A Modular Approach to Büchi Determinization

Dana Fisman and Yoad Lustig

CONCUR'15, Madrid

Motivation

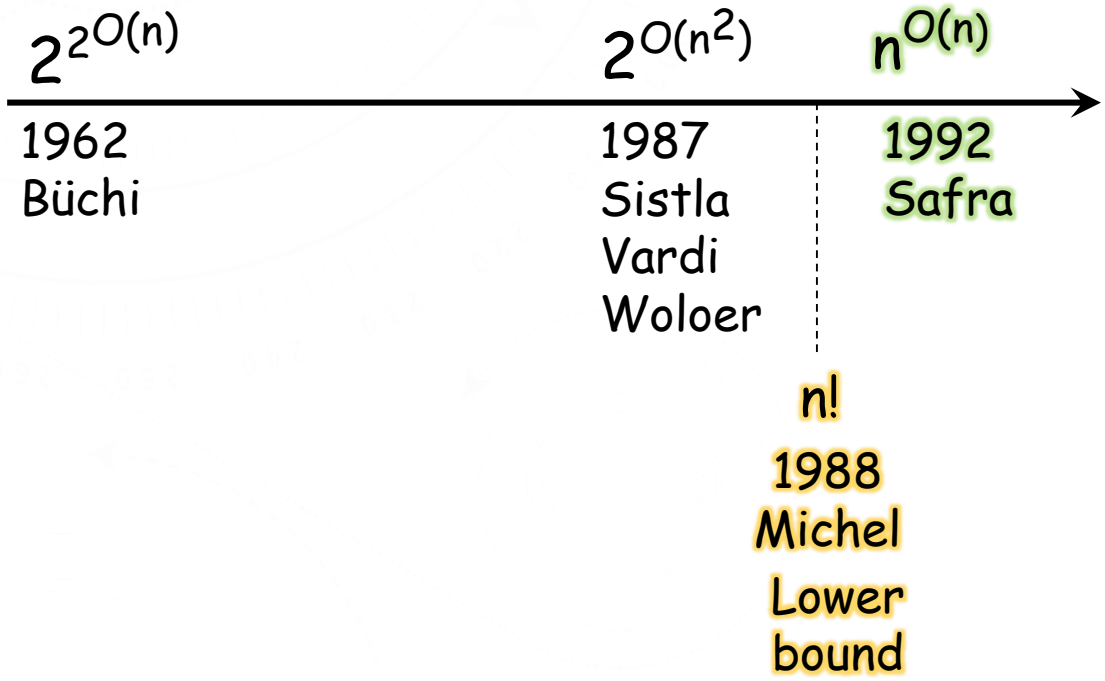
- Büchi determinization is a fundamental problem

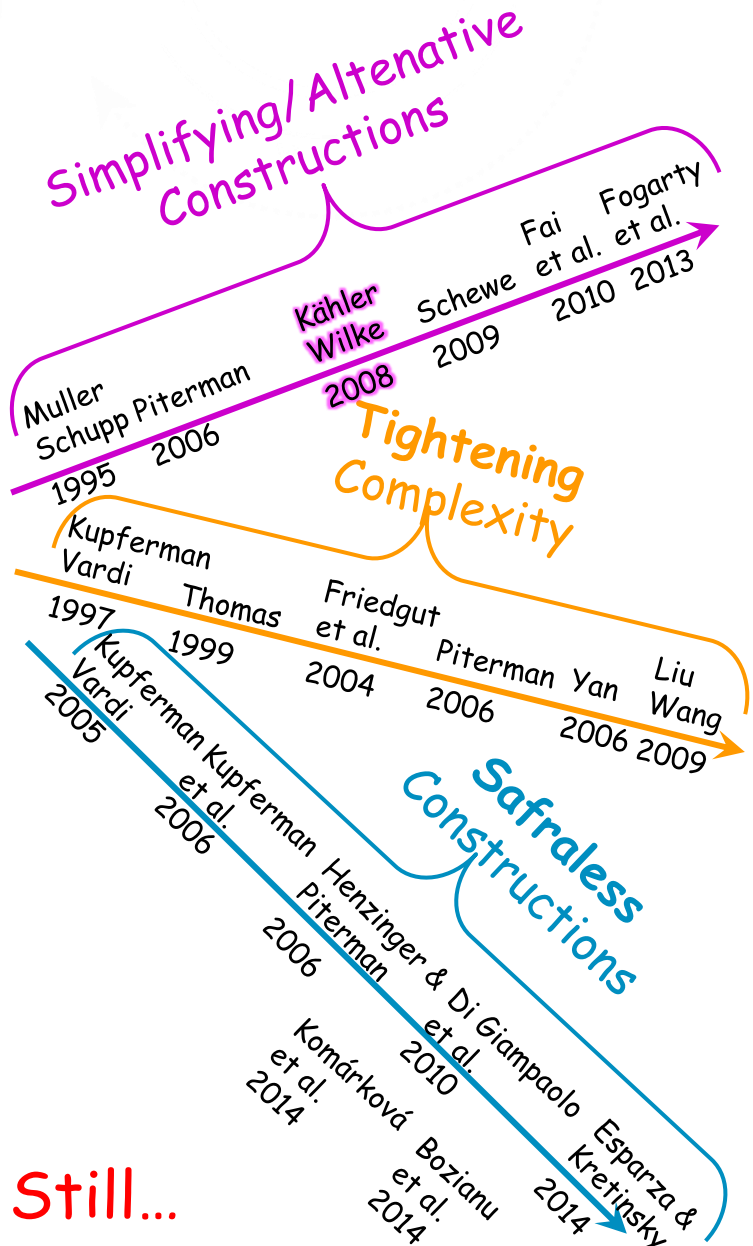
- With applications in
  - Reactive synthesis
  - Multi-agent systems
  - Probabilistic Verification

History (very briefly)

Problem is solved!

Solution achieves the optimal $n^{O(n)}$





Motivation

- Wanted:

  a solution with

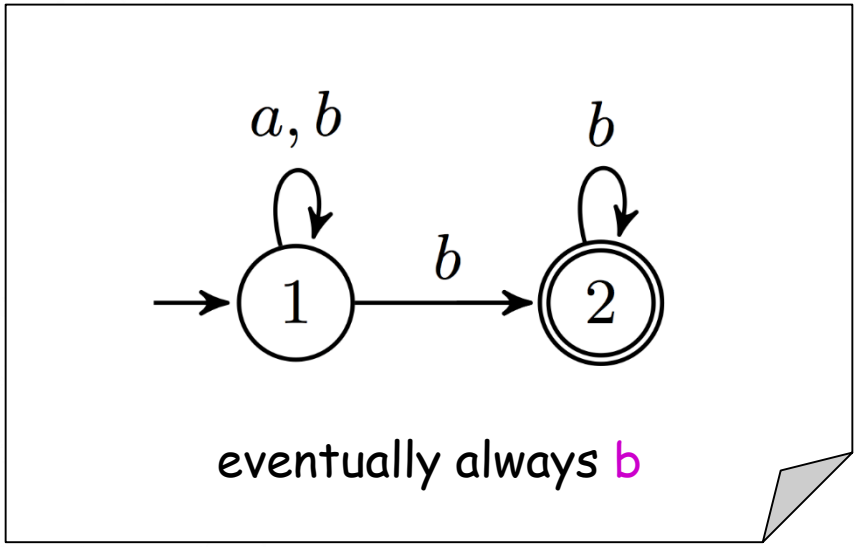
  - Few easy to understand ideas
  - No magic
  - Modular / gradual
  - Complexity in $n^{O(n)}$

Outline

- Some definitions
- Why subset construction and simple approached fail
- Kähler & Wilke's trees (our starting point)
- Overview of our approach
- Construction details

Büchi Automata

- Looks like an NFA, but definition of acceptance is different.
- An infinite word is accepted if it visits one of the accepting states infinitely often.



Other w-Automata

- There are other w-automata differing in the way acceptance is defined
  - Muller
  - Rabin
  - Streett
  - Parity
- We use three letter acronyms

  DPA = deterministic Parity Automaton

  NBA = non-deterministic Büchi Automaton

Determinization

- DBA are less expressive than NBA
- Determinization of Büchi automata refers to transforming them into an equivalent deterministic automaton, using another acceptance criteria (DMA, DRA, DSA, DPA)

Parity Automaton

- In a Parity Automaton:
  - Every state has a color (rank) in [0..k]
  - A run is accepting if the min color visited inf. often is odd.
- Can think of it as having:
  - K buzzers

  - K bells

  - Accept if the min bell to ring inf. often is smaller than the min buzzer to buzz inf. often

A Büchi automaton is a parity automaton with one bell and no buzzers

Determinization on Finite Words

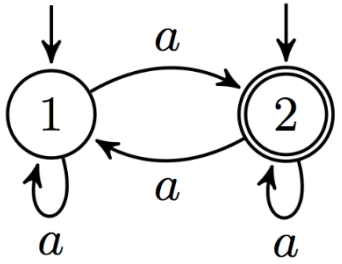
By Subset Construction

- A state in the constructed DFA automaton records the subset of states the NFA can be in after reading the respective word

