6^{th} Week

Type-2 Computability, Multi-Valued Functions, and State Complexity

Synopsis.

- Multi-Valued Partial CFL Functions
- CFLMV Hierarchy
- State Complexity for LSH
- Function-Oracle Turing Machines
- Type-2 Computability

May 14, 2018. 23:59

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

- K. Tadaki, T. Yamakami, J. C. H. Lin. Theory of one-tape linear-time Turing machines. Theor. Comput. Sci. 411(1): 22-43 (2010)
- T. Yamakami. Not all multi-valued partial CFL functions are refined by single-valued functions (extended abstract). In Proc. of IFIP TCS 2014, LNCS, vol. 8705, pp. 136-150 (2014)
- T. Yamakami. Structural complexity of multi-valued partial functions computed by nondeterministic pushdown automata (extended abstract). ICTCS 2014, CEUR Workshop Proceedings 1231, CEUR-WS.org 2014, pp. 225-236 (2014)
- T. Yamakami. State complexity characterizations of parameterized degree-bounded graph connectivity, sub-linear-bounded computation, and the linear space hypothesis. Preprint, March 2018. To appear shortly at arXiv.org.

- (Continued to the next slide)

Main References by T. Yamakami II

- ✎ T. Yamakami. Structural properties for feasibly computable classes of type two. Mathematical Systems Theory 25(3): 177-201 (1992)
- ✎ T. Yamakami. Feasible computability and resource bounded topology. Inf. Comput. 116(2): 214-230 (1995)
- ✎ S. A. Cook, R. Impagliazzo, and T. Yamakami. A tight relationship between generic oracles and type-2 complexity Theory. Inf. Comput. 137(2): 159-170 (1997)

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. CFL∩co-CFL vs. CFLSV
6. Functional Pumping Lemma
7. Function Class NFAMV
8. Boolean Operators
9. Basic Properties

Multi-Valued Partial Functions

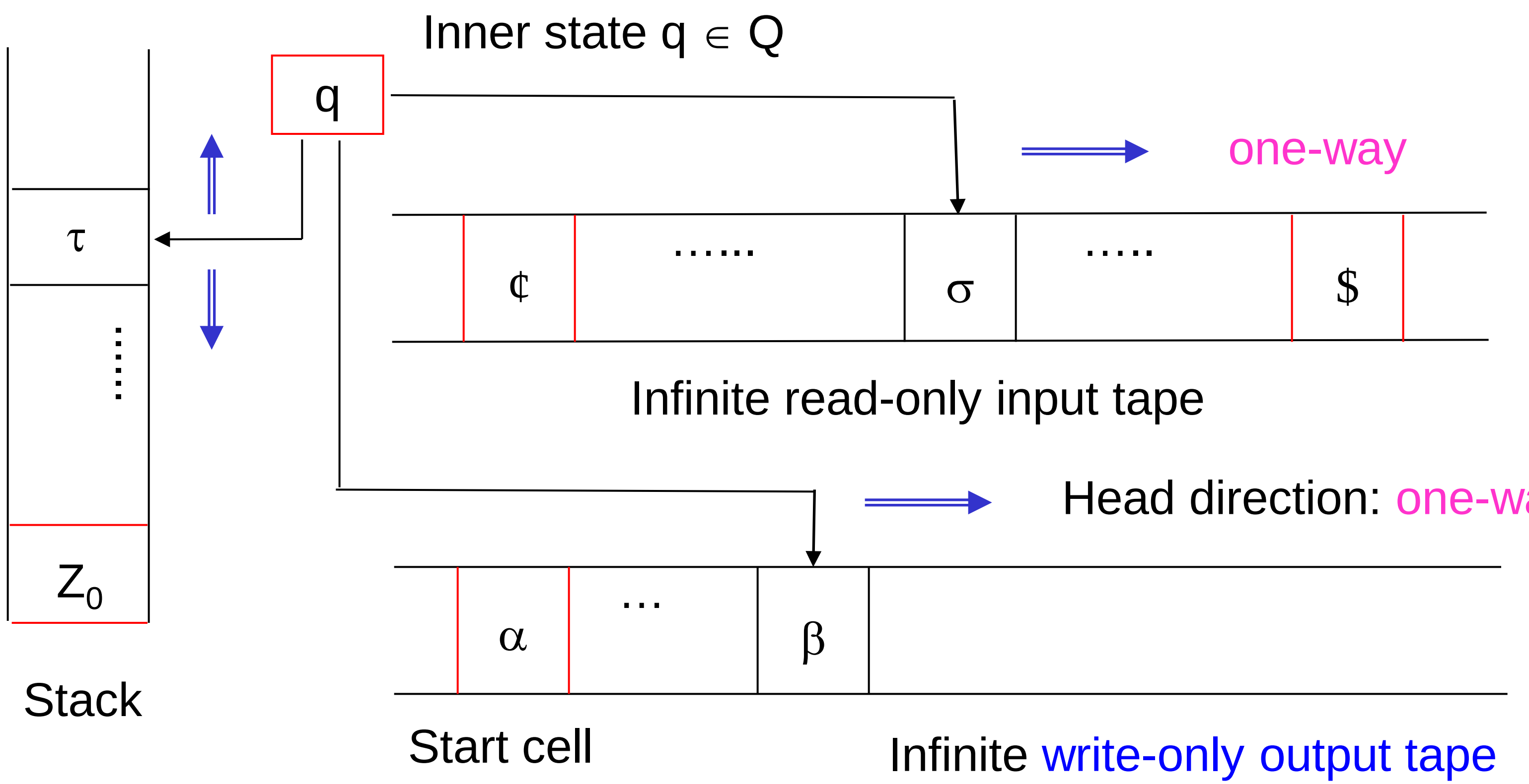
- A (standard) function is designed to produce only one output value per each input.
- We can allow a function to output more than one value simultaneously, or even allow it to output no value at all.
- A total function is a standard function f such that, for any input x, its output f(x) always exists.
- 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 ≤ 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.

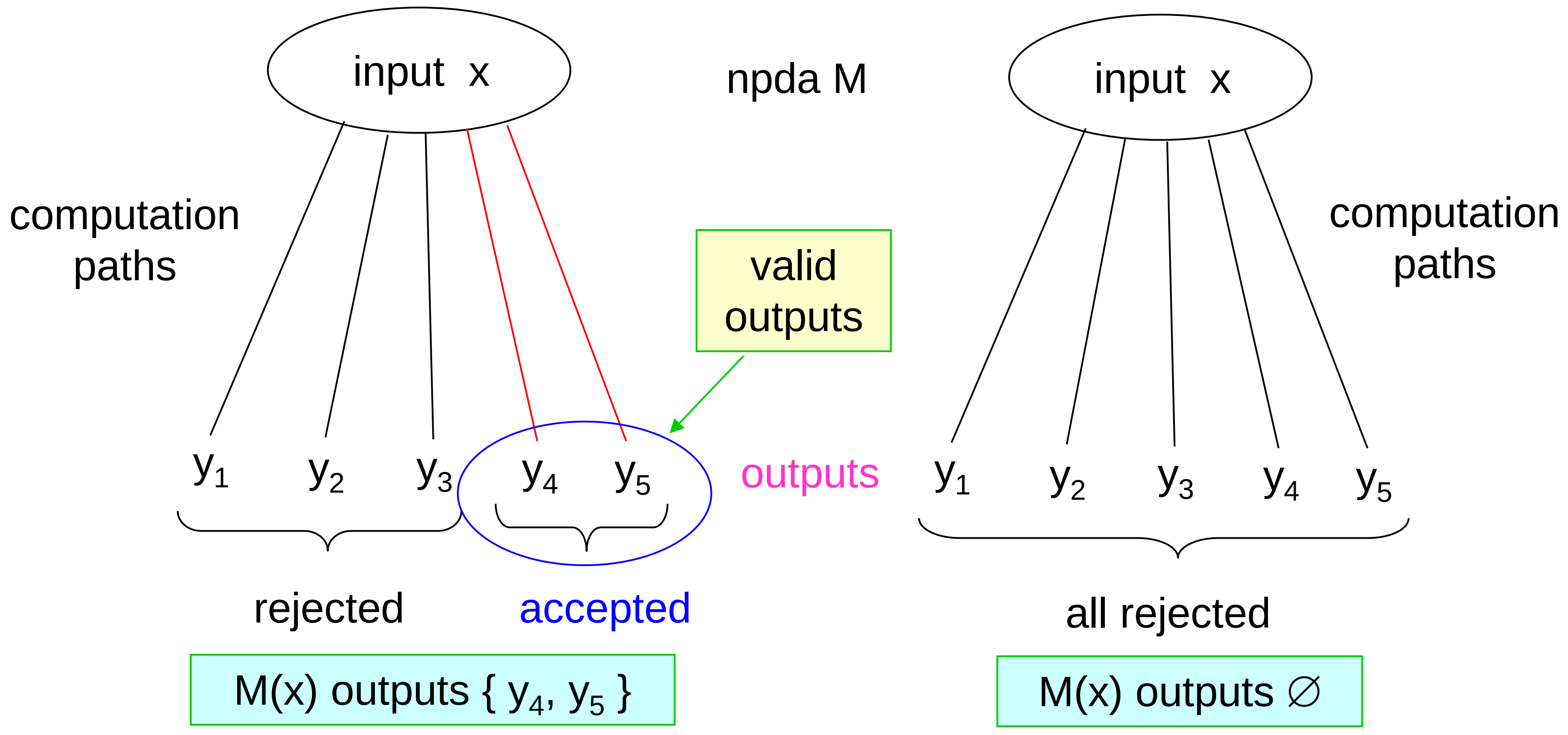
- We say that a 1npda M computes a multi-valued partial function f: $\Sigma_1^* \to \wp(\Sigma_2^*)$ if M satisfies the following:
  1. for any $x \in \mathrm{dom}(f)$, M produces exactly all values in f(x) along accepting computation paths, and
  2. for any string $x \in \Sigma_1^* - \mathrm{dom}(f)$,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



