A Perfect Model for Bounded Verification

Javier Esparza

Technical University of Munich

Joint Work with

Pierre Ganty and Rupak Majumdar

Automata-Theoretic Approach to Model Checking

System: abstract machine

Execution: sequence of states/actions = word

Behavior: set of executions= language

Property: set of "good" potential executions = language

set of "bad" potential executions = language

Model Checking: all executions are good = language inclusion

no executions are bad = language disjointness

Verifier's Paradise

- Languages of systems and properties belong to a class of languages that
  - is closed under Boolean operations, and
  - has a decidable emptiness problem.
- In this case:
  - Inclusion and disjointness decidable
  - Properties closed under Boolean operations
  - Characteristic properties
  - Most liberal implementations

Perfect Language Classes

A language class is *perfect* if

- closed under Boolean operations and
- emptiness is decidable.

Goal: find perfect language classes that can model interesting systems and properties.

Hardware : Regular languages

Sequential software: Visibly pushdown languages

Concurrent Software

System: Multithreaded program with sequential threads, finite data, rendezvous synchronization

- Executions of *i*-th thread in isolation: CFL $L{\downarrow}i$
- Executions of the system: $L{\downarrow}1 \cap \ldots \cap L{\downarrow}k$
- Property: regular language

Language-theoretic problem equivalent to ``no bad behaviours´´:

Disjointness: Given CFLs $L{\downarrow}1$, ..., $L{\downarrow}k$,

is $L{\downarrow}1 \cap \ldots \cap L{\downarrow}k \neq \emptyset$?

Concurrent Software

System: Multithreaded program with sequential threads, finite data, rendezvous synchronization

- Executions of *i*-th thread in isolation: CFL $L{\downarrow}i$
- Executions of the system: $L{\downarrow}1 \cap \ldots \cap L{\downarrow}k$
- Property: regular language

Language-theoretic [illegible] equivalent to ``no bad behaviours´´:

Undecidable, even for k=2

Disjointness: Given CFLs $L{\downarrow}1$, ..., [illegible]

is $L{\downarrow}1 \cap \ldots \cap L{\downarrow}k \neq \emptyset$?

Concurrent Software

- Disjointness for CFLs undecidable

  $\Rightarrow$ no perfect class contains the CFLs

- Perfect classes orthogonal to CFLs:
  - Reversal bounded counter machines [Ibarra et al]
  - Context-bounded multi-stack VPL [Parlato *et al*]
  - Not so natural system models

Verifier's Purgatory: Underapproximation

- Example 1: Bounded Model-Checking
  - Unroll loops a fixed number of times

    → explore only executions up to a bounded depth
- Example 2: Context-bounded reachability
  - Explore only executions up to a bounded number of context-switches
- Example 3: Bounded languages
  - More on this later

Verifier's Purgatory: Underapproximation

- Example 1: **Bounded** Model-Checking
  - Unroll loops a fixed number of times

    → explore only executions up to a bounded depth
- Example 2: Context-**bounded** reachability
  - Explore only executions up to a bounded number of context-switches
- Example 3: **Bounded** languages
  - More on this later

Bounded Verification

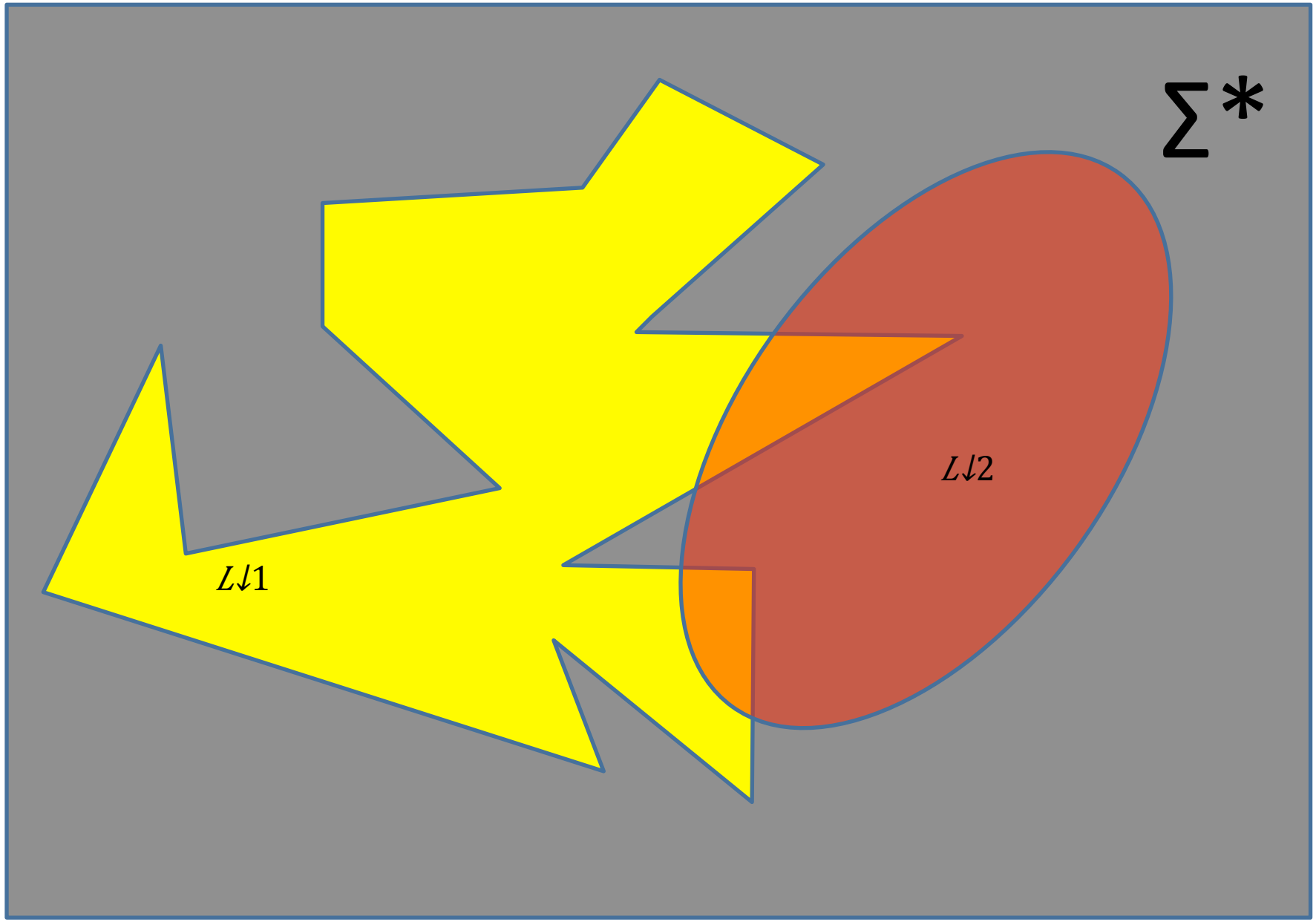
Bounded verification problem:

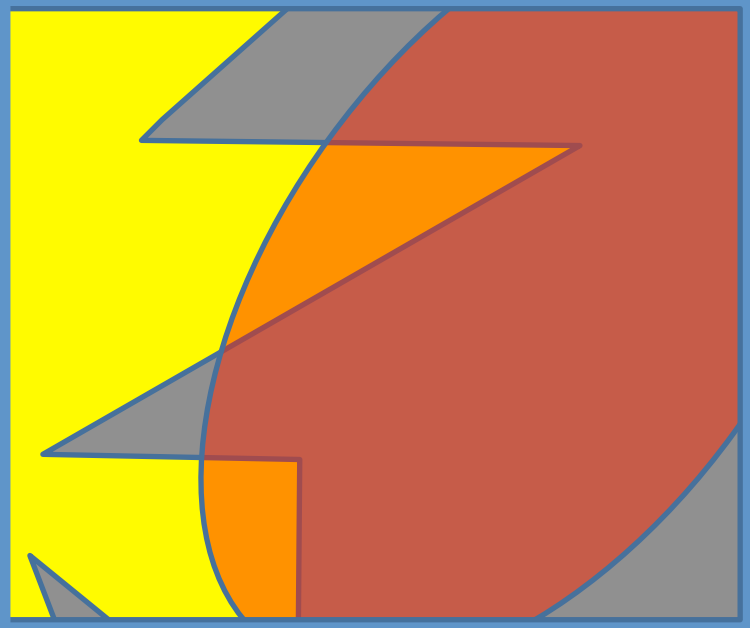
- Given $L{\downarrow}1$, …, $L{\downarrow}k$, fixed language P (*pattern*),
- Check if $L{\downarrow}1 \cap \ldots \cap L{\downarrow}k \cap P \neq \emptyset$