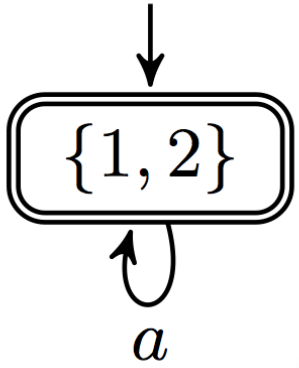
Subset Construction Fails

$\Sigma = \{ a \}$

Only one word to consider: a^ω

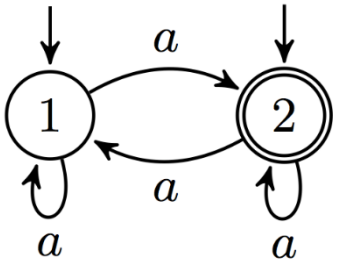


accepts a^ω

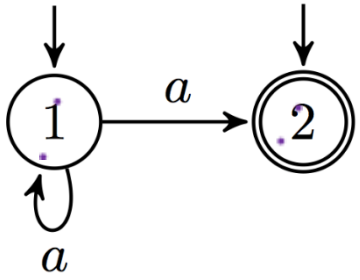


accepts a^ω

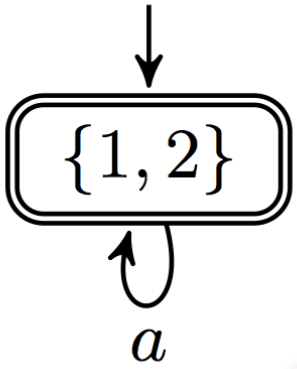
Subset Construction Fails



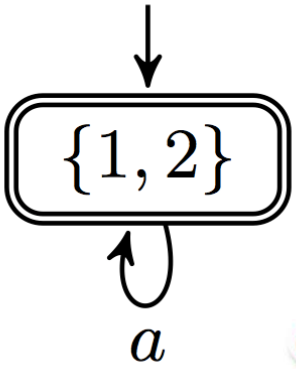
accepts a^ω



accepts nothing



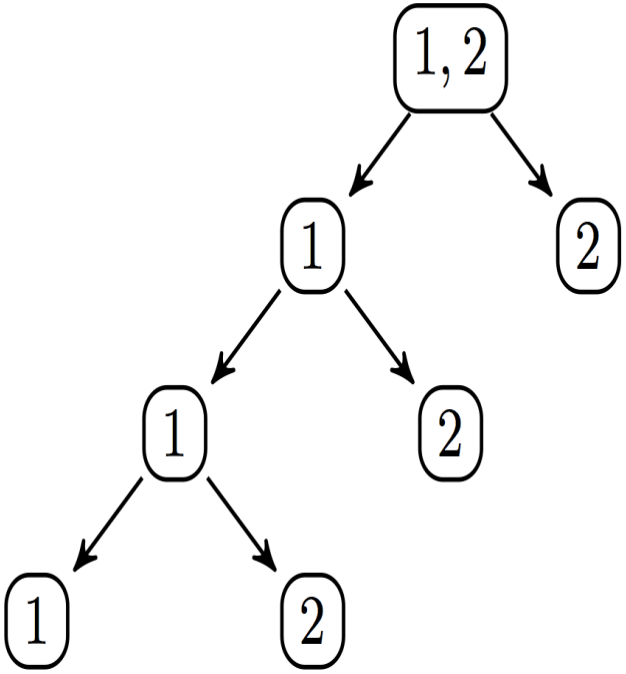
accepts a^ω



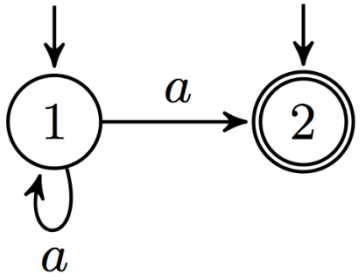
accepts a^ω

What went wrong?

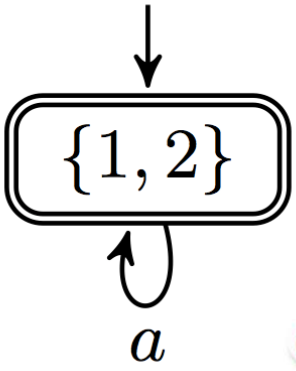
Run Tree:



State 2 is on every layer.
But it is not visited infinitely often.



accepts nothing



accepts a^ω

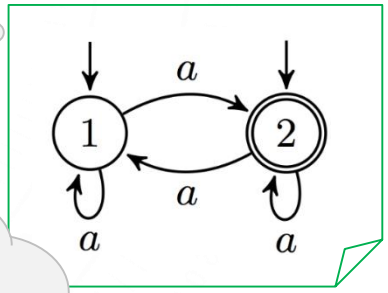
Need to record more!

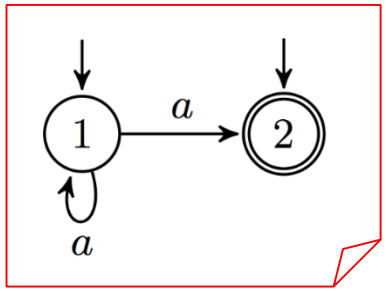
- Kähler and Wilke proposed the following structures to summarize runs of NBAs
  - Split Tree SP^A_w
  - Reduced Tree RD^A_w
  - Skeleton Tree SK^A_w
- All are defined per a given ω-word w and maintain the following property:
  - If there exists an accepting run on w in A then there exists an accepting path in the tree.

Split Tree

Follows the successors, but splits them into two sets:

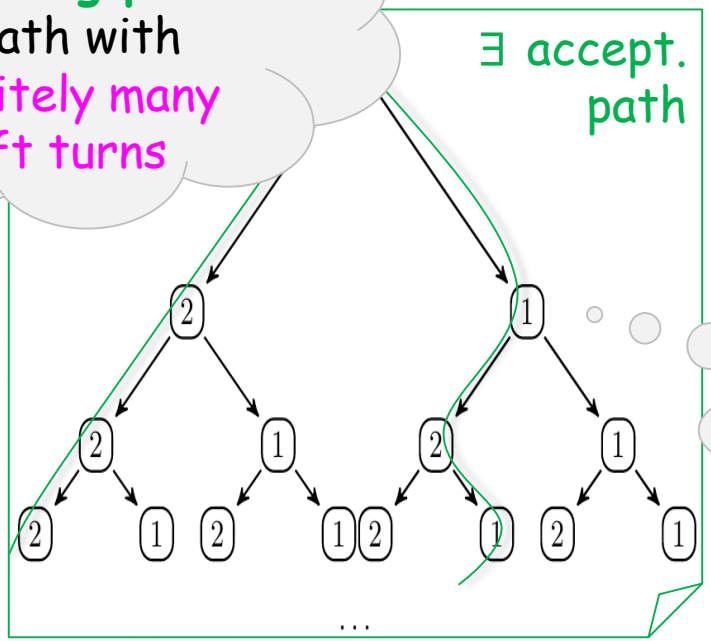
non-accepting on the right and accepting on the left



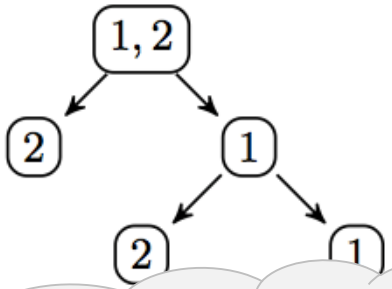


accepting path
= path with
infinitely many
left turns

∃ accept. path



no accept. path



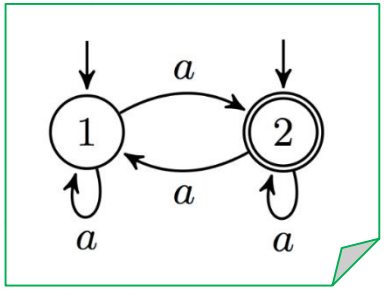
problem:
the width is
unbounded!

solution:
the reduced
tree!

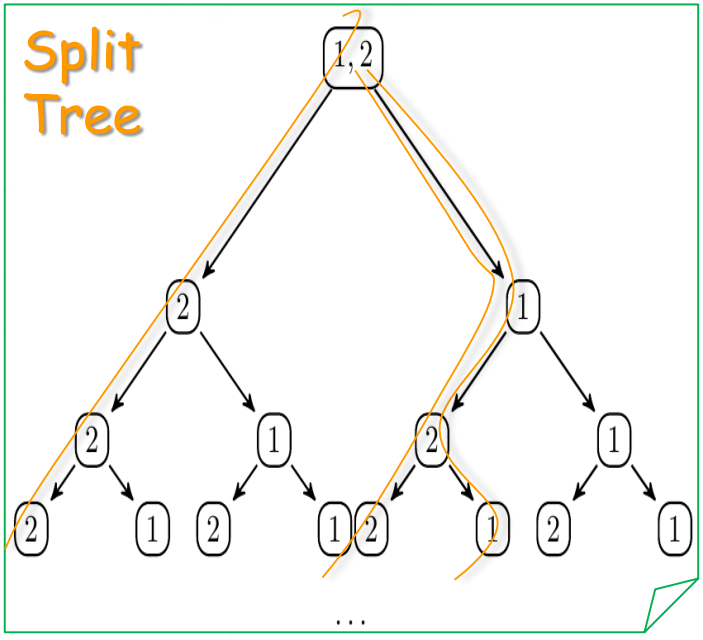
Reduced Tree

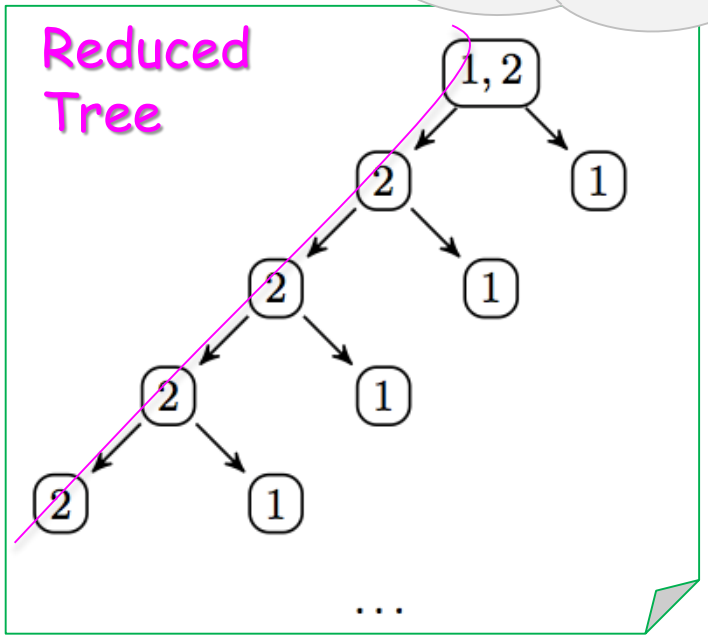
Eliminate from a node states that appeared in a node to its left (on the same layer)

4 3 01/



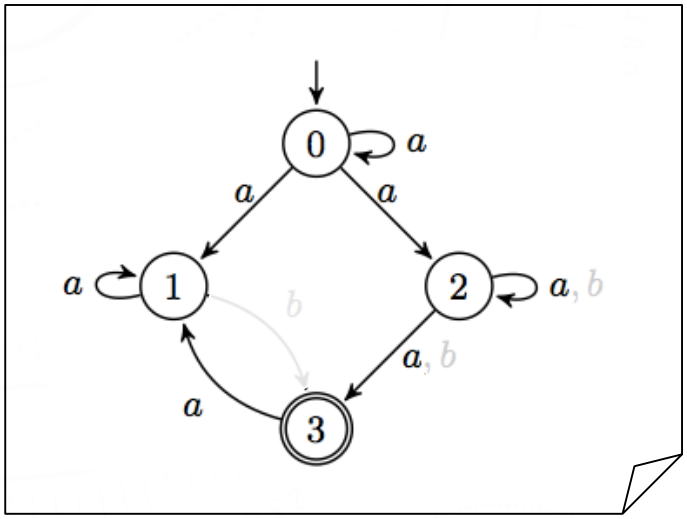
The width of the reduced tree is thus bounded by n

