Interplay between formal languages and complexity classes

Nutan Limaye

The Institute of Mathematical Sciences

Chennai, India

nutan@imsc.res.in

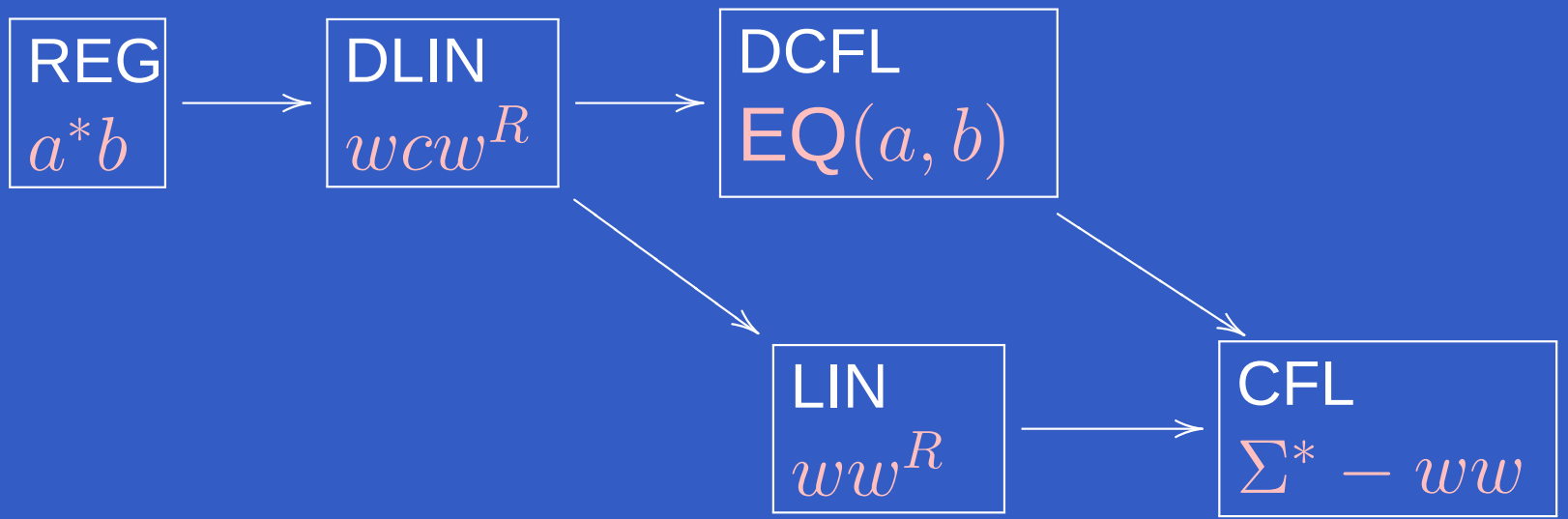
joint work with

Meena Mahajan and Antoine Meyer

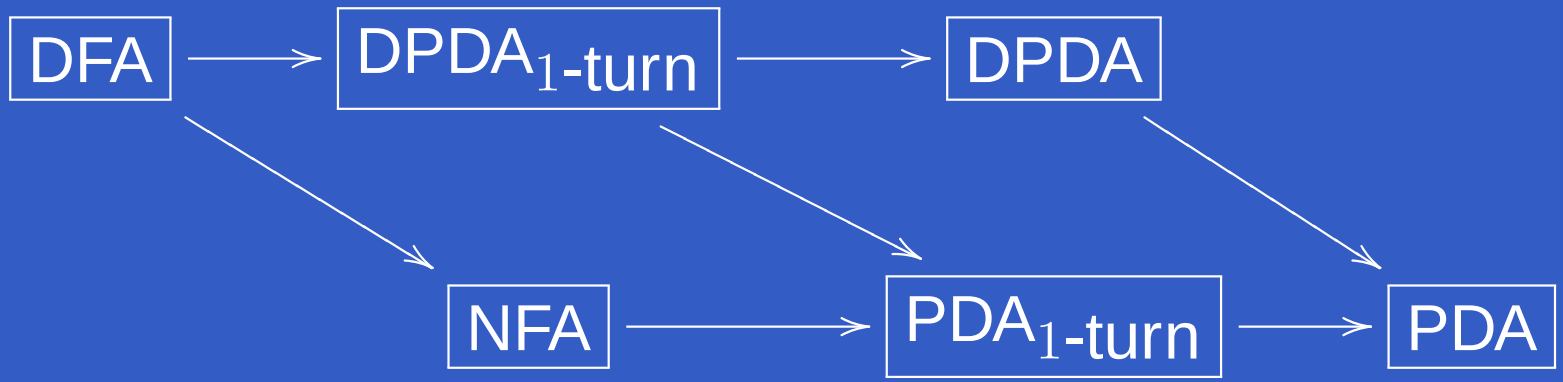
to appear in CSR 2008

Formal Languages and Automata