- If Yes: Found a bug
- If No: All behaviors satisfying pattern P are good

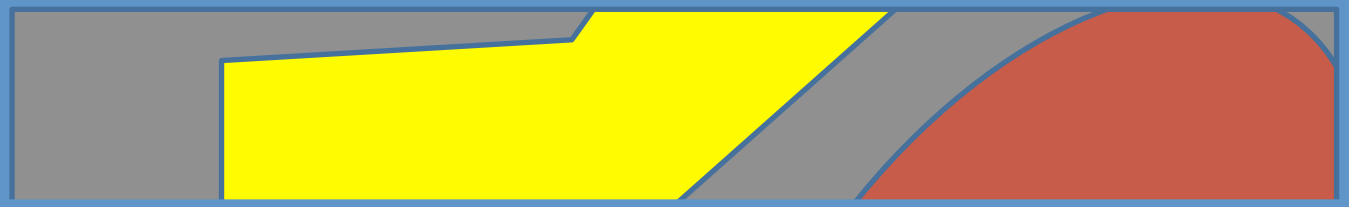
Intuitively: analysis ``within the window P''


Σ^*


Σ*
L↓2
L↓1



P



P'

Bounded Verifier's Paradise

A language class $\boldsymbol{C}$ is perfect modulo a pattern P if:

- closed under Boolean operations modulo P

  For every $L, L\!\uparrow' \in \boldsymbol{C}$:

  – $(L \cap L\!\uparrow') \bmod P = (L \cap P) \cap (L\!\uparrow' \cap P) \in \boldsymbol{C}$

  – $(\Sigma\!\uparrow\!* \setminus L) \bmod P = \overline{L} \cap P \in \boldsymbol{C}$

- emptiness modulo P is decidable

  $L \bmod P = \emptyset$ is decidable

$\boldsymbol{C}$ is perfect modulo a class $\boldsymbol{P}$ of patterns if it is perfect modulo every $P \in \boldsymbol{P}$

Perfect Classes Modulo Patterns

Classic Goal: find language class $\mathcal{C}$ such that

- $\mathcal{C}$ perfect
- $\mathcal{C}$ models interesting systems

Perfect Classes Modulo Patterns

Classic Goal: find language class $\mathcal{C}$ such that

- $\mathcal{C}$ perfect
- $\mathcal{C}$ models interesting systems.

New Goal: find pair of classes $\mathcal{C}$, $\mathcal{P}$ such that

- $\mathcal{C}$ perfect modulo $\mathcal{P}$
- $\mathcal{P}$ models a useful class of behaviours
- $\mathcal{C}$ models interesting systems.

$\boldsymbol{P}=\boldsymbol{F}=$ finite languages

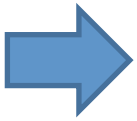
$\boldsymbol{C}$ perfect modulo $\boldsymbol{P}$ if:

- closed under Boolean operations modulo $\boldsymbol{P}$
- emptiness modulo $\boldsymbol{P}$ decidable

$\boldsymbol{P}=\boldsymbol{F}=$ finite languages

$\boldsymbol{C}$ perfect modulo $\boldsymbol{P}$ if:

- closed under Boolean operations modulo $\boldsymbol{P}$
- emptiness modulo $\boldsymbol{P}$ decidable



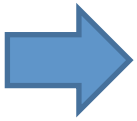
$\boldsymbol{C}$ perfect modulo $\boldsymbol{F}$ if:

- closed under Boolean operations modulo $\boldsymbol{F}$
- word problem for $\boldsymbol{C}$ decidable

$\boldsymbol{P}=\boldsymbol{F}=$ finite languages

$\boldsymbol{C}$ perfect modulo $\boldsymbol{P}$ if:

- closed under Boolean operations modulo $\boldsymbol{P}$
- emptiness modulo $\boldsymbol{P}$ decidable



$\boldsymbol{C}$ perfect modulo $\boldsymbol{F}$ if:

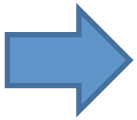
- closed under Boolean operations modulo $\boldsymbol{F}$
- word problem for $\boldsymbol{C}$ decidable

The recursive languages are perfect modulo $\boldsymbol{F}$

$\boldsymbol{P}=\boldsymbol{F}=$ finite languages

$\boldsymbol{C}$ perfect modulo $\boldsymbol{P}$ if:

- closed under Boolean operations modulo $\boldsymbol{P}$
- emptiness modulo $\boldsymbol{P}$ decidable



$\boldsymbol{C}$ perfect modulo $\boldsymbol{F}$ if:

- closed under Boolean operations modulo $\boldsymbol{F}$
- word problem for $\boldsymbol{C}$ decidable

The recursive languages are perfect modulo $\boldsymbol{F}$

*Can we go beyond **F** ?*

Candidate for $\boldsymbol{P}$: Bounded Languages

A bounded language is a language of the form

$$w1\downarrow 1\uparrow * \;\; w2\downarrow 2\uparrow * \;\; \ldots wn\downarrow n\uparrow *$$

for some strings $w1, \ldots, wn$

Example: $(ab)\uparrow * \; c\uparrow * \; (bca)\uparrow *$

Studied in the 1960s by Ginsburg and Spanier

Why are BLs Good Patterns?

1. Context-bounded reachability [QadeerRehof] is a special case

2. Most programs are "almost bounded" in their synchronization usage [Kahlon]
   - Producer consumer
   - Program phases

Are CFLs perfect modulo BL?

Emptiness decidable

Disjointness decidable:

Theorem [GinsburgSpanier '64, E.Ganty11]

For CFLs $L{\downarrow}1$,$L{\downarrow}2$ and bounded language B:

$L{\downarrow}1 \cap L{\downarrow}2 \cap B \neq \emptyset$ is NP-complete

Parikh Images

Parikh image of a word counts the number of occurrences of each letter in the word

Let $\Sigma = \{ a, b, c, d \}$

$$Parikh(abbdcdabd) = (2, 3, 1, 3)$$

$Parikh(L) = \{ Parikh(w) \mid w \in L \}$

Parikh Images

1. [VermaSeidlSchwentick05] For every CFG G, there is a formula $\varphi{\downarrow}G$ of existential Presburger arithmetic (EPA) of linear size representing $Parikh(L(G))$

2. [VonzurGathenSieveking78] Satisfiability of EPA formulas is NP-complete

Example: $L=\{a{\uparrow}n\ b{\uparrow}3n \mid n\ even\}$

Formula for $Parikh(L)$: $\#a=3\cdot\#b \wedge \exists x\ \#a=2\cdot x$

CFL Disjointness mod BL

To show: deciding $L{\downarrow}1 \cap L{\downarrow}2 \cap B \neq \emptyset$ is in NP

Special Case: $B = a{\downarrow}1{\uparrow}* \; a{\downarrow}2{\uparrow}* \; \ldots a{\downarrow}n{\uparrow}*$

$a{\downarrow}1{\uparrow}k{\downarrow}1 \;\; a{\downarrow}2{\uparrow}k{\downarrow}2 \;\; \ldots a{\downarrow}n{\uparrow}k{\downarrow}n \;\; \in L{\downarrow}1 \cap L{\downarrow}2 \cap B$

iff $(k{\downarrow}1, \ldots, k{\downarrow}n) \in Parikh(L{\downarrow}1 \cap B)$ and
$(k{\downarrow}1, \ldots, k{\downarrow}n) \in Parikh(L{\downarrow}2 \cap B)$

iff $(k{\downarrow}1, \ldots, k{\downarrow}n)$ is model of $\varphi{\downarrow}1 \wedge \varphi{\downarrow}2$