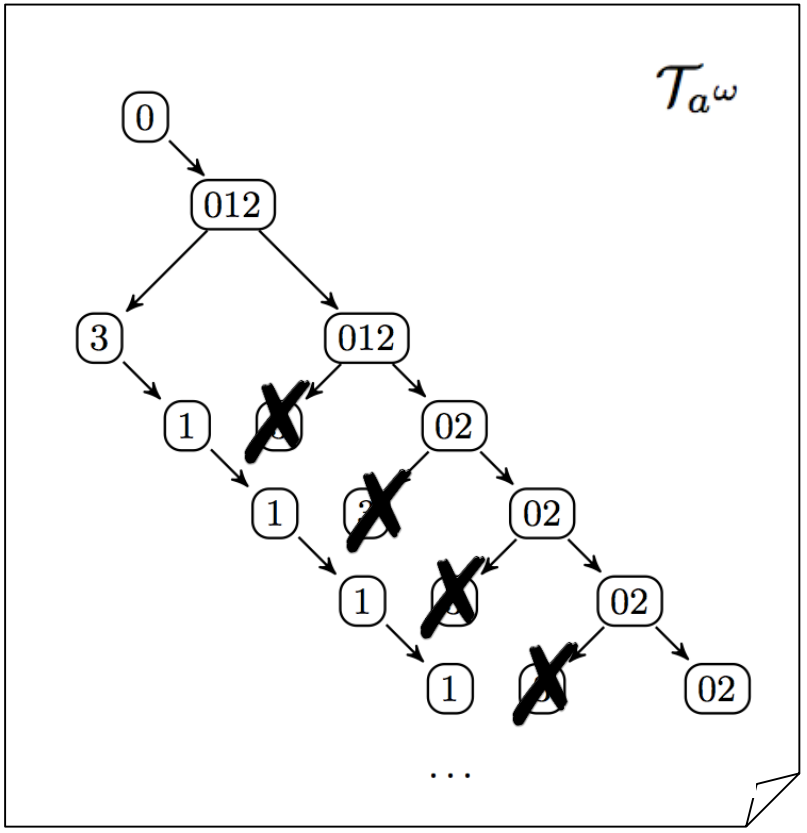




Skeleton Tree

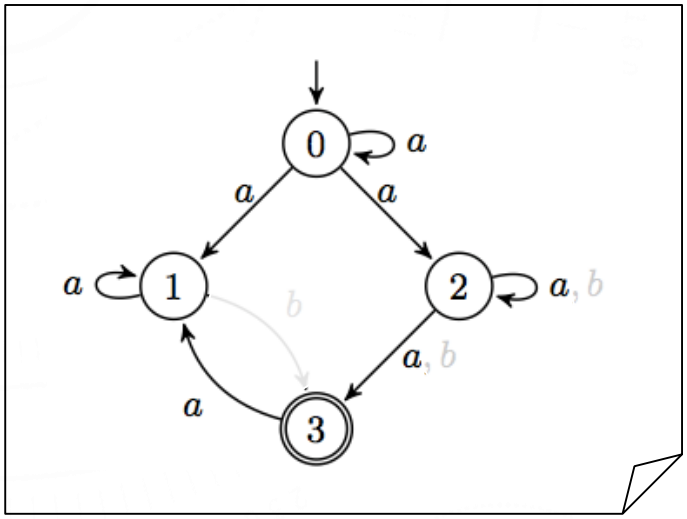
Eliminate dangling nodes (only infinite paths remain)



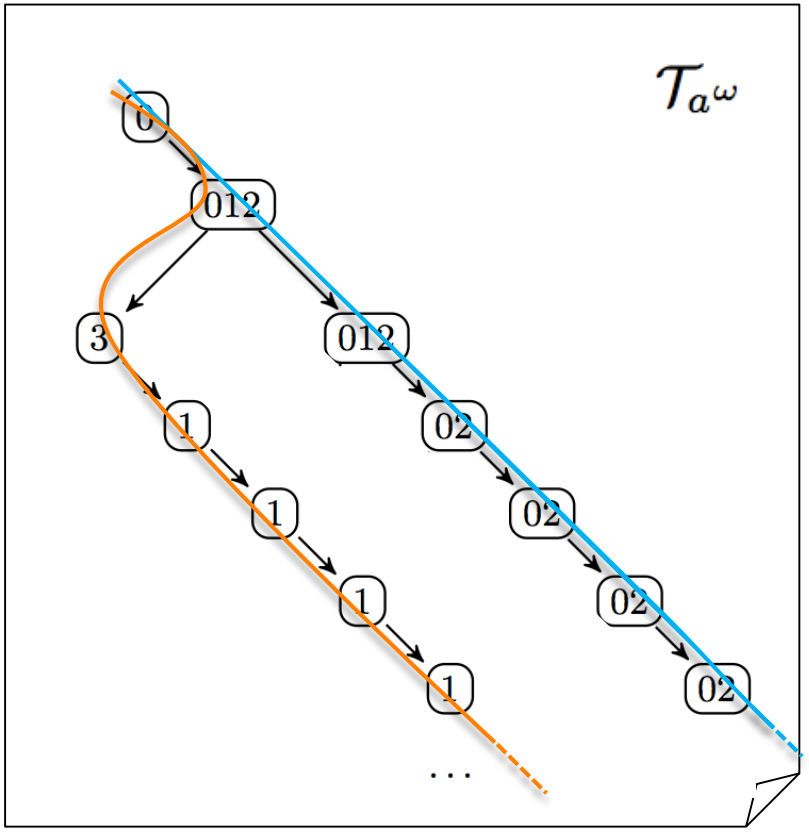


Skeleton Tree

Eliminate dangling nodes (only infinite paths remain)

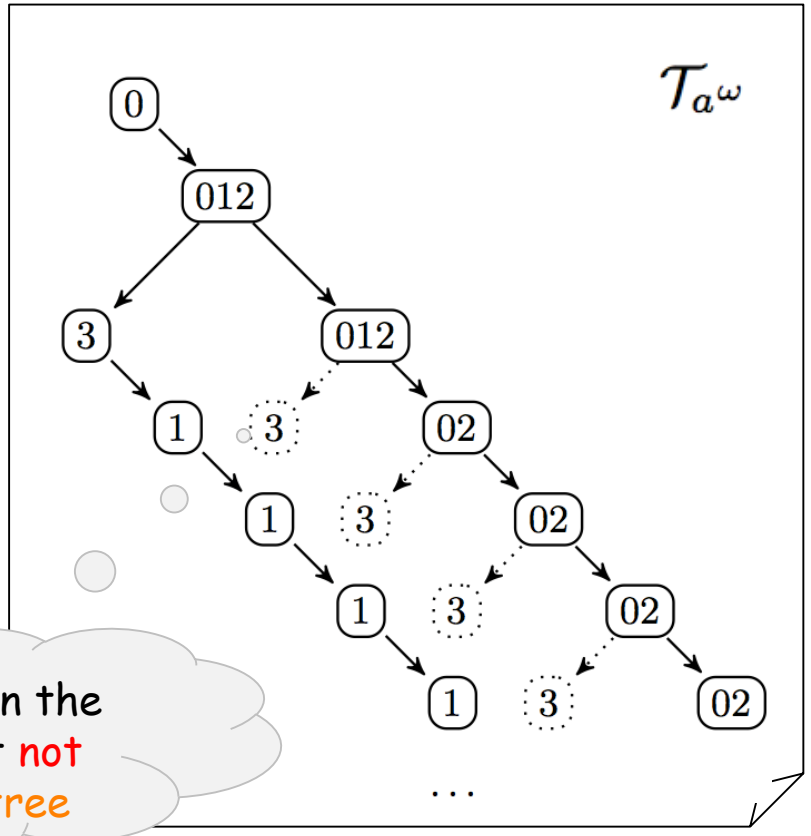


We call the number of paths in the skeleton tree the width.



Here the width is 2.

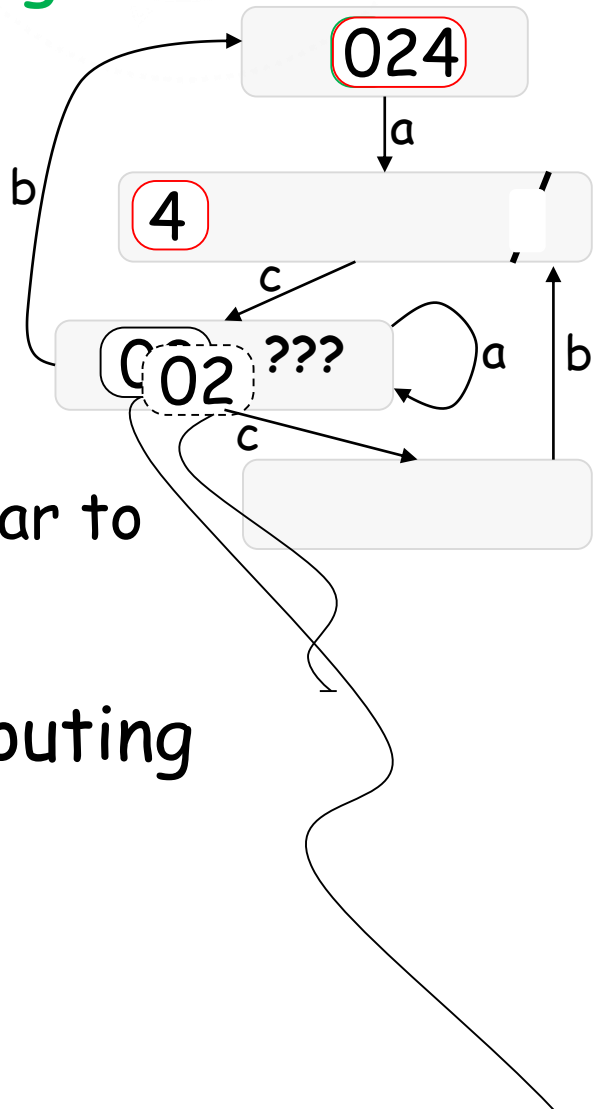
Reduced & Skeleton Tree



A dotted node is in the reduced tree but not in the skeleton tree

The Slice Automaton

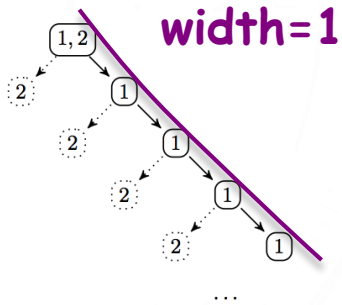
- We can build a det. aut computing the slices of the reduced trees
  - Slice by slice
  - Splitting the successors to accepting/non-accepting
  - Removing states that already appear to the left
- We cannot build a det. aut. computing the slices of the skeleton tree (requires knowing which node eventually has no decedents)



