Tutorial

Graph Decompositions and the Verification of Concurrent Recursive Programs II

K Narayan Kumar

Chennai Mathematical Institute, India.

IMS, Singapore, August 2016

Why Tree-width?

Why Tree-width?

Corollary to Seese's Theorem:
If C is any MSO definable family of graphs then, for any k, checking MSO satisfiability among graphs in C with tree-width at most k is decidable.

Applying Seese's Theorem

* Nested words are an MSO definable class with tree-width 2

* CBMs with at most k contexts is a MSO definable class with tree-width k+1

Applying Seese's Theorem

* Nested words are an MSO definable class with tree-width 2

> The MSO theory of nested words is decidable.

* CBMs with at most k contexts is a MSO definable class with tree-width k+1

Applying Seese's Theorem

* Nested words are an MSO definable class with tree-width 2

> The MSO theory of nested words is decidable.

* CBMs with at most k contexts is a MSO definable class with tree-width k+1

> The MSO theory of bounded-context CBMs is decidable

ENCYCLOPEDIA OF MATHEMATICS AND ITS APPLICATIONS

Graph Structure and Monadic Second-Order Logic

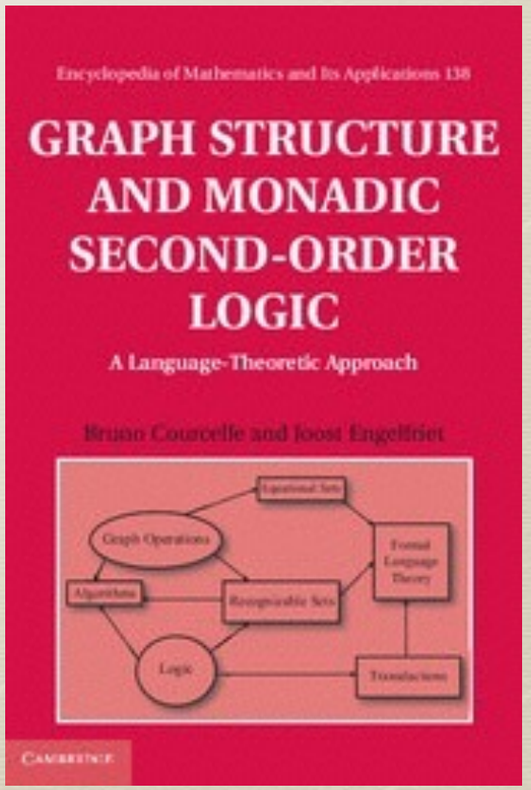
A Language-Theoretic Approach

BRUNO COURCELLE

Université de Bordeaux

JOOST ENGELFRIET

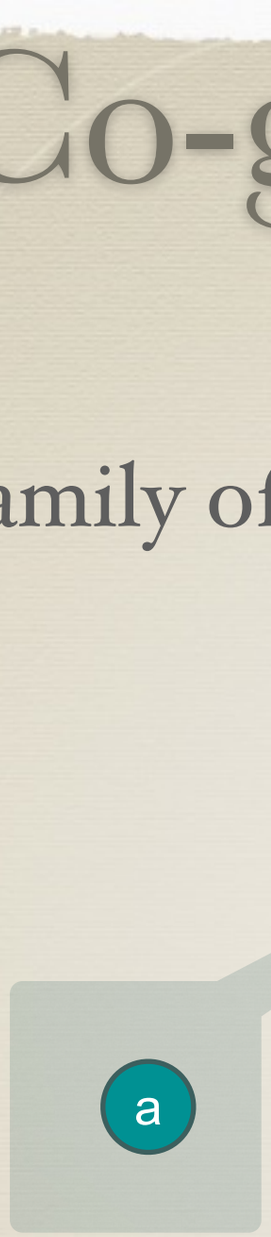
Universiteit Leiden



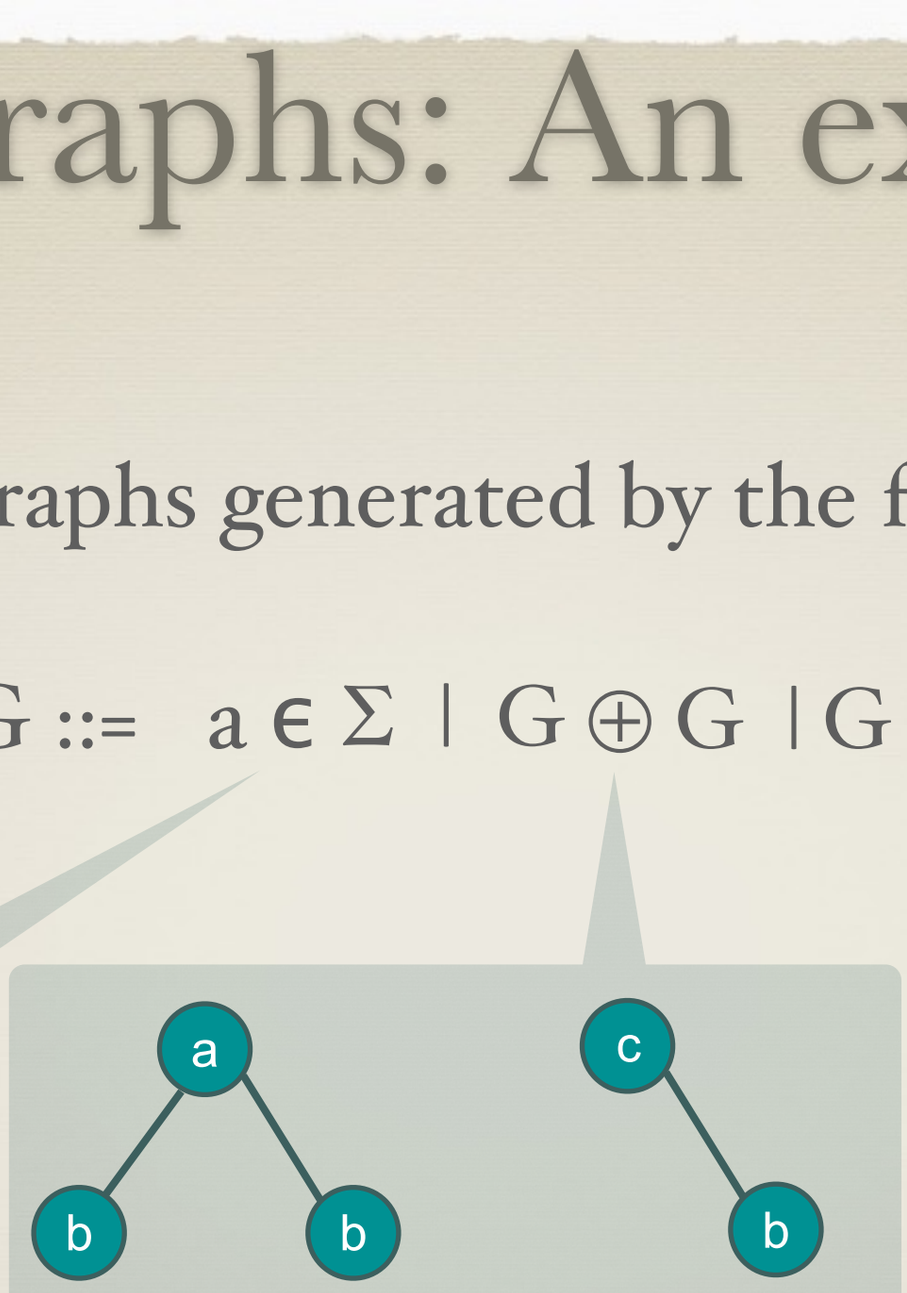
Co-graphs: An example

* Family of graphs generated by the following algebra:

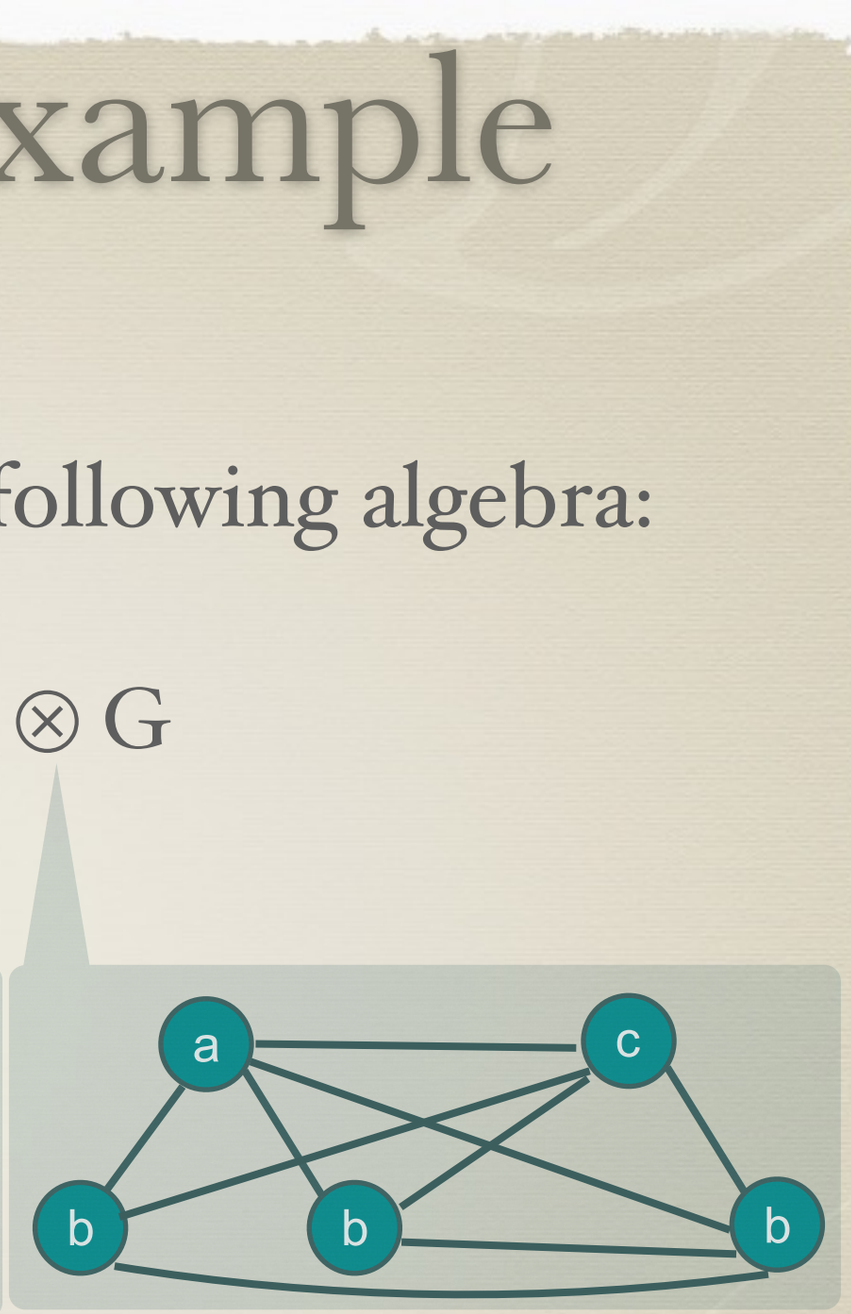
$$G ::= a \in \Sigma \mid G \oplus G \mid G \otimes G$$



Single vertex labelled a

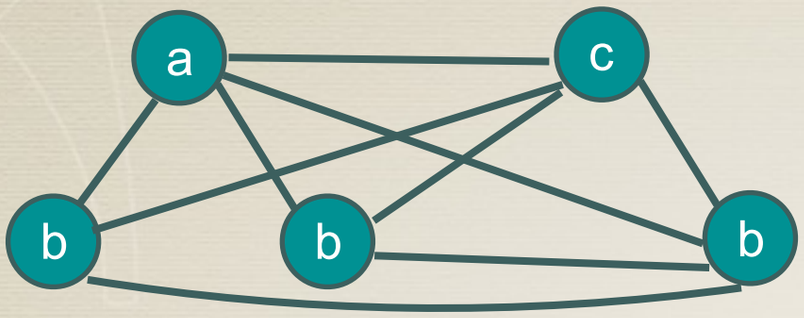


Disjoint union

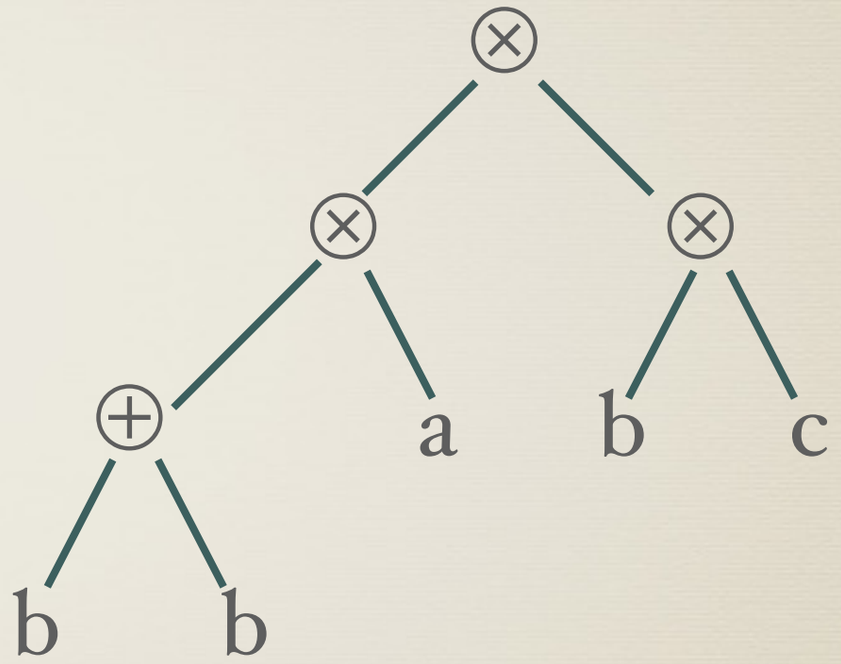


Disjoint union and connect all pairs

Interpretation on Trees

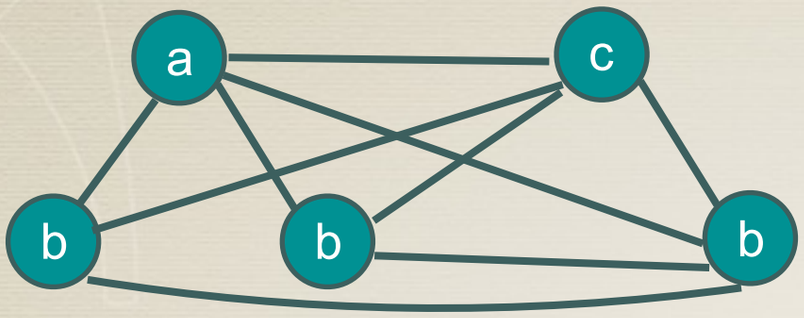


$$((b \oplus b) \otimes a) \otimes (b \otimes c)$$

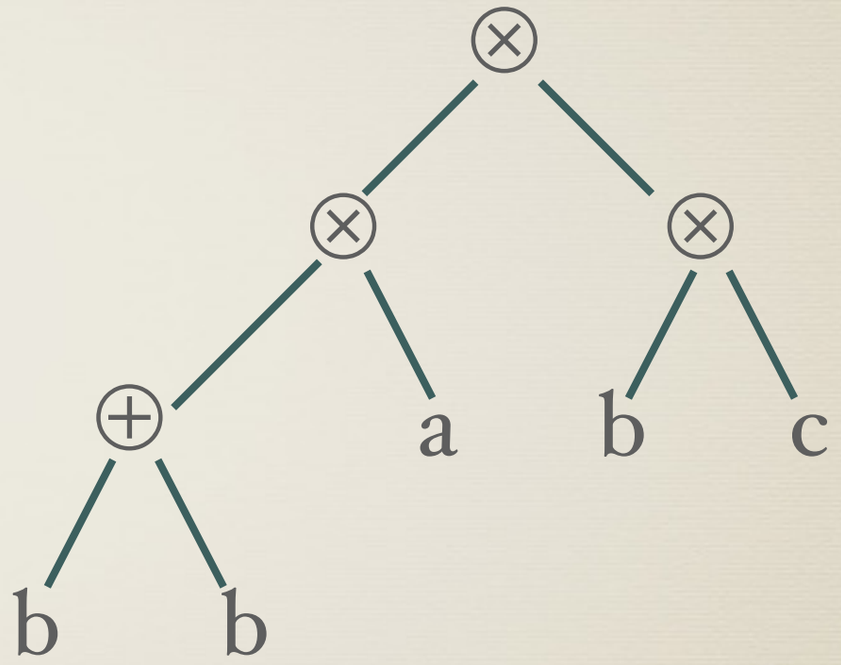


Graph	Tree
There is a vertex x	There is a leaf x
There is a set of vertices X	There is a set of leaves X
a(x)	a(x)
E(x,y)	There is path from x to y whose highest node is a ⊗

Interpretation on Trees



$$((b \oplus b) \otimes a) \otimes (b \otimes c)$$



Graph	Tree
There is a vertex x	There is a leaf x
There is a set of vertices X	There is a set of leaves X
a(x)	a(x)
E(x,y)	There is path from x to y whose highest node is a ⊗

We translate a formula Φ over graphs labelled with Σ to a formula Φ_{tree} over trees labelled with $\Sigma \cup \{\oplus, \otimes\}$.

MSO decidability for Co-graphs

* The collection of trees labelled by $\Sigma \cup \{\oplus, \otimes\}$ that constitute valid co-graph expressions is a regular tree language.

  Expressible in MSO over trees (φ_{co})

A formula Φ in MSO over graphs is satisfiable over co-graphs
iff
the formula $\Phi_{tree} \wedge \varphi_{co}$ is satisfiable over trees.

The MSO theory of co-graphs is decidable.

Advantages of the algebraic approach

Advantages of the algebraic approach

* The tree interpretation is quite transparent.

Advantages of the algebraic approach

* The tree interpretation is quite transparent.
* Membership checking

Advantages of the algebraic approach

* The tree interpretation is quite transparent.
* Membership checking
* Are all paths co-graphs?

Advantages of the algebraic approach

* The tree interpretation is quite transparent.
* Membership checking
* Are all paths co-graphs?

Advantages of the algebraic approach

* The tree interpretation is quite transparent.
* Membership checking
* Are all paths co-graphs?
* What labels the root?

  $\otimes$ or $\oplus$

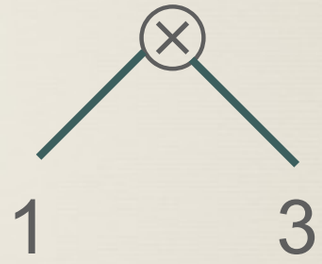
Advantages of the algebraic approach

* The tree interpretation is quite transparent.
* Membership checking
* Are all paths co-graphs?
* What labels the root?

Advantages of the algebraic approach

* The tree interpretation is quite transparent.
* Membership checking
* Are all paths co-graphs?

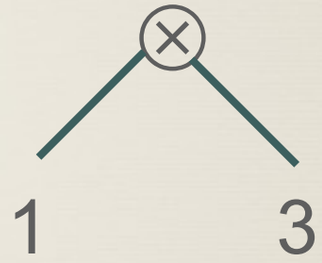
* What labels the root?



Advantages of the algebraic approach

* The tree interpretation is quite transparent.
* Membership checking
* Are all paths co-graphs?

* What labels the root?



No vertex with degree 3!

Advantages of the algebraic approach

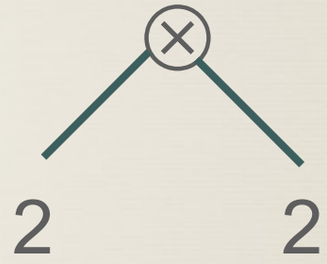
* The tree interpretation is quite transparent.
* Membership checking
* Are all paths co-graphs?

* What labels the root?

Advantages of the algebraic approach

* The tree interpretation is quite transparent.
* Membership checking
* Are all paths co-graphs?

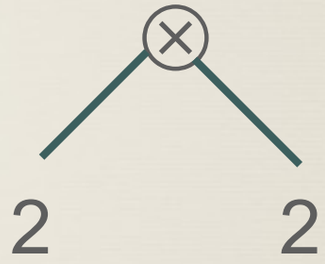
* What labels the root?



Advantages of the algebraic approach

* The tree interpretation is quite transparent.
* Membership checking
* Are all paths co-graphs?

* What labels the root?



No 2 x 2 perfect matching!

Advantages of the algebraic approach

* The tree interpretation is quite transparent.
* Membership checking
* Are all paths co-graphs?

* What labels the root?

Advantages of the algebraic approach

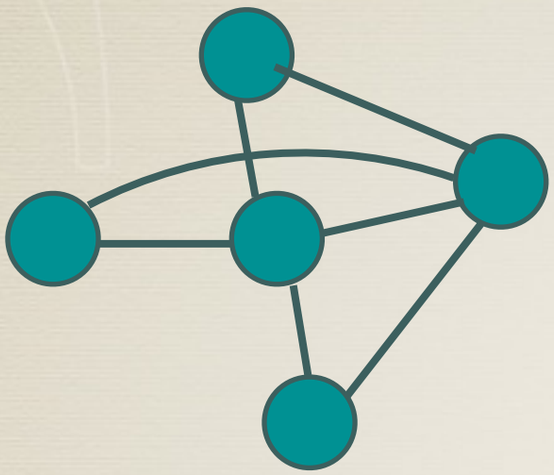
* The tree interpretation is quite transparent.
* Membership checking
* Are all paths co-graphs?

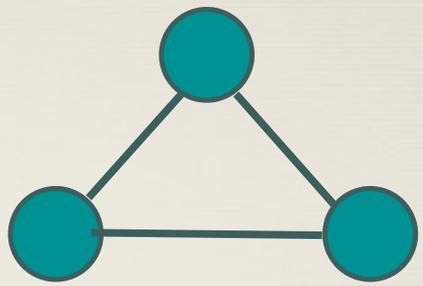
* What labels the root?

The path of length 3 is not a co-graph

Using the Co-graph Algebra

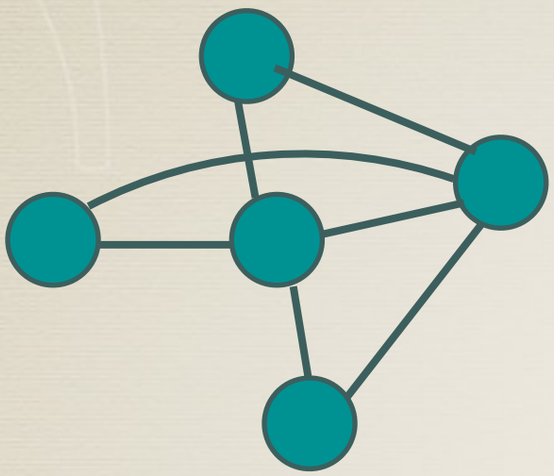
Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

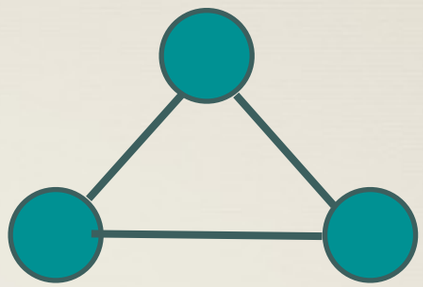




Using the Co-graph Algebra

Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

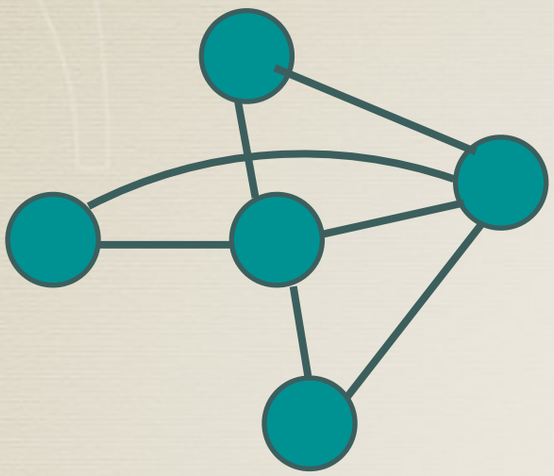


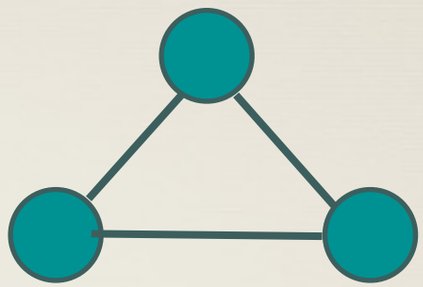


Is the MSO theory of C decidable?