But: CFLs not perfect modulo BL

- CFL not closed under intersection mod BL
  - $B = a\uparrow* \; b\uparrow* \; c\uparrow*$
  - $L\downarrow 1 = a\uparrow n \; b\uparrow n \; c\uparrow* \;\; , \; L\downarrow 2 = a\uparrow* \; b\uparrow n \; c\uparrow n$
  - $(L\downarrow 1 \cap L\downarrow 2 \;) \; mod \; B = a\uparrow n \; b\uparrow n \; c\uparrow n$
  - No CFL L satisfies $L \cap B = a\uparrow n \; b\uparrow n \; c\uparrow n$

Main Result

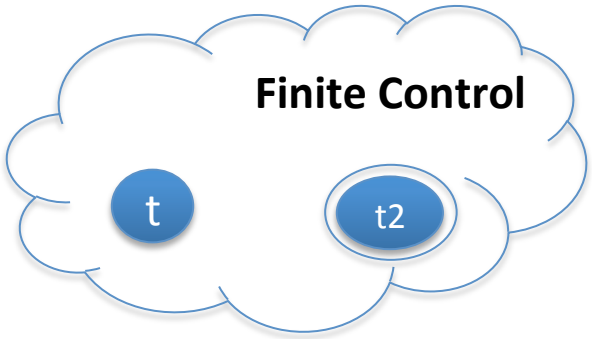
[E.GantyMajumdar LICS 12]

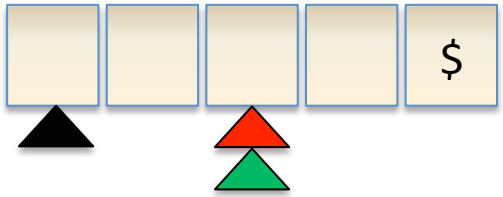
The class of languages recognized by

Multi-head pushdown automata

1. contains all CFLs
2. is perfect modulo BLs
3. models standard concurrency models
   - (recursive) multithreaded programs
   - (recursive) counter machines
   - (recursive) communicating finite state machines

Multi-head PDAs





tape containing input with end marker $

Stack

Transition: q, top_of_stack, letters read by heads → q', stack operation, i-th head moves right

Run, **acceptance, language** defined as usual (require all heads fall off right)

Two Heads are Better than One

- Can use two heads to accept the intersection of two CFLs
- Can use two heads to check accepting computations of a TM:
  - Move two heads across consecutive configurations, checking that the second follows from the first.
  - It follows: Emptiness of 2HNFA is undecidable

Main Result

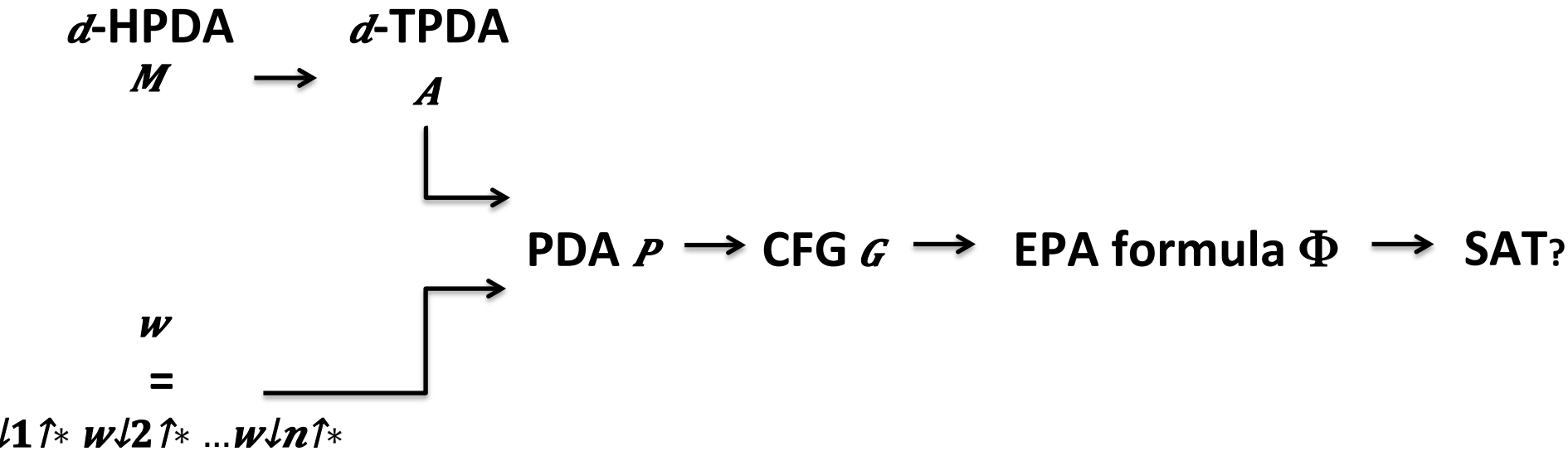
Given d-HPDA M, BL B,

Checking $L(M) \cap B = \emptyset$ is coNEXP-complete

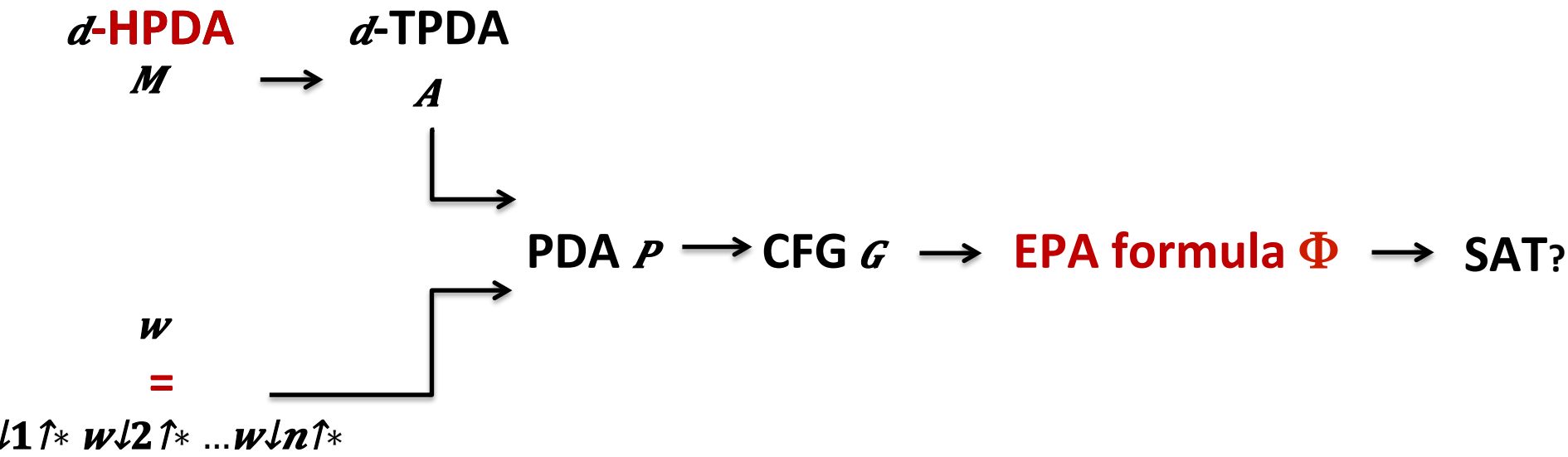
(The exponential dependency is only on d)

- Decidability was known from [Ibarra74]

Proof Outline



Proof Outline



Input: M, w

M accepts $w{\downarrow}1{\uparrow}k{\downarrow}1\ \ w{\downarrow}2{\uparrow}k{\downarrow}2\ \ \ldots w{\downarrow}n{\uparrow}k{\downarrow}n$

iff

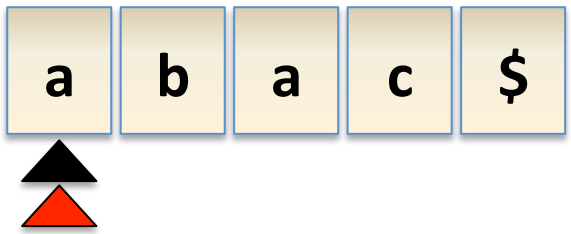
$(k{\downarrow}1\ ,\ \ldots,\ k{\downarrow}n\ )$ model of Φ