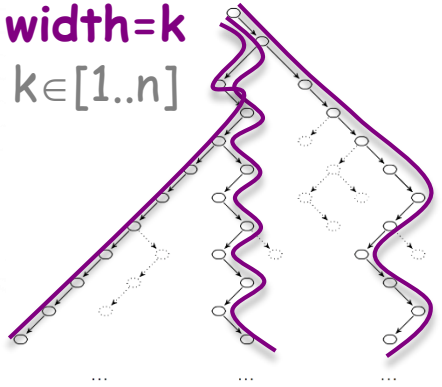
Our contribution starts here

Mission

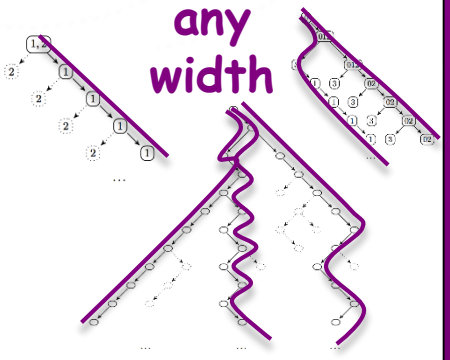
First



Then

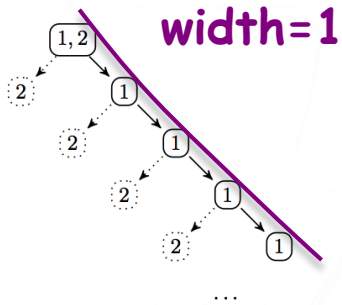


Finally



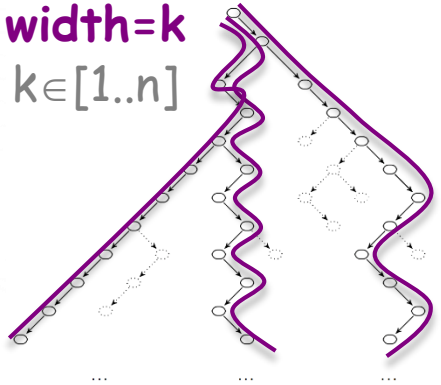
Mission

First



Devise a DPW P_1 that answers correctly only for words of width=1

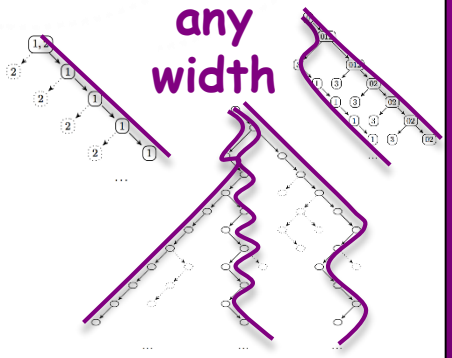
Then



Generalize to a DPW P_k that answers correctly only for words of width=k

given k∈[1..n]

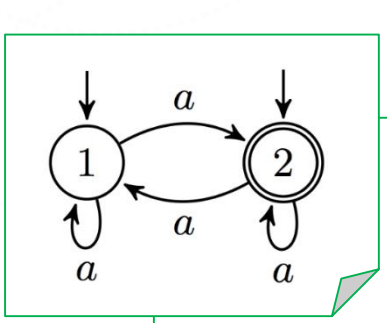
Finally

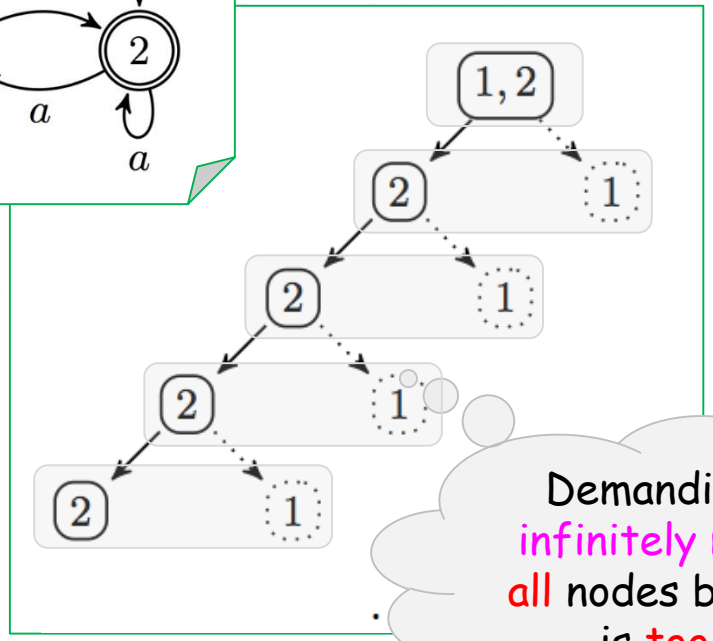


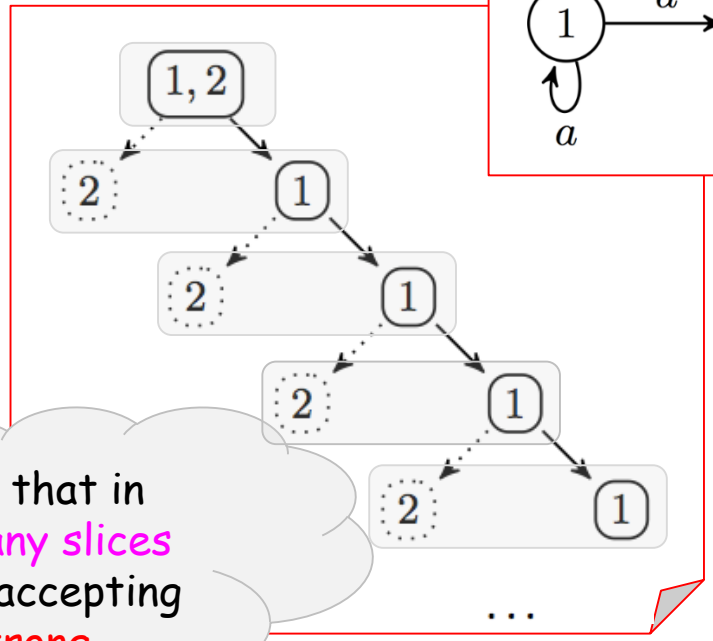
Obtain a DPW P that answers correctly for words of any width by running $P_1 \| P_2 \| \ldots \| P_k$

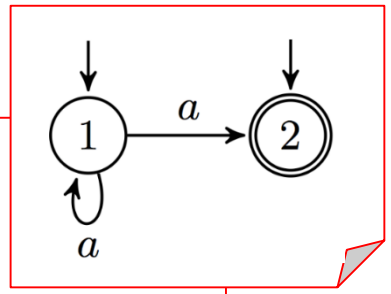
When the width is 1

- Need to answer if the single infinite path is accepting
- We know how to compute the slices ☺
- But we don't know which node is a skeleton node ☹





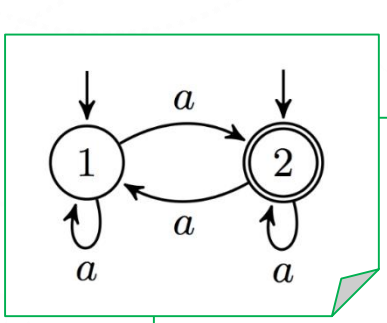


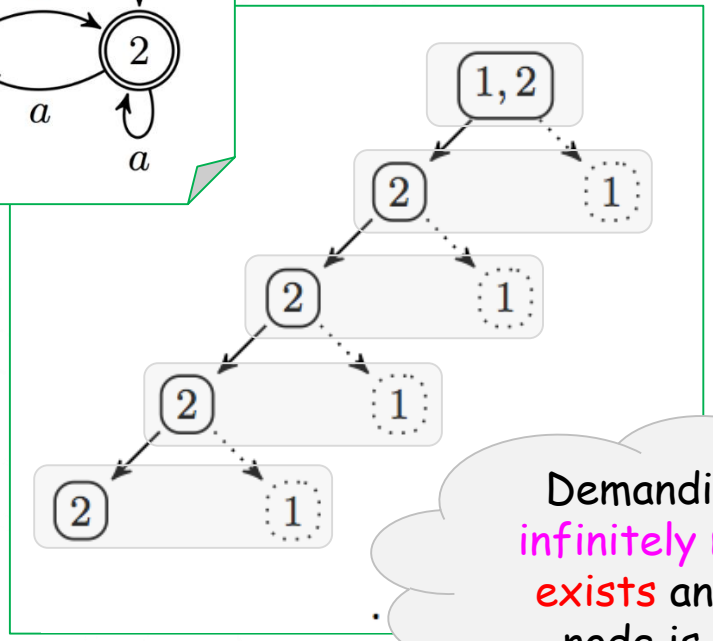


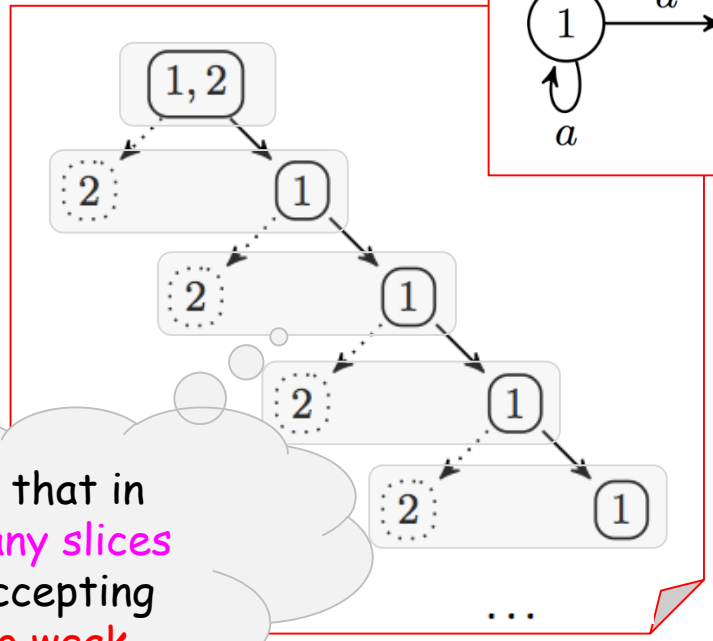
Demanding that in infinitely many slices all nodes be accepting is too strong