Using the Co-graph Algebra

Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }



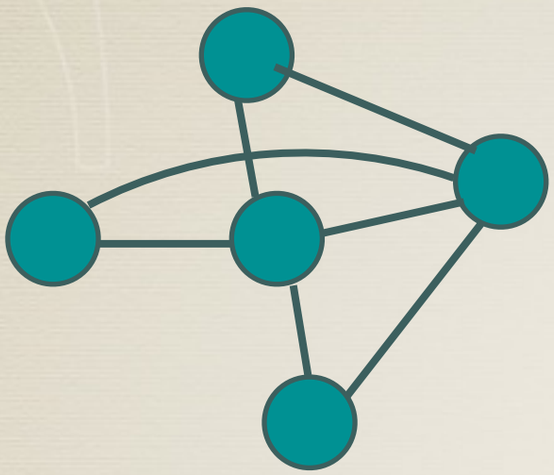


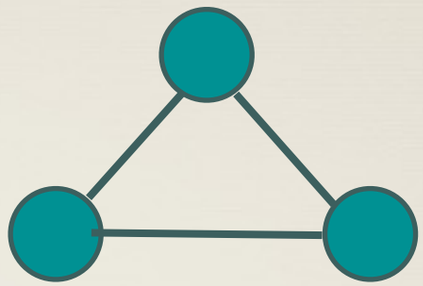
Is the MSO theory of C decidable?

The class is certainly MSO definable.

Using the Co-graph Algebra

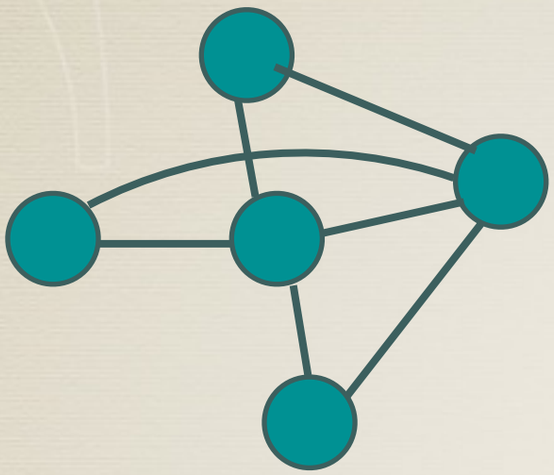
Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

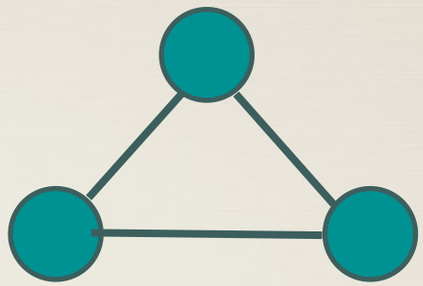




Using the Co-graph Algebra

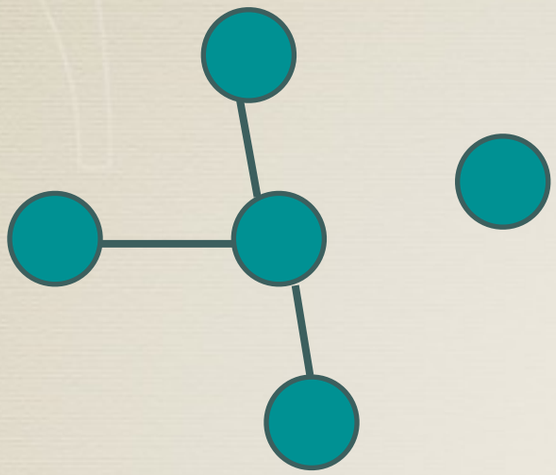
Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

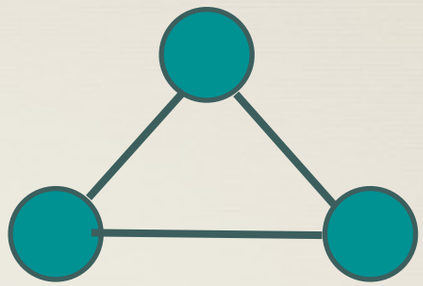




Using the Co-graph Algebra

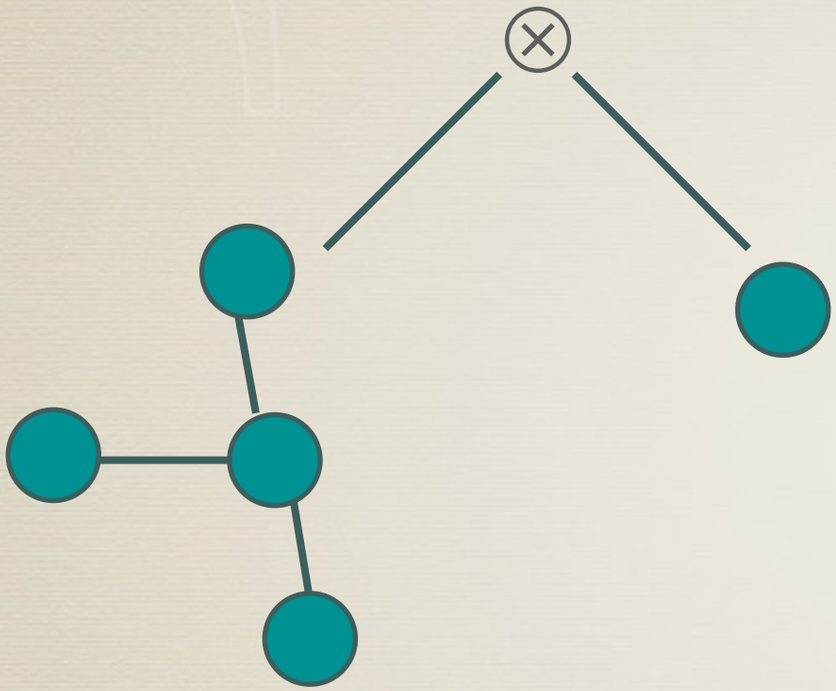
Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

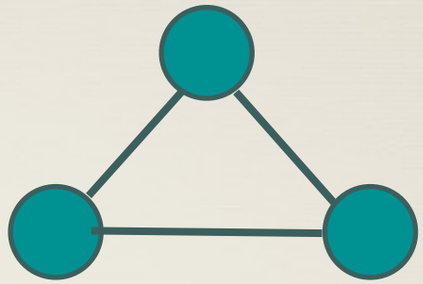




Using the Co-graph Algebra

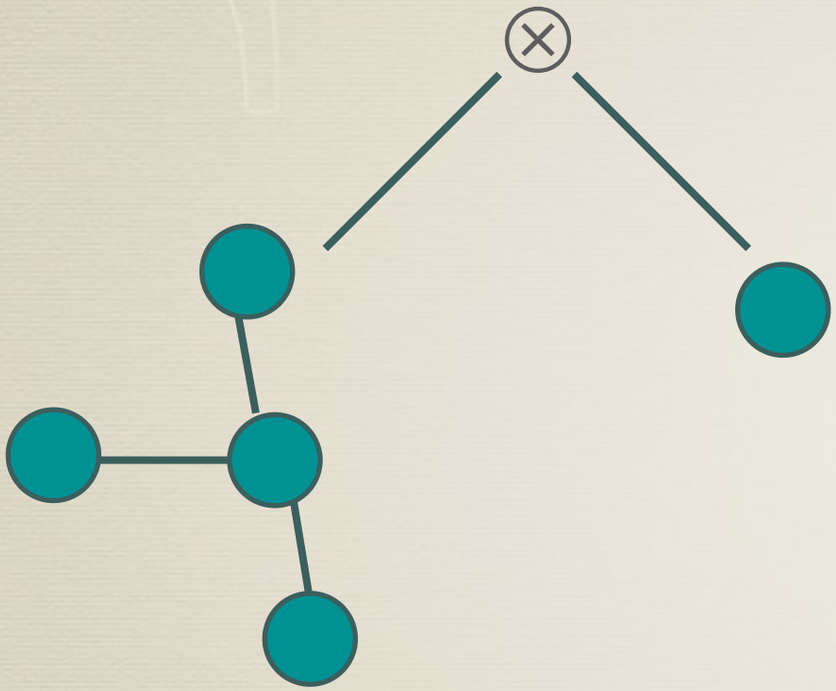
Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

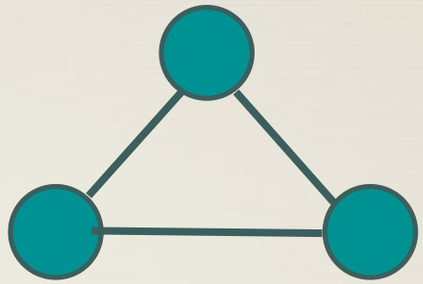




Using the Co-graph Algebra

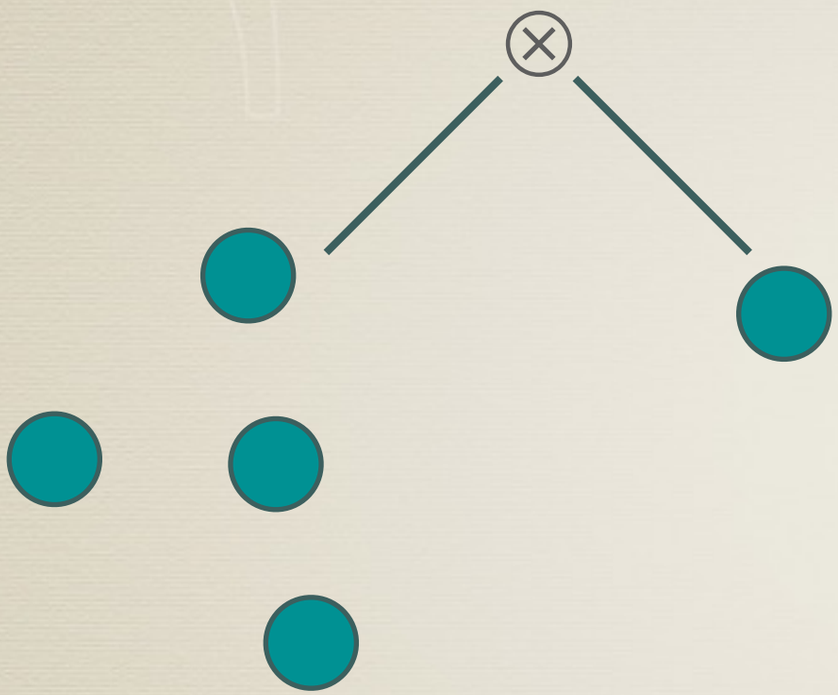
Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

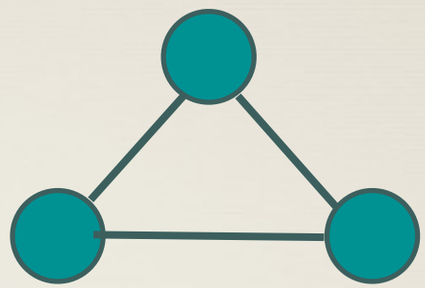




Using the Co-graph Algebra

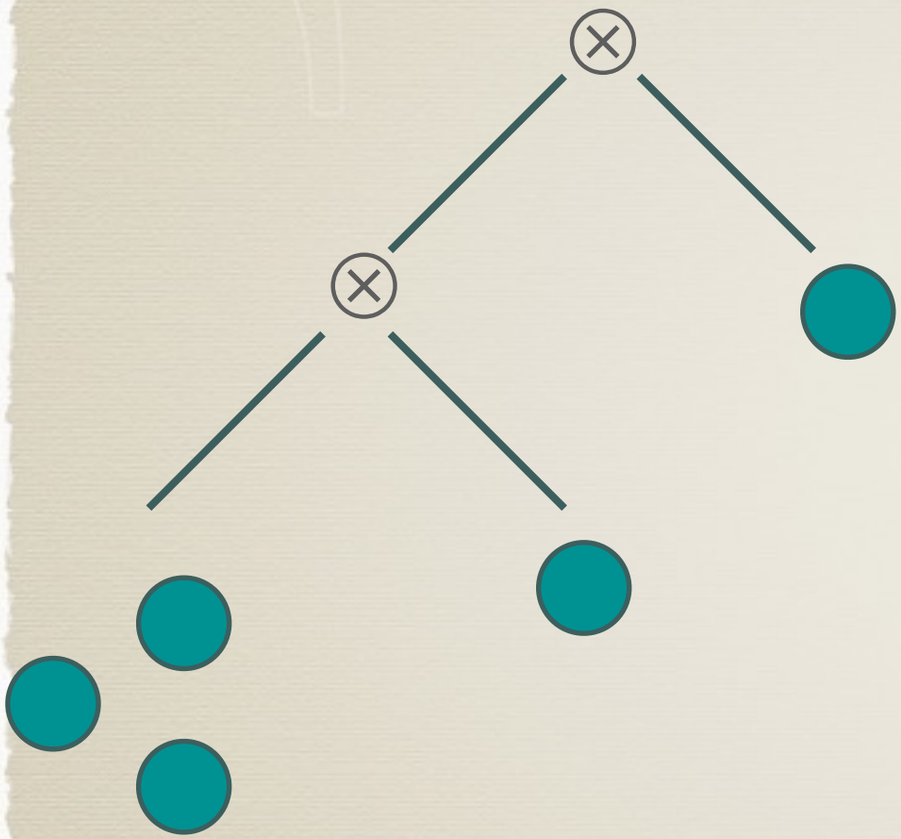
Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

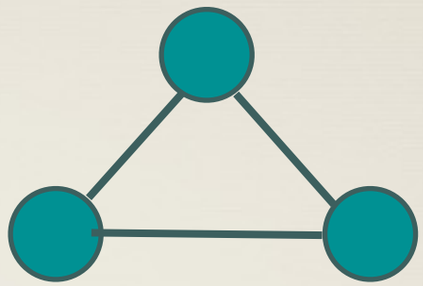




Using the Co-graph Algebra

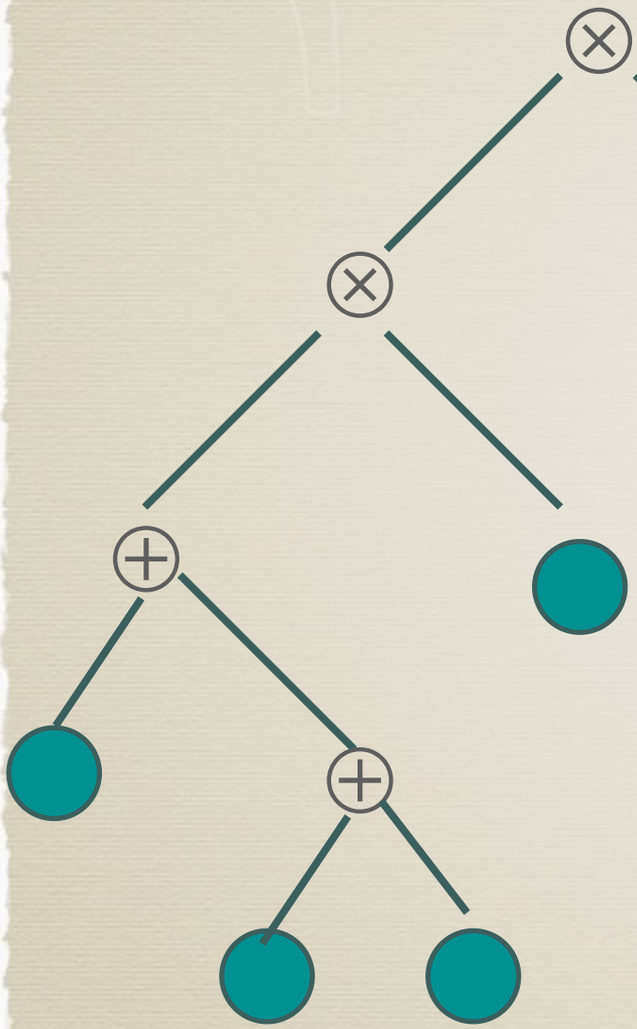
Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

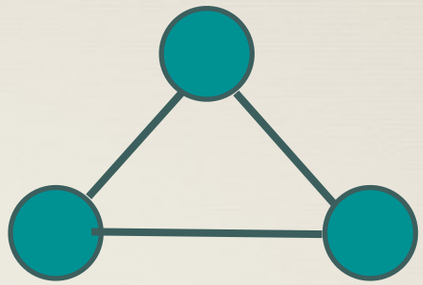




Using the Co-graph Algebra

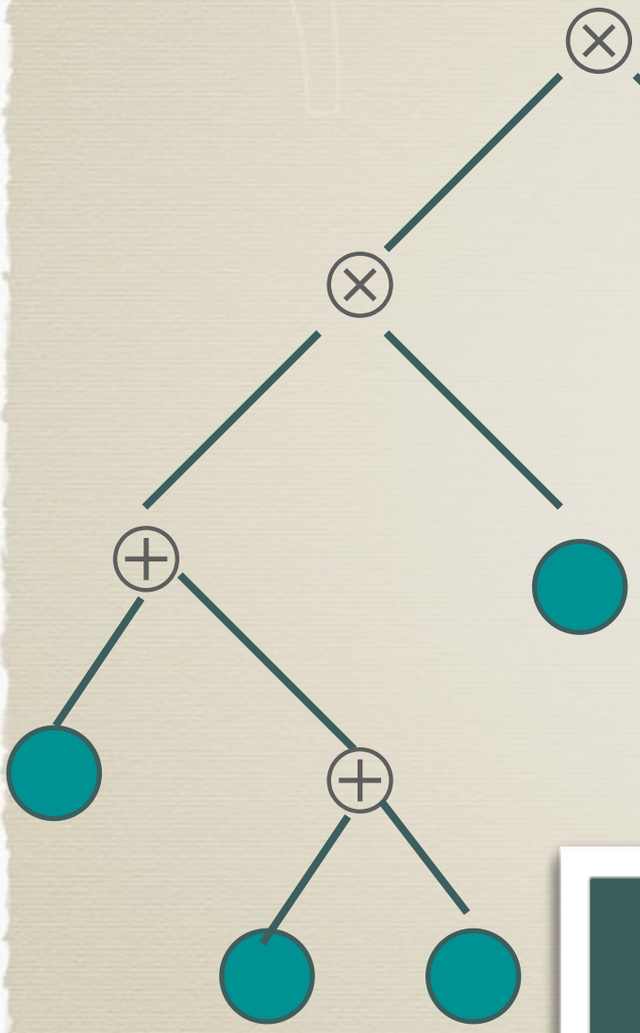
Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

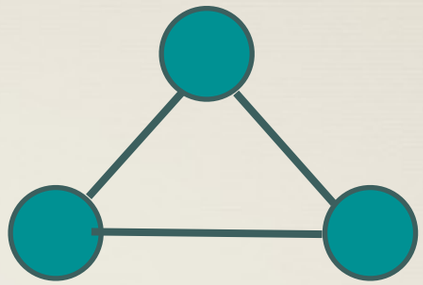




Using the Co-graph Algebra

Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }





Every graph in C is a co-graph.

Using the Co-graph Algebra

Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

Using the Co-graph Algebra

Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

Size one graphs in C are co-graphs.

Using the Co-graph Algebra

Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

Size one graphs in C are co-graphs.

Case 1: If the graph is not connected then we inductively construct expressions for each part.

Using the Co-graph Algebra

Let C = { G | {u,v} is an edge then N(u) $\subseteq$N(v) or N(v)$\subseteq$N(u) }

Size one graphs in C are co-graphs.

Case 1: If the graph is not connected then we inductively construct expressions for each part.

Case 2: If the graph is connected. $\otimes$ at the top.

Using the Co-graph Algebra

Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

Size one graphs in C are co-graphs.

Case 1: If the graph is not connected then we inductively construct expressions for each part.

Case 2: If the graph is connected. ⊗ at the top.

> Divide it into two parts so that the complete bipartite graph on the division is a subgraph

Using the Co-graph Algebra

Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

Case 2: If the graph is connected. ⊗ at the top.

Using the Co-graph Algebra

Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

Case 2: If the graph is connected. ⊗ at the top.

u be a maximal degree vertex.

Using the Co-graph Algebra

Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

Case 2: If the graph is connected. ⊗ at the top.

- u be a maximal degree vertex.
- v be the closest vertex that is not a neighbour

Using the Co-graph Algebra

Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

Case 2: If the graph is connected. ⊗ at the top.

- u be a maximal degree vertex.
- v be the closest vertex that is not a neighbour

Therefore, we have

Using the Co-graph Algebra

Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

Case 2: If the graph is connected. ⊗ at the top.

- u be a maximal degree vertex.
- v be the closest vertex that is not a neighbour

Therefore, we have

N(u) ⊇ N(p) and so v in N(p).

u is a neighbour of every vertex in G

Using the Co-graph Algebra

Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

Case 2: If the graph is connected. ⊗ at the top.

- u be a maximal de
- v [illegible] a neighbour

There[illegible], we have

N(u) ⊇ N(p) and so v in N(p).

u is a neighbour of every vertex in G

Every graph in C is a co-graph. MSO theory of C is decidable

Using the Co-graph Algebra

Let C = { G | {u,v} is an edge then N(u) ⊆N(v) or N(v)⊆N(u) }

Case 2: If the graph is connected. ⊗ at the top.

- u be a maximal degree
- v [illegible] neighbour

There [illegible] have

N(u) ⊇ N(p) and so v in N(p).

u is a neighbour of every vertex in G

MSO theory of quasi-threshold graphs is decidable.

Advantages of the algebraic approach

Advantages of the algebraic approach

* The tree interpretation is quite transparent.

Advantages of the algebraic approach

* The tree interpretation is quite transparent.

* Helps in establishing membership/containment in class.

Advantages of the algebraic approach

* The tree interpretation is quite transparent.

* Helps in establishing membership/containment in class.

* Fewer or less powerful operators might help.

Advantages of the algebraic approach

* The tree interpretation is quite transparent.

* Helps in establishing membership/containment in class.

* Fewer or less powerful operators might help.
    * For quasi-threshold graphs, our search was guided by the limited set of operations available.

Clique-width

Clique-width

* A generalisation of the co-graph algebra.

Clique-width

* A generalisation of the co-graph algebra.
    * Vertices are coloured.

Clique-width

* A generalisation of the co-graph algebra.
  * Vertices are coloured.
  * Disjoint Union

Clique-width

* A generalisation of the co-graph algebra.
  * Vertices are coloured.
  * Disjoint Union
  * Add edges between all vertices coloured A with all vertices coloured B

Clique-width

* A generalisation of the co-graph algebra.
  * Vertices are coloured.
  * Disjoint Union
  * Add edges between all vertices coloured A with all vertices coloured B
  * Relabel a colour.

Clique-width

* A generalisation of the co-graph algebra.
  * Vertices are coloured.
  * Disjoint Union
  * Add edges between all vertices coloured A with all vertices coloured B
  * Relabel a colour.

The smallest number of colours needed to describe a graph in the CW algebra.

Clique-width

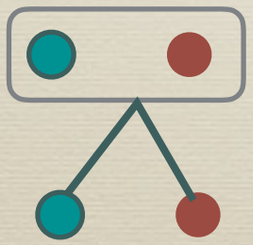
* A generalisation of the co-graph algebra.
  * Vertices are coloured.
  * Disjoint Union
  * Add edges between all
    vertices col
* Relabel a col

Co-graphs are exactly the class of graphs with clique-width = 2

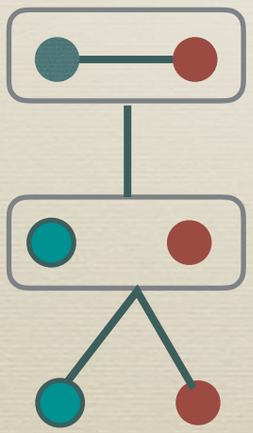
The smallest number of colours needed to describe a graph in the CW algebra.