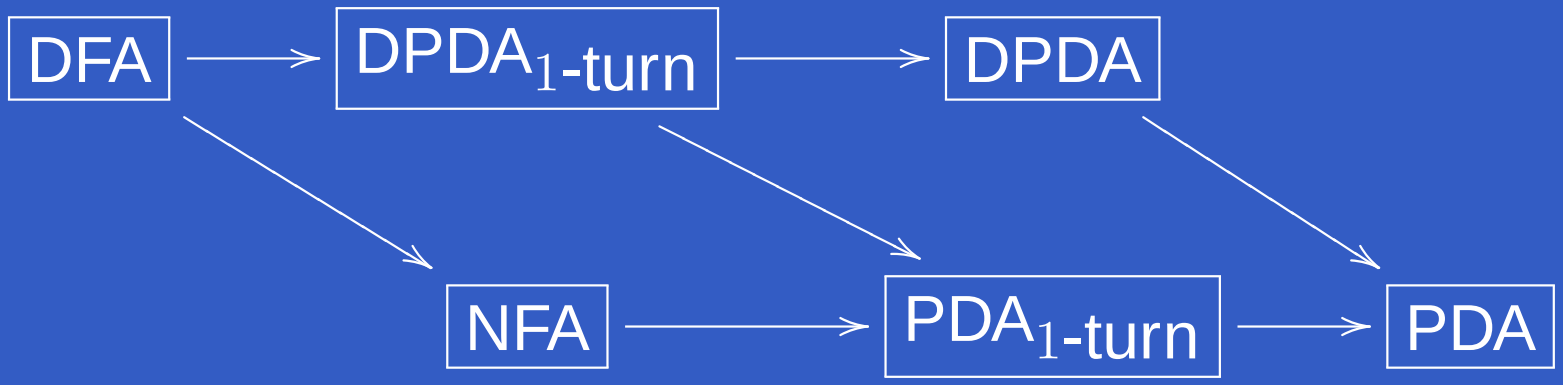
Some formal language classes



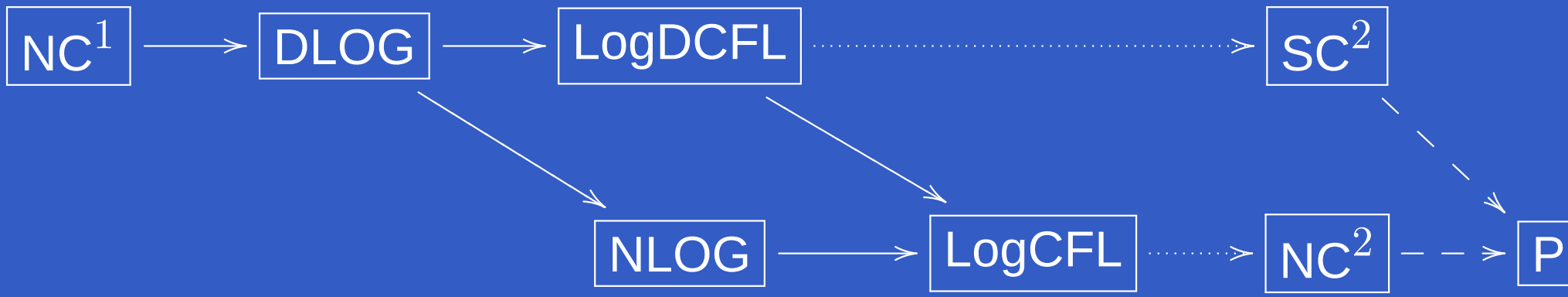
Corresponding automata



Automata and Complexity Classes



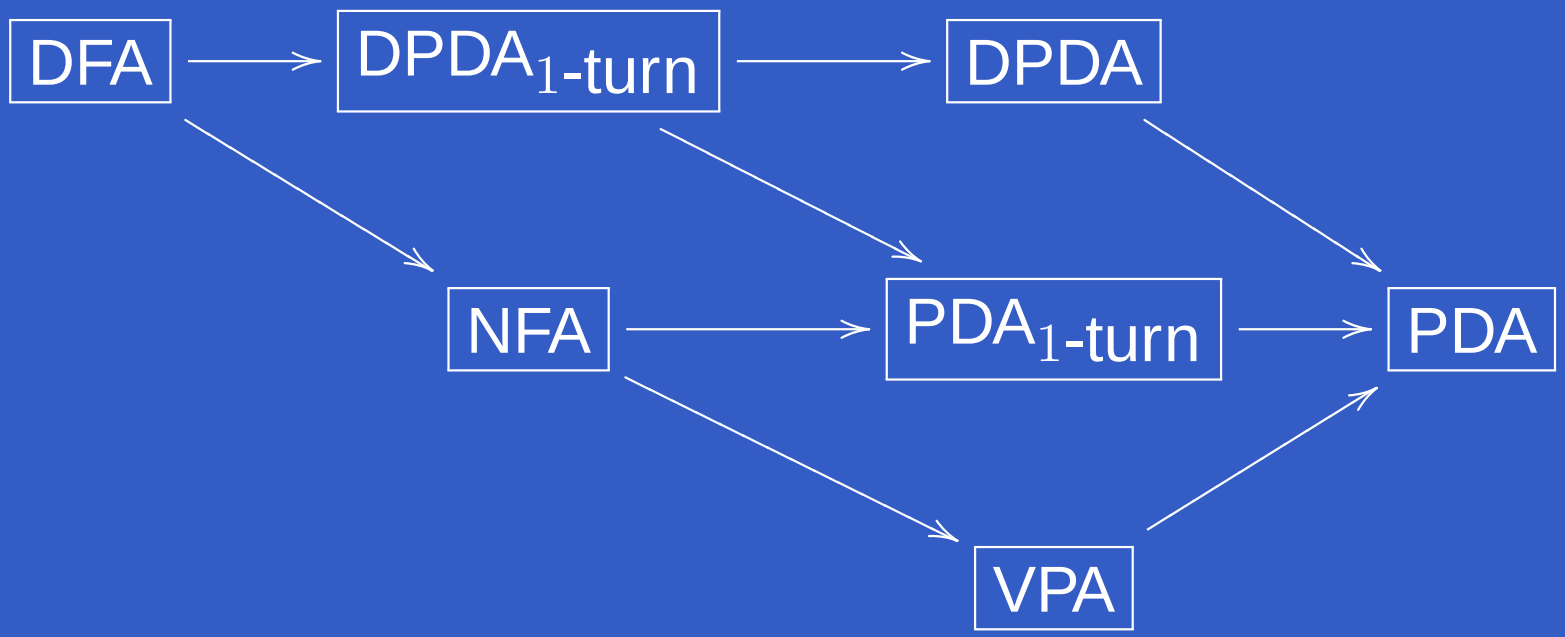
Closures (under very restrictive reductions) yield complexity classes: *[Barrington, Cook, Holzer-Lange, Sudborough]*