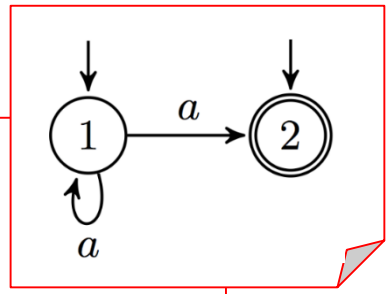
When the width is 1

- Need to answer if the single infinite path is accepting
- We know how to compute the slices ☺
- But we don't know which node is a skeleton node ☹









Demanding that in infinitely many slices exists an accepting node is too weak

When the width is 1

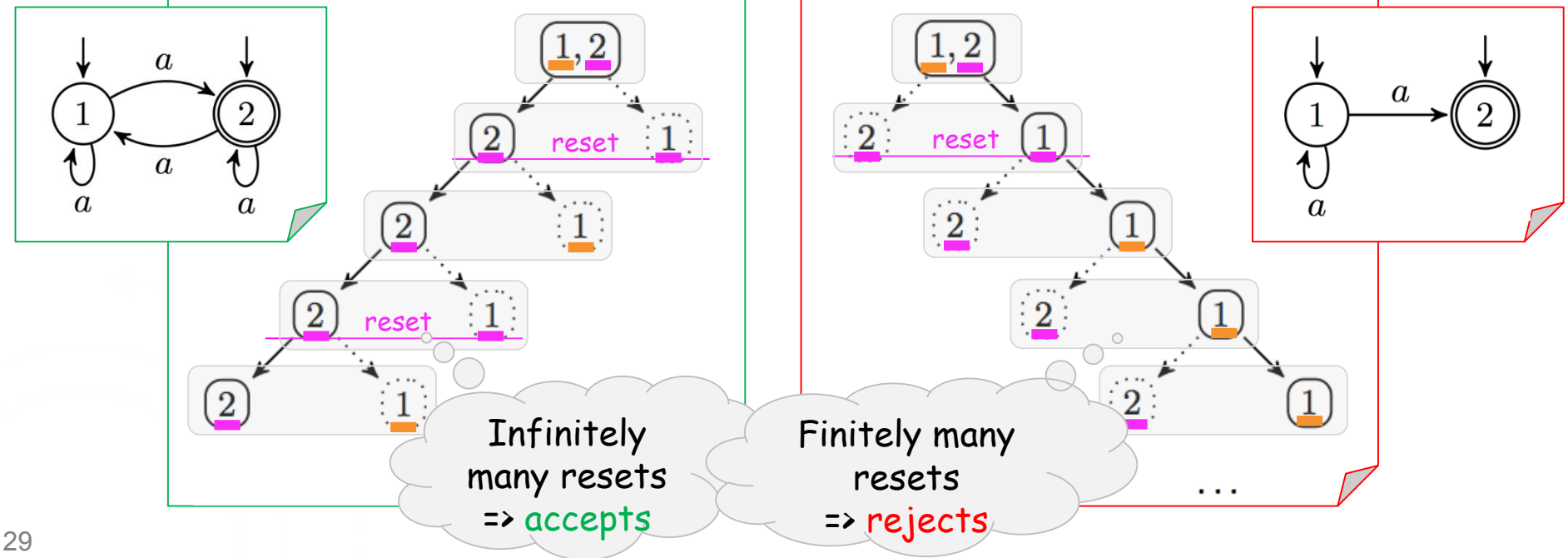
- The Miyano-Hayashi construction asks whether **ALL** infinite paths are accepting
- In the case that there is just **ONE** infinite path, this is the same as asking if **THE** single infinite path is accepting.
- MH-construction:
  - For each node, we track whether it is on a path that visited an accepting state recently or not.
  - We denote it as visited or owes.
  - If all nodes on a layer are marked visited we perform a "reset" – start charging everyone for a visit again

When the width is 1

Fisman Lustig

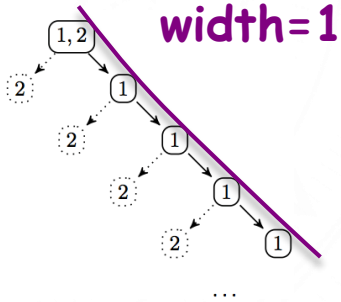
Conclusion:
Desired P_1 can simply be a DBW implementing Miyano-Hayashi on the Reduced Tree

Running Example



Mission

First ✓



Devise a DPW P_1 that answers correctly only for words of width=1

Then

width=k
k∈[1..n]

Generalize to a DPW P_k that answers correctly only for words of width=k

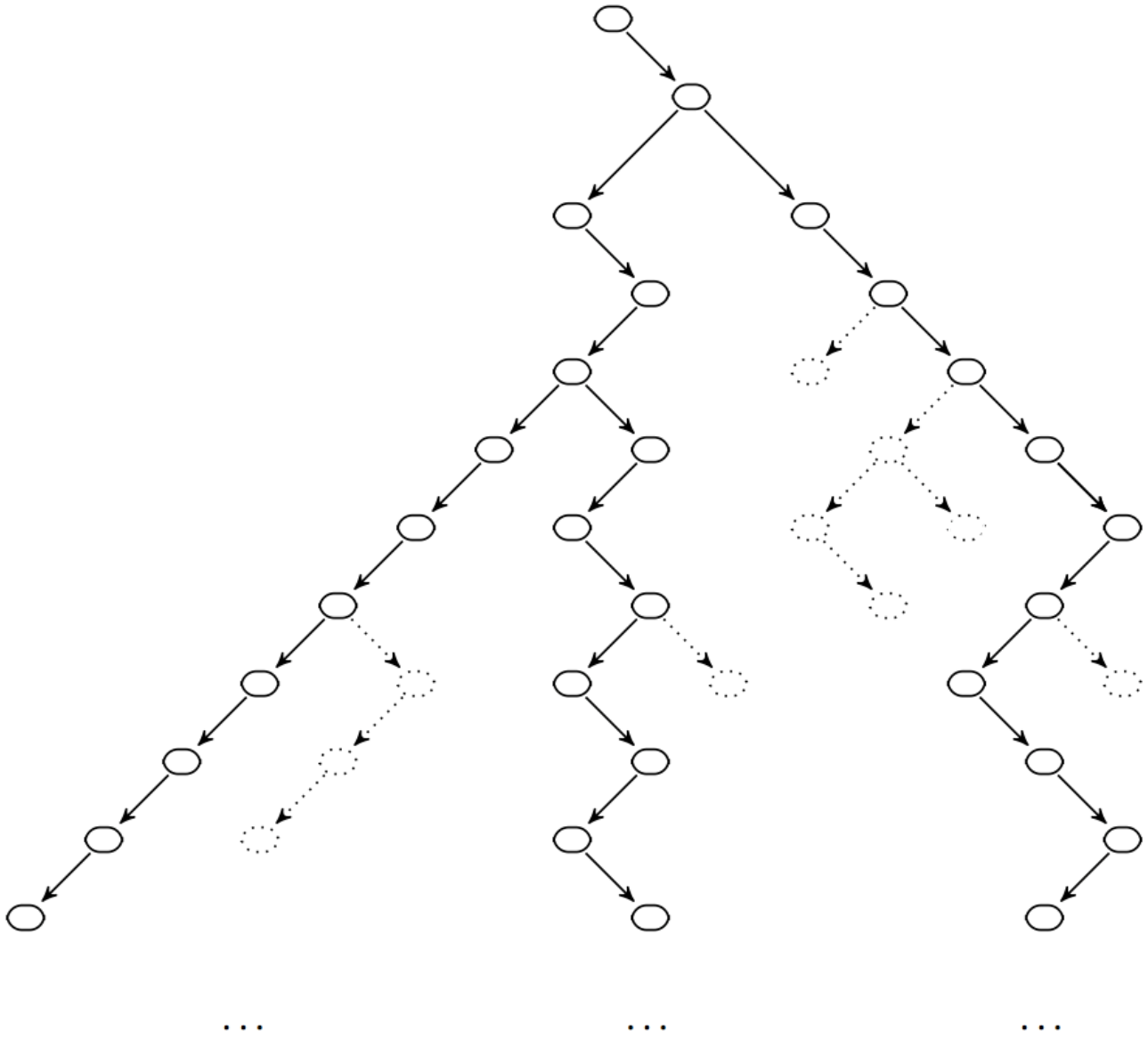
given k∈[1..n]