* Vertices are coloured.
* Disjoint Union
* Add edges between all vertices coloured A with all vertices coloured B
* Relabel a colour.

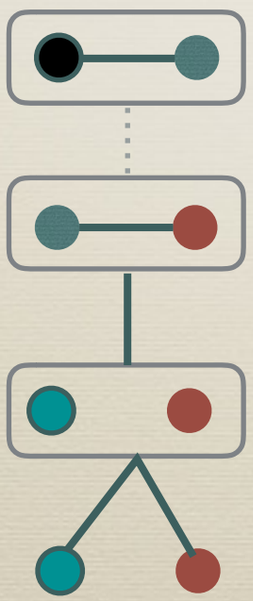
* Vertices are coloured.
* Disjoint Union
* Add edges between all vertices coloured A with all vertices coloured B
* Relabel a colour.



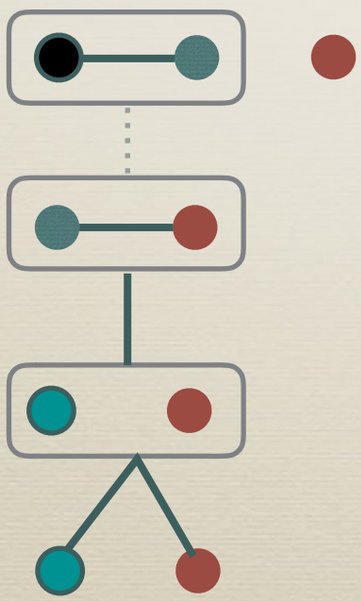
* Vertices are coloured.
* Disjoint Union
* Add edges between all vertices coloured A with all vertices coloured B
* Relabel a colour.



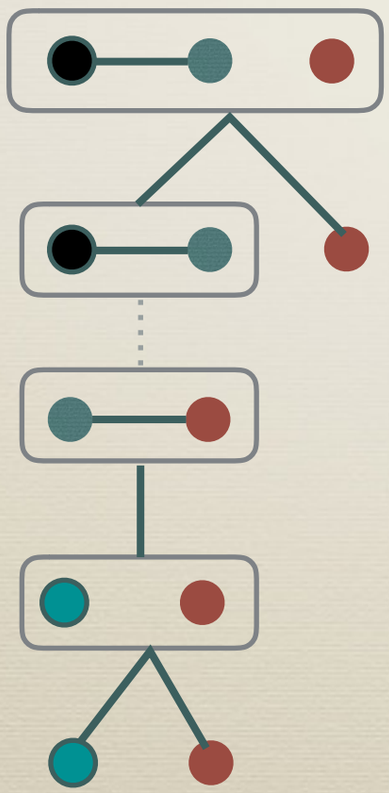
* Vertices are coloured.
* Disjoint Union
* Add edges between all vertices coloured A with all vertices coloured B
* Relabel a colour.

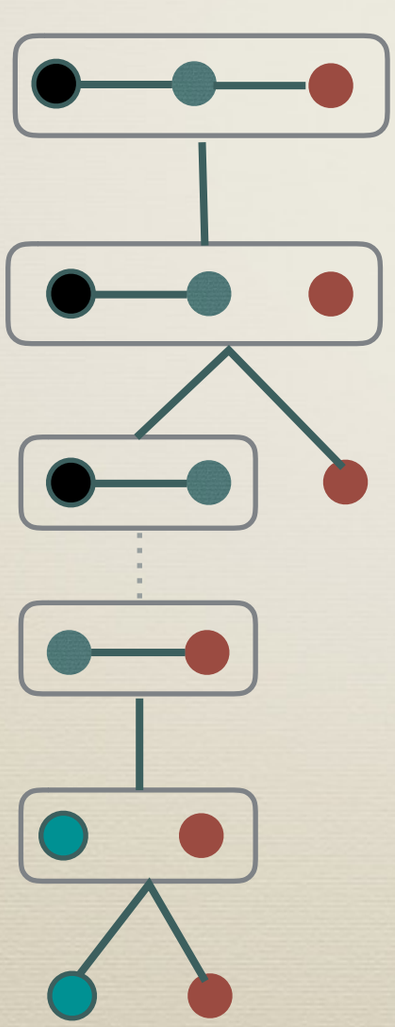


* Vertices are coloured.
* Disjoint Union
* Add edges between all vertices coloured A with all vertices coloured B
* Relabel a colour.

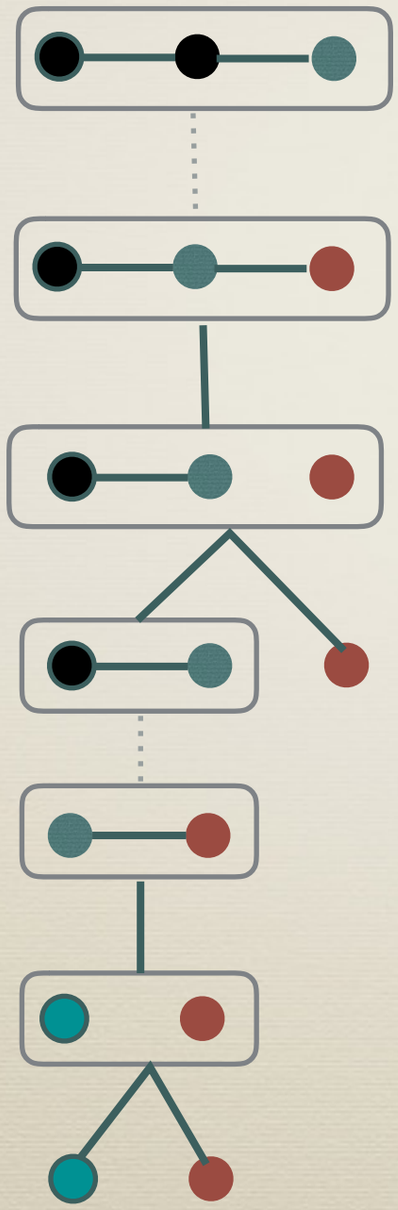


* Vertices are coloured.
* Disjoint Union
* Add edges between all vertices coloured A with all vertices coloured B
* Relabel a colour.

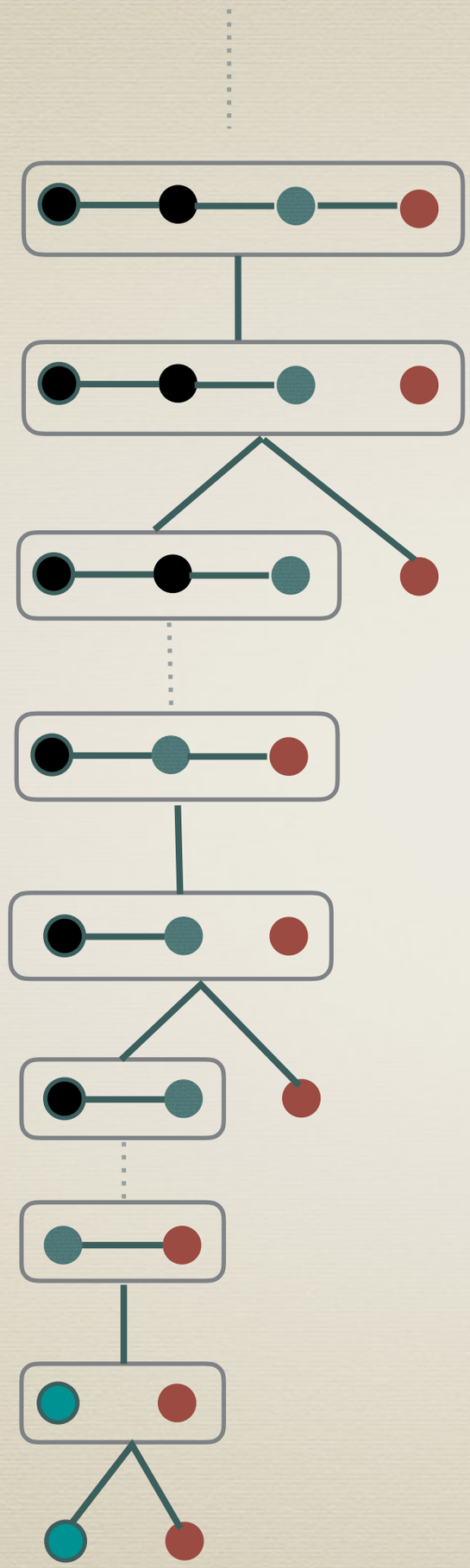




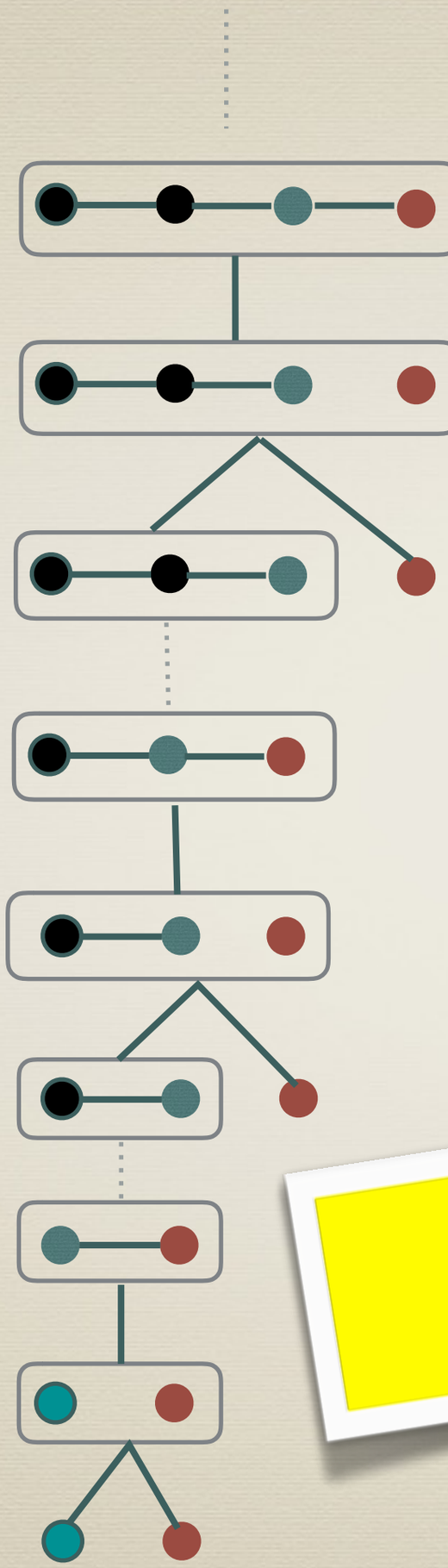
* Vertices are coloured.
* Disjoint Union
* Add edges between all vertices coloured A with all vertices coloured B
* Relabel a colour.



* Vertices are coloured.
* Disjoint Union
* Add edges between all vertices coloured A with all vertices coloured B
* Relabel a colour.



* Vertices are coloured.
* Disjoint Union
* Add edges between all vertices coloured A with all vertices coloured B
* Relabel a colour.



* Vertices are coloured.
* Disjoint Union
* Add edges between all vertices coloured A with all vertices coloured B
* Relabel a colour.

Paths have clique-width 3

Clique-width

Theorem:. (CourcelleOlariu'00)

1. For any k, checking MSO satisfiability among the class of graphs with clique-width at most k is decidable.

Clique-width

Theorem:. (CourcelleOlariu'00)

1. For any k, checking MSO satisfiability among the class of graphs with clique-width at most k is decidable.

* $CW(G) \leq 2^{O(TW(G))}$.

Clique-width

Theorem:. (CourcelleOlariu'00)

1. For any k, checking MSO satisfiability among the class of graphs with clique-width at most k is decidable.

* $CW(G) \leq 2^{O(TW(G))}$.

  * C is bounded tree-width => C is bounded clique-width

Clique-width

Theorem:. (CourcelleOlariu'00)

1. For any k, checking MSO satisfiability among the class of graphs with clique-width at most k is decidable.

* $CW(G) \leq 2^{O(TW(G))}$.
  * C is bounded tree-width => C is bounded clique-width
  * The family of cliques has clique-width 1 and unbounded tree-width

Clique-width

Theorem:. (CourcelleOlariu'00)

1. For any k, checking MSO satisfiability among the class of graphs with clique-width at most k is decidable.

* $CW(G) \leq 2^{O(TW(G))}$.
  * C is bounded tree-width => C is bounded clique-width
  * The family of cliques has clique-width 1 and unbounded tree-width

MSO decidability is via reduction to trees.

Advantages of the algebraic approach

Advantages of the algebraic approach

* The tree interpretation is quite transparent.

Advantages of the algebraic approach

* The tree interpretation is quite transparent.

* Helps in establishing membership/containment in class.

Advantages of the algebraic approach

* The tree interpretation is quite transparent.
* Helps in establishing membership/containment in class.
* Can be more expressive!

Equivalence for CBMs

Let C be a class of bounded degree MSO definable graphs.
TFAE

1. C has a decidable MSO theory
2. C can be interpreted in binary trees
3. C has bounded tree-width
4. C has bounded clique-width

Outline

* Models
* Behaviours
* Specifications
* Verification via graph decompositions
  * Split-width
* Conclusion

Split-width

* A way to decompose graphs to obtain a tree interpretation.
* Specifically designed for CBMs
* CBMs have bounded degree

Let C be MSO definable class of CBMs
TFAE

1. C has a decidable MSO theory
2. C can be interpreted in binary trees
3. C has bounded tree-width
4. C has bounded clique-width
5. C has bounded split-width

Split-width Operations

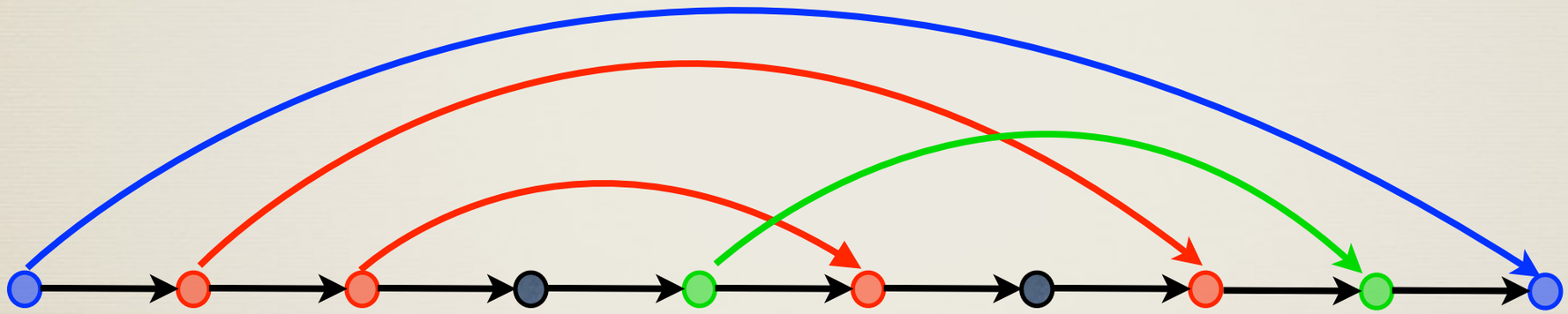
The Cut Operation:

Pick a set of process edges and delete them.

Split-width Operations

The Cut Operation:

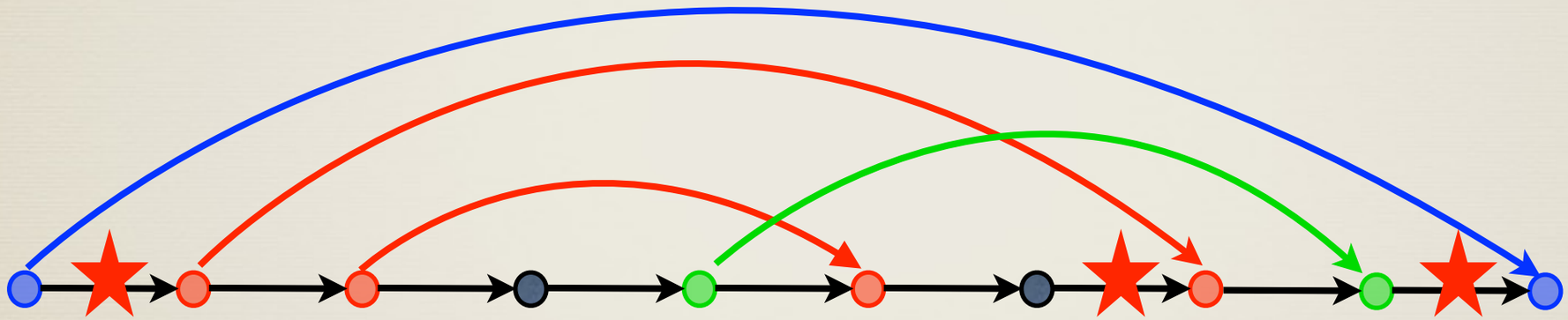
Pick a set of process edges and delete them.



Split-width Operations

The Cut Operation:

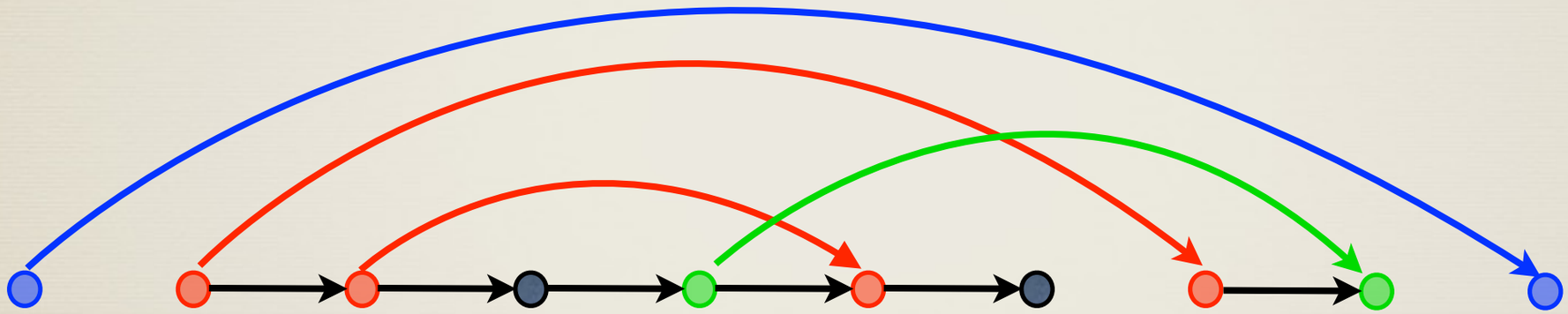
Pick a set of process edges and delete them.



Split-width Operations

The Cut Operation:

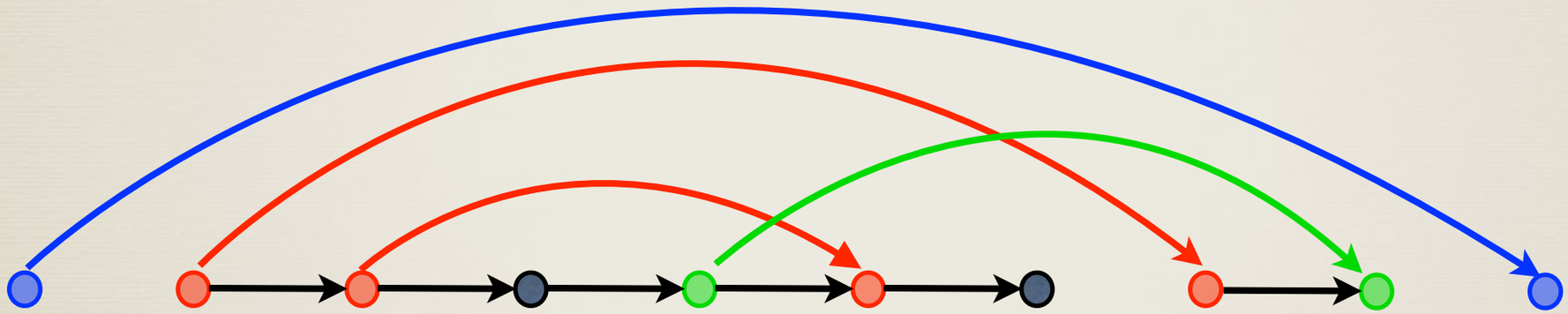
Pick a set of process edges and delete them.



Split-width Operations

The Cut Operation:

Pick a set of process edges and delete them.

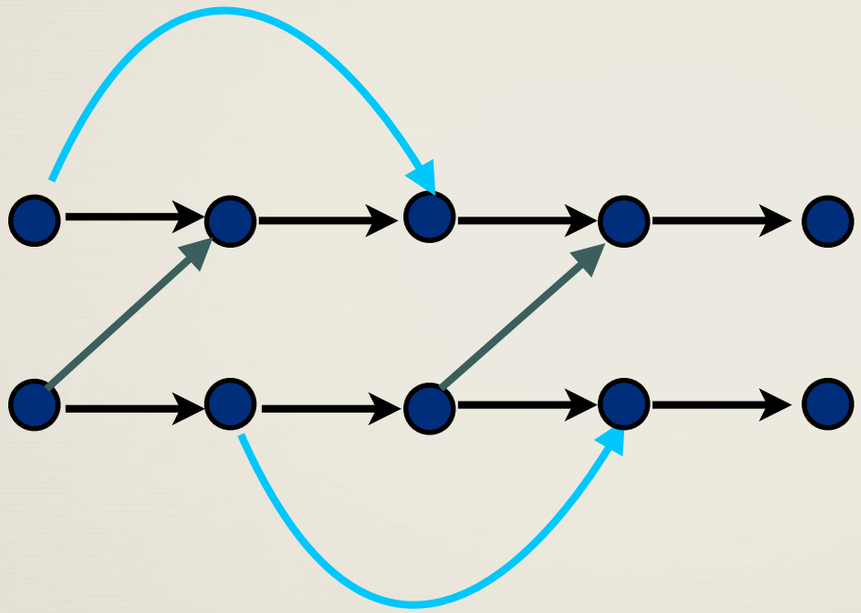


Split CBM with 3 holes

Split-width Operations

The Cut Operation:

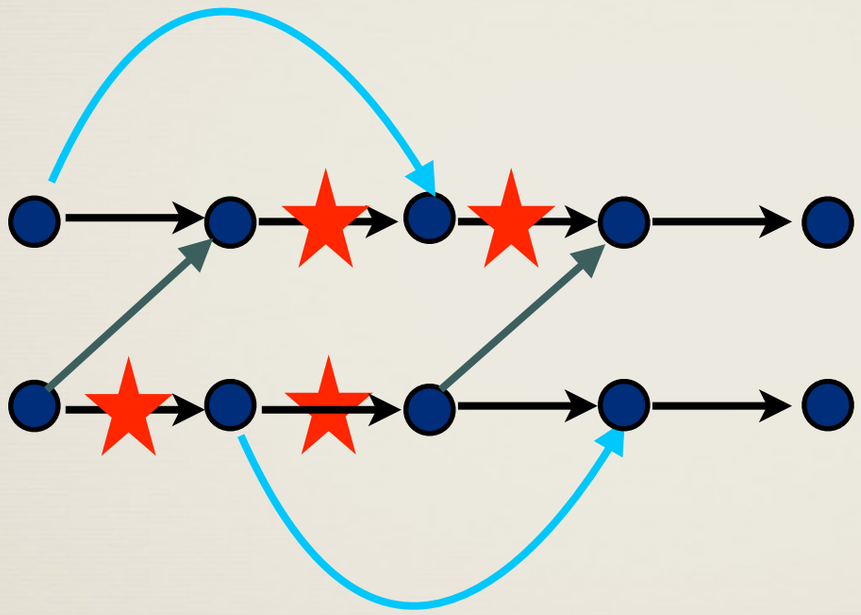
Pick a set of process edges and delete them.