Formal Definition

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

$$\delta : (Q - Q_{halt}) \times (\breve{\Sigma} \cup \{\lambda\}) \times \Theta \to P(Q \times \Gamma^* \times (\Gamma \cup \{\lambda\}))$$

$\breve{\Sigma} = \Sigma \cup \{ ¢, \$ \}$

$Q_{halt} = Q_{acc} \cup Q_{rej}$

- Termination condition of M:
  - All computation paths (both accepting and rejecting) should terminate (reaching halting states) **within O(n) time**.
  - $ACC_M(x)$ = set of accepting computation paths of M on x

This is because all context-free languages are recognized by O(n)-time npda's.

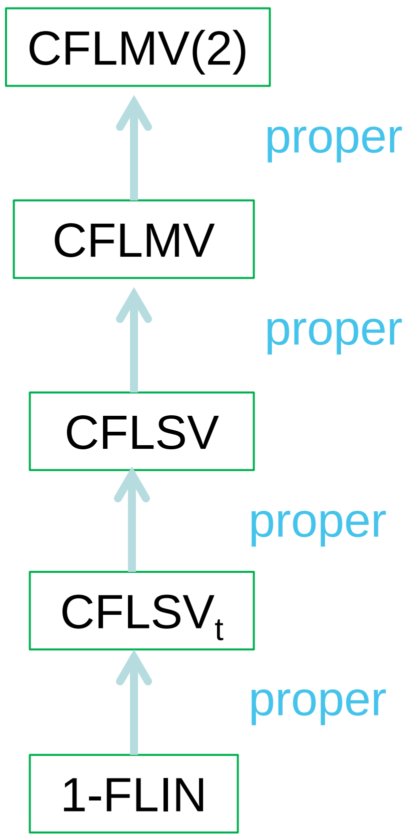
Multi-Valued Partial CFL Functions

♠ 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

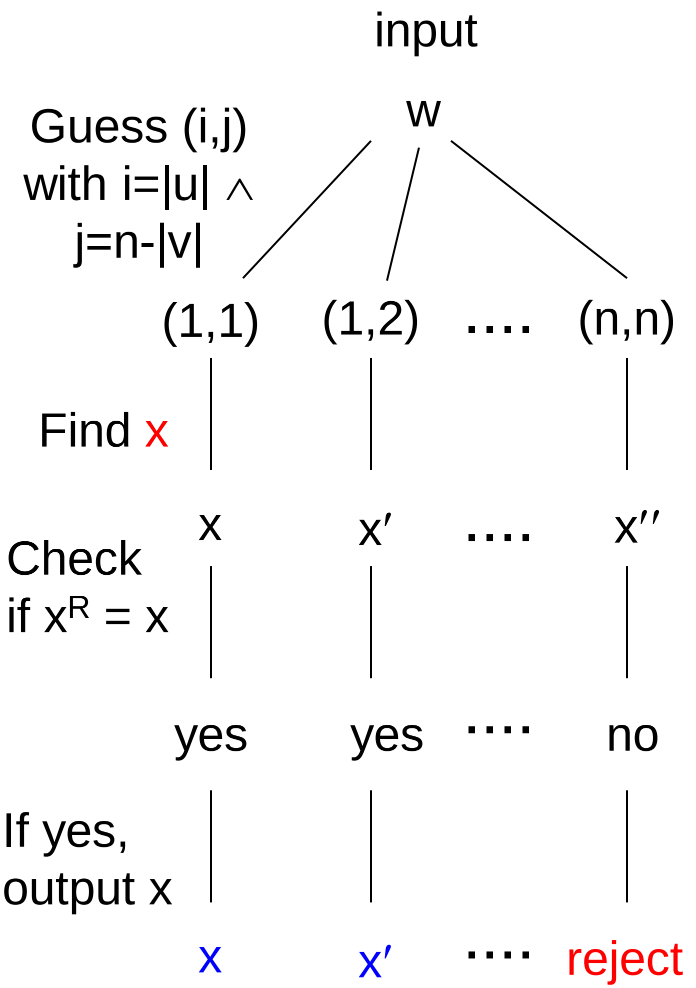
- CFLMV, CFLSV, and $CFLSV_t$ are analogues of NPMV, NPSV, and $NPSV_t$, respectively.

Containment & separation



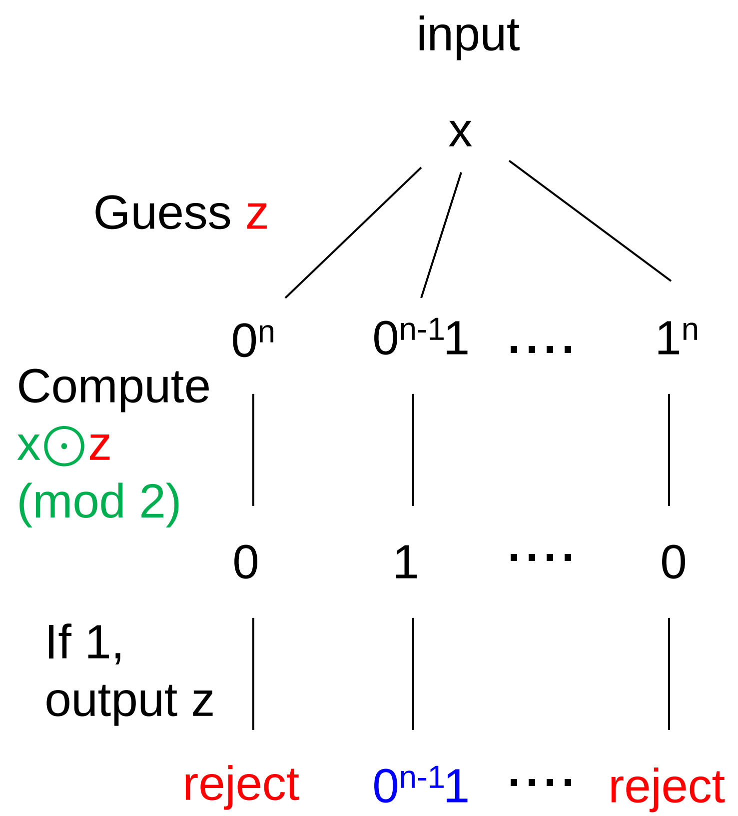
Examples: PAL

- Here, we take a look at two simple examples.
- $\mathrm{PAL}(w) = \{\, x \mid \exists u,v\, [\, w = uxv \,] \wedge x = x^R \,\}$ for all $w \in \{0,1\}^*$.
- I.e., PAL(w) outputs all possible palindrome blocks in w.
- The right-hand side illustration shows how to compute PAL.
- Thus, PAL is in 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$\cap$co-CFL vs. CFLSV

- CFLSV is closely related to the language family CFL$\cap$co-CFL.