Finally

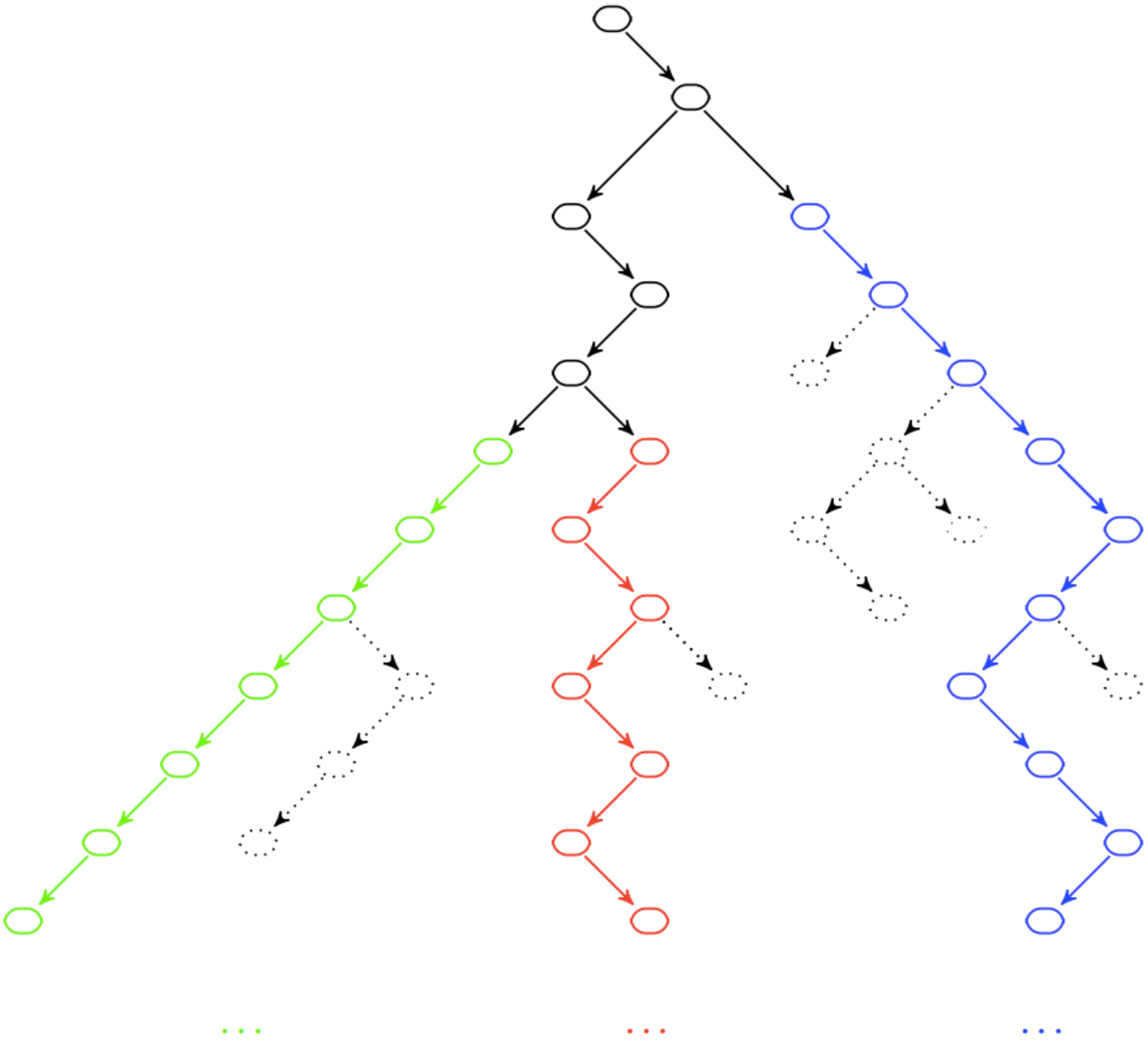
any width

Obtain a DPW P that answers correctly for words of any width by running $P_1 || P_2 || \ldots || P_k$

When the width is not 1 ?

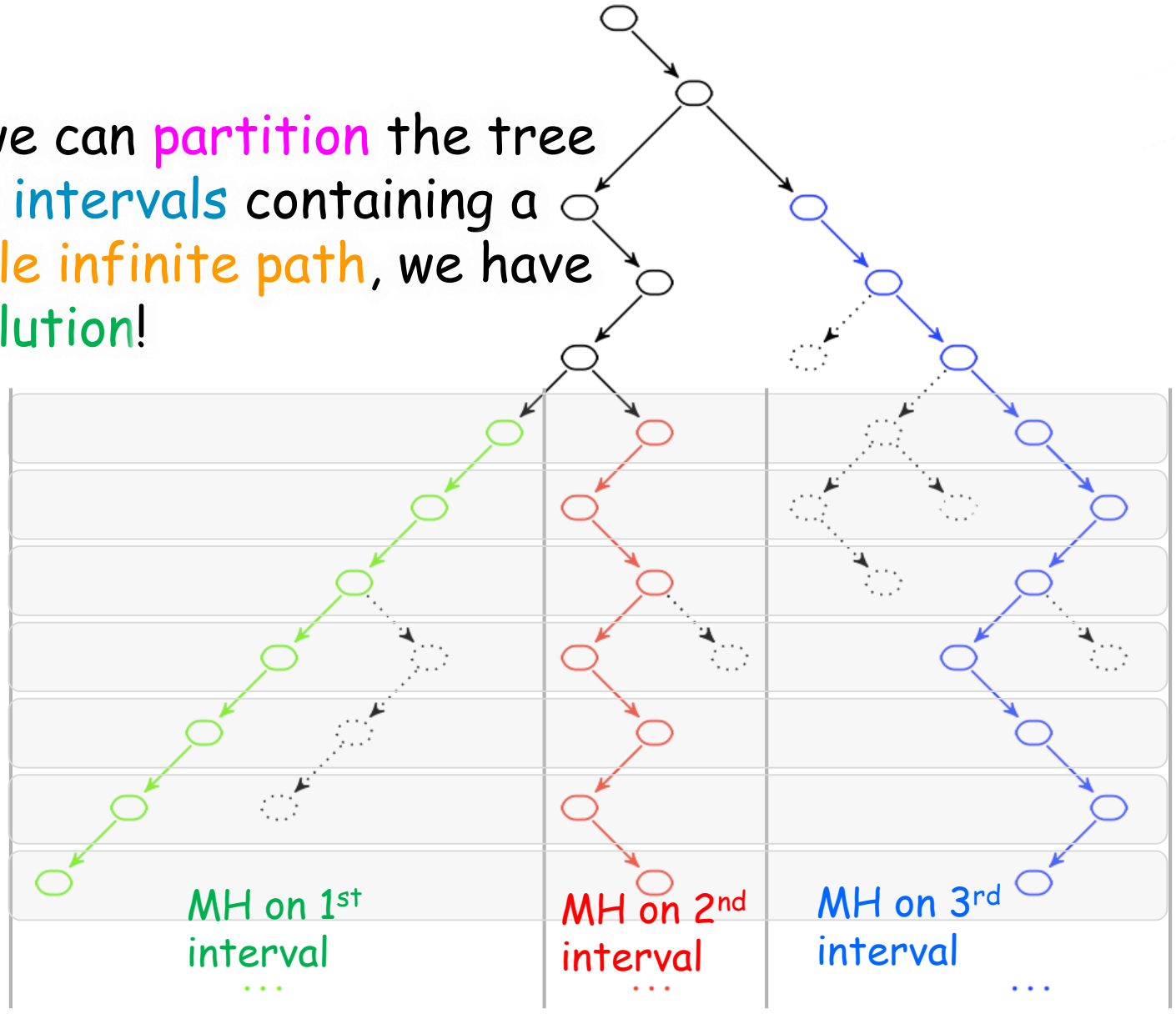


Here the width is 3



Key Observation !

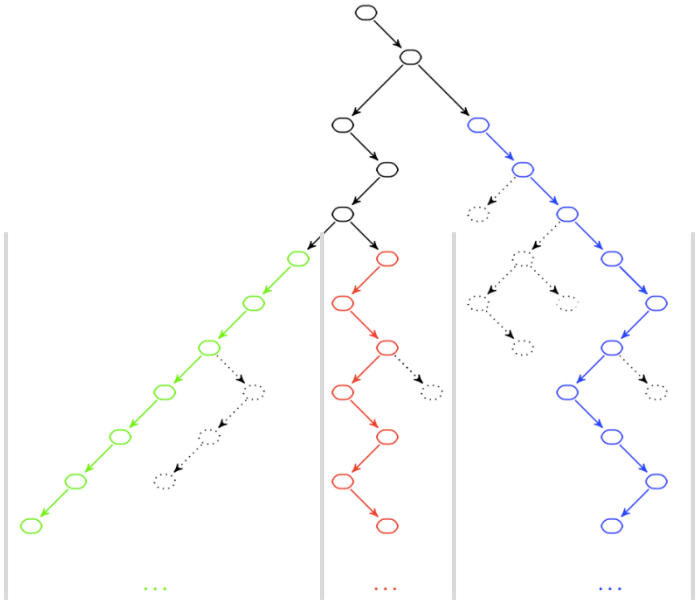
If we can partition the tree into intervals containing a single infinite path, we have a solution!



But how?

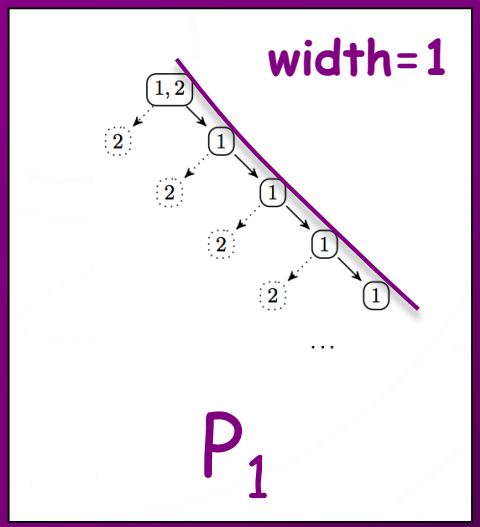
But how can we find this conceptual partition?

- First let's try to answer a simpler question:
  - What is the width?
- Or put as a decision problem:
  - Is the width < k ?