A different restriction of PDA

VPA: Visibly Pushdown Automata *[Mehlhorn]*

- non-deterministic
- multiple-turns
- real-time (no ϵ-moves)
- stack movement restricted



What are VPAs?

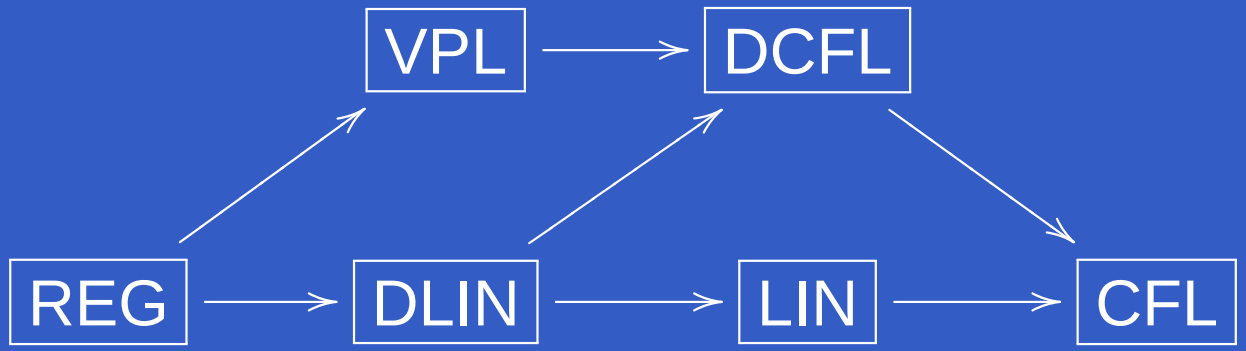
- The input alphabet is partitioned into three sets: Push, Pop, Internal.
- Whether the VPA pushes, or pops, or does neither, depends *only* on the current letter.
- The next state, what symbol is to be pushed, can depend on the current state and stack-top too.

What can VPAs do?

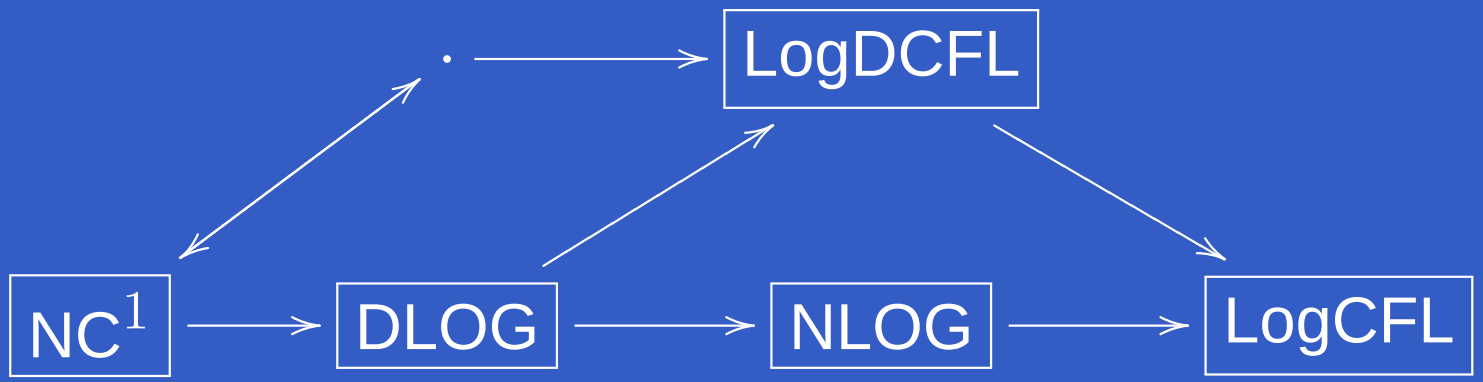
- Can accept $\{a^n b^n \mid n \geq 1\}$; so stronger than NFA.
- Incomparable with $\text{DPDA}_{1\text{-turn}}$:
  Can accept $\{a^n b^n a^m b^m \mid n, m \geq 1\}$,
  Cannot accept $\{a^n b a^n \mid n \geq 1\}$.
- Can be determinized, so within DPDA.
  Cannot accept $\{w \mid |w|_a = |w|_b\}$; so weaker than DPDA.