- Recall that χ_A is the characteristic function of a language A.

- Lemma: [Yamakami (2016)]

  Let A be any language.

  $A \in$ CFL$\cap$co-CFL $\Leftrightarrow$ $\chi_A \in$ CFLSV
- We can replace CFLSV by CFLSV_t and CFLMV.

Functional Pumping Lemma for CFLMV

- Pumping Lemma for CFLMV: [Yamakami (2014)]

Let Σ and Γ be any alphabets and let $f:\Sigma^*\to\wp(\Gamma^*)$ be any function in CFLMV. There exist 3 numbers $m \in N^+$ and $c,d \in N$ 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) $ab^ipq^ir \in f(uv^ixy^iz)$.

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

$f_1 = g_1 \wedge g_2$

- $f_1 \in F \wedge G$

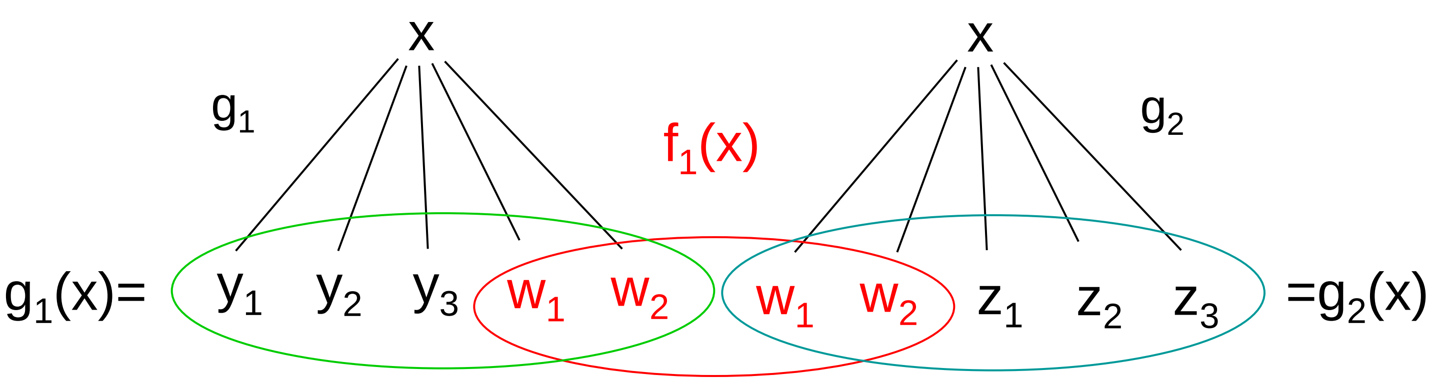
$$\Leftrightarrow \quad \exists g_1 \in F\ \exists g_2 \in G \text{ s.t. } \forall x\,[\,f_1(x) = g_1(x) \cap g_2(x)\,]$$

❑ Disjunction of F and G

$f_1 = g_1 \vee g_2$

- $f_2 \in F \vee G$

$$\Leftrightarrow \quad \exists g_1 \in F\ \exists g_2 \in G \text{ s.t. } \forall x\,[\,f_2(x) = g_1(x) \cup g_2(x)\,]$$



Simple Examples of $f \vee g$ and $f \wedge g$

- Here, we present two simple examples.
- Consider the following f and g.
  - $f(x) = \{ a^n b^n \mid n=|x| \}$
  - $g(x) = \{ a^n b^{2n} \mid n=|x| \}$
  - $(f \vee g)(x) = \{ a^n b^n, a^n b^{2n} \mid n=|x| \}$
- Consider the following f and g.
  - $f(x) = \{ a^n b^n c^m \mid n=|x|, m \geq 0 \}$
  - $g(x) = \{ a^m b^n c^n \mid n=|x|, m \geq 0 \}$
  - $(f \wedge g)(x) = \{ a^n b^n c^n \mid n=|x| \}$

Function Classes CFLMV(k)

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

- Lemma: [Yamakami (2014)]
  1) CFLMV(max{k,m}) $\subseteq$ CFLMV(k)$\vee$CFLMV(m) $\subseteq$ CFLMV(km).
  2) CFLMV(max{k,m}) $\subseteq$ CFLMV(k)$\wedge$CFLMV(m) $\subseteq$ CFLMV(k+m).
  3) CFLSV(k) $\neq$ CFLSV(k+1) for any k$\geq$1.

Difference/Complement of Functions

- We define the difference/complement of function classes.

❑ Difference between F and G

- $f \in F \ominus G \iff \exists g_1 \in F\ \exists g_2 \in G$ s.t. $\forall x\ [\ f(x) = g_1(x) - g_2(x)\ ]$

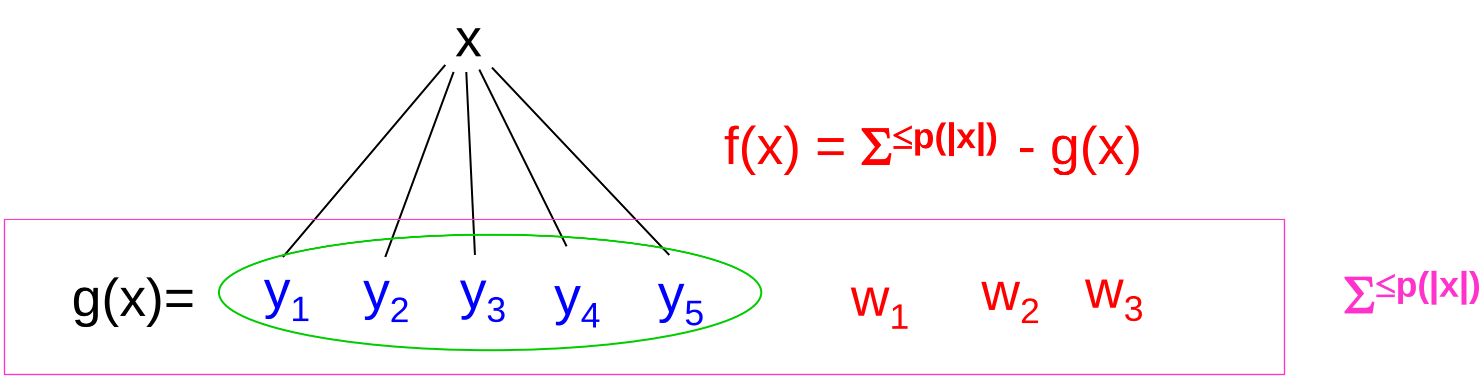
set difference

❑ Complement of F

- $f \in$ co-F

$\iff \exists g \in F\ \exists p$: linear polynomial $\exists n_0$: constant s.t.

$\forall (x,y)$ with $|x| \geq n_0\ [\ y \in f(x) \leftrightarrow |y| \leq p(|x|) \wedge y \in g(x)\ ]$



Boolean Operations

- Using two operators $\ominus$ and co-, we define the following function classes.
  - co-CFLMV (complement)
  - CFLMV $\ominus$ 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 $IP^c(x) = \{ z \mid |x|\geq|z|, x\odot z0^{|x|-|z|} \equiv 0 \pmod 2 \}$ for any x.
- It follows that $IP^c \in$ co-$CFLMV_t$ since $IP_2 \in CFLMV_t$ and $IP^c(x) = \Sigma^{\leq|x|} - IP_2(x)$ for any x.

Basic Properties

- The following basic properties hold.