Split-width Operations

The Cut Operation:

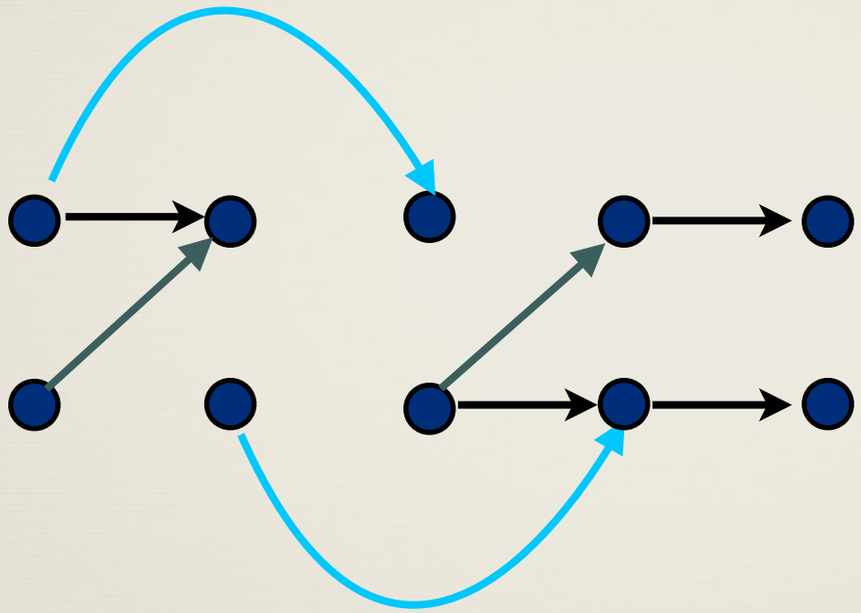
Pick a set of process edges and delete them.



Split-width Operations

The Cut Operation:

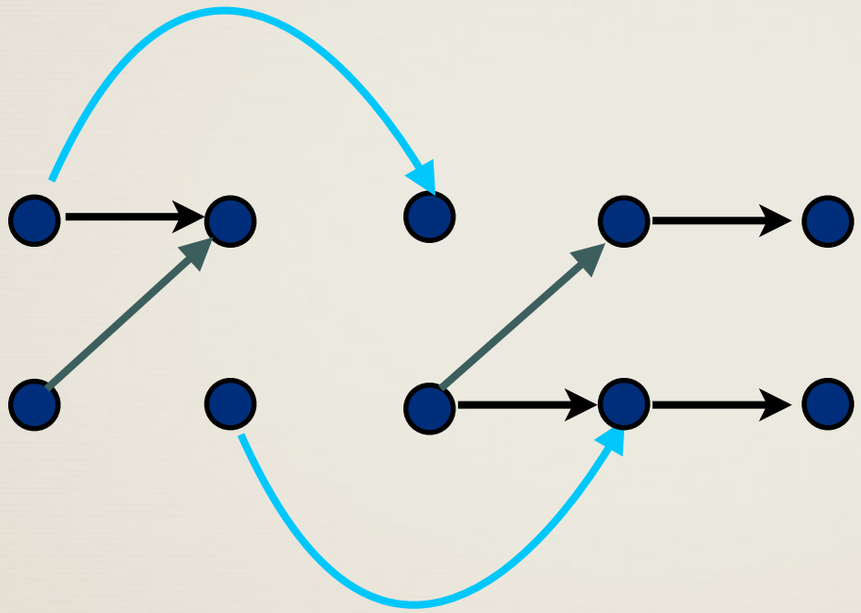
Pick a set of process edges and delete them.



Split-width Operations

The Cut Operation:

Pick a set of process edges and delete them.

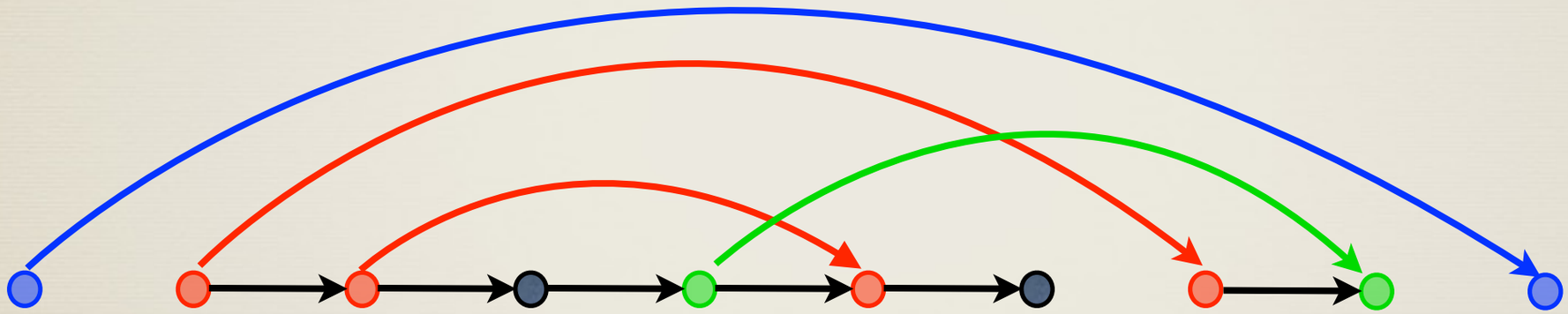


Split CBM with 4 holes

Split-width Operations

The Split Operation:

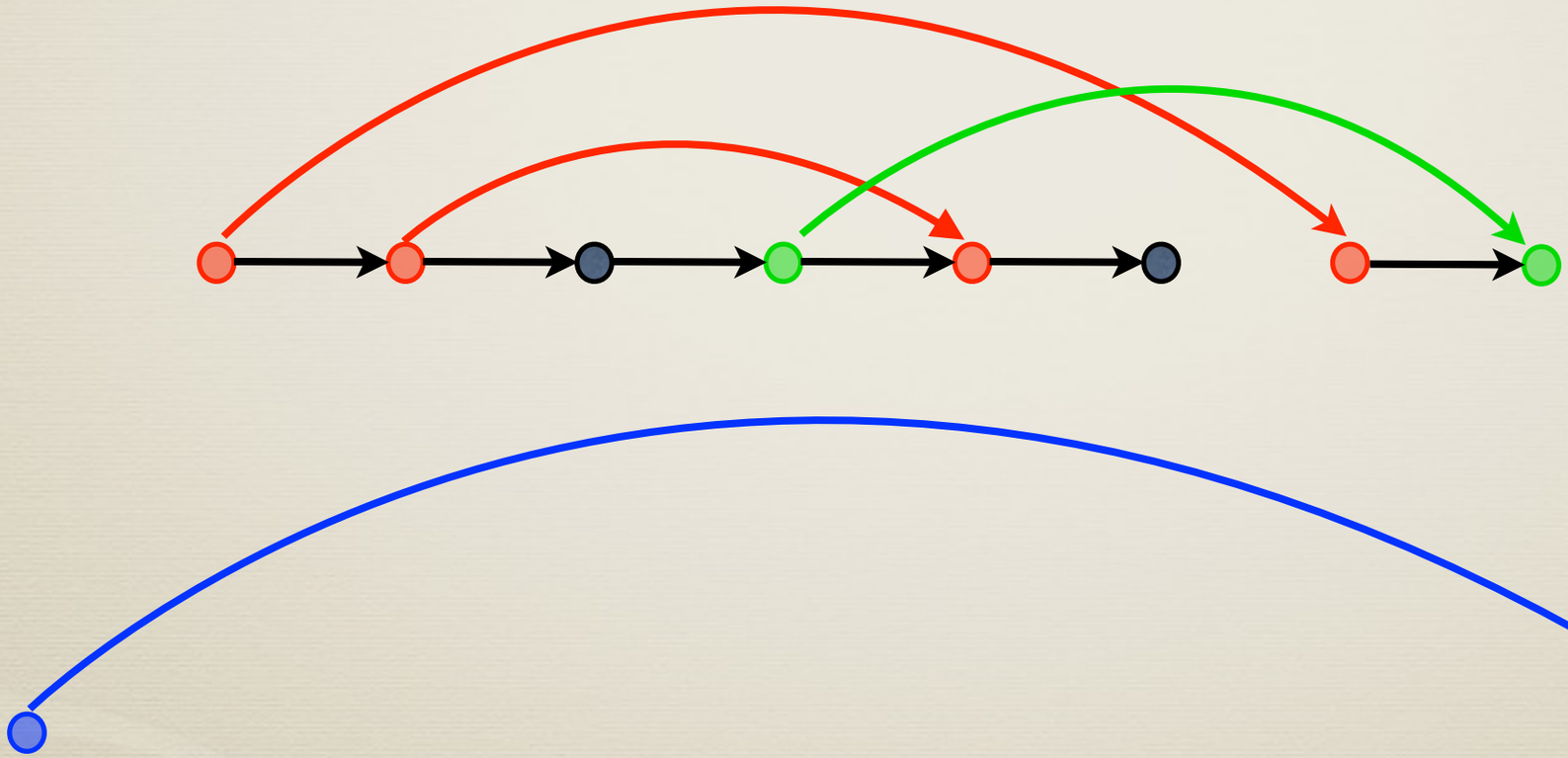
Separate out two disconnected parts into 2 split-CBMs



Split-width Operations

The Split Operation:

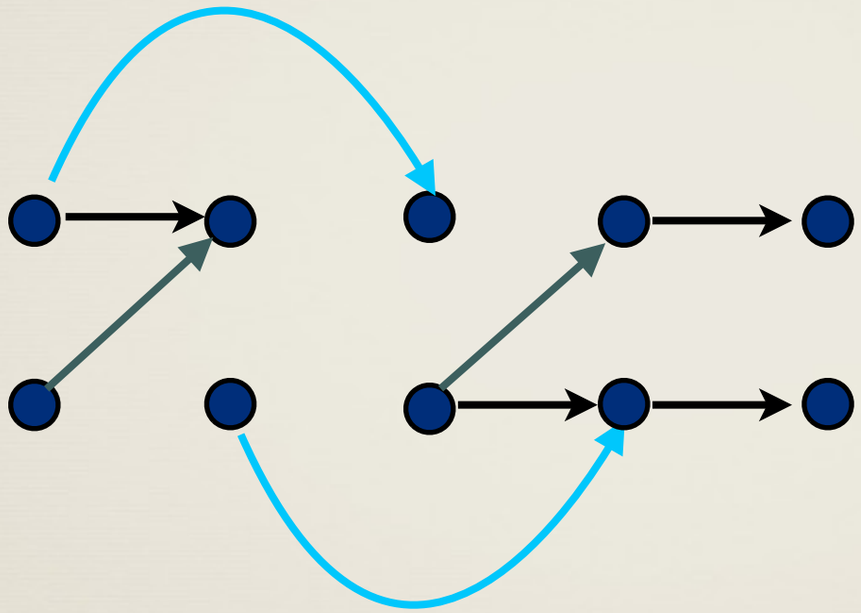
Separate out two disconnected parts into 2 split-CBMs



Split-width Operations

The Split Operation:

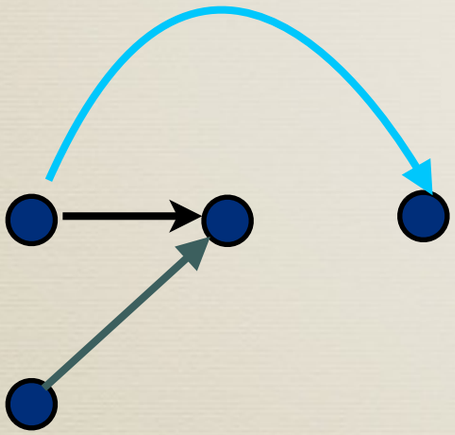
Separate out two disconnected parts into 2 split-CBMs

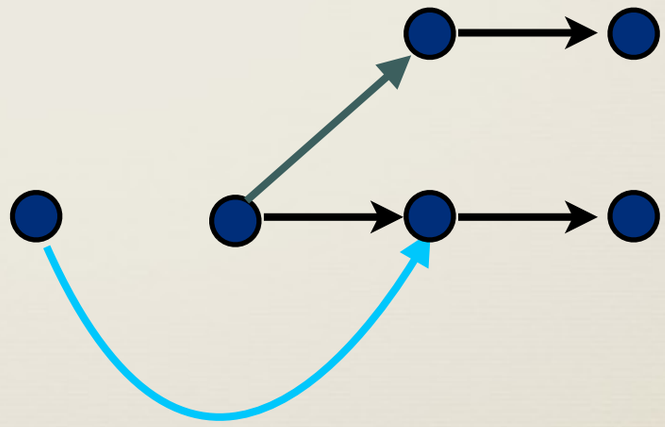


Split-width Operations

The Split Operation:

Separate out two disconnected parts into 2 split-CBMs

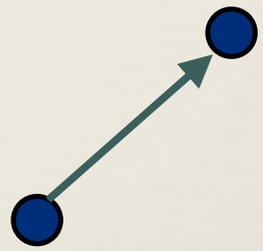


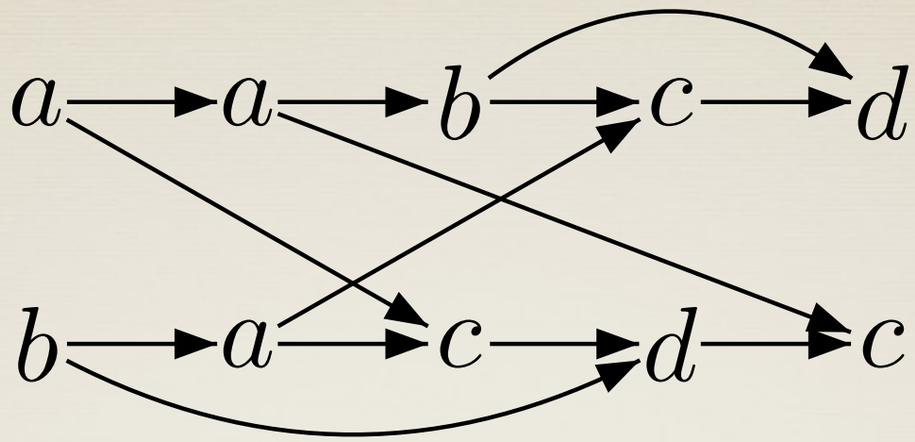


The Basic Split CBMs

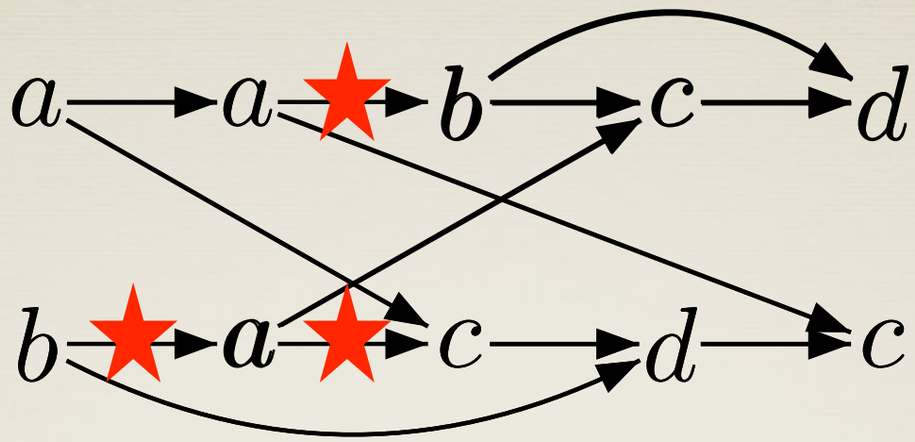
An Internal event:

A Communication edge:

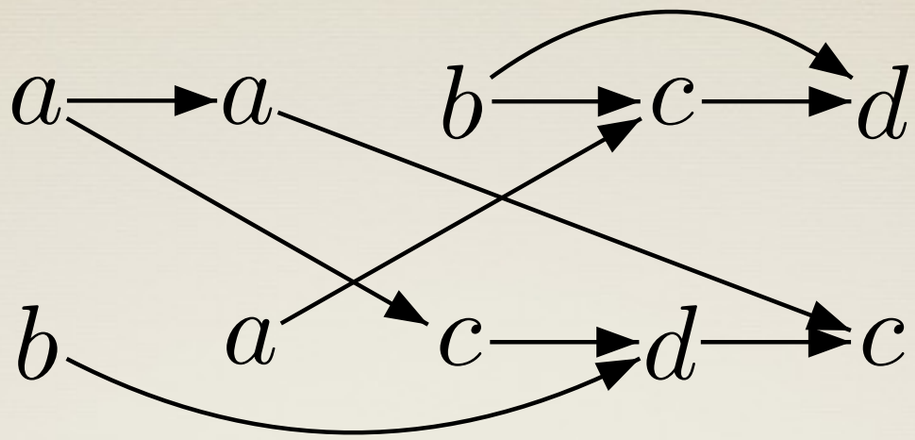




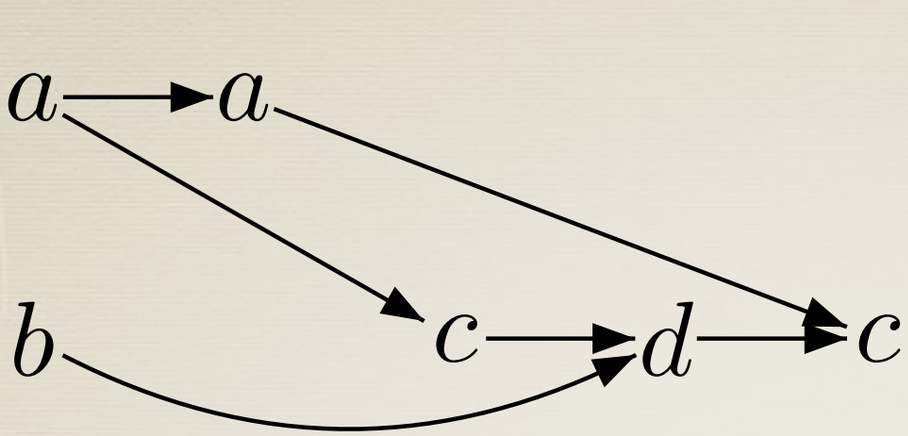
SPLIT DECOMPOSITION OF CBMs

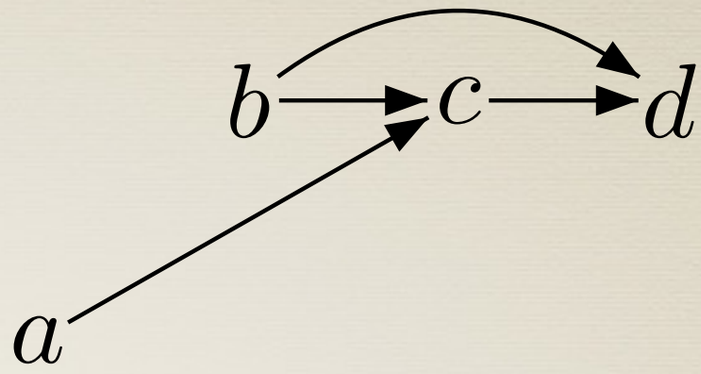


SPLIT DECOMPOSITION OF CBMs

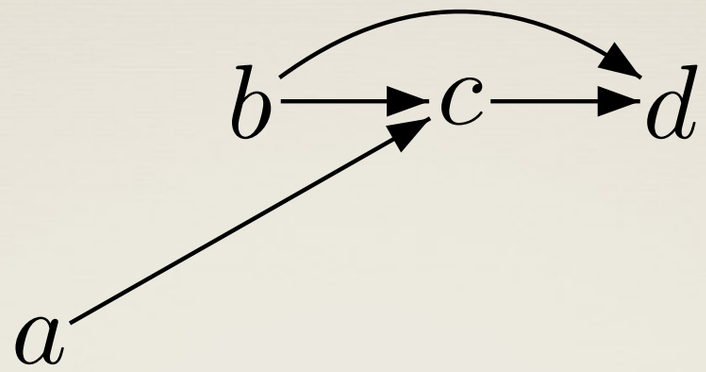


SPLIT DECOMPOSITION OF CBMs

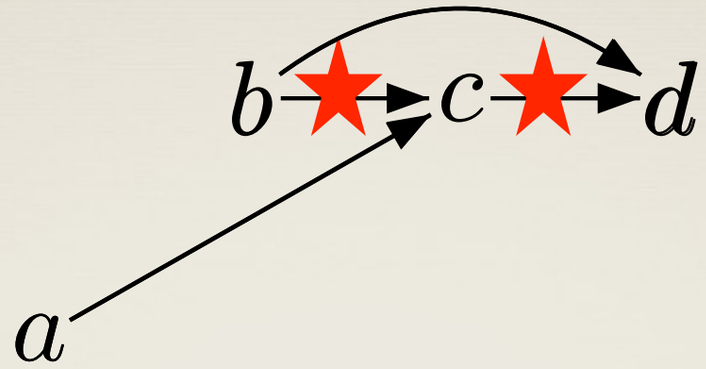




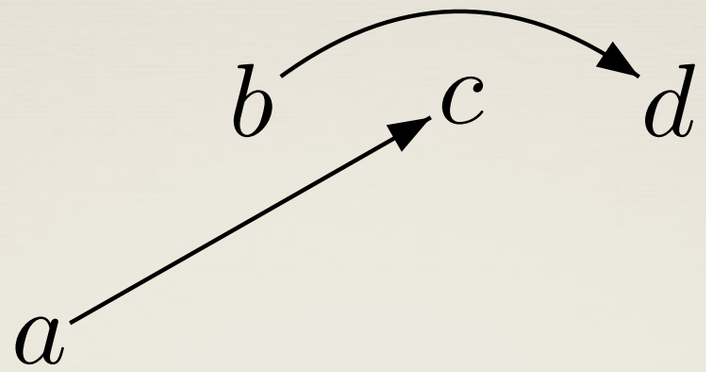
SPLIT DECOMPOSITION OF CBMs



SPLIT DECOMPOSITION OF CBMs

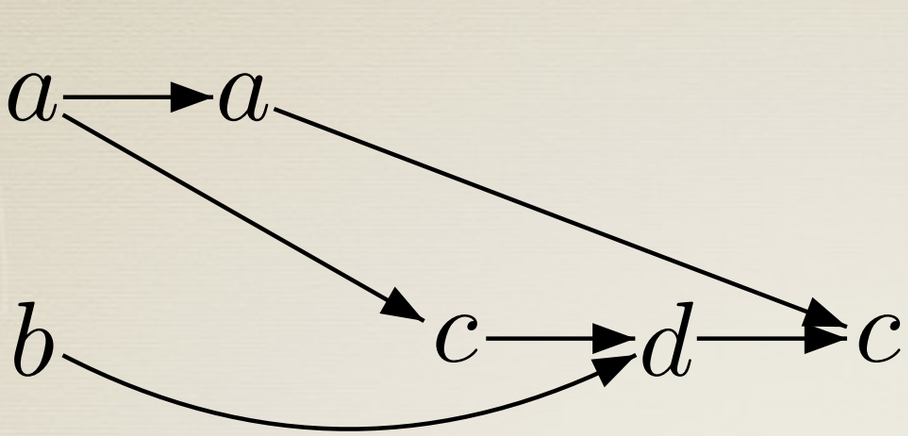


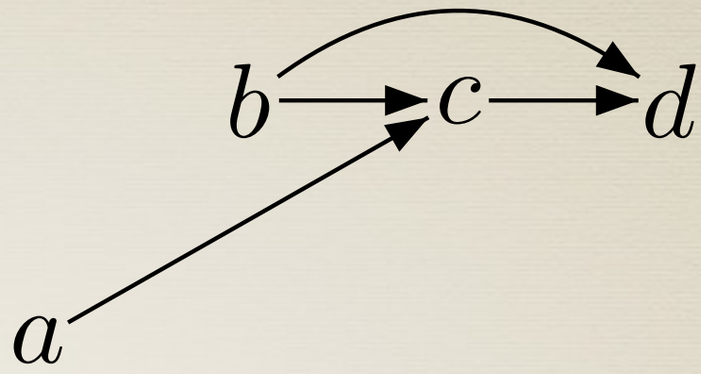
SPLIT DECOMPOSITION OF CBMs



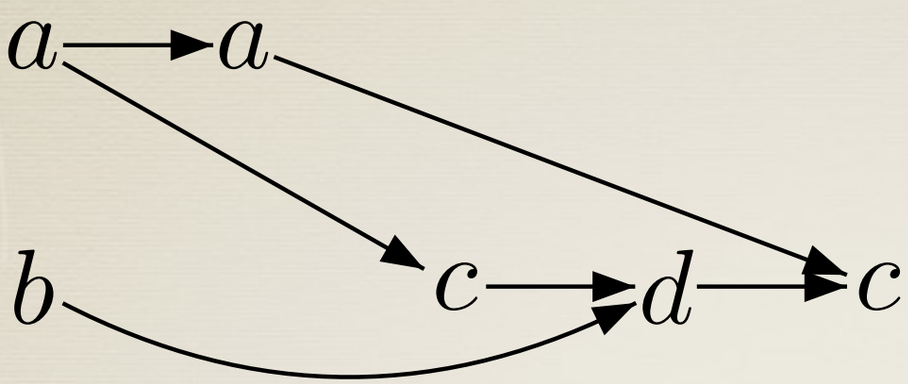
SPLIT DECOMPOSITION OF CBMs

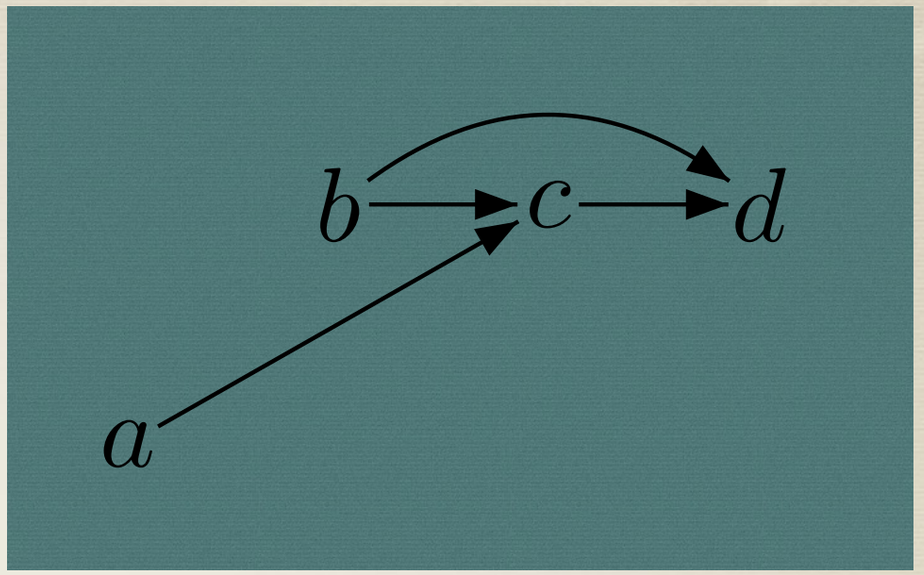
SPLIT DECOMPOSITION OF CBMs



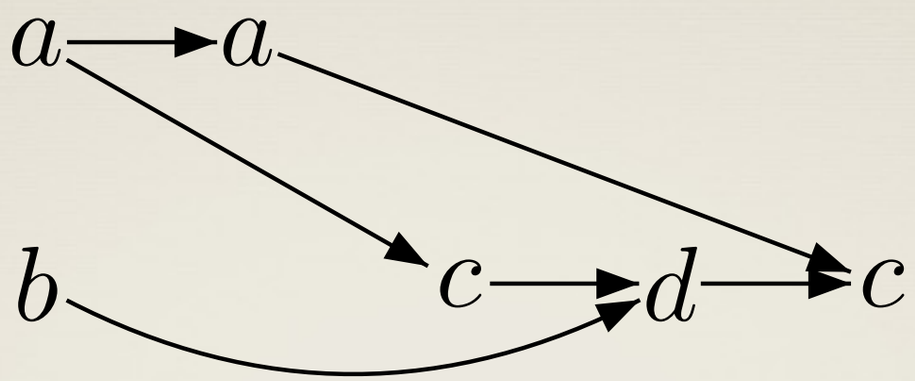


SPLIT DECOMPOSITION OF CBMs

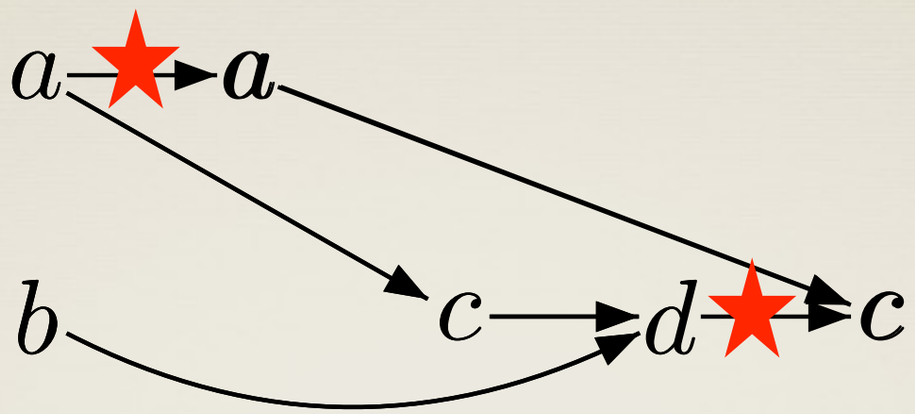




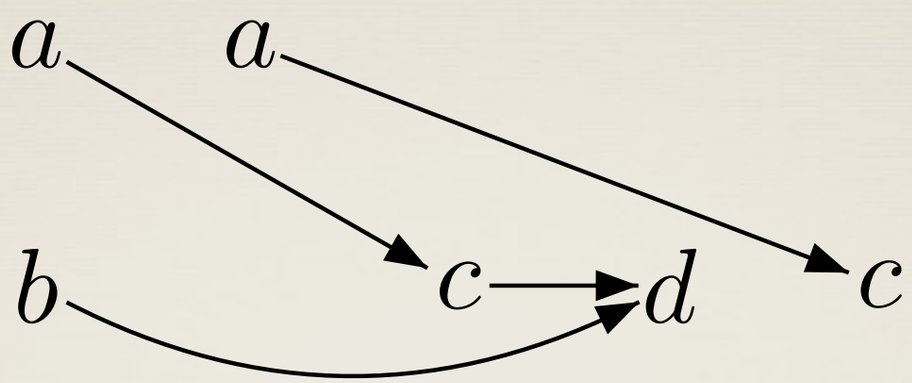
SPLIT DECOMPOSITION OF CBMs



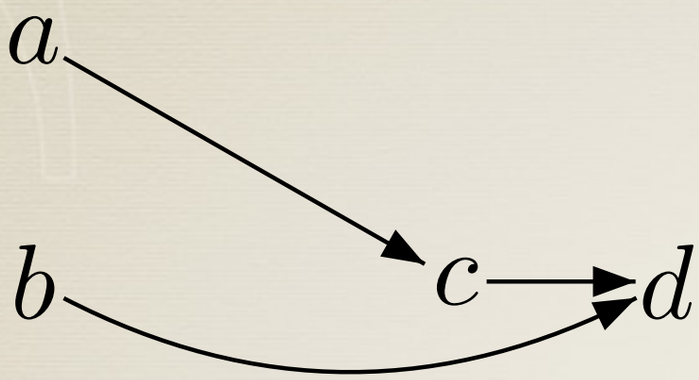
SPLIT DECOMPOSITION OF CBMs

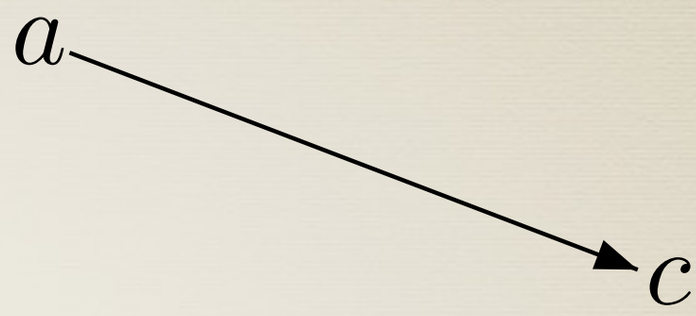


SPLIT DECOMPOSITION OF CBMs

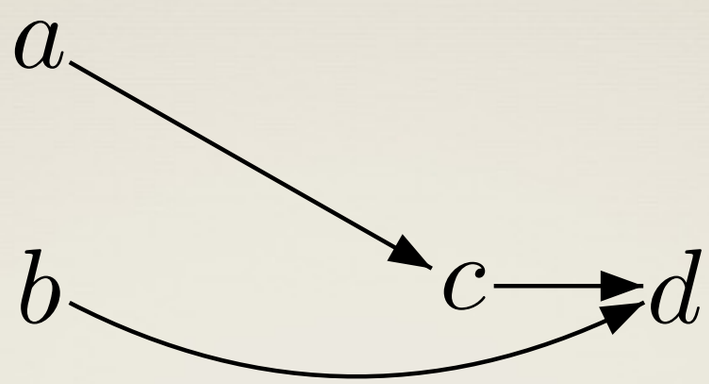


SPLIT DECOMPOSITION OF CBMs

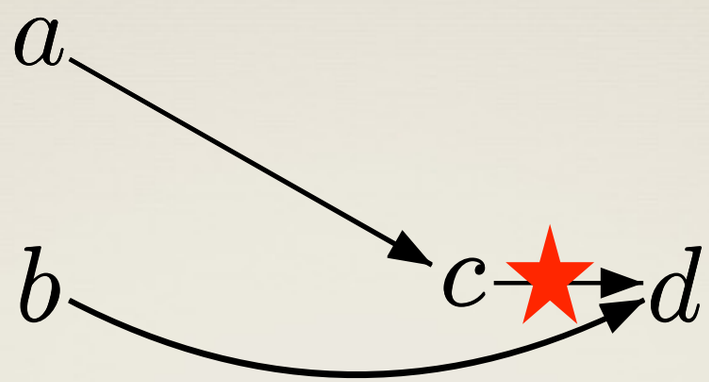




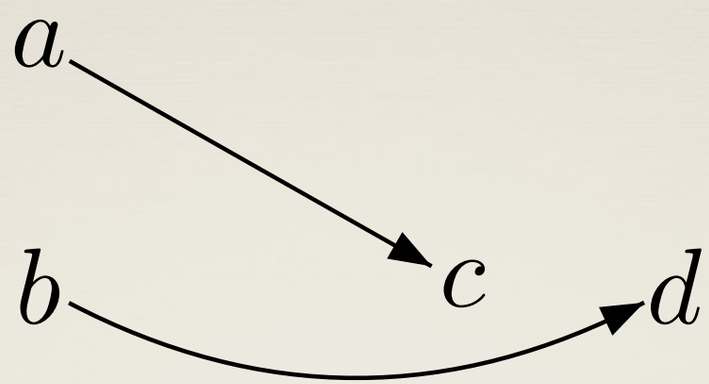
SPLIT DECOMPOSITION OF CBMs



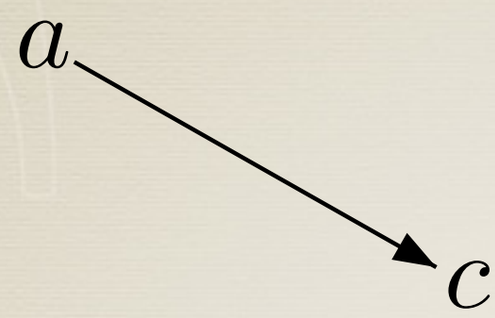
SPLIT DECOMPOSITION OF CBMs



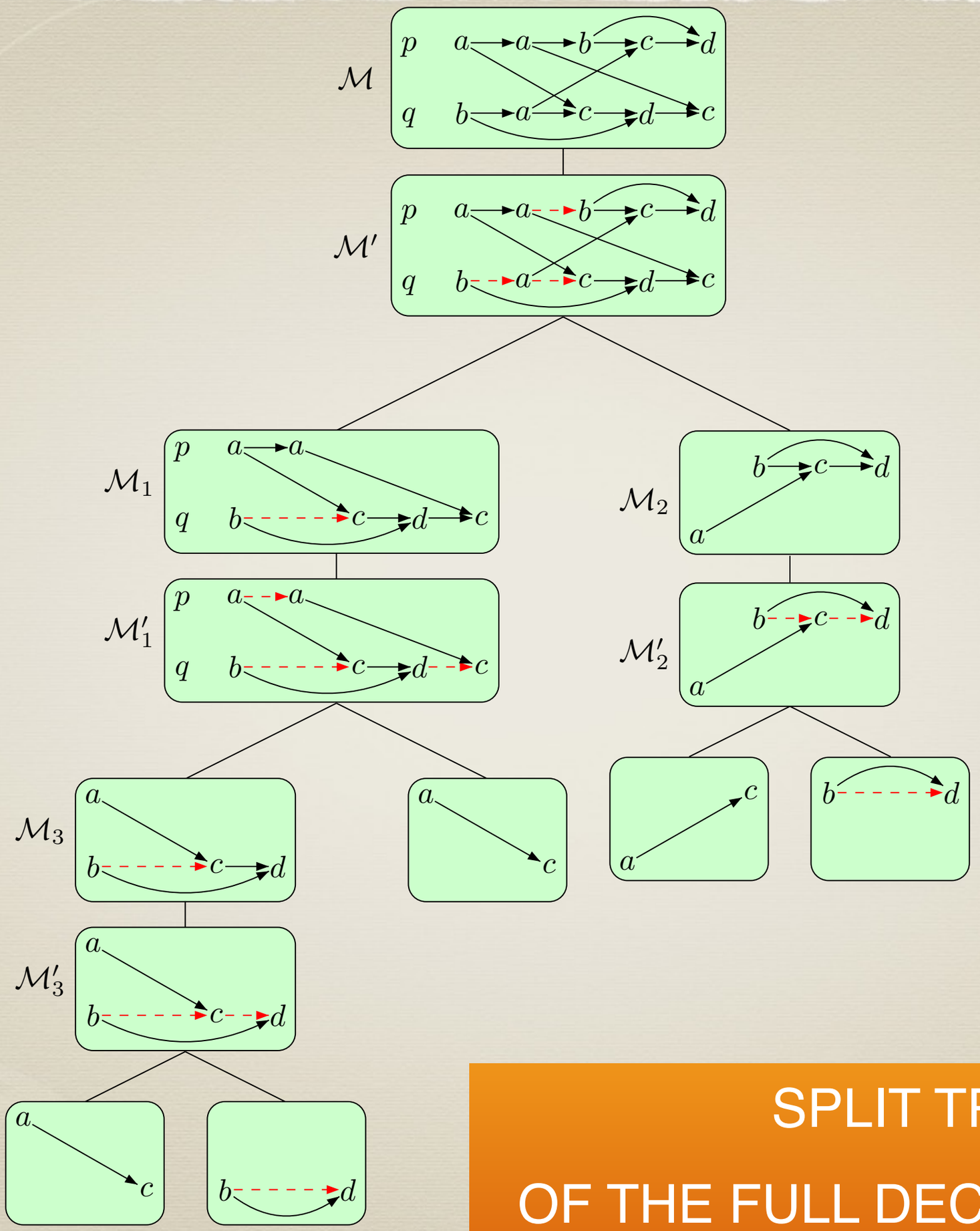
SPLIT DECOMPOSITION OF CBMs



SPLIT DECOMPOSITION OF CBMs

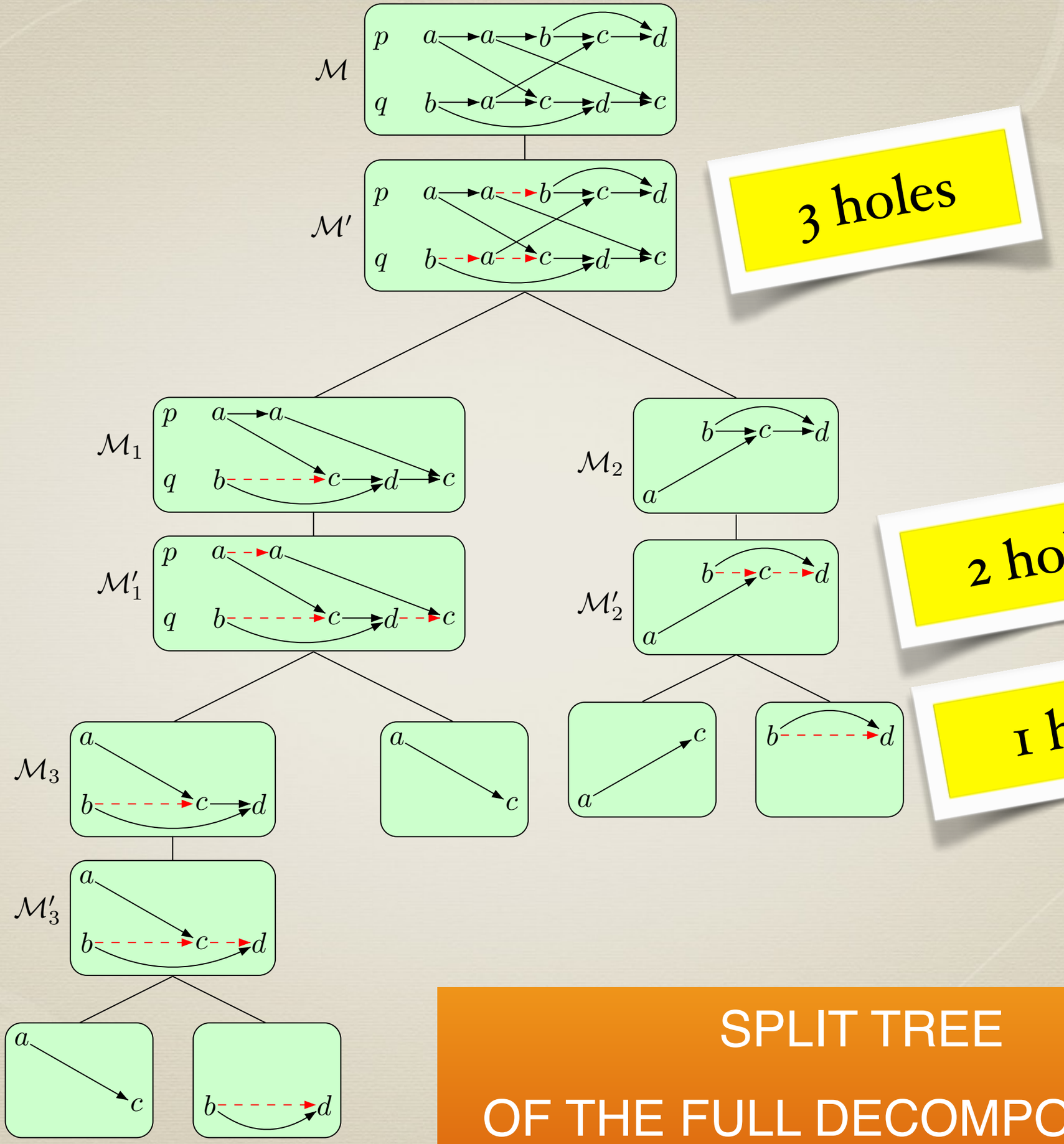


SPLIT DECOMPOSITION OF CBMs



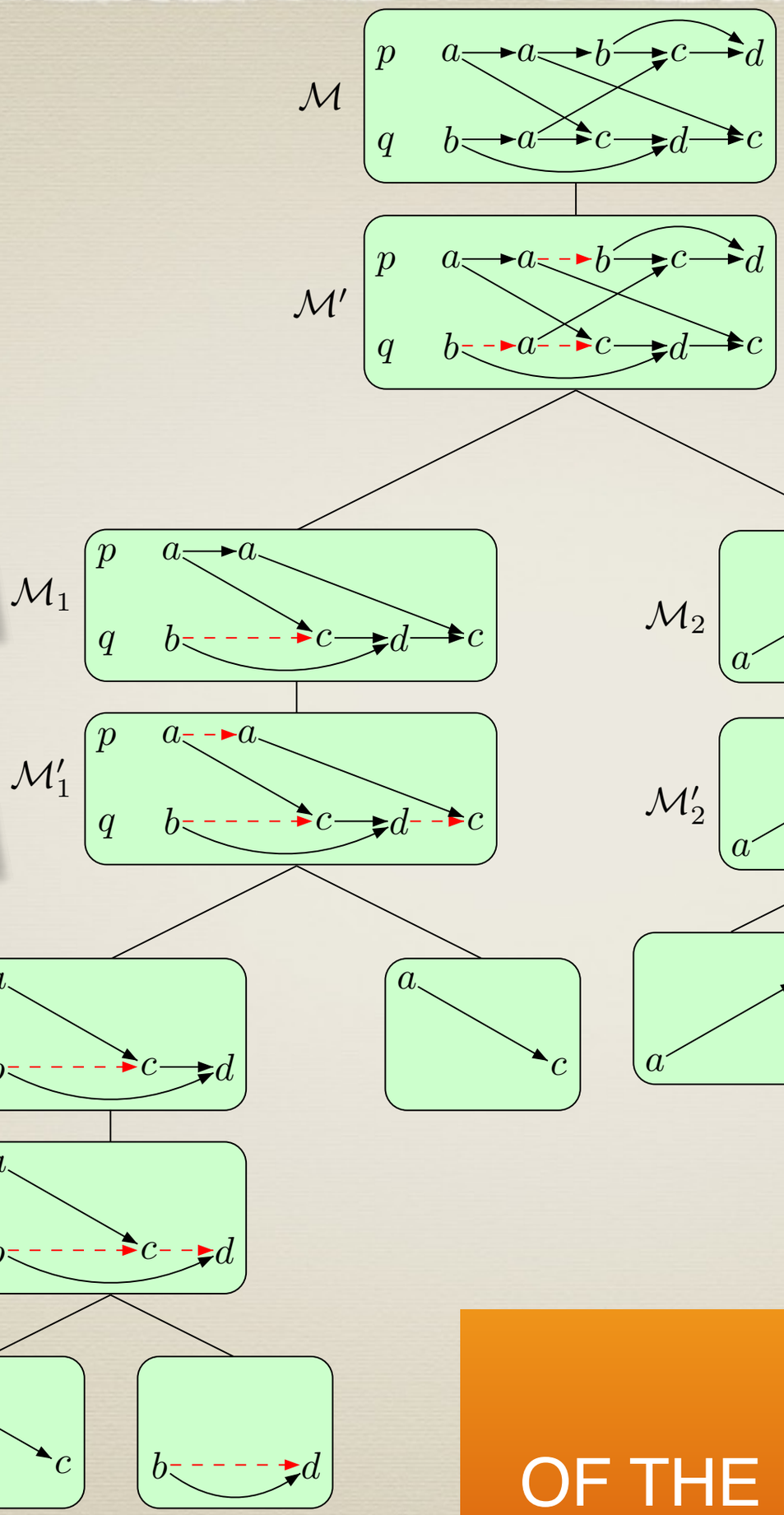
SPLIT TREE

OF THE FULL DECOMPOSITION



SPLIT TREE

OF THE FULL DECOMPOSITION



3 holes

1 hole

3 holes

1 hole

2 holes

2 holes

1 hole

SPLIT TREE

OF THE FULL DECOMPOSITION

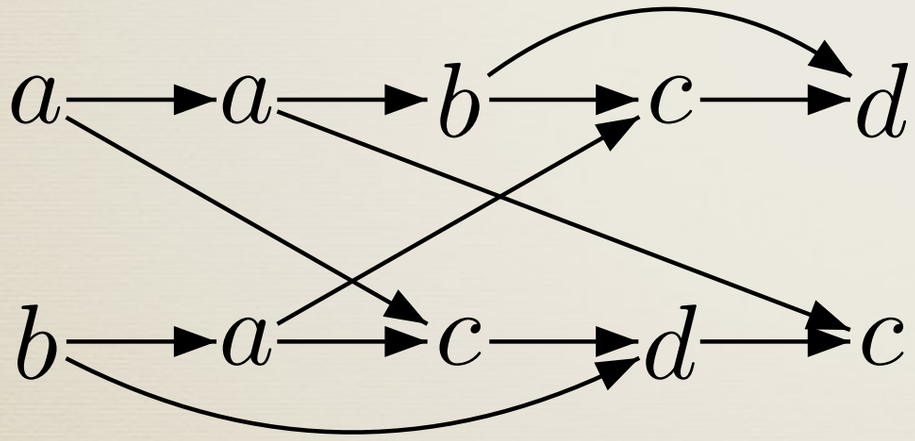
Split-width

* The width of a decomposition is the maximum number of holes in any split-CBM in the decomposition.
* Split-width of a CBM is the minimum of the widths of its decompositions.