Output: formula Φ of EPA

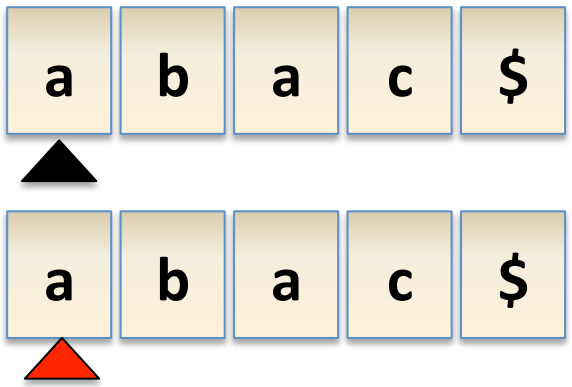
Proof Outline

Specification:

M accepts



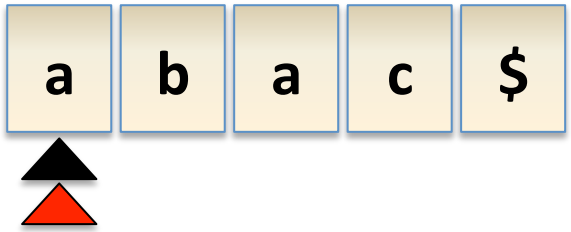
A accepts

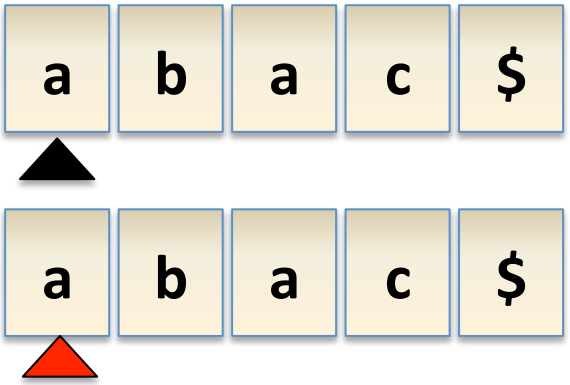


Specification:

M accepts

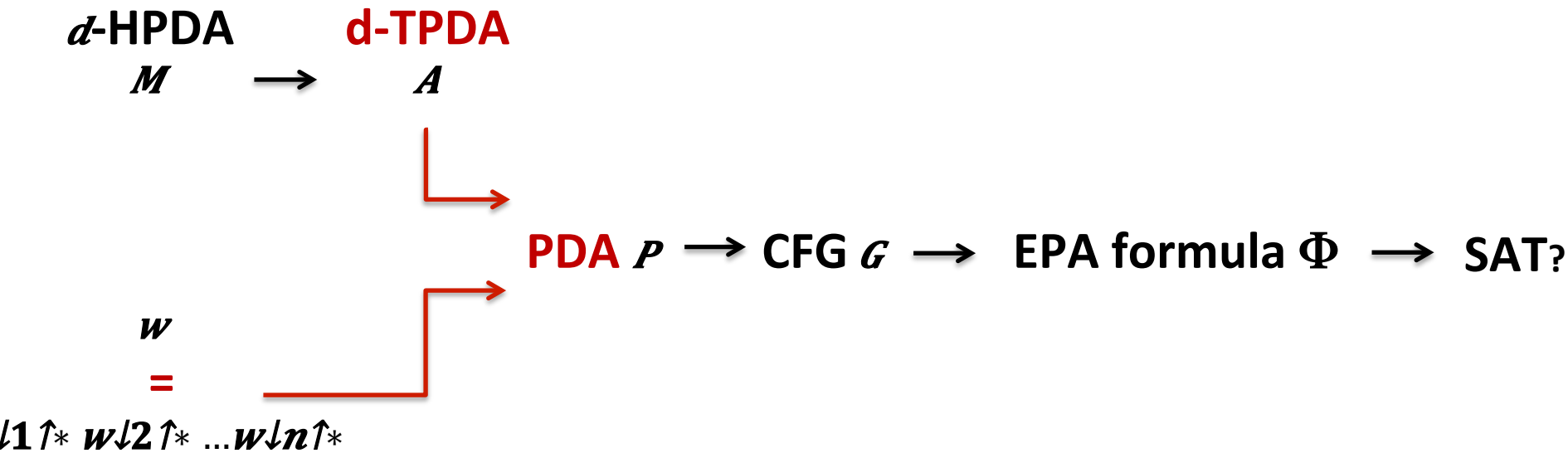
A accepts



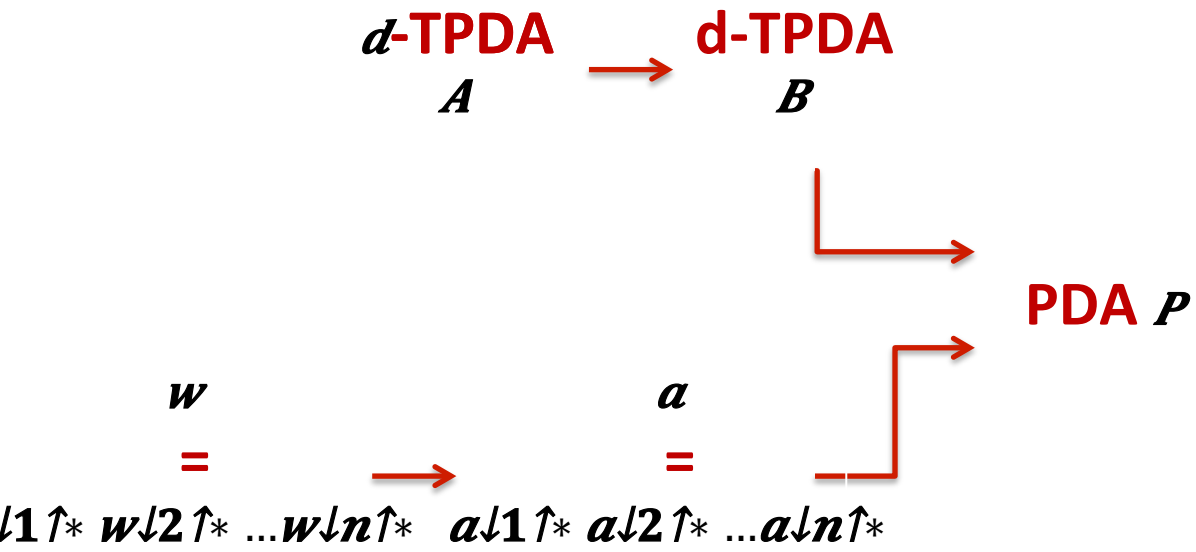


Linear blowup

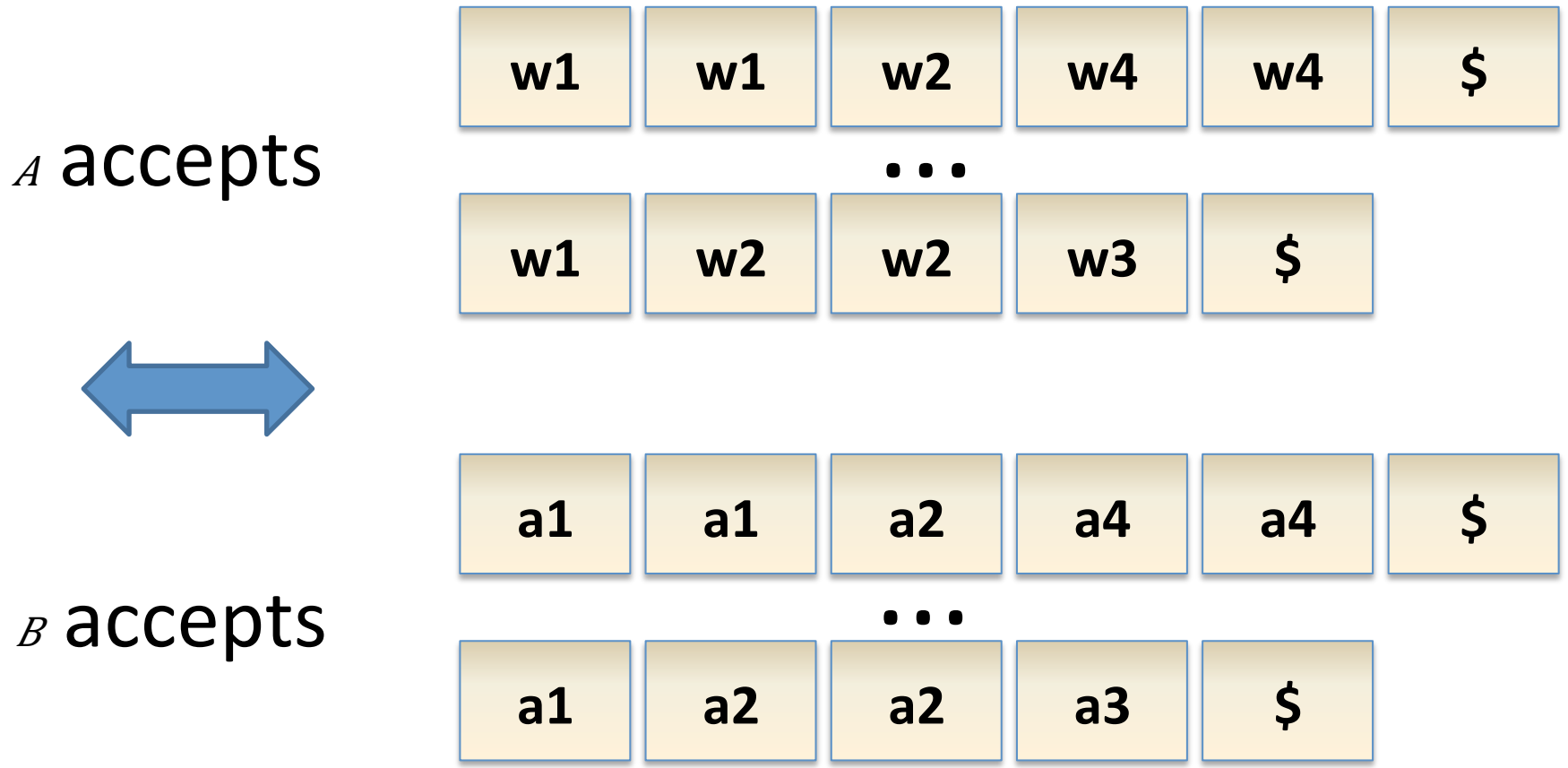
Proof Outline



Proof Outline

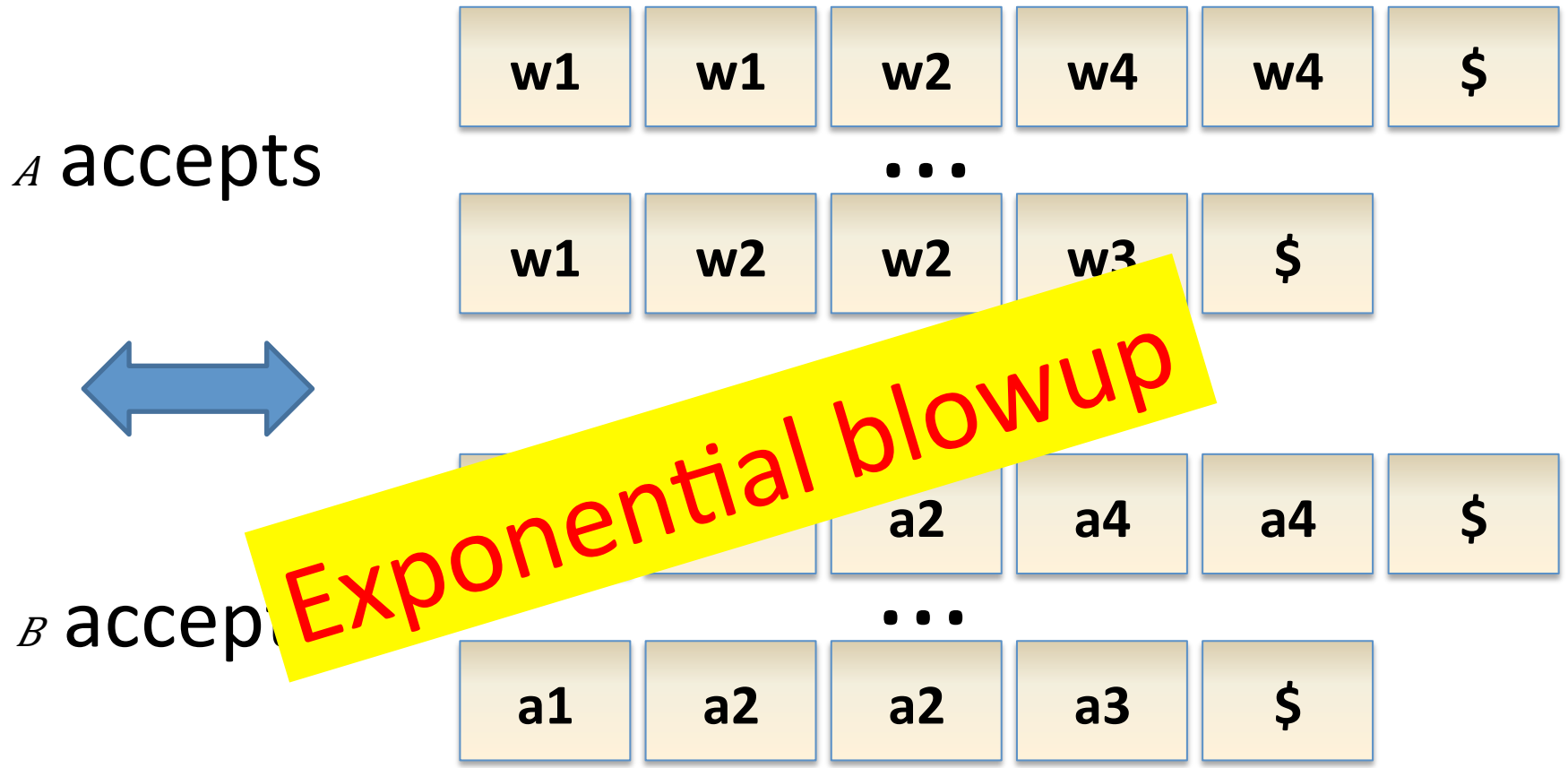


- Specification from A to B



where a1, …, an new alphabet

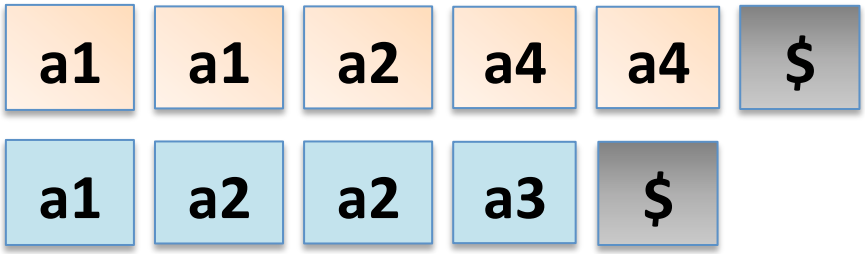
- Specification from A to B



where a1, …, an new alphabet

- Specification from B to P

B accepts



iff P accepts at least one word of the shuffle

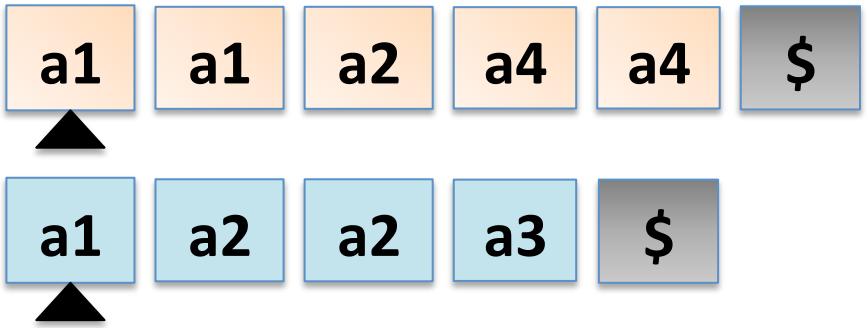
[a1,1][a1,1][a2,1][a4,1][a4,1]

|||

[a1,2][a2,2][a2,2][a3,2]