- Proposition: [Yamakami (2014)]
  1. co-(co-CFLMV) = CFLMV
  2. co-CFLMV = NFAMV $\ominus$ CFLMV
  3. CFLMV $\ominus$ CFLMV = CFLMV $\wedge$ co-CFLMV
  4. CFLMV $\neq$ co-CFLMV
  5. $\mathrm{CFLMV}_t \neq \text{co-}\mathrm{CFLMV}_t$

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., $\mathrm{CFLSV}_t \neq \mathrm{CFLSV} \neq \mathrm{CFLMV}$ holds.
- Let f,g be any two functions from Σ^* to $\wp(\Gamma^*)$.

  g is a refinement of f (notationally, $f \sqsubseteq_{ref} g$)

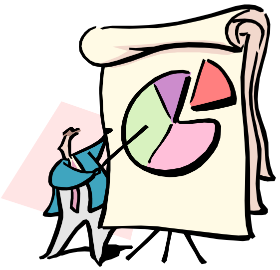
  $\Leftrightarrow \forall x \in \Sigma^*$

  1. $f(x) \neq \varnothing \Leftrightarrow g(x) \neq \varnothing$
  2. $g(x) \subseteq f(x)$ (as set inclusion).

Refinement is also known as uniformization.

- For two function classes F and G,

  $F \sqsubseteq_{ref} G \Leftrightarrow \forall f \in F\ \exists g \in G\ [\, f \sqsubseteq_{ref} g\,]$
- NOTE: $F \subseteq G \Rightarrow F \sqsubseteq_{ref} G$.



Example: maxPAL

- Let us see an example of refinement.
- Recall $PAL(w) = \{ x \mid \exists u,v [w = uxv] \wedge x = x^R \}$.
- For each $w \in \{ 0,1 \}^*$, we define

  maxPAL(w) = maximum element in PAL(w),

  where "maximum" is according to a dictionary order.
- maxPAL is a single-valued total function.
- (Claim) PAL $\sqsubseteq_{ref}$ maxPAL (PAL is refined by maxPAL)

❑ Proof: This is because dom(PAL) = dom(maxPAL) and maxPAL(x) $\subseteq$ 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.

Refinement Separation: CFLMV II

- Here, we claim the desired separation result.
- Theorem: [Yamakami (2014)]

  UCFL2V $\not\subseteq_{\mathrm{ref}}$ CFLSV.

- The above theorem implies that CFLMV $\not\subseteq_{\mathrm{ref}}$ 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. $h_3 \in$ UCFL2V.
  2. h_3 has no refinement in CFLSV.

Refinement Separation: CFLMV III

- The desired function h_3 is defined as follows.

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

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

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

$$h_3(w) = \begin{cases} \left\{ 0^i 1^j \mid (i, j) \in I_3, x_i^R = x_j \right\} & \text{if } w = x_1 \# x_2 \# x_3 \in L, \\ \varnothing & \text{if } w \notin L. \end{cases}$$

QED

- For example,
  - ✓ $h_3(001\#100\#000) = \{ 0^1 1^2 \}$
  - ✓ $h_3(001\#100\#001) = \{ 0^1 1^2, 0^2 1^3 \}$
  - ✓ $h_3(111\#011\#101) = \varnothing$

$001^R = 100$

$001^R = 100$, $100^R = 001$

1-FLIN(partial), 1-NLINMV, and 1-NLINSV

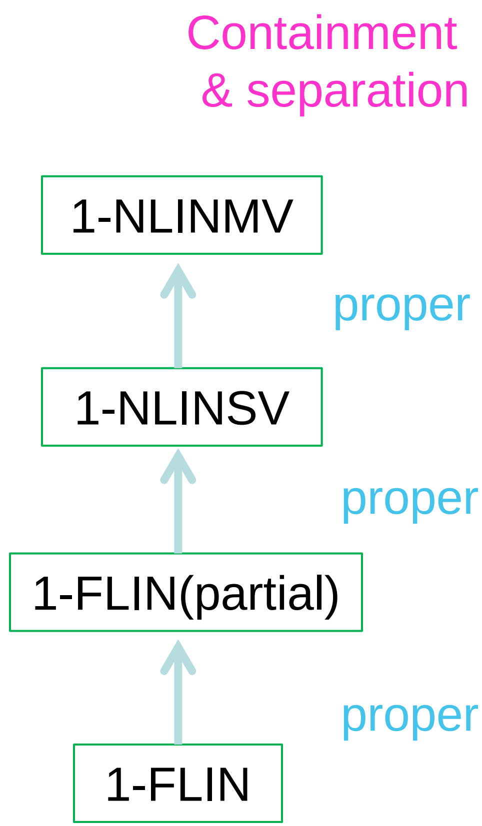
- Recall 1-FLIN from Week 1.
- Here, we relax the function condition of 1-FLIN to obtain 1-FLIN(partial), which is composed of all partial functions computable by 1DTM in linear time with no extra output.
- In other words, if we restrict all partial functions in 1-FLIN(partial) to be total, we immediately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

- A multi-valued partial function f: $\Sigma_1^* \to \wp(\Sigma_2^*)$ is in 1-NLINMV if there exists a 1NTM M such that
  1. for any string $x \in dom(f)$, M produces exactly all values in f(x) along accepting computation paths, and
  2. for any string $x \in \Sigma_1^*$-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 $f: \Sigma_1^* \to \Sigma_2^*$ is length-preserving if, for any input $x \in \Sigma_1^*$, $|f(x)| = |x|$ holds.
- Theorem: [Tadaki-Yamakami-Lin (2010)]

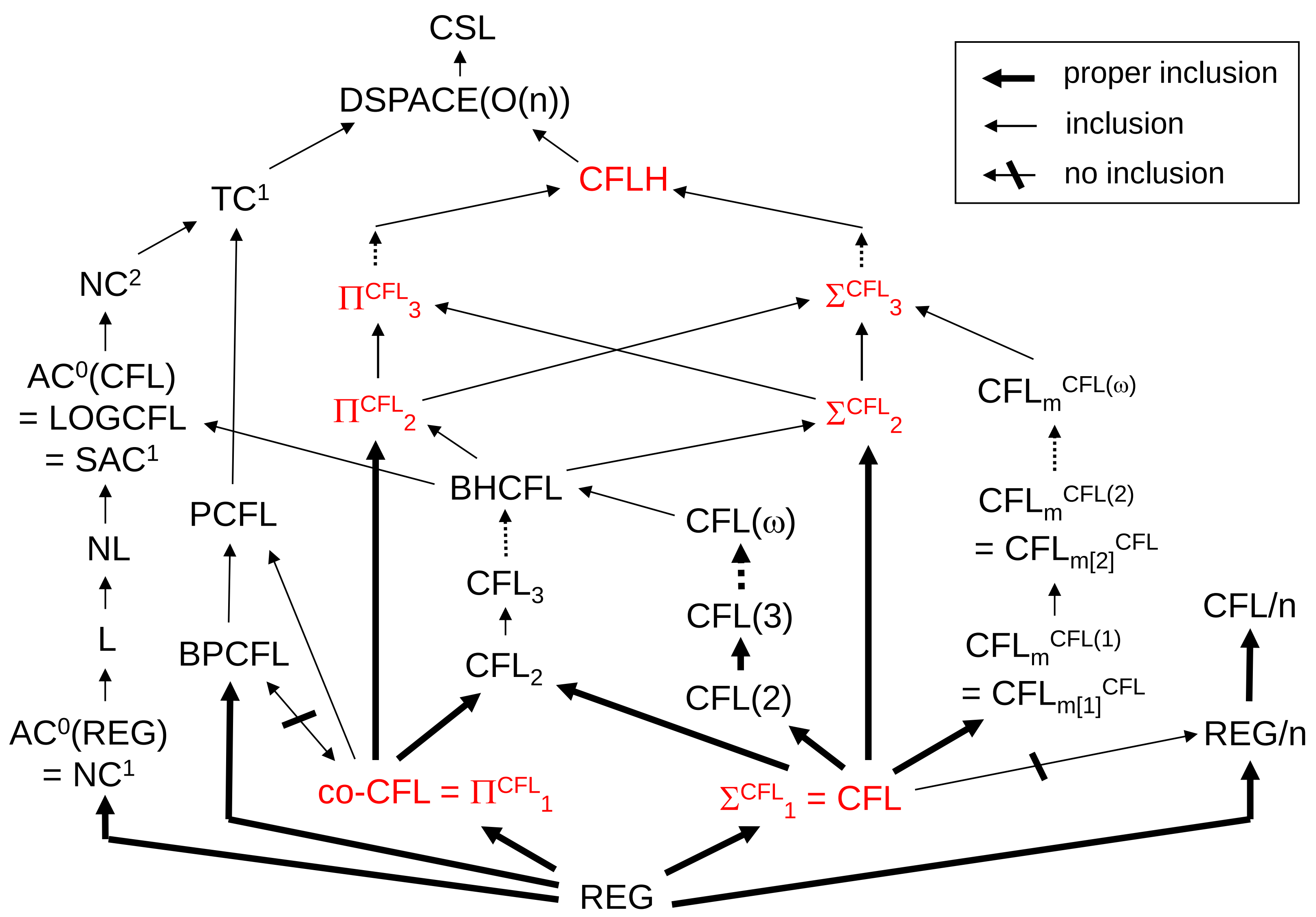
  Every length-preserving 1-NLINMV function has a 1-FLIN(partial) refinement.
- (*) This will be used for one-way functions in Week 7.