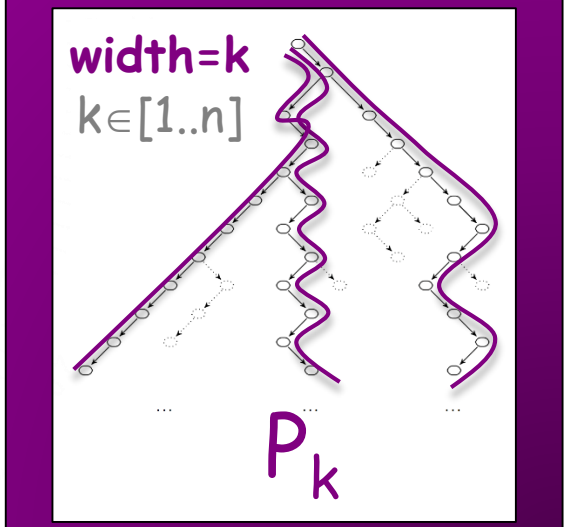


Mission

First



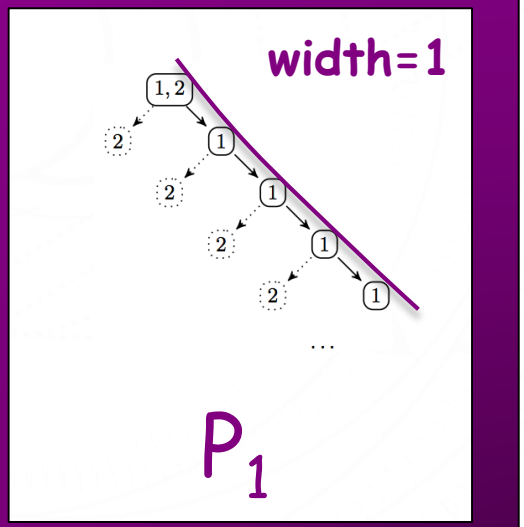
Then



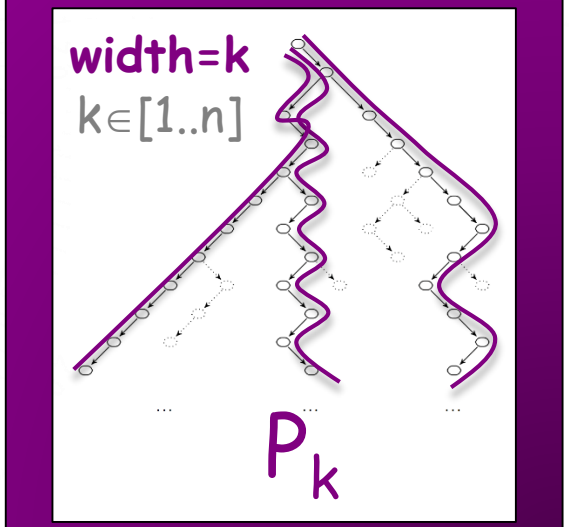
Finally

Mission

First

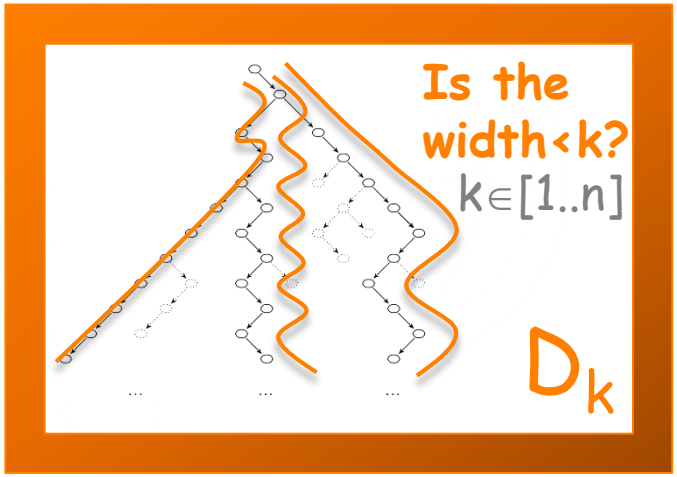


afterwards



Finally

then

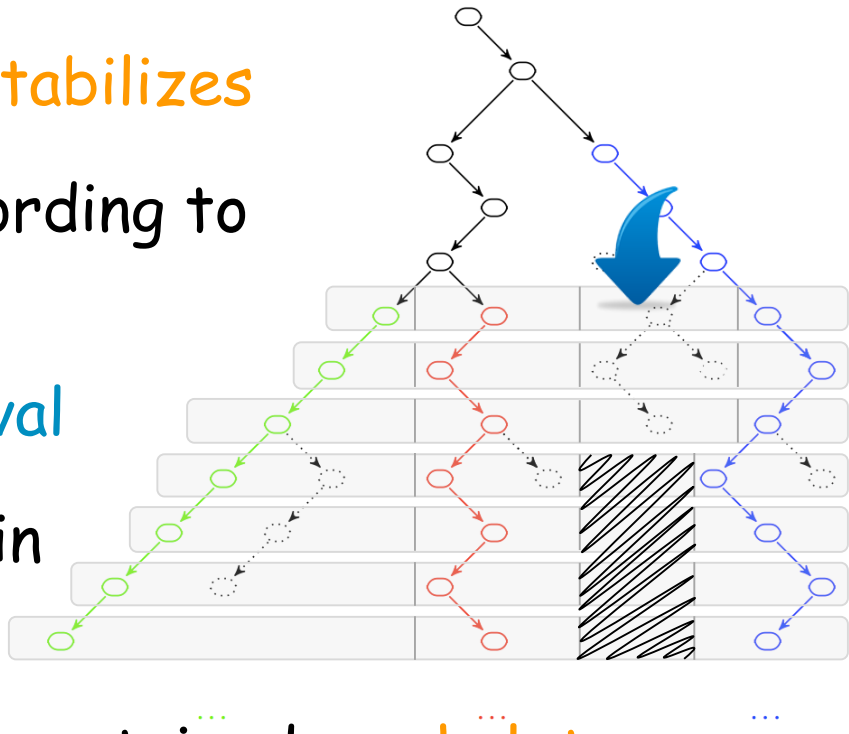


Deciding if width < k (idea)

- The width is monotonically non-decreasing
- Suppose we can start after it stabilizes
- We determine the interval according to the current nodes:

  each node in its own interval

- We develop the successor within the intervals



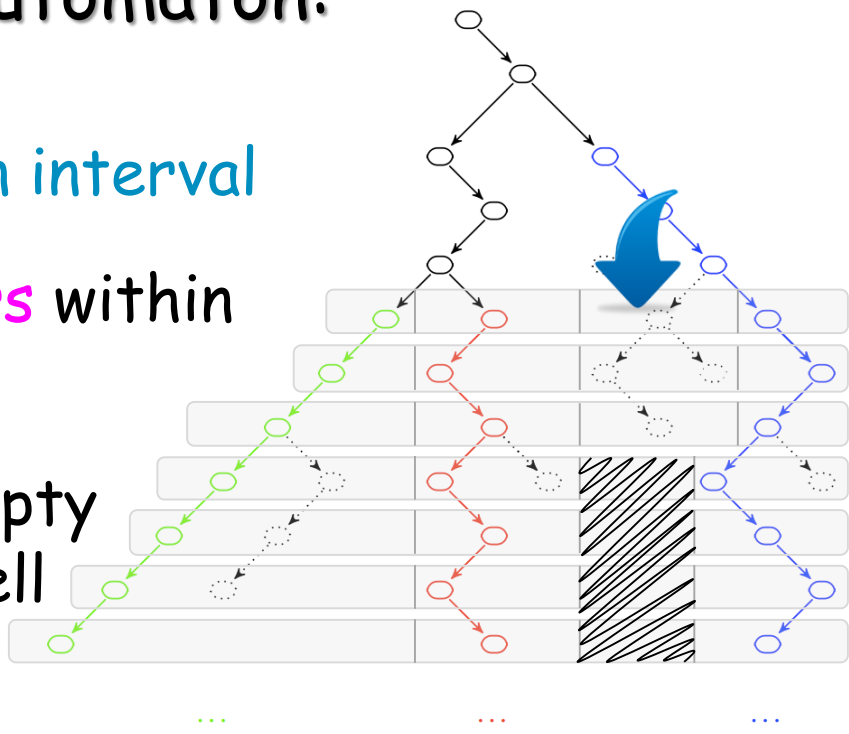
- Intervals that at the beginning contained no skeleton nodes will eventually become empty!

Deciding if width < k (const.)

Fisman Lustig

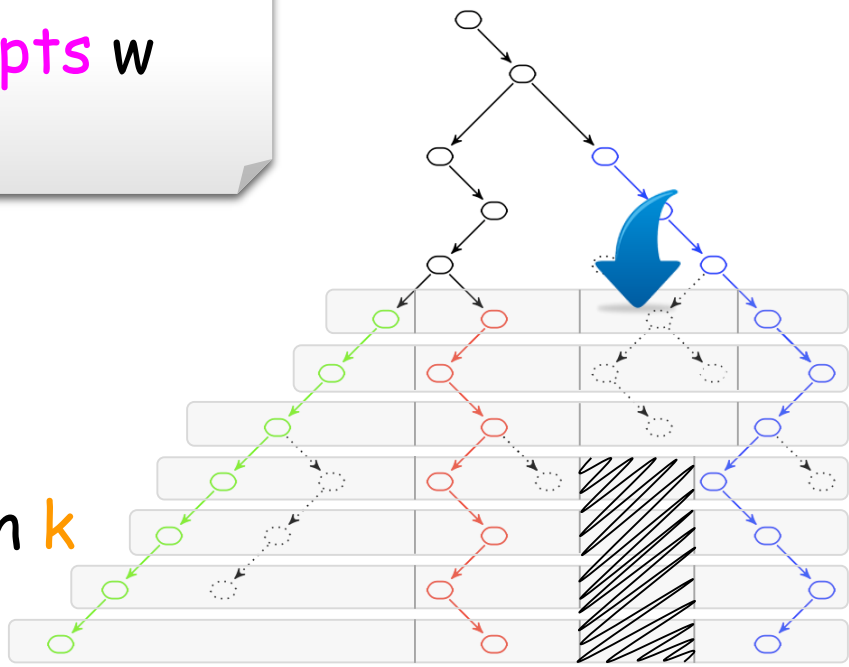
Build a deterministic Büchi automaton:

- **Shred**: put each node in its own interval
- **Develop**: develop the successors within the intervals
- **If** there are less than k non-empty intervals: Shred and ring the bell



Deciding if width < k

Theorem:
The constructed DBW accepts w iff width(w) < k



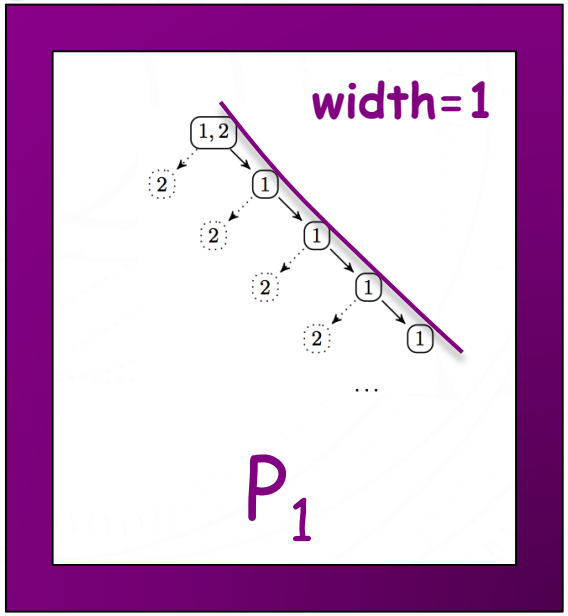
Proof:

- If the width is less than k:
  - Inf. often we'll have less than k intervals and ring the bell
- If the width is k or more, say k*
  - Eventually we will have k* non-empty intervals forever long (and we'll stop ringing the bell)

Claim:
When width=k each skeleton-path gets trapped in its own interval!