Split-width

* The width of a decomposition is the maximum number of holes in any split-CBM in the decomposition.

* Split-width of a CBM is the minimum of the widths of its decompositions.

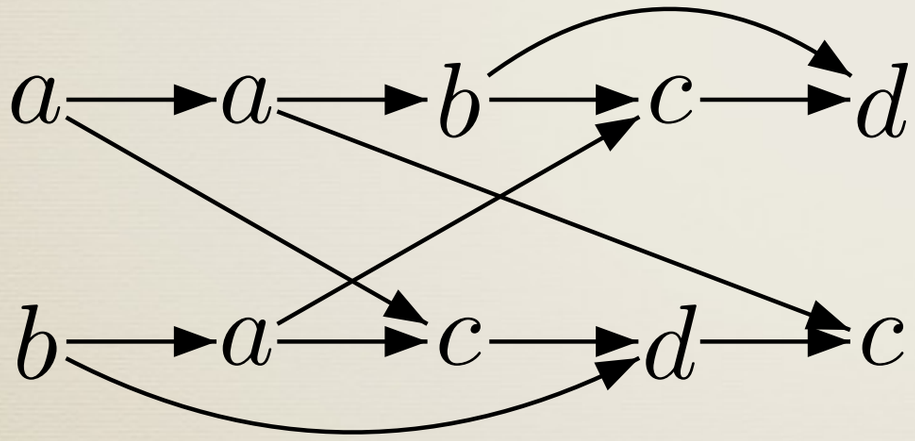


A CBM with split-width = 3

Split-width

* The width of a decomposition is the maximum number of holes in any split-CBM in the decomposition.

* Split-width of a CBM is the minimum of the widths of its decompositions.

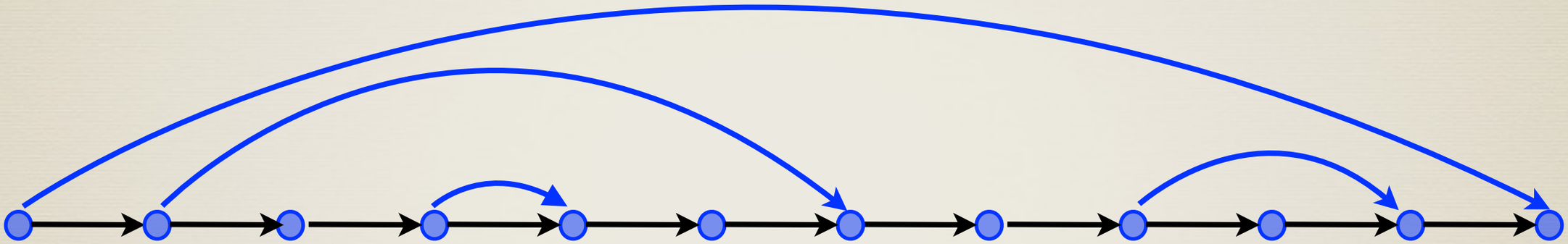


A CBM with split-width = 3

Split-width of a set of CBMs is the maximum of their split-widths

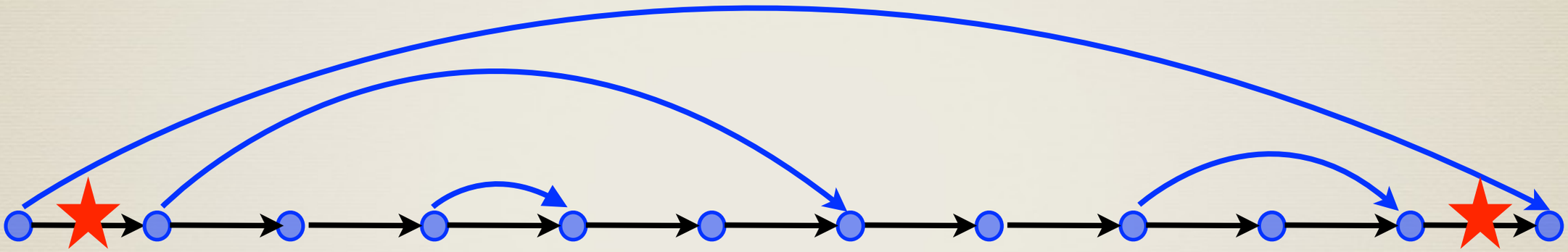
Split-width of nested words

* The class of nested words has split-width bounded by 2



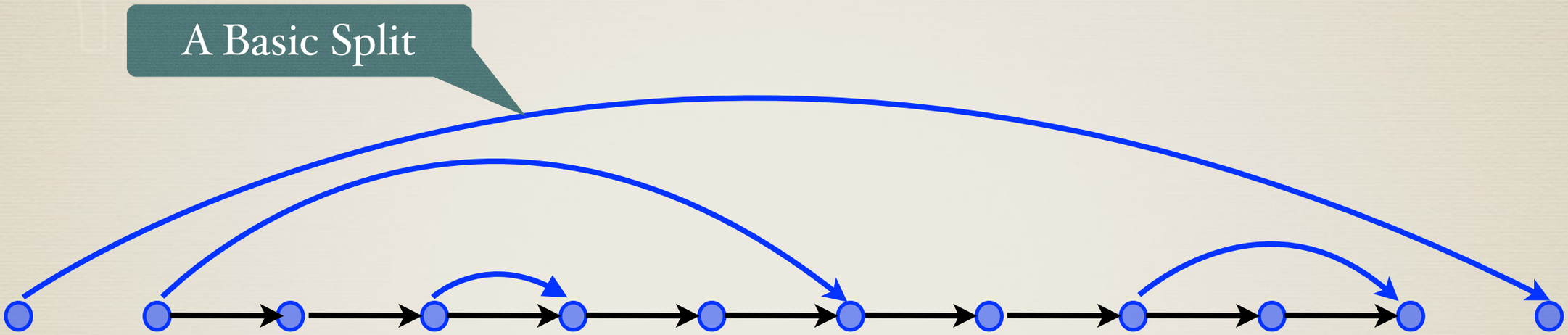
Split-width of nested words

* The class of nested words has split-width bounded by 2



Split-width of nested words

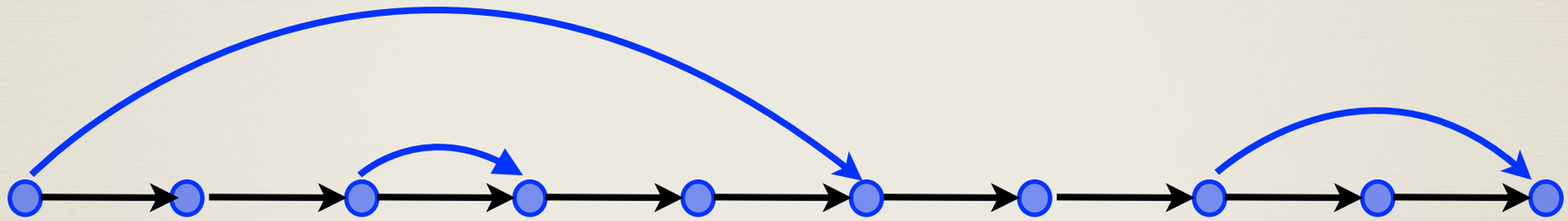
* The class of nested words has split-width bounded by 2



2 holes

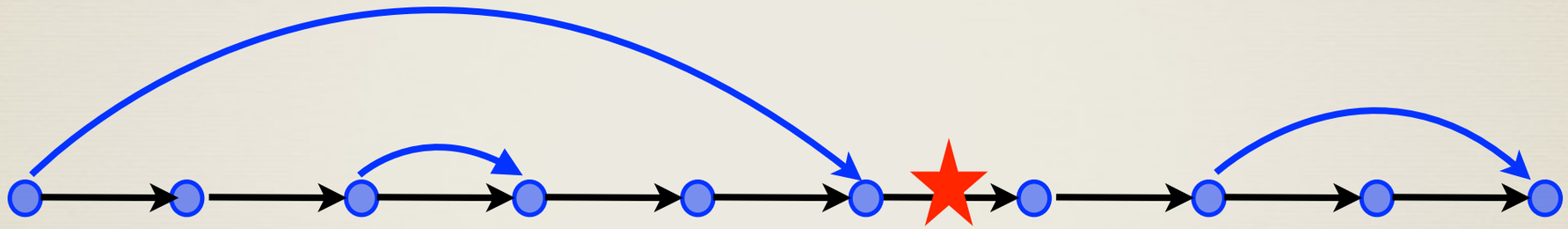
Split-width of nested words

* The class of nested words has split-width bounded by 2



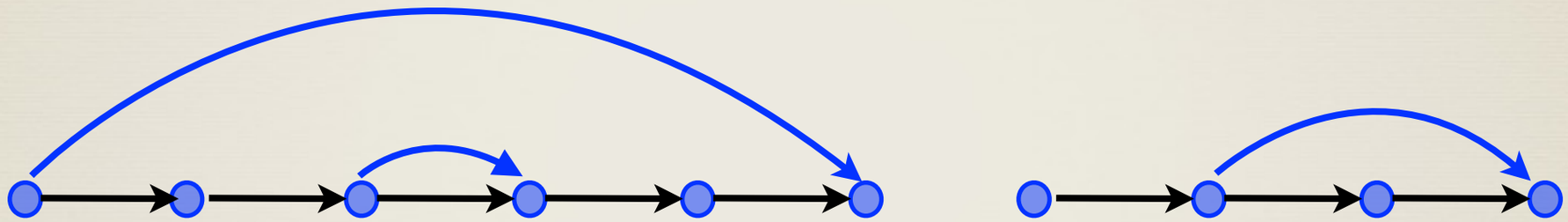
Split-width of nested words

* The class of nested words has split-width bounded by 2



Split-width of nested words

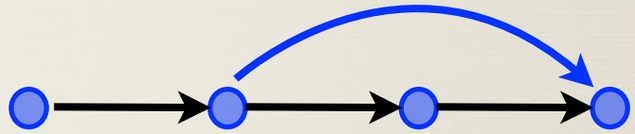
* The class of nested words has split-width bounded by 2

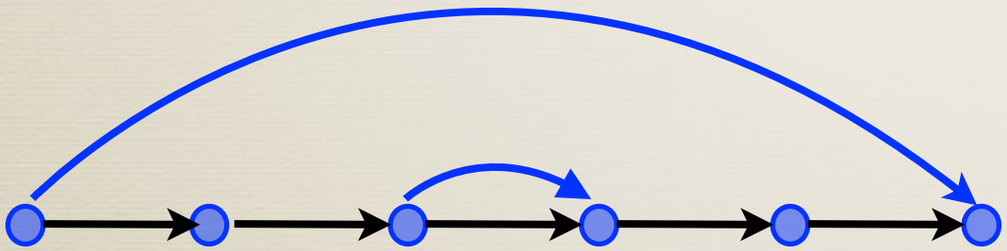


1 hole

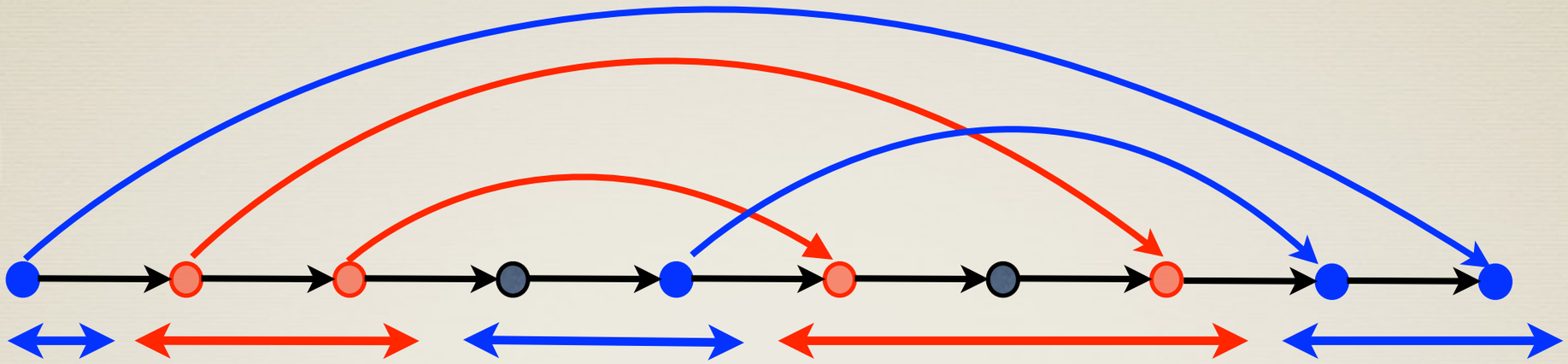
Split-width of nested words

* The class of nested words has split-width bounded by 2

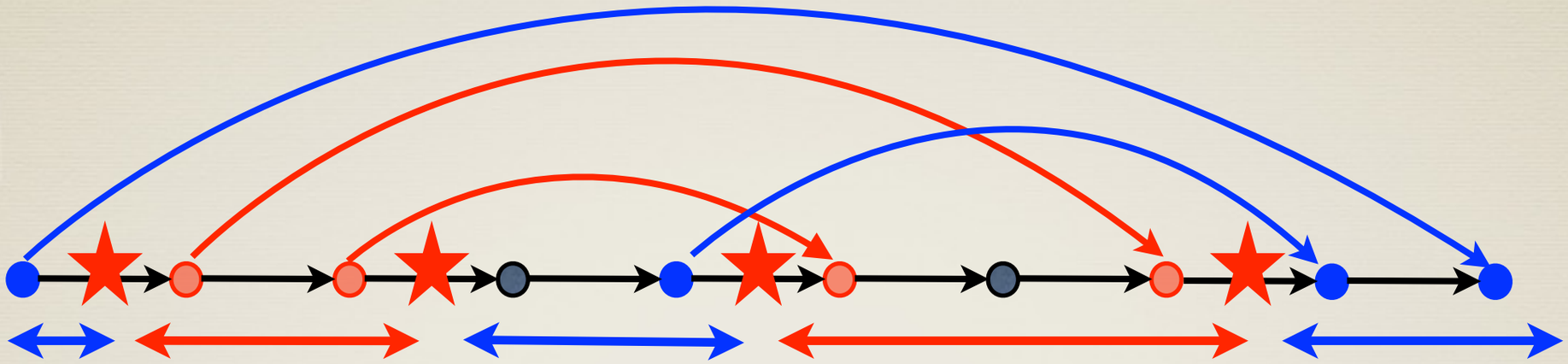




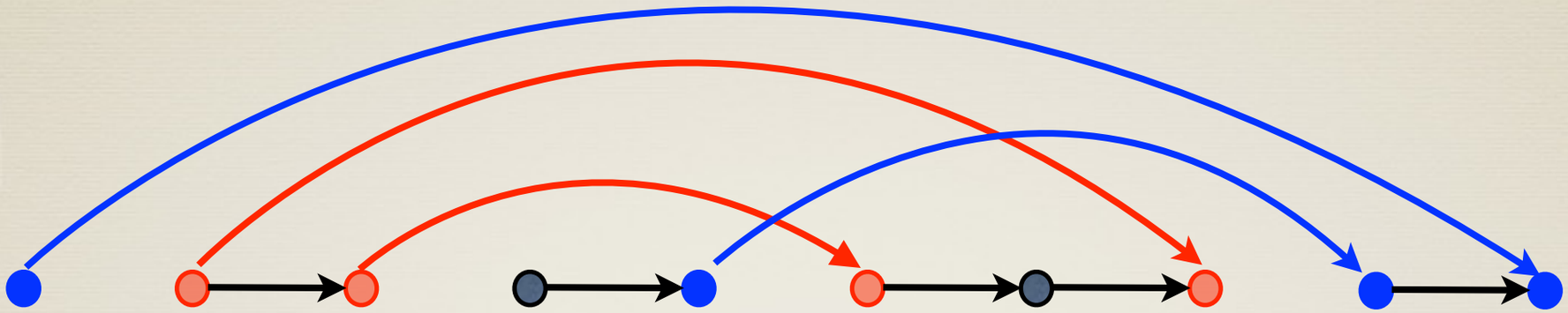
Bounded-context Runs



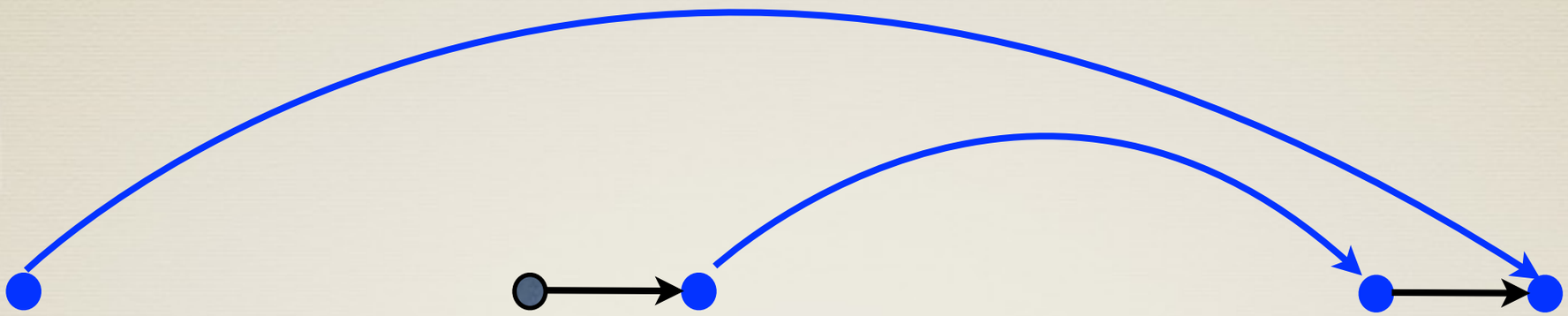
Bounded-context Runs



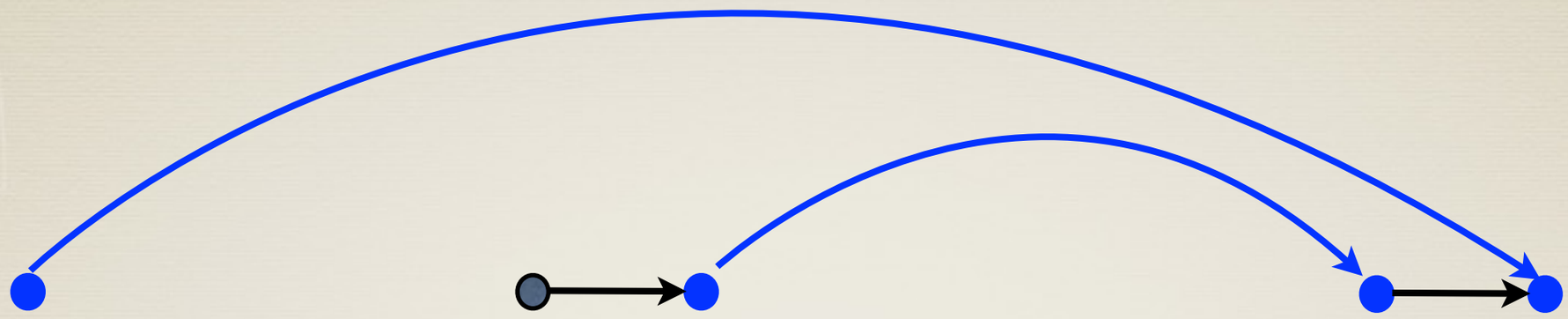
Bounded-context Runs



Bounded-context Runs



Bounded-context Runs



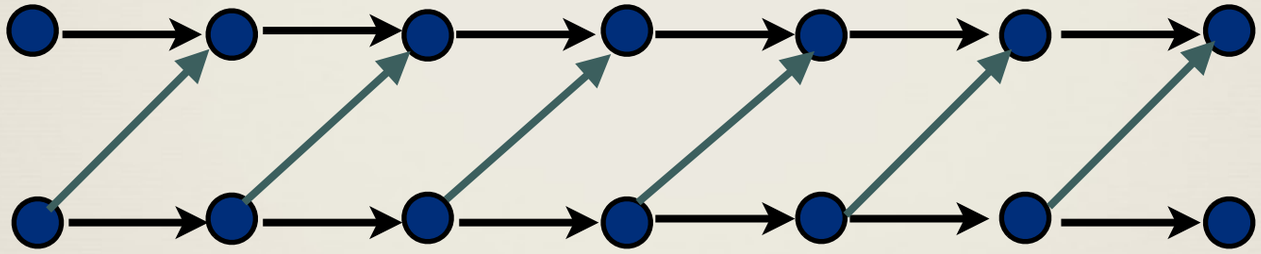
A k context CBM has split-width k+1

Existentially k bounded MSCs

There is a linearisation where no channel contains more than k values at any point along the linearization.

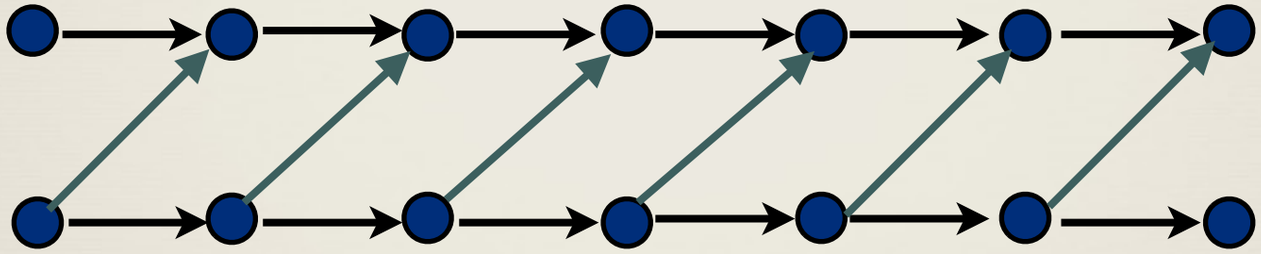
Existentially k bounded MSCs

There is a linearisation where no channel contains more than k values at any point along the linearization.



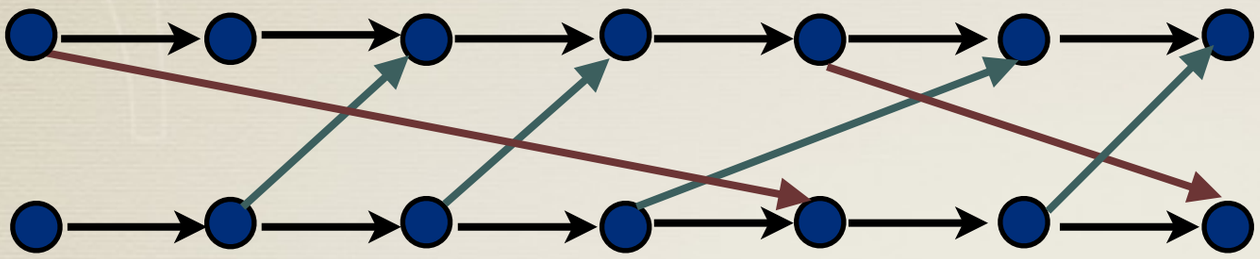
Existentially k bounded MSCs

There is a linearisation where no channel contains more than k values at any point along the linearization.