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)



The CFLMV Hierarchy

- Similarly to CFL^A (relative to A), we can relativize CFLMV to oracle A and obtain $CFLMV^A$ by attaching query tapes to underlying 1npda's with output tapes.
- We then define the CFLMV hierarch as follows.

$$\Sigma_1^{CFL}MV = CFLMV^A; \;\; \Sigma_{k+1}^{CFL}MV = CFLMV^{\Sigma_k^{CFL}}$$

- Similarly, we define the CFLSV hierarchy by setting:

$$\Sigma_k^{CFL}SV = \{ f \in \Sigma_k^{CFL}MV \mid f \text{ is single-valued } \}$$

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

1. $\Sigma^{CFL}{}_k SV \sqsubseteq_{\mathrm{ref}} \Sigma^{CFL}{}_k MV$.
2. $\Sigma^{CFL}{}_k SV = \Sigma^{CFL}{}_{k+1} SV \Rightarrow \Sigma^{CFL}{}_k = \Sigma^{CFL}{}_{k+1}$
3. $\Sigma^{CFL}{}_k = \Sigma^{CFL}{}_{k+1} \Rightarrow \Sigma^{CFL}{}_k SV = \Sigma^{CFL}{}_{k+1} SV$

Refinement Separations and Collapses

- We have seen CFLMV $\not\sqsubseteq_{ref}$ CFLSV. This is equivalent to Σ^{CFL}_{0}MV $\not\sqsubseteq_{ref}$ Σ^{CFL}_{0}SV.
- (Open Problem) Is Σ^{CFL}_{k}MV $\not\sqsubseteq_{ref}$ Σ^{CFL}_{k}SV for each $k \geq 2$?
- Related to this question, we obtain the following.
- Lemma: [Yamakami (2014)] ($k \geq 1$)
  - Σ^{CFL}_{k}MV $\sqsubseteq_{ref}$ Σ^{CFL}_{k+1}SV
- Theorem: [Yamakami (2014)] ($k \geq 2$)
  - $\Sigma^{CFL}_{k} = \Sigma^{CFL}_{k+1} \Rightarrow \Sigma^{CFL}_{k+1}$MV $\sqsubseteq_{ref}$ Σ^{CFL}_{k+1}SV.
- Corollary: [Yamakami (2014)] ($k \geq 2$)
  - Σ^{CFL}_{k}MV $\sqsubseteq_{ref}$ Σ^{CFL}_{k}SV $\Rightarrow$ 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 REG//F $\Leftrightarrow$ there are a language B $\in$ REG and a function h $\in$ F such that, for any x,

$$x \in L \Leftrightarrow \exists y \in h(x) \text{ s.t. } \begin{bmatrix} x \\ y \end{bmatrix} \in B$$

- Analogously, CFL//F is defined using CFL instead of REG.

Basic Properties

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

- Proposition: [Yamakami (2014)]
  1. $\mathrm{REG//NFASV_t} \not\subseteq \mathrm{CFL}$ and $\mathrm{CFL} \not\subseteq \mathrm{REG//NFAMV}$.
  2. $\mathrm{REG//NFASV_t} = \text{co-}(\mathrm{REG//NFASV_t})$
  3. $\mathrm{REG//NFAMV} \neq \text{co-}(\mathrm{REG//NFAMV})$
  4. $\mathrm{CFL} \cap \text{co-}\mathrm{CFL} \neq \mathrm{REG//CFLSV_t}$
- (*) The last claim is compared to $\mathrm{NP} \cap \text{co-}\mathrm{NP} = \mathrm{P//NPSV_t}$. [Köbler-Thierauf (1994)]

- Proposition: [Yamakami (2014)]
  - $\Sigma^{\mathrm{CFL}}_k \cap \Pi^{\mathrm{CFL}}_k = \mathrm{REG//}\Sigma^{\mathrm{CFL}}_k \mathrm{SV_t}$. 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

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

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

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

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

Separations

- We show a simple separation result.

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

❑ Proof Sketch:

- Define $f_{dup\#}(x) = \{ x\#x \}$ for any $x \in \{0,1\}^*$.
- Clearly, $f_{dup\#}(x) \in \mathrm{CFLSV}^{(2)}_t$.
- However, if $f_{dup\#}(x) \in \mathrm{CFLSV}_t$, then the language $\mathrm{DUP}_\#$ $= \{ x\#x \mid x \in \{0,1\}^* \}$ must belong to CFL.
- Since $\mathrm{DUP}_\# \notin \mathrm{CFL}$, we conclude $f_{dup\#}(x) \notin \mathrm{CFLSV}_t$.

QED

OptCFL

- Krentel (1988) introduced a function class OptP, which consists of the optimal cost functions of NP optimization problems.
- Similarly, Yamakami (2014) considered its pushdown-automaton version, which is called OptCFL.
- We assume the standard lexicographic order on Σ^*.
- A function f: $\Sigma^* \to \Sigma^*$ is in OptCFL $\Leftrightarrow$ there exists a 1npda M with a write-only output tape s.t.

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

where opt $\in$ { 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 $\Sigma^{CFL}_{k}SV \not\subseteq_{ref} \Sigma^{CFL}_{k}MV$ for all $k \geq 2$.
3. Prove that OptCFL $\not\subseteq \Sigma^{CFL}_{2}SV_{t}$ or OptCFL $\not\subseteq \Sigma^{CFL}_{3}SV_{t}$.

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

- We consider a family $\{M_n\}_{n\in N}$ of finite automata, each M_n of which is of the form $(Q_n,\Sigma_n,\delta_n,q_{0n},Q_{acc,n},Q_{rej,n})$.
- We often take the same input alphabet $\Sigma_n = \Sigma$ for all n.

- Note that the state complexity of this family $\{M_n\}_{n\in N}$ becomes a function $st(n) = |Q_n|$ 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 $\{M_n\}_{n\in N}$ be any family of finite automata, each M_n of which is of the form $(Q_n,\Sigma_n,\delta_n,q_{0n},Q_{acc,n},Q_{rej,n})$.
- This family $\{M_n\}_{n\in N}$ is called L-uniform if there exists a log-space DTM A with a write-only output tape such that, for any length $n\in N$, 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,

$$M \text{ accepts } x \leftrightarrow N \text{ accepts } x.$$

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

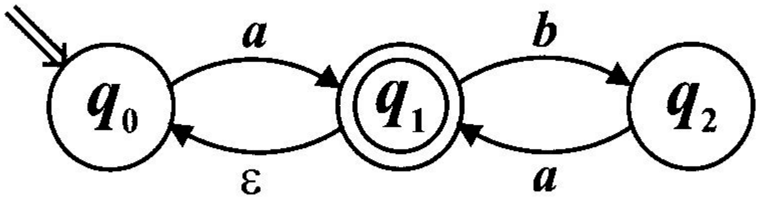


Fig.1

- Fig.1 is a 1nfa with 3 states, and Fig.2 is its equivalent 1dfa with 4 states.

