP^A \neq \left(\overline{NP} \cap co-\overline{NP}\right)[A]$$

  $$BPP^B \neq \overline{BPP} [B]$$
Close Connection to Generic Oracles

- Recall the notion of generic oracle from Week 4.
- There is a close connection between type-2 computability and generic oracle.

- Let $C$ and $D$ be classes of computable type-2 relations.
- Assume that $C$ and $D$ are closed under $\leq_{p_m}$-reductions.
- For any generic oracle $G$,

$$
\overline{C} \subseteq \overline{D} \iff C[G] \subseteq D[G]
$$
Power of Generic Oracles

• Recall that there are oracles A and B for which

\[ NP^A \cap co-NP^A \neq (\overline{NP} \cap co-\overline{NP})[A]\]

\[ BPP^B \neq \overline{BPP} \ [B]\]

• However, we can show the following for generic oracles.

• Proposition: [Cook-Impagliazzo-Yamakami (1997)]
For any generic oracle G,

\[ NP^G \cap co-NP^G = (\overline{NP} \cap co-\overline{NP})[G]\]

\[ BPP^G = \overline{BPP} \ [G]\]
Open Problems

• Develop a theory of computability of higher types.
• Find more complexity classes $C$ such that
  1. there is an oracle $A$ satisfying
     $$\overline{C^A} \neq \overline{C} [A]$$
  2. for all generic oracle $G$,
     $$\overline{C^G} = \overline{C} [G]$$
Thank you for listening
Q & A

I’m happy to take your question!
END