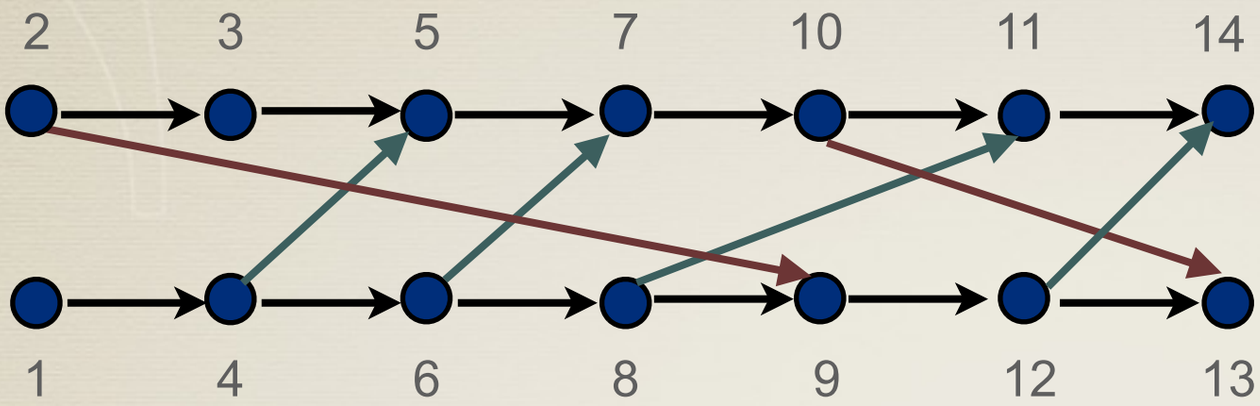


An existentially 1 bounded behaviour.

Existentially k bounded MSCs

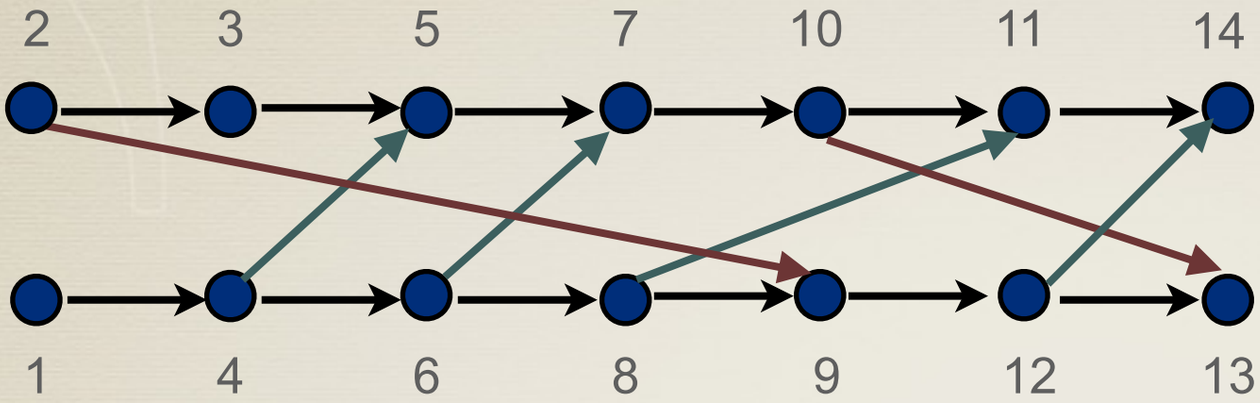


Existentially k bounded MSCs

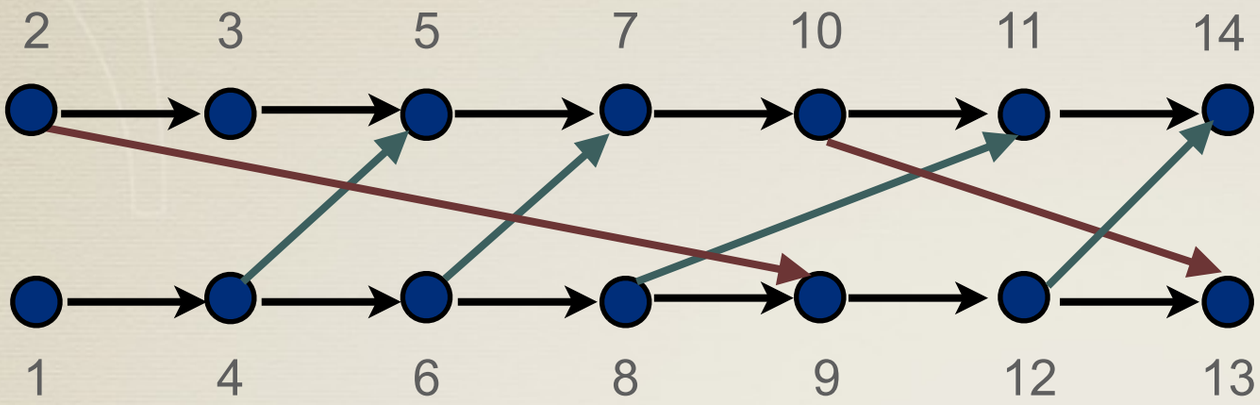


a 2-bounded sequentialisation

Existentially k bounded MSCs

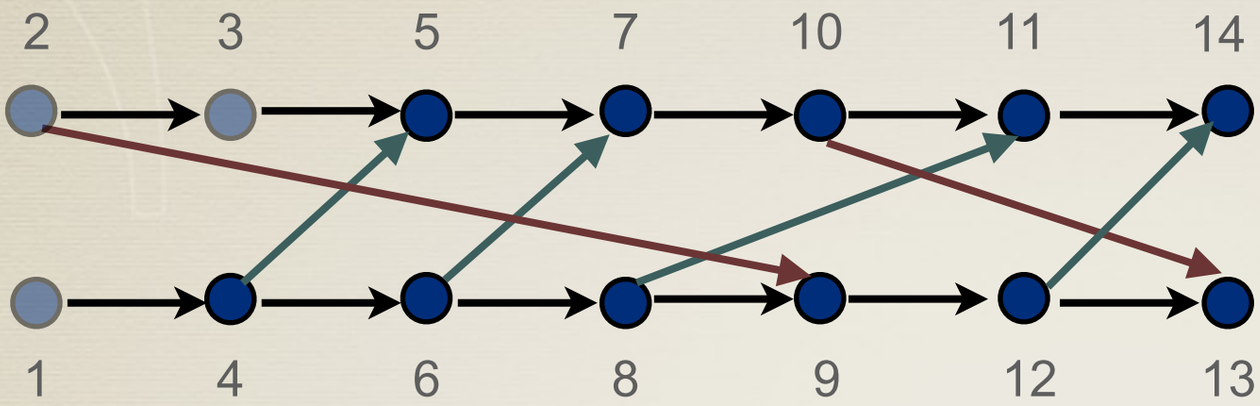


Existentially k bounded MSCs



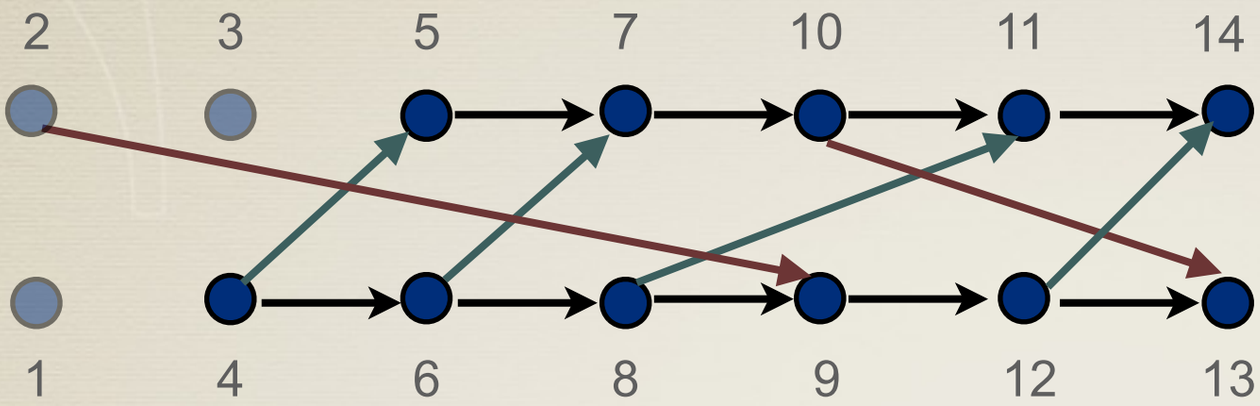
* Cut process edges from the first k+1 events in the k bounded sequentialisation.
* Remove message edges, internal events that can be.
* Expand to have k+1 events and repeat this process.

Existentially k bounded MSCs



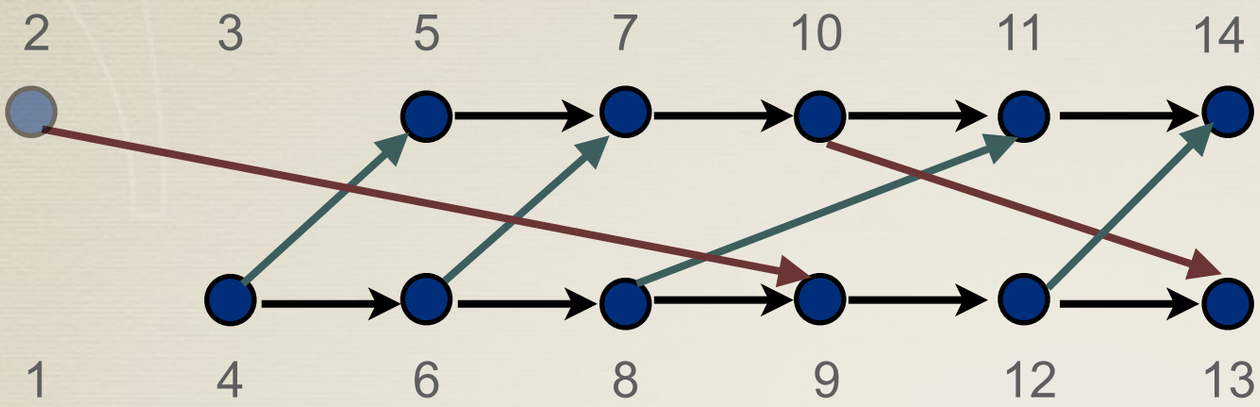
* Cut process edges from the first k+1 events in the k bounded sequentialisation.
* Remove message edges, internal events that can be.
* Expand to have k+1 events and repeat this process.

Existentially k bounded MSCs



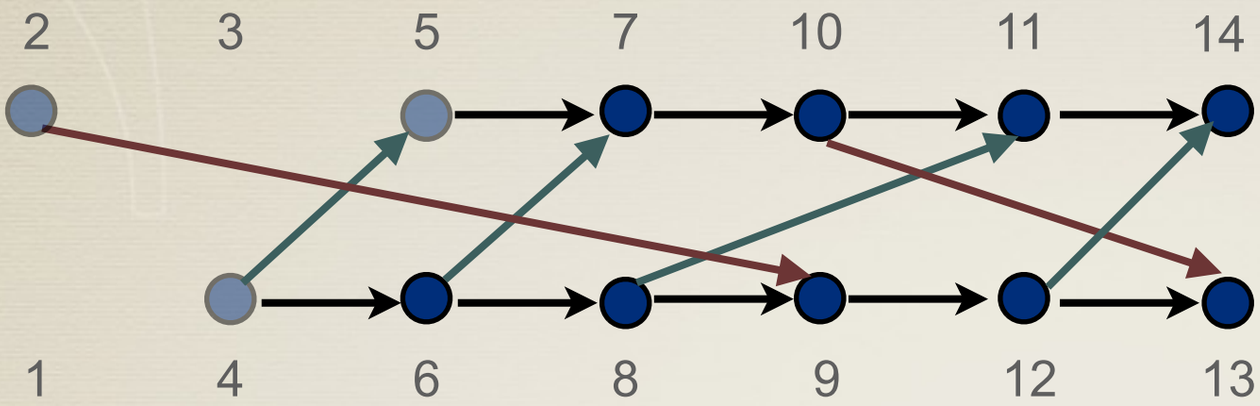
* Cut process edges from the first k+1 events in the k bounded sequentialisation.
* Remove message edges, internal events that can be.
* Expand to have k+1 events and repeat this process.

Existentially k bounded MSCs



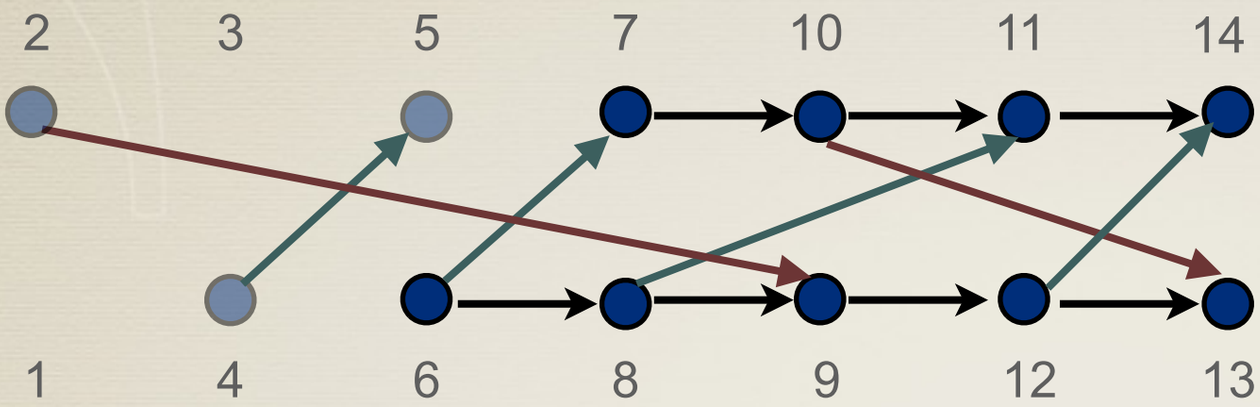
* Cut process edges from the first k+1 events in the k bounded sequentialisation.
* Remove message edges, internal events that can be.
* Expand to have k+1 events and repeat this process.

Existentially k bounded MSCs



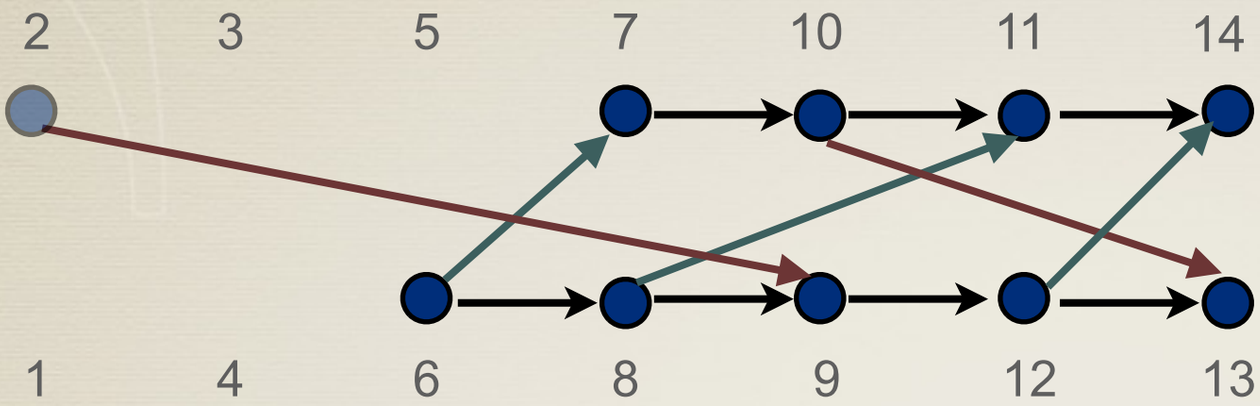
* Cut process edges from the first k+1 events in the k bounded sequentialisation.
* Remove message edges, internal events that can be.
* Expand to have k+1 events and repeat this process.

Existentially k bounded MSCs



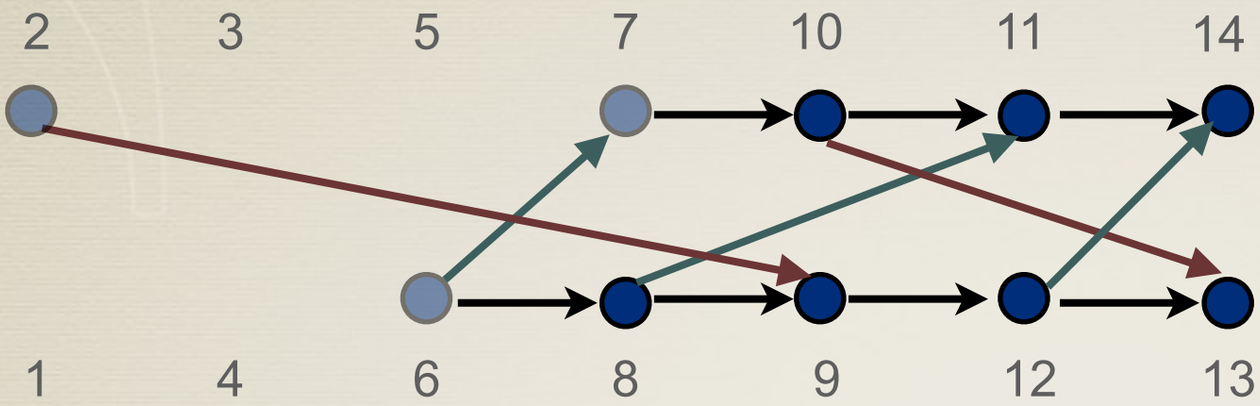
* Cut process edges from the first k+1 events in the k bounded sequentialisation.
* Remove message edges, internal events that can be.
* Expand to have k+1 events and repeat this process.

Existentially k bounded MSCs



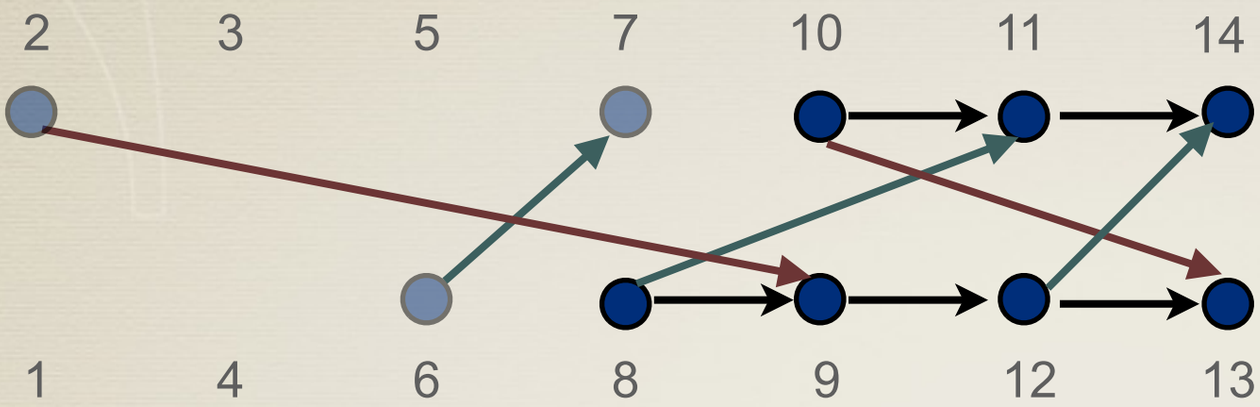
* Cut process edges from the first k+1 events in the k bounded sequentialisation.
* Remove message edges, internal events that can be.
* Expand to have k+1 events and repeat this process.

Existentially k bounded MSCs



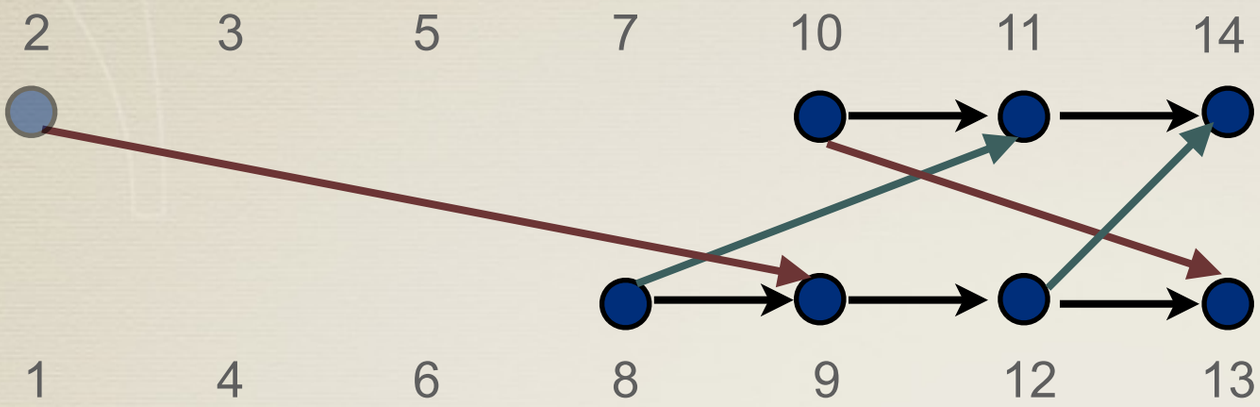
* Cut process edges from the first k+1 events in the k bounded sequentialisation.
* Remove message edges, internal events that can be.
* Expand to have k+1 events and repeat this process.

Existentially k bounded MSCs



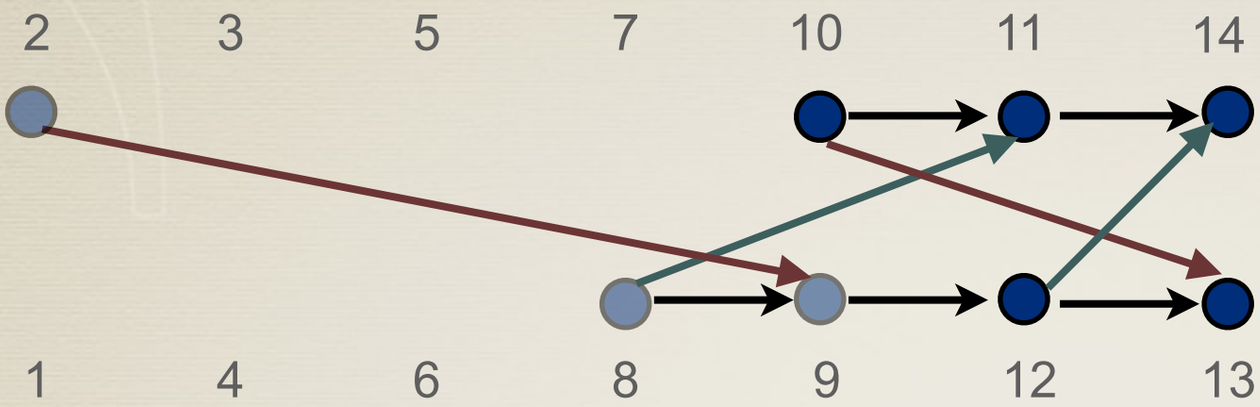
* Cut process edges from the first k+1 events in the k bounded sequentialisation.
* Remove message edges, internal events that can be.
* Expand to have k+1 events and repeat this process.

Existentially k bounded MSCs



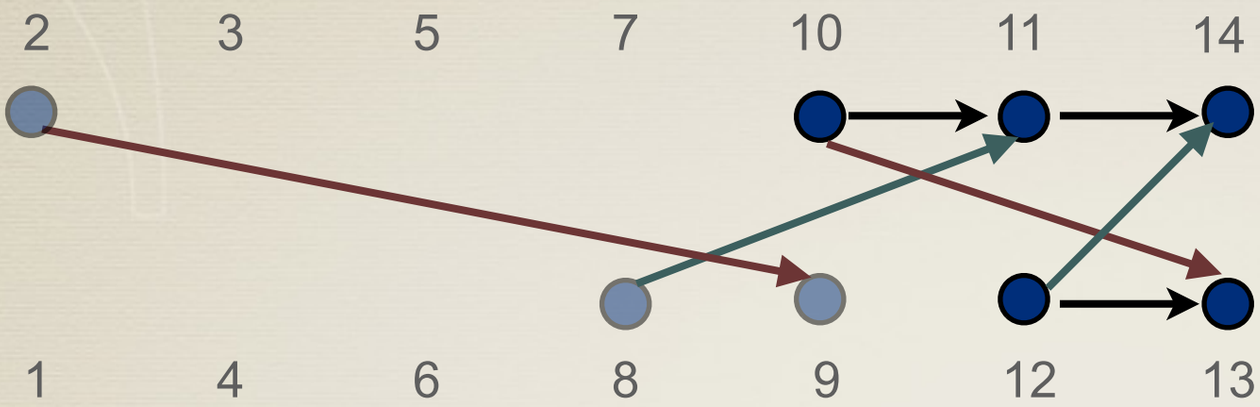
* Cut process edges from the first k+1 events in the k bounded sequentialisation.
* Remove message edges, internal events that can be.
* Expand to have k+1 events and repeat this process.