The class VPL and its closure

VPA can be determinized. *[Alur-Madhusudan]*
So the language classes picture is

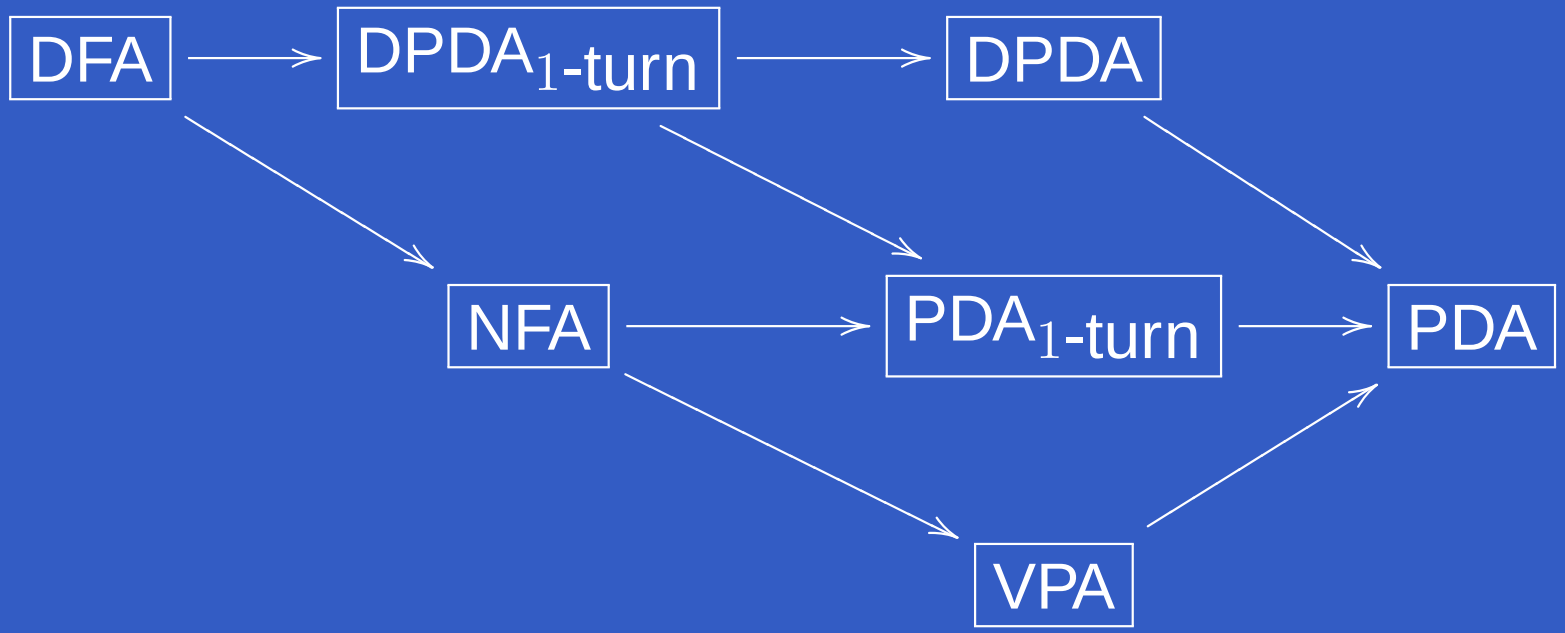


Membership testing for VPLs can be done in NC^1. *[Dymond]*
So the complexity classes picture is

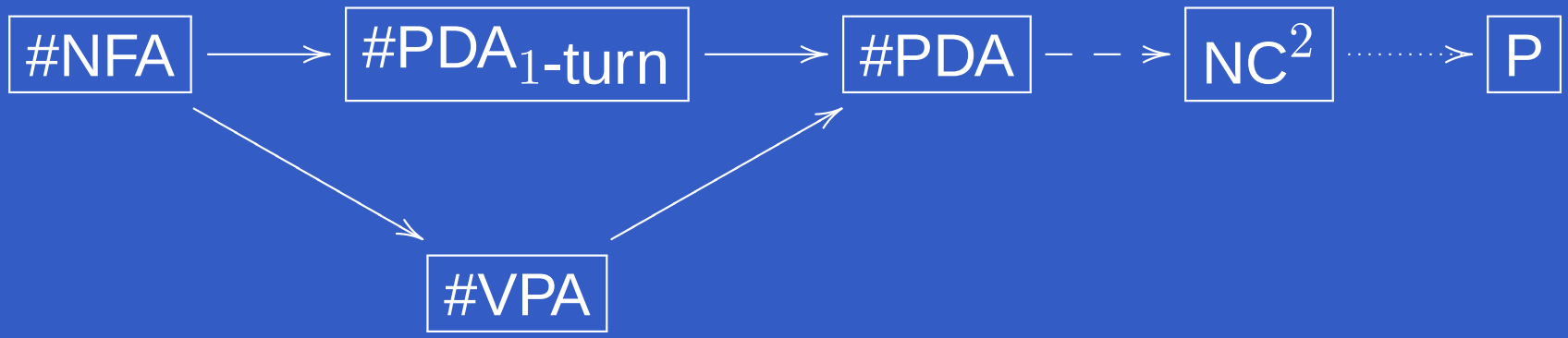


Counting classes

Recall the automata picture:



Discard deterministic classes.
Count paths in nondeterministic automata.



Why count paths?