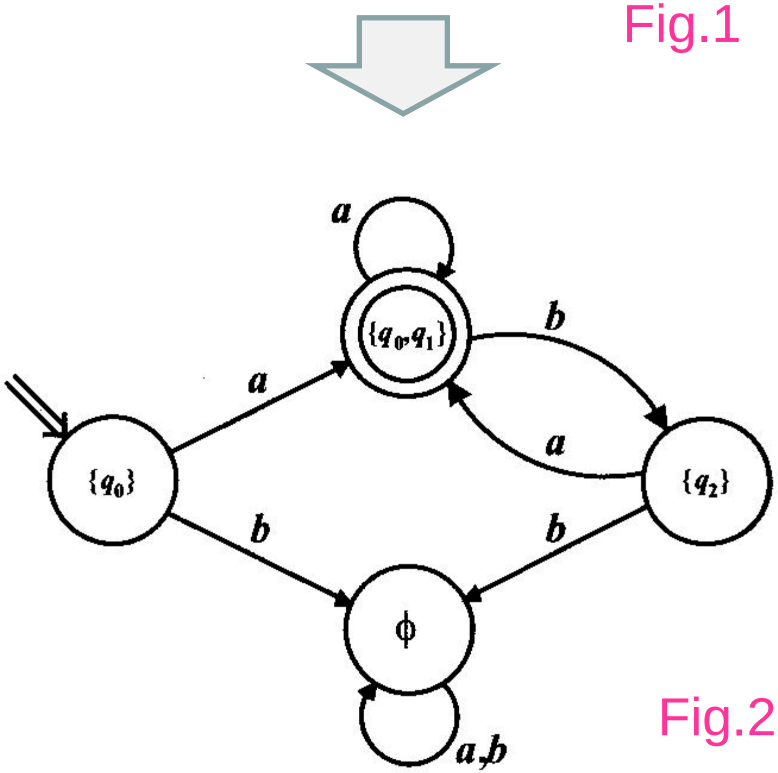


Fig.2

Characterization of NL$\subseteq$L/poly

- Recall the non-uniform class L/poly from Week 3.
- Note that we do not know whether or not NL$\subseteq$L/poly.
- Kapoutsis (2014) and Kapoutsis and Pighizzini (2015) gave a new characterization of NL$\subseteq$L/poly in terms of state-complexity of transforming 2nfa's to 2dfa's.
- (Claim) The following statements are logically equivalent.
  1. NL$\subseteq$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 $2SAT_3$) states:

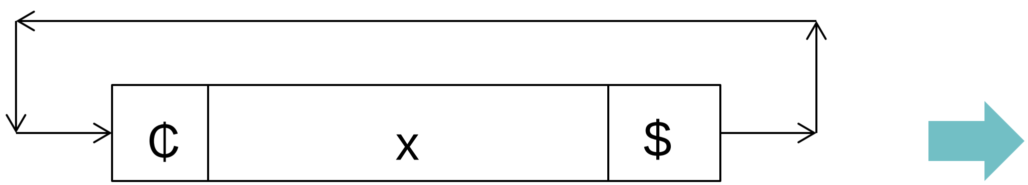
  There is no deterministic algorithm that solves $2SAT_3$ in time $p(|x|)$ using at most $m_{vbl}(x)^{\varepsilon} l(|x|)$ space on instance x for a certain polynomial p, a certain polylog function l, and a certain constant $\varepsilon \in [0,1)$.

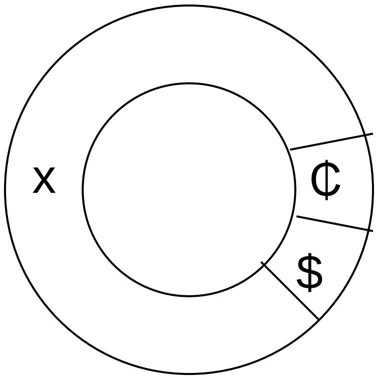
- We can replace $(2SAT_3, m_{vbl})$ by $(3DSTCON, m_{ver})$, where $m_{ver}(\langle G,s,t \rangle)$ = the number of vertices in G.

- Here, we want to give a state complexity characterization of LSH.

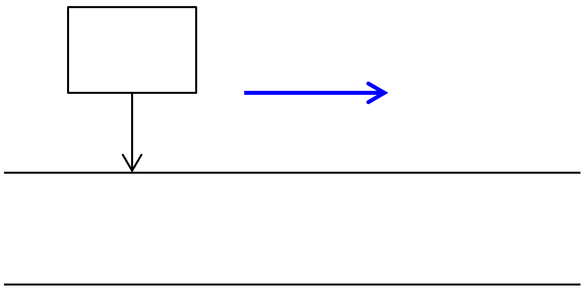
Circular Tapes and Sweeping Moves

- 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 ¢ to $. In this case, the tape head is called sweeping.





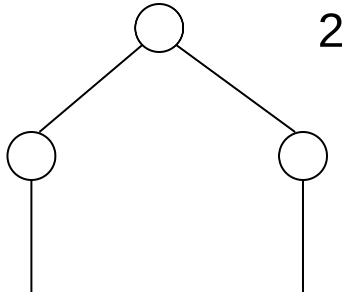
circular tape

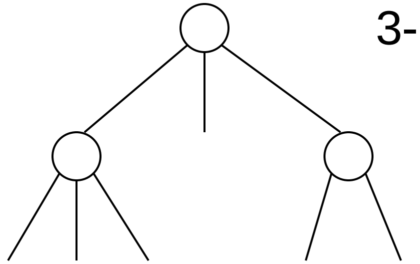


sweeping

Constant Branching

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





Constant-Branching Simple 2nfa's

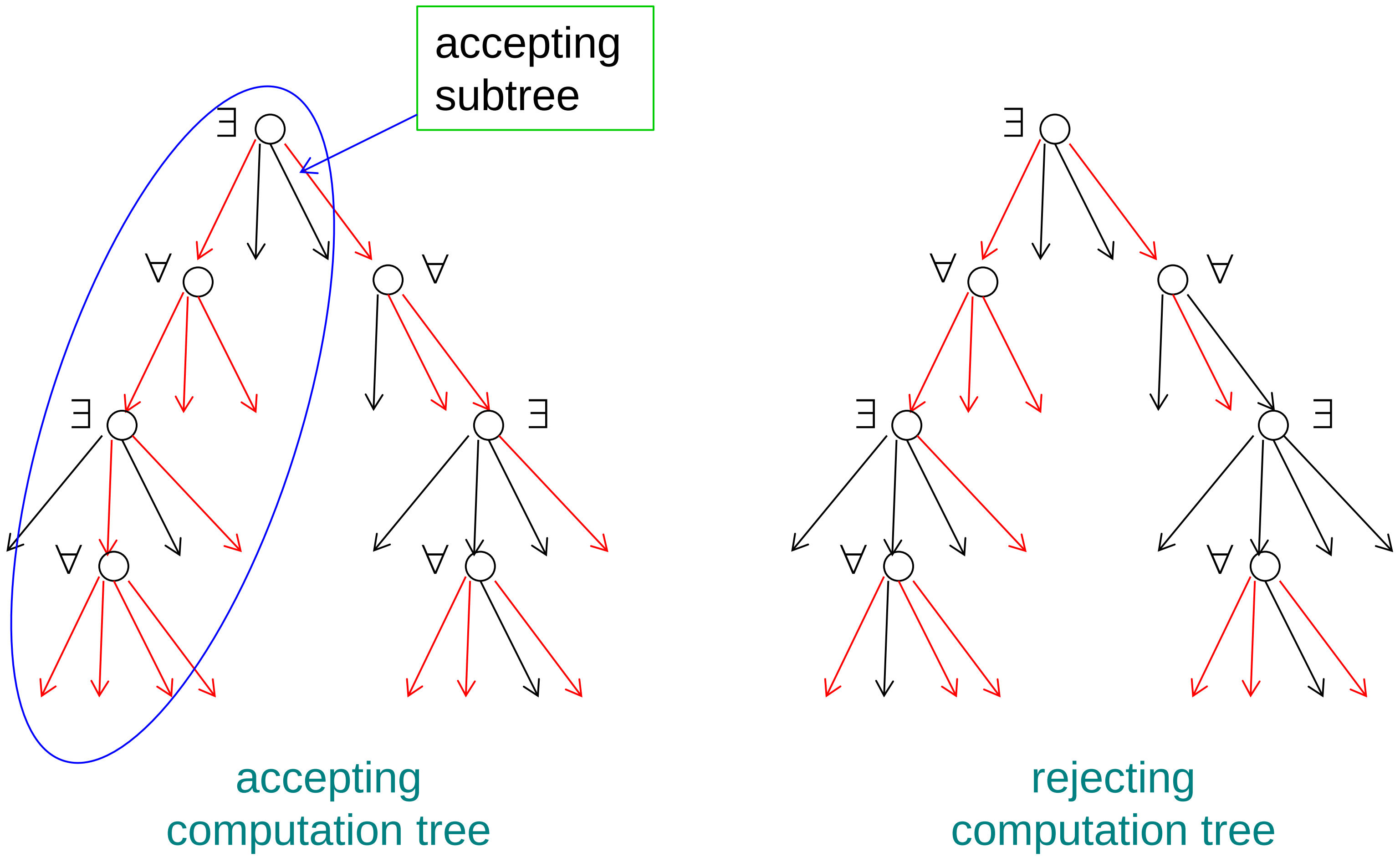
- We place certain restrictions on 2nfa's.
- We consider only 2nfa's whose input tapes are circular.