Mission

First



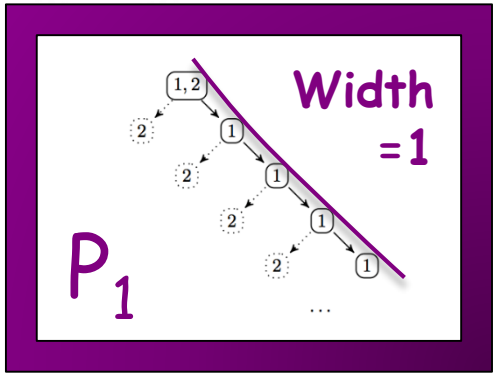
afterwards

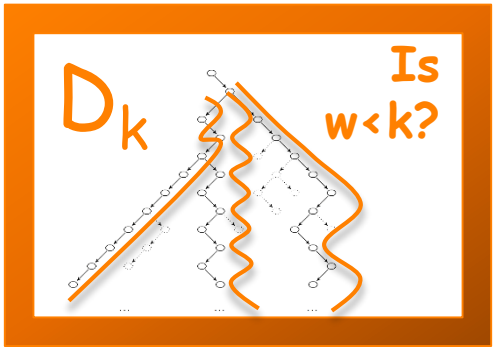
Finally

then

Determinizing for Width = k

- Idea:
  - Combine the construction for determinizing when width =1 (MHoRD)
  - And the construction for deciding if the width is < k





Determinizing for Width = k

Fisman Lustig

Build a deterministic Parity automaton:

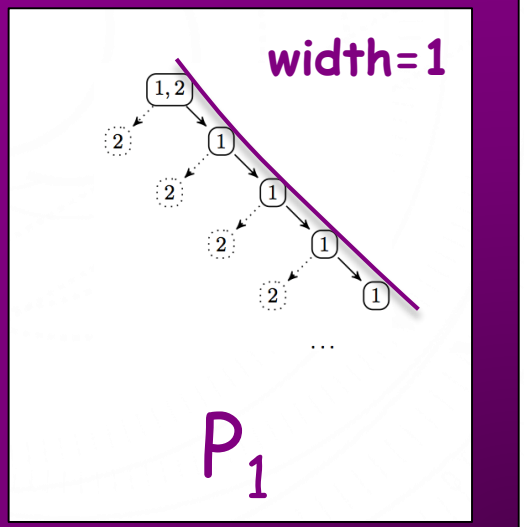
- Compute the next slice of RD
- Annotate with visited/owes
- If less than k intervals – shred and buzz the buzzer
- If in some interval all visited – reset and ring the bell

Theorem:

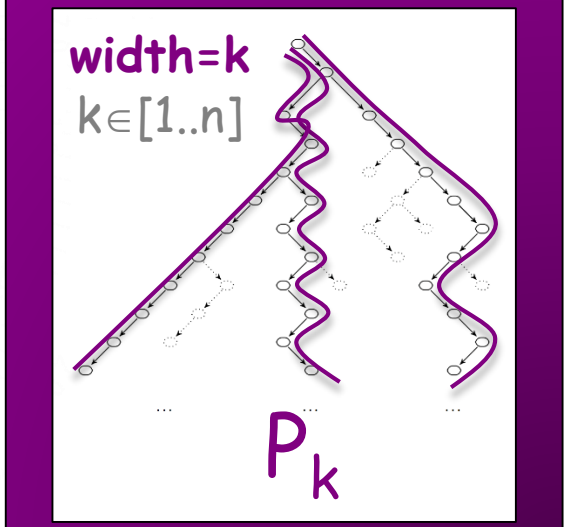
Given the width is k, the constructed DPW accepts w iff the given NBW accepts it.

Mission

First

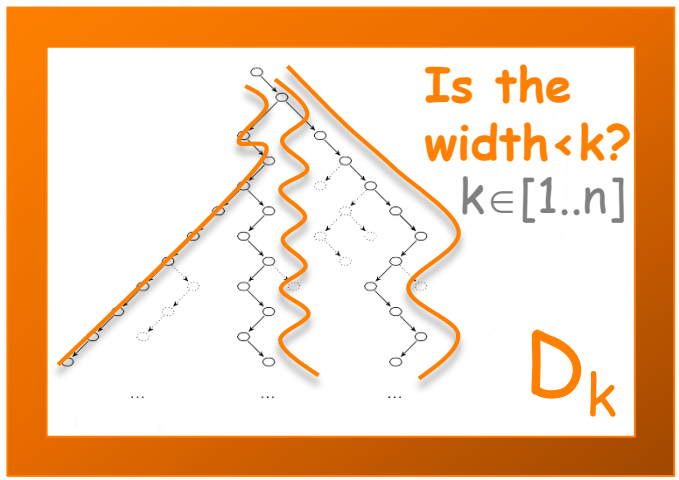


afterwards



Finally

then

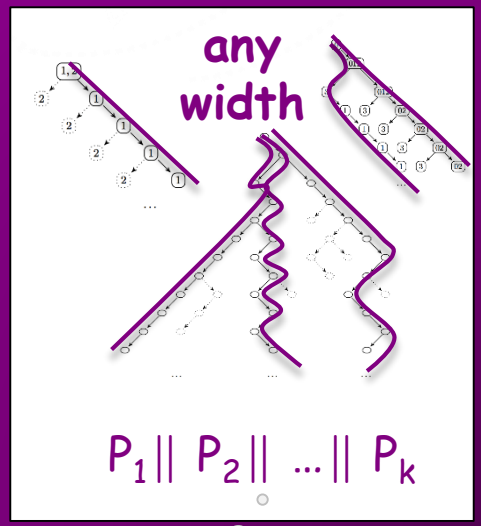


Mission

First

afterwards

Finally



then

How do we retrieve the correct answer?

Determinzing for Any Width

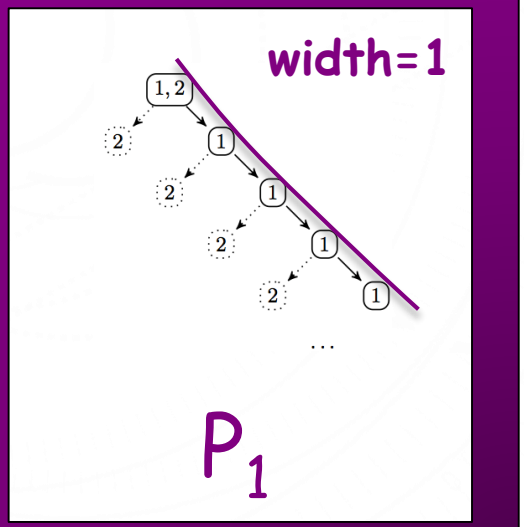
- Run the automata for all the widths in parallel
- Automaton for width =k uses the k-buzzer and the k-bell
- Proof: Assume the width is k*. Then
  - The automaton for width =k* answers correctly: rings the k-bell inf. often iff the NBA accepts (and it doesn't shred inf. often)
  - The automata for width >k* shred (i.e. >k-buzz) inf. often
  - The automata for width <k* don't shred (buzz) i.o. but may reset (<k-ring). If so that is correct –all the paths trapped in one of the intervals accept.

THM:

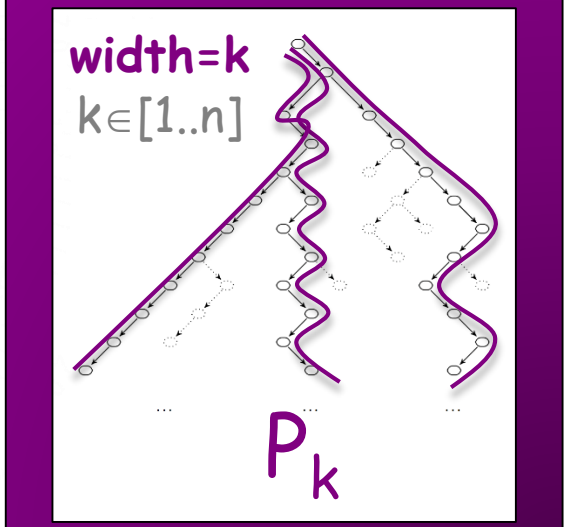
The DPA accepts w **iff** the NBA accepts w

Mission

First

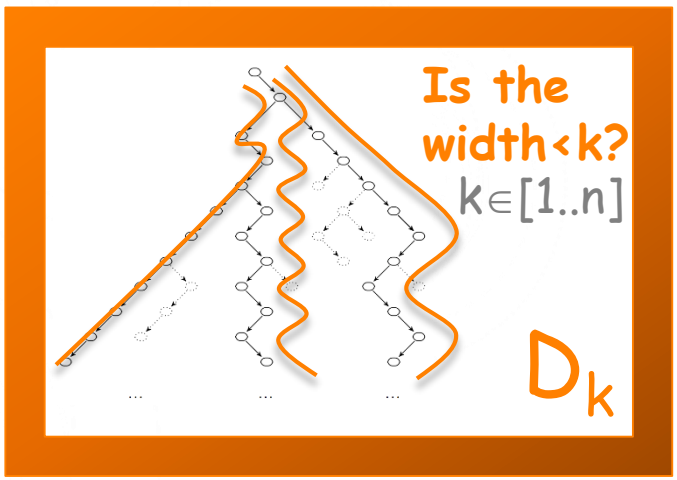


afterwards



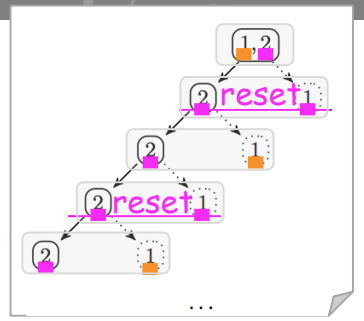
Finally

then

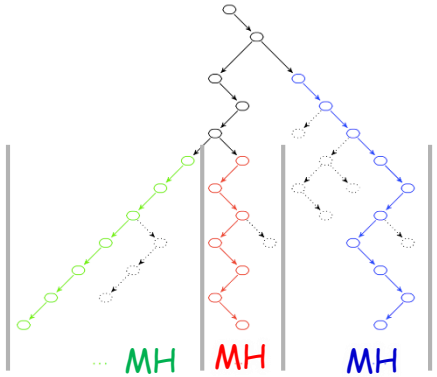


Recap

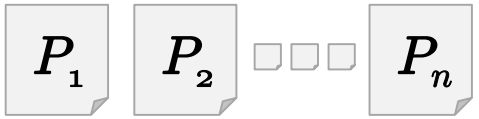
- Determinization for width=1 can be done by applying MH on the RD



- Determinization for width=k is done by trapping the presumably k skeleton-paths each in its own interval and applying MH on each interval separately



- Determinization for any width is obtained by running all width in parallel



- Complexity remains $n^{O(n)}$ (skipped in this talk)

Thank you for your attention!

Questions/ Comments?

Dana Fisman and Yoad Lustig