Counting classes capture the complexity of natural important arithmetic functions.

- #NFA $\equiv$ multiplying n integer matrices of $O(1)$ size.

  [Caussinus-Mckenzie-Therien-Vollmer]

- #PDA$_1$-turn $\equiv$ computing the determinant of an integer matrix. *[Damm, Toda, Vinay, Valiant]*

- #PDA$\equiv$ evaluating families of polynomials $p_n(x_1, x_2, \ldots, x_n)$ with polynomial-size circuit representations, where the degree of p_n is $n^{O(1)}$.

  [Vinay,Rossmanith-Niedermeier]

  (Without the degree bound, this is not known to be in P.)

Where are #VPA functions?

- Equivalence of membership testing does not imply equivalence of counting paths.

  So #VPA could be harder than #NFA.

- #VPA functions can be computed in LogDCFL.

  [Limaye-M-Rao], extending VPA parsing method of *[Braunmuhl-Verbeek]*

  So they are easier than #PDA functions.

- Open Question: Is #VPA reducible to #NFA?

What makes #VPA functions easier than #PDA?

- real-time (no ϵ moves)
- on all runs, the stack-height versus time profile is the same –
  height-determinism
- stack-height after reading a string can be computed easily –
  simple height-advice

What makes #VPA functions easier than #PDA?

- real-time (no ϵ moves)
- on all runs, the stack-height versus time profile is the same –
  height-determinism
- stack-height after reading a string can be computed easily –
  simple height-advice

Can we abstract out these properties and get a larger class of PDA with easy counting functions?

Defining rhPDA:

real-time height-determinstic pushdown automata

[Nowotka-Srba]

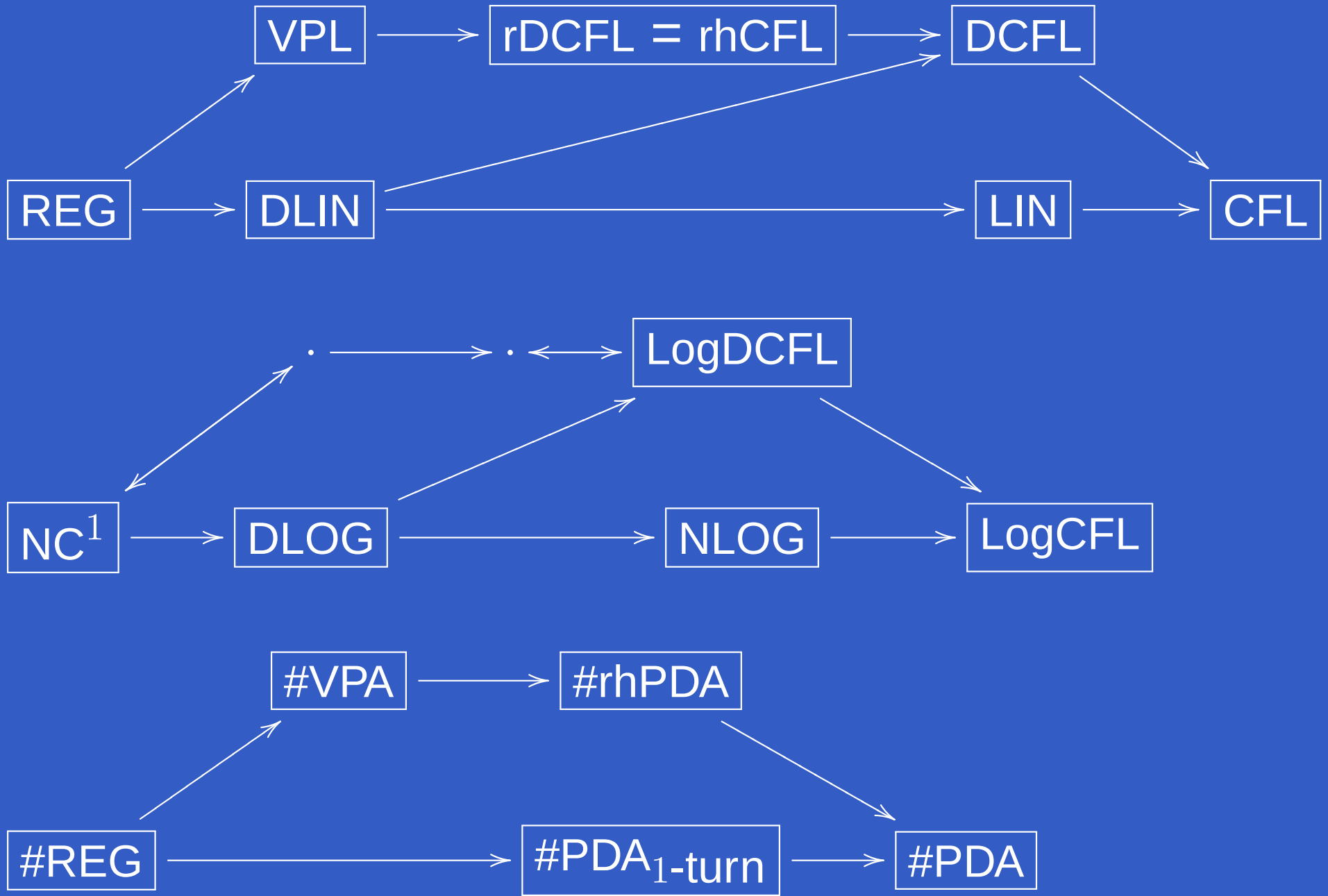
- real-time
- complete (the automaton never gets stuck midway on a run)
- height-deterministic: all runs on a word have the same stack-height-vs-time profile

Clearly, every rDPDA can be “completed” to get an rhPDA.