- Simulation B to P

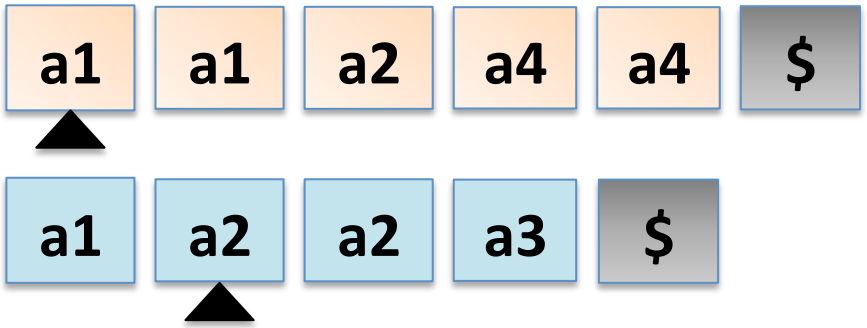
B accepts



P accepts

- Simulation B to P

B accepts

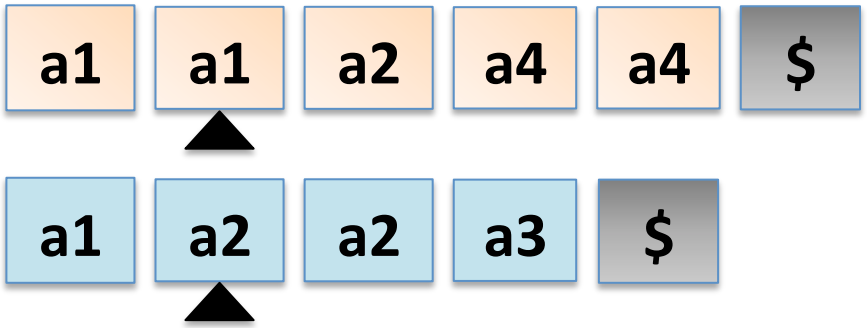


[a1,2]

P accepts

- Simulation B to P

B accepts

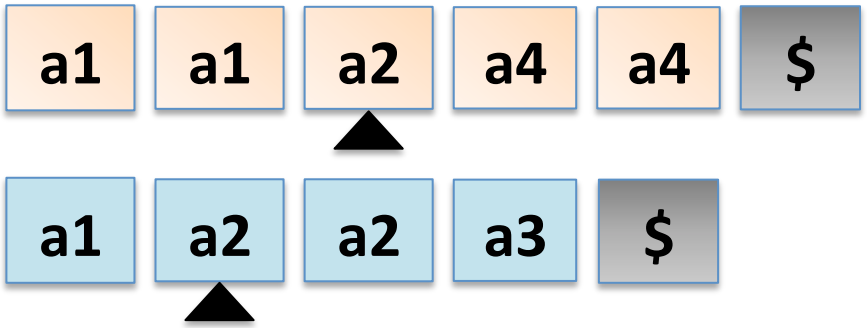


[a1,2] [a1,1]

P accepts

- Simulation B to P

B accepts

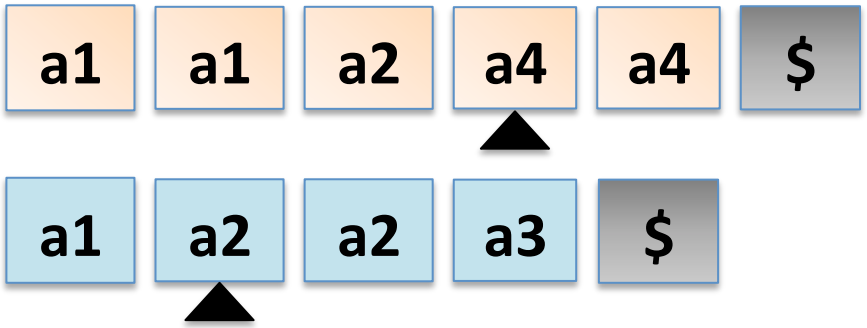


[a1,2][a1,1] [a1,1]

P accepts

- Simulation B to P

B accepts

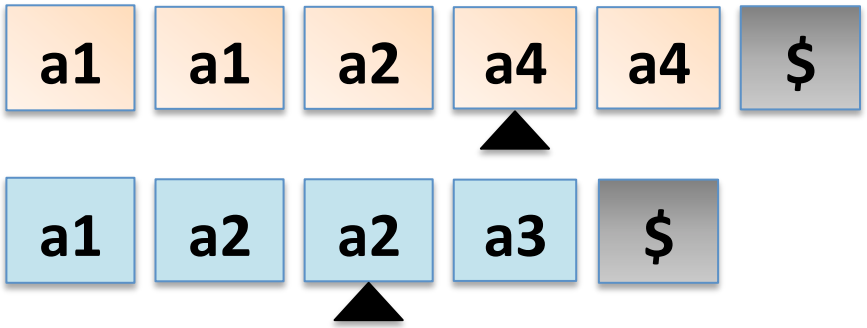


[a1,2][a1,1][a1,1] [a2,1]

P accepts

- Simulation B to P

B accepts

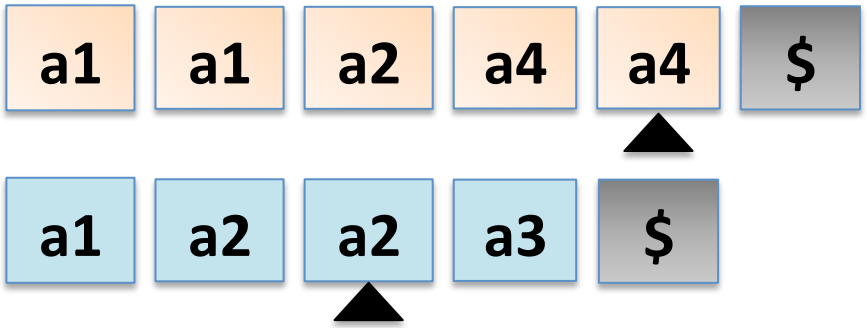


[a1,2][a1,1][a1,1][a2,1] [a2,2]

P accepts

- Simulation B to P

B accepts

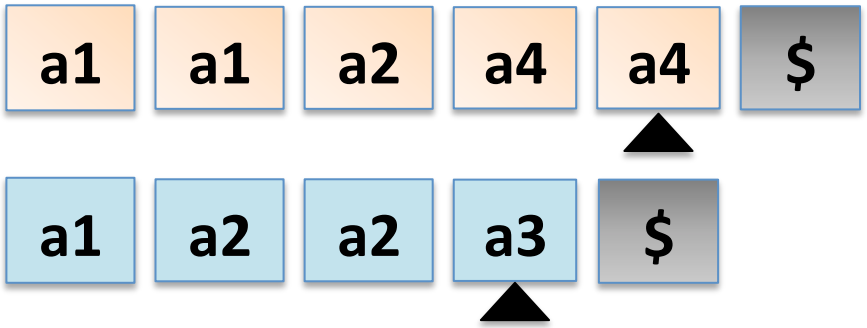


[a1,2][a1,1][a1,1][a2,1][a2,2][a4,1]

P accepts

- Simulation B to P

B accepts

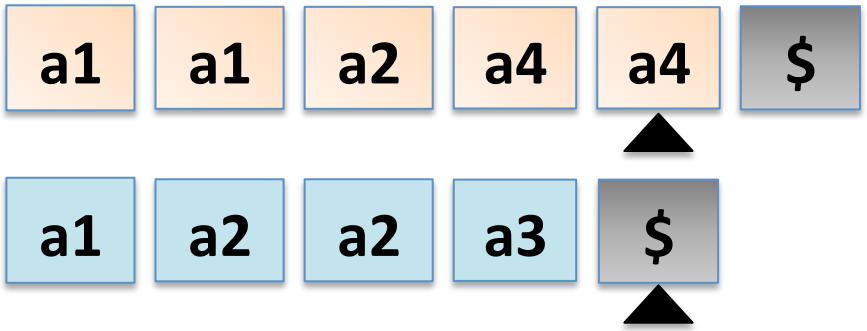


[a1,2][a1,1][a1,1][a2,1][a2,2][a4,1][a2,2]