- 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.

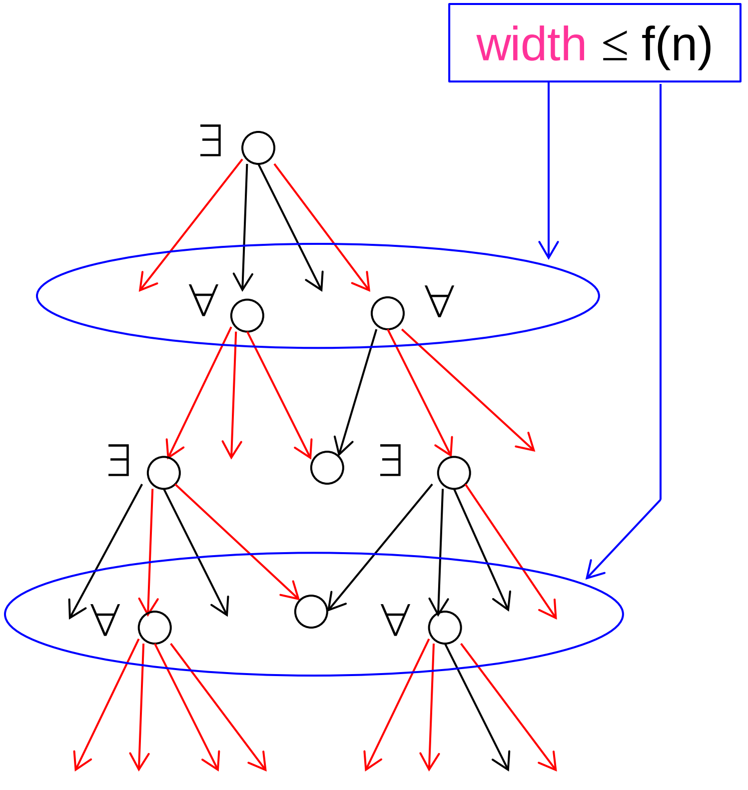
Alternating Finite Automata (revisited)

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



Narrow 2afa's

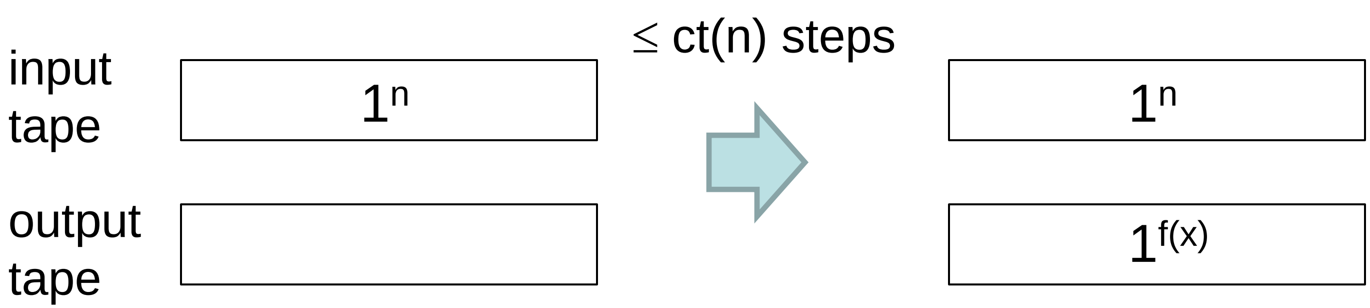
- Instead of using computation trees, we use computation graphs.
- Here, we further consider additional restrictions on 2afa's.
- Let $f:N\rightarrow N$ be a function.
- A family $\{M_n\}_{n\in N}$ of 2afa's is called f(n)-narrow if, for any $n\in N$ and any input x of length n, a $\{\forall,\exists\}$-leveled computation graph of M_n on input x has width at most f(n) at every $\forall$-level.



computation graph

t(n)-Time Space Constructibility

- We need a restricted notion of space constructibility.
- Let t,f: N $\rightarrow$ N be functions.
- A function f is called t(n)-time space constructible $\Leftrightarrow$ 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 $t,\ell : N \to N^+$ be s.t. t is log-space computable and ℓ is $O(t(n))$-time space constructible. Let L and m be a language over alphabet Σ and a log-space size parameter.

  1. $(L,m) \in$ TIME,SPACE$(t(|x|),\ell(m(x)))$, then there are two constants $c_1,c_2>0$ and an L-uniform family $\{M_{n,\ell}\}_{n,\ell\in N}$ of $c_2\ell(m(x))$-narrow 2afa's such that each $M_{n,|x|}$ has at most $c_1t(|x|)\ell(m(x))$ 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 $\{M_{n,\ell}\}_{n,\ell\in N}$ of $c_2\ell(m(x))$-narrow 2afa's such that each $M_{n,|x|}$ has at most $c_1t(|x|)\ell(m(x))$ states and computes $L(x)$ on all inputs satisfying $m(x)=n$, then (L,m) belongs to 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(nlog(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^{ε} for a certain constant ε 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 $f\in FL$ 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 $O(n \cdot \mathrm{polylog}(n))$ states to an L-uniform family of equivalent $O(n^{\varepsilon})$-narrow 2afa's is super-polynomial in n for any $\varepsilon \in [0,1)$.

Non-Uniform Linear Space Hypothesis

- 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 ($2SAT_3$,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)]

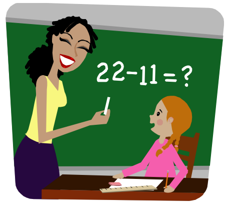
  The following statements are logically equivalent.

  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

- The following is a list of important open problems.

1. Is it possible to reduce the factor $n^{1-c/\sqrt{\log(n)}}$ to n^{ε} for a certain constant ε 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.
- Townsend (1982,1990) reformulated the polynomial-time computability of type-2 functionals.
- 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 ω to ω
- ${}^{k,l}\omega = \omega^k \times ({}^{\omega}\omega)^l$ E.g., ${}^{3,2}\omega = \omega \times \omega \times \omega \times {}^{\omega}\omega \times {}^{\omega}\omega$
- $(m,\alpha) \in {}^{k,l}\omega \Leftrightarrow m \in \omega^k$ and $\alpha \in ({}^{\omega}\omega)^l$
- A partial functional F of rank (k,l) satisfies that

  $\mathrm{Dom}(F) \subseteq {}^{k,l}\omega$ and $\mathrm{Im}(F) \subseteq \omega$.
- A total functional F of rank (k,l) satisfies that

  $\mathrm{Dom}(F) = {}^{k,l}\omega$ and $\mathrm{Im}(F) \subseteq \omega$.