Interestingly, the converse is also true:

rhPDA can be determinized! *[Nowotka-Srba]*

Revisiting some pictures



The height-advice function

- For VPA, height-advice functions are in TC^0.
- For SSPDA, height-advice functions are in NC^1.
- For rhPDA, height-advice functions are in LogDCFL.

Use to parameterize rhPDA.

Our results: For any rhPDA M,

- Height-advice function g_M in $\mathrm{NC}^1 \Longrightarrow$ membership in NC^1.
- Height-advice function g_M computable by a DFA transducer $\Longrightarrow \#M$ no harder than #VPA.
- Membership in DLOG with oracle access to g_M.

builds on *[Braunmuhl-Verbeek]*, *[Limaye-M-Rao]*

Our Main Result: #rhPDA in LogDCFL

Fixed rhPDA M. Input word w of length n.

- Idea 1: Use Chinese Remaindering.
  - Compute $a_i = \#M(w) \bmod p_i$ for $i = 1, 2, .., O(\log n)$.
    p_i is the ith prime
  - Reconstruct $\#M(w)$ in DLOG with these a_is from oracle.

Our Main Result: #rhPDA in LogDCFL

Fixed rhPDA M. Input word w of length n.

- Idea 1: Use Chinese Remaindering.
  - Compute $a_i = \#M(w) \bmod p_i$ for $i = 1, 2, .., O(\log n)$.
    p_i is the ith prime
  - Reconstruct $\#M(w)$ in DLOG with these a_is from oracle.
- Idea 2: To compute an a_i: Use height-advice (as oracle) to achieve a LogDCFL bound via Divide-and-Conquer.

Our Main Result: #rhPDA in LogDCFL

Fixed rhPDA M. Input word w of length n.

- Idea 1: Use Chinese Remaindering.
  - Compute $a_i = \#M(w) \bmod p_i$ for $i = 1, 2, .., O(\log n)$. p_i is the ith prime
  - Reconstruct $\#M(w)$ in DLOG with these a_is from oracle.
- Idea 2: To compute an a_i: Use height-advice (as oracle) to achieve a LogDCFL bound via Divide-and-Conquer.
- Idea 3: Pull down the height-advice oracle.
  - difficult in general for machines with stack.
  - easy here because the oracle queries are small ($(O \log n)$ bits long).

The Divide-and-Conquer step

(Builds on *[Braunmuhl-Verbeek]*, *[Limaye-M-Rao]*)

- Three-way branching recursion tree, with logarithmic depth.
- Location of each node specified in $O(\log n)$ bits.
- Information at nodes stacked using $O(\log n)$ bits per node.

The Divide-and-Conquer step: cont'd

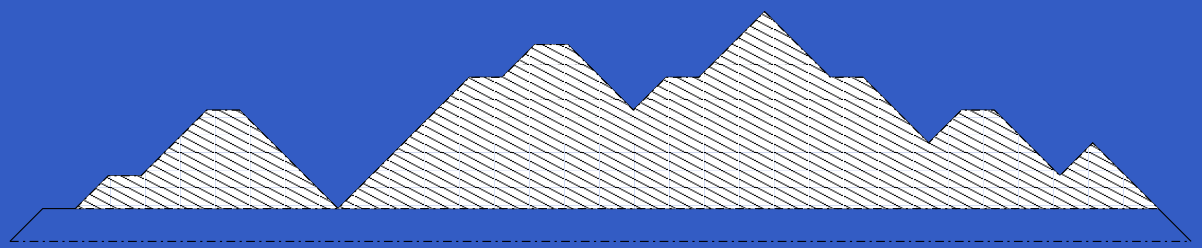
Surface configuration (q, X, i) = (state, stack-top, time)

For height-matched positions i, j, information table $T(i,j)[q, X, q', Y]$: number of paths, modulo the relevant prime p, going from (q, X, i) to (q', Y, j).

We want

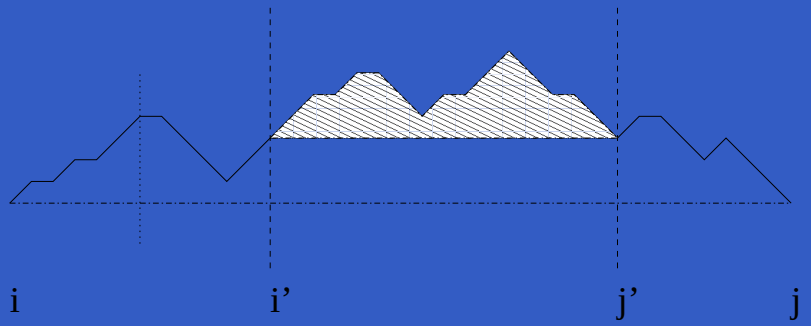
$$\sum_{q' \in F} T(0, n,)[q_0, Z_0, q', Z_0]$$

Easy if interval i, j is small.
Also easy if table of large subinterval known.