P accepts

- Simulation B to P

B accepts

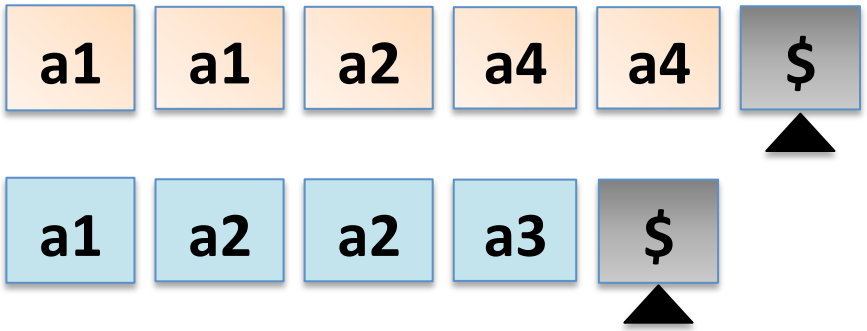


[a1,2][a1,1][a1,1][a2,1][a2,2][a4,1][a2,2][a3,2]

P accepts

- Simulation B to P

B accepts

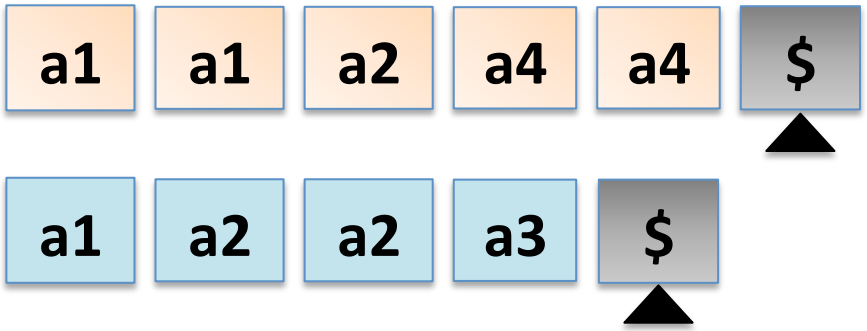


[a1,2][a1,1][a1,1][a2,1][a2,2][a4,1][a2,2][a3,2][a4,1]

P accepts

- Simulation B to P

B accepts



[a1,2][a1,1][a1,1][a2,1][a2,2][a4,1][a2,2][a3,2][a4,1]

P accepts

Linear blowup

So far ...

M accepts $w\downarrow 1 \uparrow k \downarrow 1 \;\; w\downarrow 2 \uparrow k \downarrow 2 \;\; \ldots w\downarrow n \uparrow k \downarrow n$

iff

A accepts $w\downarrow 1 \uparrow k \downarrow 1 \;\; w\downarrow 2 \uparrow k \downarrow 2 \;\; \ldots w\downarrow n \uparrow k \downarrow n$

…

$w\downarrow 1 \uparrow k \downarrow 1 \;\; w\downarrow 2 \uparrow k \downarrow 2 \;\; \ldots w\downarrow n \uparrow k \downarrow n$

iff

B accepts $a\downarrow 1 \uparrow k \downarrow 1 \;\; a\downarrow 2 \uparrow k \downarrow 2 \;\; \ldots a\downarrow n \uparrow k \downarrow n$

…

$a\downarrow 1 \uparrow k \downarrow 1 \;\; a\downarrow 2 \uparrow k \downarrow 2 \;\; \ldots a\downarrow n \uparrow k \downarrow n$

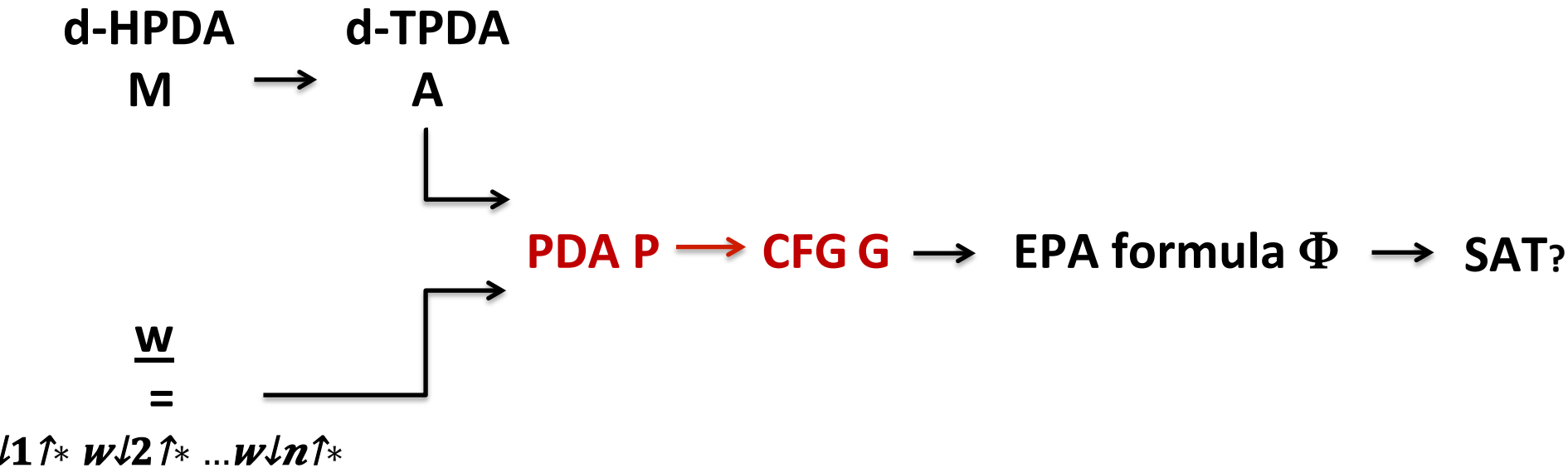
iff

P accepts some element of

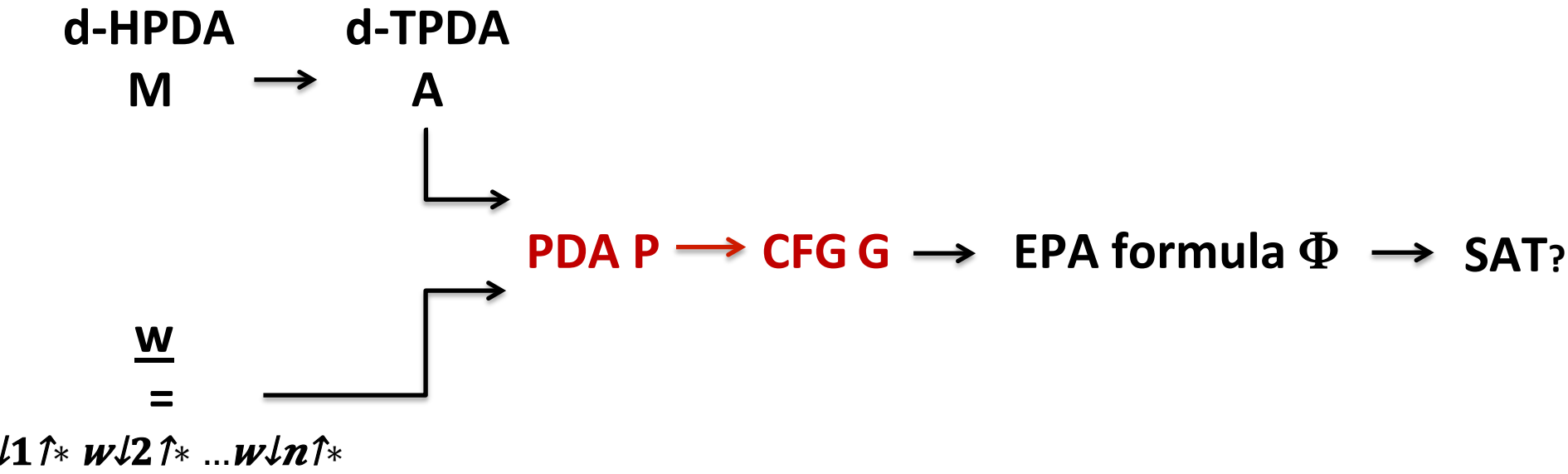
$[a\downarrow 1 , 1]\downarrow\uparrow k \downarrow 1 \;\; [a\downarrow 2 , 1]\downarrow\uparrow k \downarrow 2 \;\; \ldots [a\downarrow n , 1]\downarrow\uparrow k \downarrow n$ |||…|||