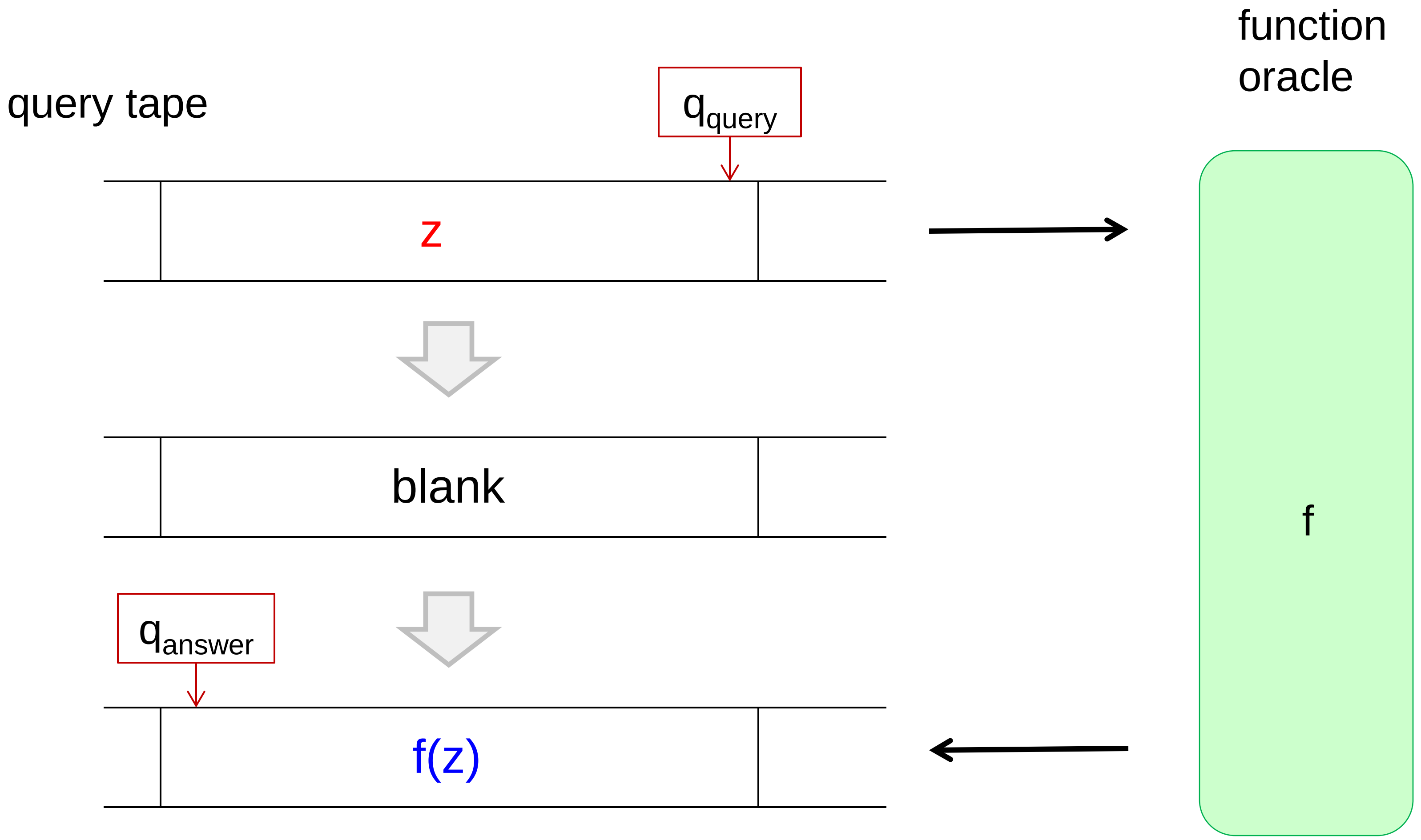
- A relation R of rank (k,l) is a subset of ${}^{k,l}\omega$. (namely, $R \subseteq {}^{k,l}\omega$.)

Function-Oracle Turing Machines

- 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 makes a query to the function oracle f by writing a query word z on the query tape.
  2. M enters a query state q_{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_{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



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.
- Ptf = class of all polynomial-time computable total functionals

- Let A be any language.
- 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 Ptf(C) = $\cup_{A \in C}$ Ptf(A).

The Polynomial Hierarchy of Type 2

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

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

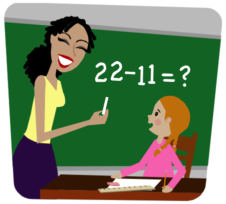
$$\square_0^{0,p} = Ptf$$

$$\Sigma_{k+1}^{0,p} = \{\left(\exists x \le F(m,\alpha)\right) R(x,m,\alpha) \mid R \in \Pi_k^{0,p}, F \in \square_0^{0,p}\}$$

$$\Pi_{k+1}^{0,p} = \{\left(\forall x \le F(m,\alpha)\right) R(x,m,\alpha) \mid R \in \Sigma_k^{0,p}, F \in \square_0^{0,p}\}$$

$$\square_{k+1}^{0,p} = Ptf\left(\Sigma_k^{0,p}\right)$$

$$\Delta_{k+1}^{0,p} = \{R \mid \chi_R \in \square_{k+1}^{0,p}\}$$



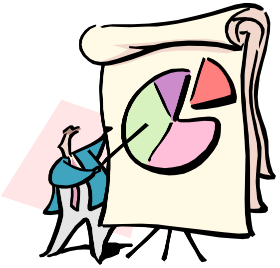
Hierarchy Theorem

- Townsend (1990) proved the following.
- Hierarchy Theorem: for all n≥1,

$$\begin{cases} \Delta_n^{0,p} \neq \Sigma_n^{0,p} \neq \Pi_n^{0,p} \\ \square_n^{0,p} \neq \square_{n+1}^{0,p} \end{cases}$$

- Next, we define $\Delta_{k+1}^{NP} = \left\{ R \mid \chi_R \in Ptf\left(\Sigma_{k-1}^{0,p} \cup \Sigma_k^p\right) \right\}$
- 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≥1,

$$\begin{cases} \Delta_n^p = \Sigma_n^p \Leftrightarrow \Delta_n^{0,p} = \Delta_{n+1}^{NP} \\ \Sigma_n^p = \Pi_n^p \Leftrightarrow \Delta_{n+1}^{NP} \subseteq \Sigma_n^{0,p} \cap \Pi_n^{0,p} \end{cases}$$



Relativization and Type-2 Computation

- Let C be any “typical” type-1 complexity class.
- Let $\overline{\overline{C}}$ 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:

  P, NP, BPP, NP$\cap$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 $\overline{\overline{C}}$ of type-2 relations, let $\overline{\overline{C}}[A] = \{R[A] \mid R \in C\}$

Regular Complexity Classes

- Let C be any "typical" type-1 complexity class and let $\overline{\overline{C}}$ 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, $C^A = \overline{\overline{C}}[A]$

- (Claim)

  P and NP are regular.

  Namely, for any oracle A, it follows that

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

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

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

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

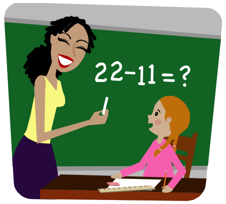
$$BPP^B \neq \overline{\overline{BPP}}\ [B]$$

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,

$$\overline{\overline{C}} \subseteq \overline{\overline{D}} \Leftrightarrow C[G] \subseteq D[G]$$



Power of Generic Oracles

- Recall that there are oracles A and B for which

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

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

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

- Proposition: [Cook-Impagliazzo-Yamakami (1997)]

For any generic oracle G,

$$NP^{G} \cap co\text{-}NP^{G} = \left(\overline{\overline{NP}} \cap co\text{-}\overline{\overline{NP}}\right)[G]$$

$$BPP^{G} = \overline{\overline{BPP}}\ [G]$$

Open Problems

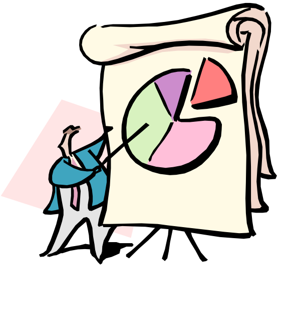
- Develop a theory of computability of higher types.
- Find more complexity classes C such that

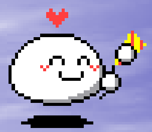
1. there is an oracle A satisfying

$$C^A \neq \overline{\overline{C}}\,[A]$$

2. for all generic oracle G,

$$C^G = \overline{\overline{C}}\,[G]$$





Thank you for listening

Q & A

I’m happy to take your question!

END