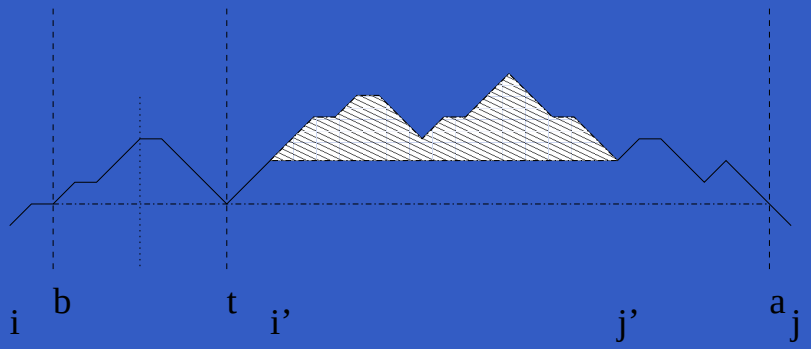


Finding $T(i,j)$ recursively

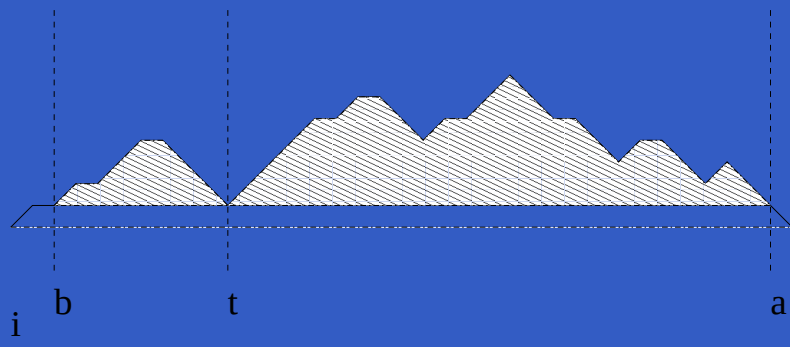
To find $T(i,j)|T'(i',j')$,



get t, b, a; find $T"(b,a) = T_1(b,t) \times T_2(t,a)|T'(i',j')$,



hence get
$T(i,j)|T'(i',j')$
$= T(i,j)|T"(b,a)$



Multi-pushdown models

- A PDA with two stacks can recognise r.e. languages.

Multi-pushdown models

- A PDA with two stacks can recognise r.e. languages.
- Restricted models are useful for many model checking,parsing applications.

Multi-pushdown models

- A PDA with two stacks can recognise r.e. languages.
- Restricted models are useful for many model checking,parsing applications.
- What kind of restrictions?

Multi-pushdown models

- A PDA with two stacks can recognise r.e. languages.
- Restricted models are useful for many model checking,parsing applications.
- What kind of restrictions?
  - Order as a restriction: Push on any stack; pop only from **first** non-empty.

Multi-pushdown models

- A PDA with two stacks can recognise r.e. languages.
- Restricted models are useful for many model checking,parsing applications.
- What kind of restrictions?
  - Order as a restriction: Push on any stack; pop only from **first** non-empty.
  - Visibility as a restriction: Input alphabet decides the complete stack-profile for all the stacks.

Ordered n-pushdown machines: PD_n

- Contain all of CFLs. Proper subclass of context sensitive languages. *[La Torre-Madhusudan-Parlato]*

Ordered n-pushdown machines: PD_n

- Contain all of CFLs. Proper subclass of context sensitive languages. *[La Torre-Madhusudan-Parlato]*
- Membership at least as hard as that for CFLs; LogCFL.

Ordered n-pushdown machines: PD_n

- Contain all of CFLs. Proper subclass of context sensitive languages. *[La Torre-Madhusudan-Parlato]*
- Membership at least as hard as that for CFLs; LogCFL.
- A generalisation of *[Cocke-Kasami-Younger]* algorithm gives a P-time upper bound. *[Cherubini-Pietro]*

Ordered n-pushdown machines: PD_n

- Contain all of CFLs. Proper subclass of context sensitive languages. *[La Torre-Madhusudan-Parlato]*
- Membership at least as hard as that for CFLs; LogCFL.
- A generalisation of *[Cocke-Kasami-Younger]* algorithm gives a P-time upper bound. *[Cherubini-Pietro]*
- The CKY algorithm can be implemented in LogCFL.

Ordered n-pushdown machines: PD_n

- Contain all of CFLs. Proper subclass of context sensitive languages. *[La Torre-Madhusudan-Parlato]*
- Membership at least as hard as that for CFLs; LogCFL.
- A generalisation of *[Cocke-Kasami-Younger]* algorithm gives a P-time upper bound. *[Cherubini-Pietro]*
- The CKY algorithm can be implemented in LogCFL.
- Our Result: The *[Cherubini-Pietro]* algorithm can be implemented in LogCFL.

Visible multi-pushdown machines: MVPA

- Contain all of VPLs. Proper subclass of context sensitive languages. *[Carotenuto-Murano-Peron]*

Visible multi-pushdown machines: MVPA

- Contain all of VPLs. Proper subclass of context sensitive languages. *[Carotenuto-Murano-Peron]*
- Accept $\{a^n b^n c^n \mid n \geq 0\}$ in two *phases*.

Visible multi-pushdown machines: MVPA

- Contain all of VPLs. Proper subclass of context sensitive languages. *[Carotenuto-Murano-Peron]*
- Accept $\{a^n b^n c^n \mid n \geq 0\}$ in two *phases*.