$[a\downarrow 1 , d]\downarrow\uparrow k \downarrow 1 \;\; [a\downarrow 2 , d]\downarrow\uparrow k \downarrow 2 \;\; \ldots [a\downarrow n , d]\downarrow\uparrow k \downarrow n$

Proof Outline



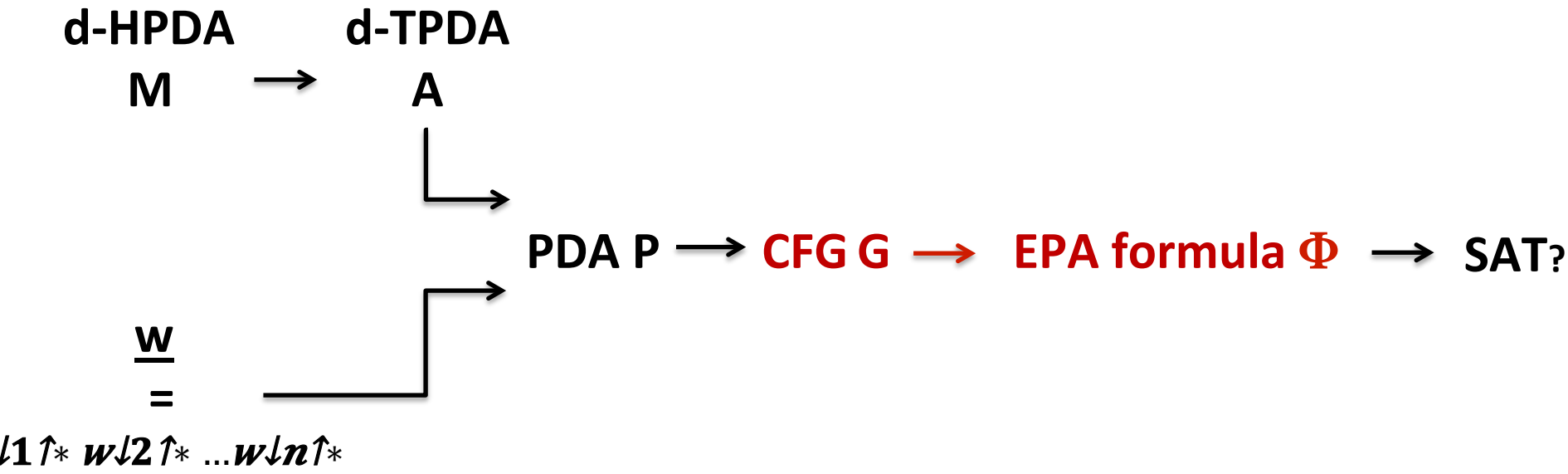
Proof Outline



Standard language-preserving translation

Cubic blowup

Proof Outline



So far…

M accepts $w\downarrow 1 \uparrow k \downarrow 1 \;\; w\downarrow 2 \uparrow k \downarrow 2 \;\; \ldots w\downarrow n \uparrow k \downarrow n$

iff

P accepts some element of

$[a\downarrow 1, 1]\downarrow\uparrow k\downarrow 1 \;\; [a\downarrow 2, 1]\downarrow\uparrow k\downarrow 2 \;\; \ldots [a\downarrow n, 1]\downarrow\uparrow k\downarrow n$ ||| … |||

$[a\downarrow 1, d]\downarrow\uparrow k\downarrow 1 \;\; [a\downarrow 2, d]\downarrow\uparrow k\downarrow 2 \;\; \ldots [a\downarrow n, d]\downarrow\uparrow k\downarrow n$

iff

$Parikh(L(G))$ contains $(k\downarrow 1, \ldots, k\downarrow n, \ldots, k\downarrow 1, \ldots, k\downarrow n)$

d times

So far

M accepts $w\downarrow 1\uparrow k\downarrow 1 \;\; w\downarrow 2\uparrow k\downarrow 2 \;\; \ldots w\downarrow n\uparrow k\downarrow n$

iff

Parikh(L(G)) contains $(k\downarrow 1\,, \ldots, k\downarrow n\,, \ldots, k\downarrow 1\,, \ldots, k\downarrow n\,)$

iff

$(k\downarrow 1\,, \ldots, k\downarrow n\,)$ is a model of

$$\exists y\downarrow 1 \ldots \exists z\downarrow n \quad \varphi\downarrow G\,(x\downarrow 1\,, \ldots, x\downarrow n\,, y\downarrow 1\uparrow 1\,, \ldots, y\downarrow n\uparrow 1\,, \ldots, y\downarrow 1\uparrow d-1\,, \ldots, y\downarrow n\uparrow d-1\,) \wedge$$

$$x\downarrow 1 = y\downarrow 1\uparrow 1 = \ldots = y\downarrow 1\uparrow d-1 \wedge \ldots \wedge x\downarrow n = y\downarrow n\uparrow 1 = \ldots = y\downarrow n\uparrow d-1$$

[VSS05] For every CFG G, there is a formula $\varphi\downarrow G$ of existential Presburger arithmetic (EPA) of linear size representing $Parikh(L(G))$

So far

M accepts $w\downarrow 1\uparrow k\downarrow 1 \;\; w\downarrow 2\uparrow k\downarrow 2 \;\; \ldots w\downarrow n\uparrow k\downarrow n$

iff

Parikh(L(G)) contains $(k\downarrow 1 , \ldots, k\downarrow n , \ldots, k\downarrow 1 , \ldots, k\downarrow n)$

iff

d times

$(k\downarrow 1 , \ldots, k\downarrow n)$ is a model of

$\exists y\downarrow 1 \ldots \exists z\downarrow n \quad \varphi\downarrow G \; (x\downarrow 1 , \ldots, x\downarrow n , y\downarrow 1 \uparrow \ldots \uparrow d-1 , \ldots, y\downarrow n\uparrow d-1) \wedge$

$x\downarrow 1 = y\downarrow 1\uparrow 1 = \ldots \qquad y\downarrow n\uparrow 1 = \ldots = y\downarrow n\uparrow d-1$

Linear blowup

Complexity

- One exponential blowup (A to B)
- Exponential in the number of heads only
  - And only necessary if the $w\downarrow i$´s have at least 2 characters
- The problem is NP-complete for patterns of the form $a\downarrow 1\uparrow *\ a\downarrow 2\uparrow *\ \ldots a\downarrow n\uparrow *$

01)*

Lower Bound: coEXPTIME-hard

- Reduce from acceptance of APSPACE TM
- Works already for (01)* as bounded language
- Key Idea: Use k heads to store configurations in $\{0,1\}k$
- Using stack, can store AND-configurations

Closure Properties mod BLs

- Closure under union:
  - Use non-determinism
- Closure under intersection:
  - Given $k\downarrow 1$ -HPDA and $k\downarrow 2$ -HPDA, construct a $(k\downarrow 1 + k\downarrow 2)$-HPDA that simulates both machines
- So, closed under union/intersection mod BL

Closure under negation

Given d-HPDA $\mathcal{M}$, BL $\mathcal{B}$:

- Compute the existential PA formula  (with constants in unary)
- Compute a Q-free formula Ψ equivalent to $\neg\Phi$
- Build a MHPDA for each basic constraint and combine using Boolean operators

Optimality

Let $\mathcal{U}$ = Finite unions of BL + Finite languages

$\mathcal{U}$ is the largest class of regular languages s.t. MHPDA are perfect modulo $\mathcal{U}$

Applications to Verification

Example 1: Recursive multithreaded programs

Example 2: Recursive counter machines

Example 3: Recursive communicating FSMs

Recursive Multithreaded Programs

- Thread → PDA
- Language of multithreaded program: intersection of the languages of the PDA

Recursive Counter Machines

- Pushdown system with k counters
- Each counter can be incremented or decremented or tested for zero
- Head 0 checks a sequence of transitions is syntactically feasible
- Heads 1 to k : Head i checks the sequence is semantically possible with respect to the i-th counter
  (simulates counter using the stack)

Recursive CFSM

- FSMs communicating through queues
- Similar construction
- For each queue: use 2 heads to ensure dequeues and enqueues match for each queue

Conclusion

- First nontrivial bounded paradise
- MHPDA are an expressive model for which bounded verification is possible
  - Many common models easily compiled into it
  - Resulting proofs separate the compilation from the combinatorics