$aabbcc$	$B\ \bot$	$C\ \bot$	
$aabbcc$	$BB\ \bot$	$CC\ \bot$	
$aabbcc$	$B\ \bot$	$CC\ \bot$	phase 1
$aabbcc$	$\bot$	$CC\ \bot$	phase 1
$aabbcc$	$\bot$	$C\ \bot$	phase 2
$aabbcc$	$\bot$	$\bot$	phase 2

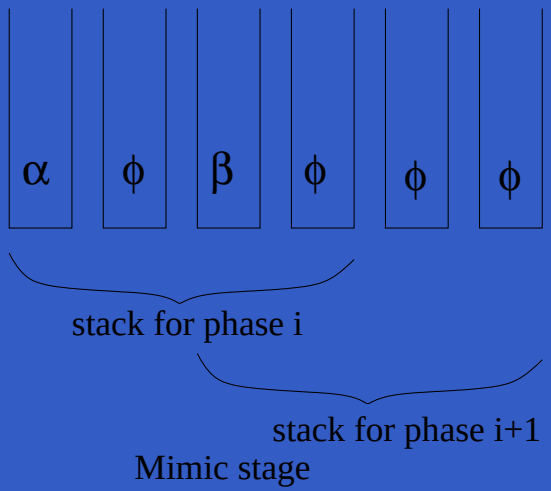
Conversion

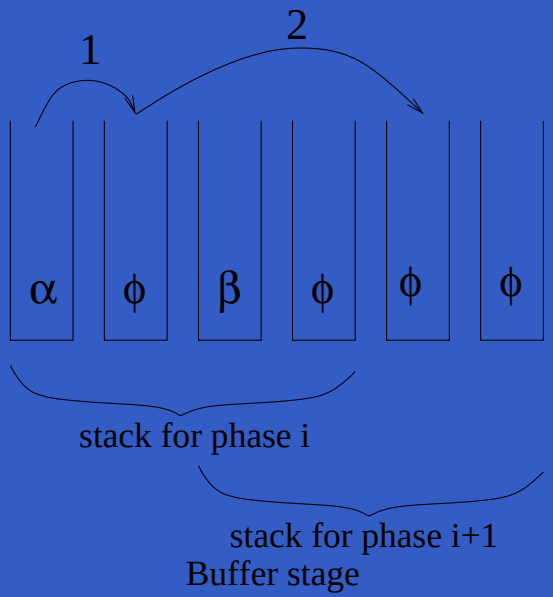
Fix a MVPA M and a number k.
Given a input string w, a string w' and a PD_l M' can be obtained in NC^1 such that

$w \in L(M)$ iff $w' \in L(M')$ and l linear in k.

Conversion

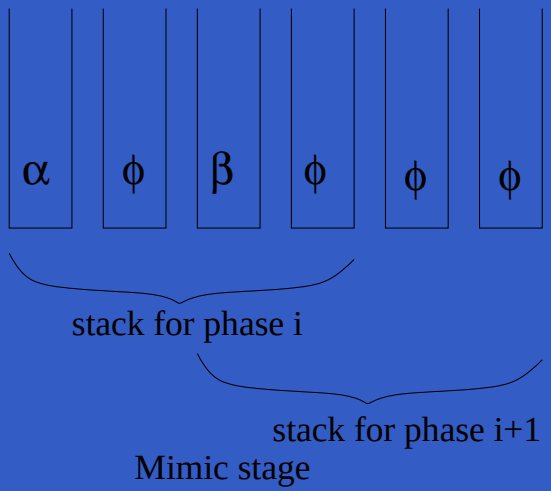
Fix a MVPA M and a number k.
Given a input string w, a string w' and a PD_l M' can be obtained in NC^1 such that
$w \in L(M)$ iff $w' \in L(M')$ and l linear in k.

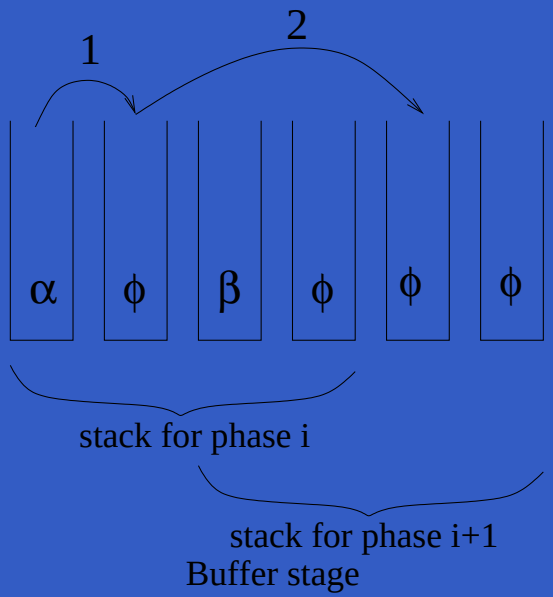




Conversion

Fix a MVPA M and a number k.
Given a input string w, a string w' and a PD_l M' can be obtained in NC^1 such that
$w \in L(M)$ iff $w' \in L(M')$ and l linear in k.





Our Result: Membership problem for MVPA for fixed k is also in LogCFL.

Open Problems

- Characterize the complexity of #VPA.
- Obtain and study and arithmetization of DLOG by finding a non-deterministic model capturing DLOG.
- Find a better (?) arithmetization of LogDCFL.
- Arithmetize multi-pushdown machines and analyse their complexity.

Thank you!