Existentially k bounded MSCs



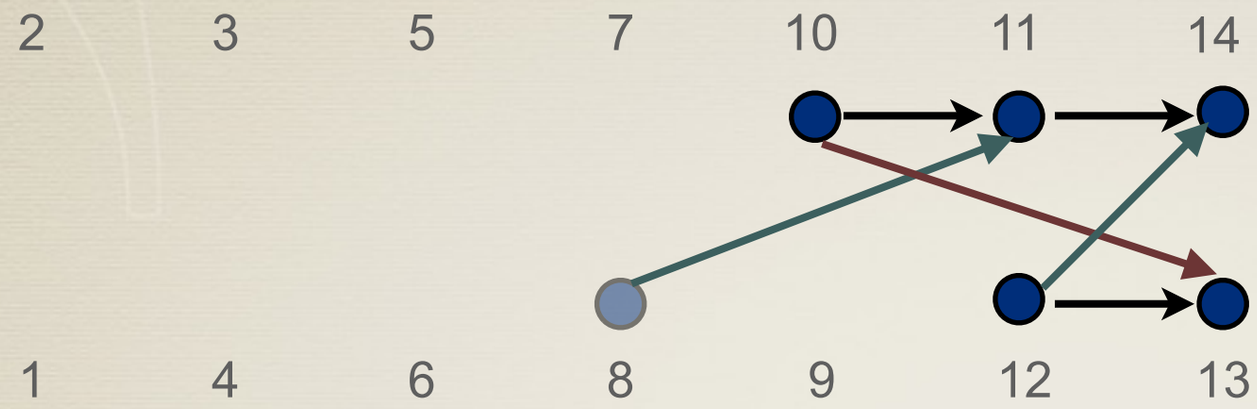
* Cut process edges from the first k+1 events in the k bounded sequentialisation.
* Remove message edges, internal events that can be.
* Expand to have k+1 events and repeat this process.

Existentially k bounded MSCs



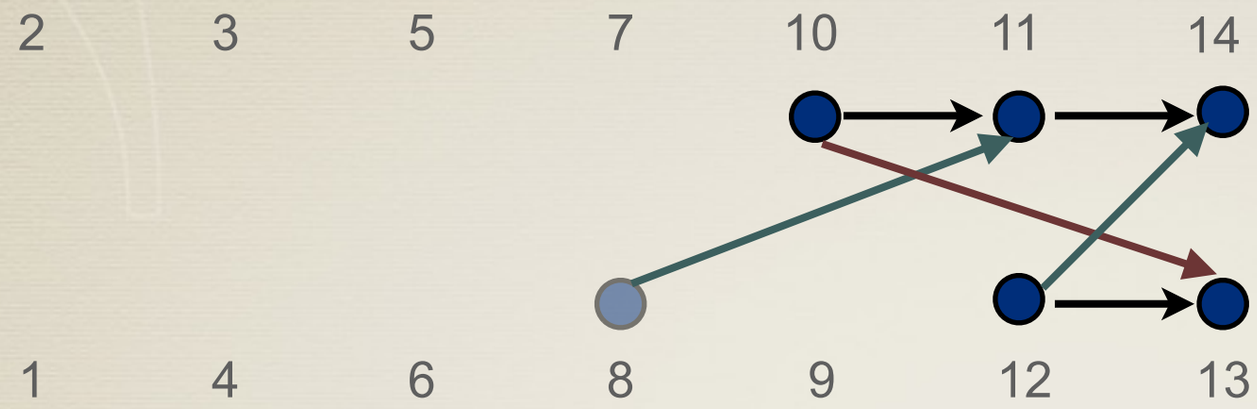
* Cut process edges from the first k+1 events in the k bounded sequentialisation.
* Remove message edges, internal events that can be.
* Expand to have k+1 events and repeat this process.

Existentially k bounded MSCs



* Cut process edges from the first k+1 events in the k bounded sequentialisation.
* Remove message edges, internal events that can be.
* Expand to have k+1 events and repeat this process.

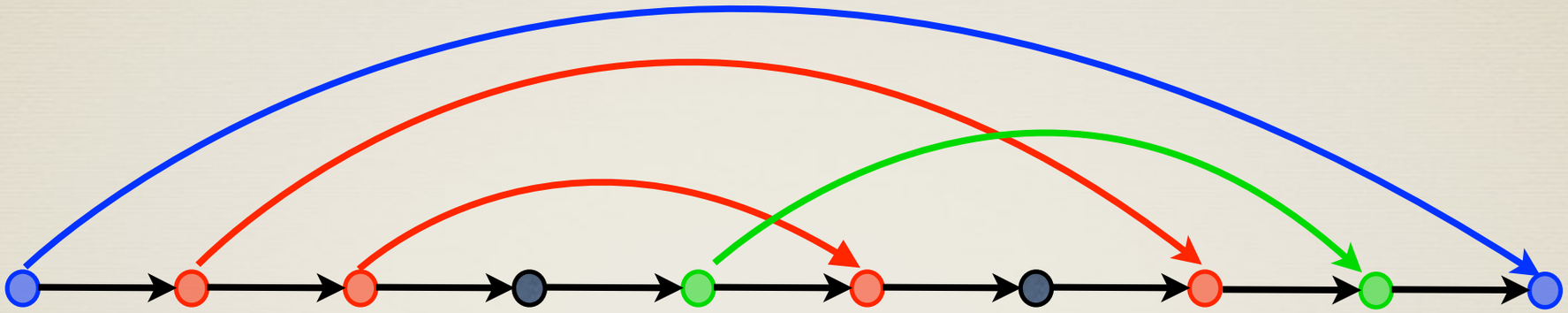
Existentially k bounded MSCs



* Cut process edges from the first k+1 events in the k bounded sequentialisation.
* Remove message edges, internal events that can be.
* Expand to have k+1 events and repeat this process.

A k existentially bounded MSC/CBM has split-width k+1

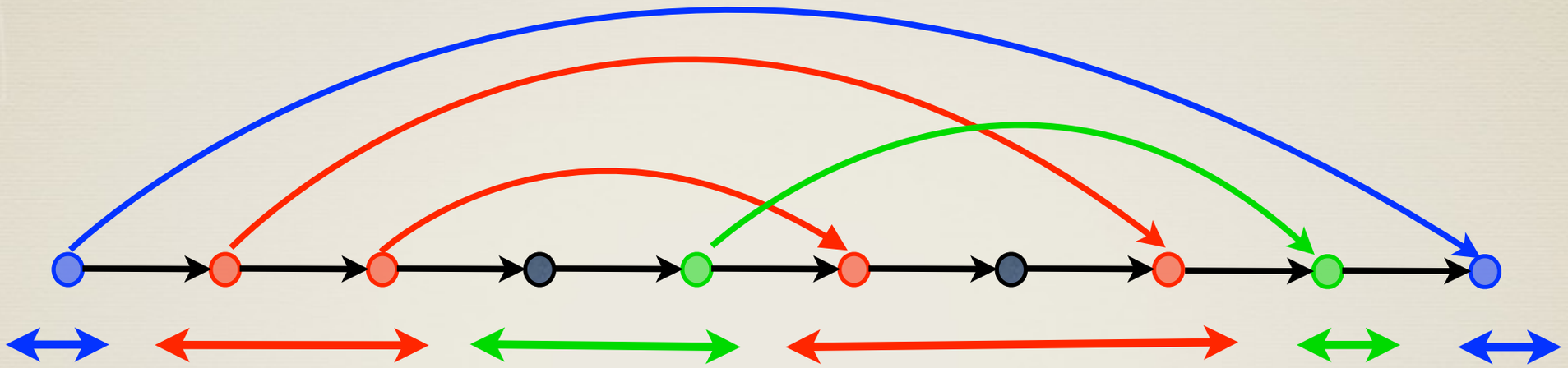
Bounded Scope Behaviours



* Between any call and the corresponding return there are at most k context-switches

* Somewhat different from other generalisations of bounded-context.

Bounded Scope Behaviours

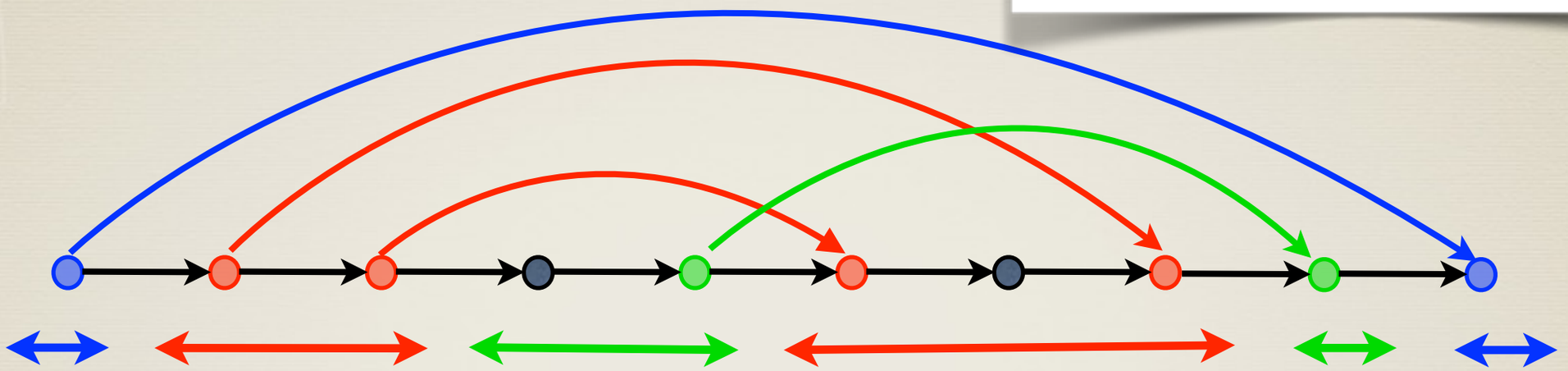


* Between any call and the corresponding return there are at most k context-switches

* Somewhat different from other generalisations of bounded-context.

Bounded Scope Behaviours

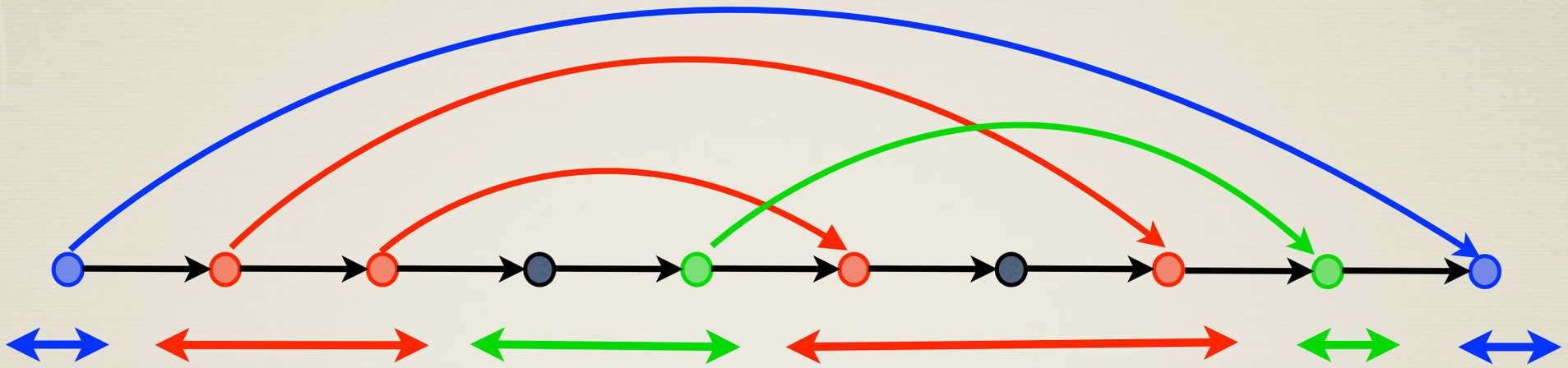
a 5 scope bounded cbm



* Between any call and the corresponding return there are at most k context-switches

* Somewhat different from other generalisations of bounded-context.

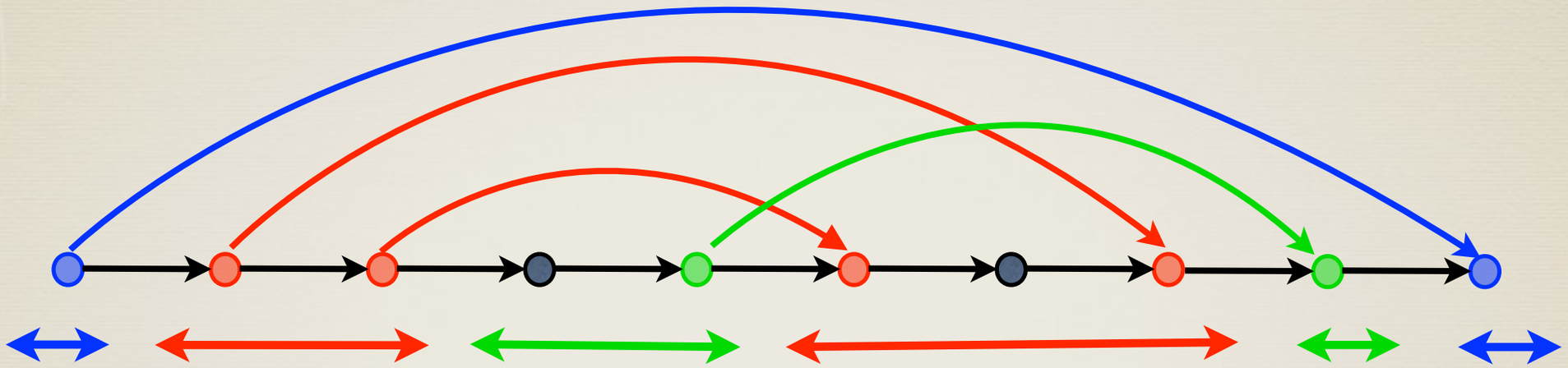
Bounded Scope Behaviours



* Between any call and the corresponding return there are at most k context-switches

* Somewhat different from other generalisations of bounded-context.

Bounded Scope Behaviours

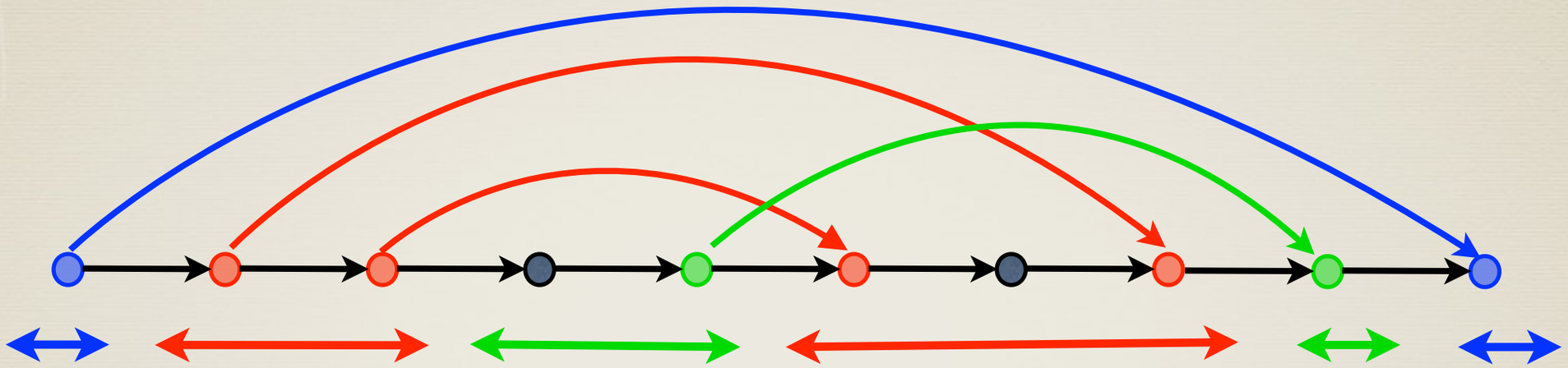


* Between any call and the corresponding return there are at most k context-switches

LaTorreNapoli'11

* Somewhat different from other generalisations of bounded-context.

Bounded Scope Behaviours

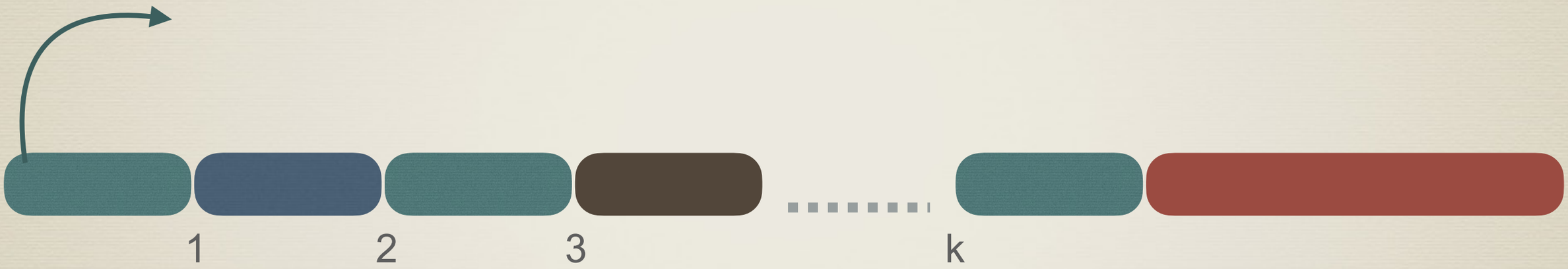


* Between any call and the corresponding return there are at most k context-switches

* Somewhat different from other generalisations of bounded-context.

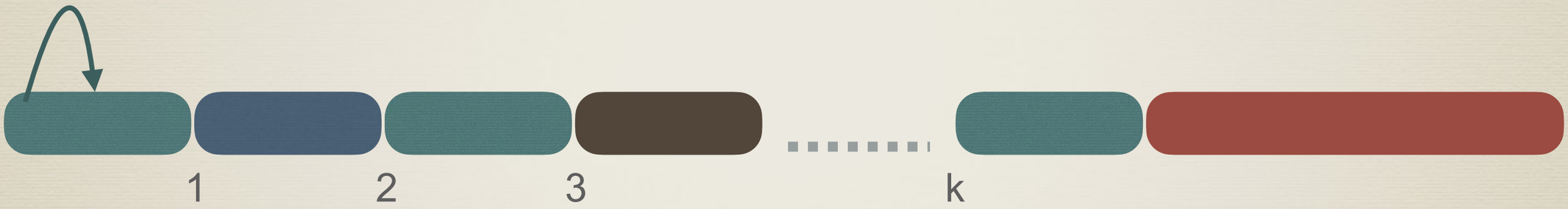
Bounded Scope Behaviours

* Between any call and the corresponding return there are at most k context-switches



Bounded Scope Behaviours

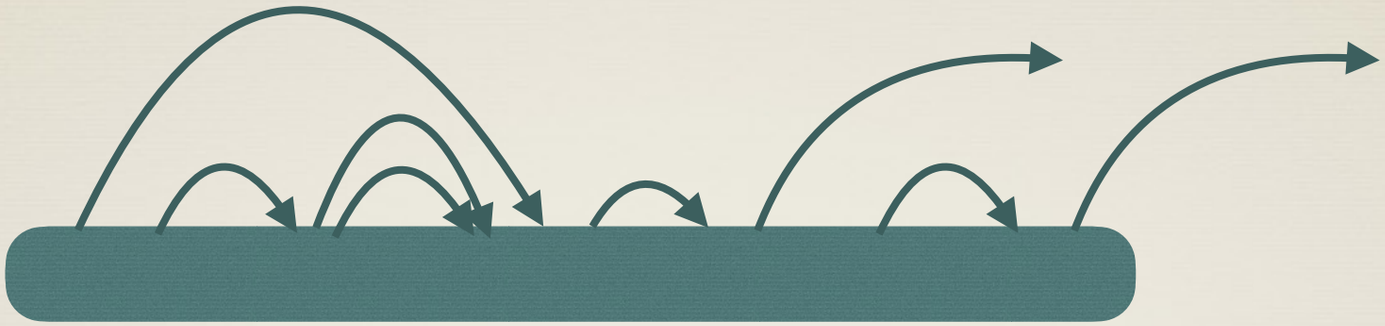
* Between any call and the corresponding return there are at most k context-switches



Case 1: The return is within the same context.

Bounded Scope Behaviours

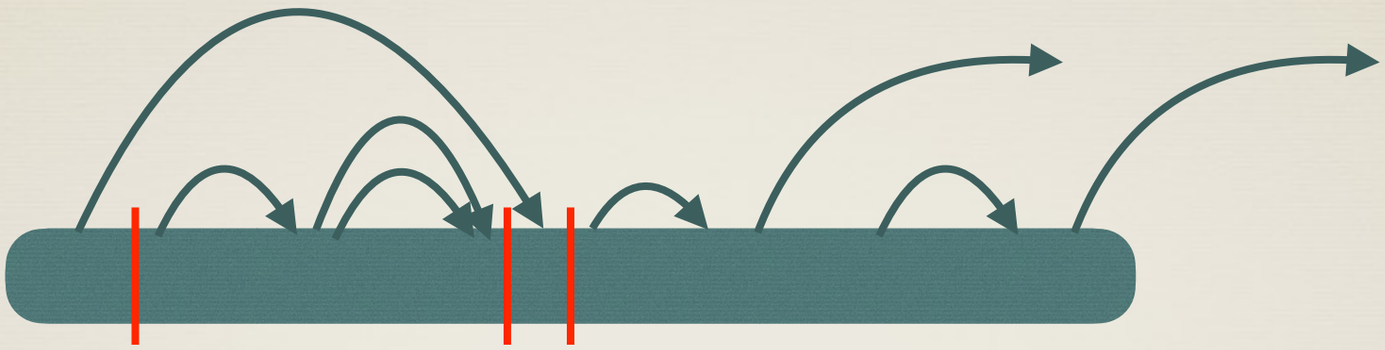
* Between any call and the corresponding return there are at most k context-switches



Case 1: The return is within the same context.

Bounded Scope Behaviours

* Between any call and the corresponding return there are at most k context-switches

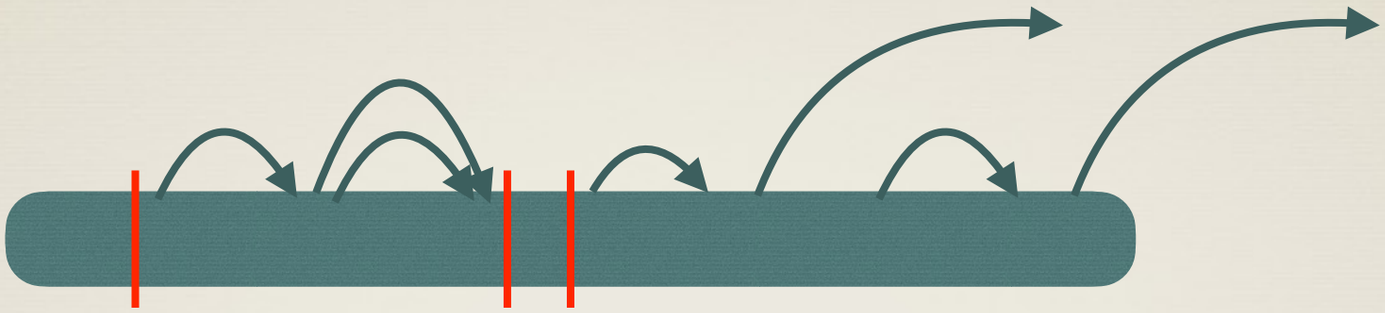


Case 1: The return is within the same context.

Cut this call-return pair. Remove.

Bounded Scope Behaviours

* Between any call and the corresponding return there are at most k context-switches

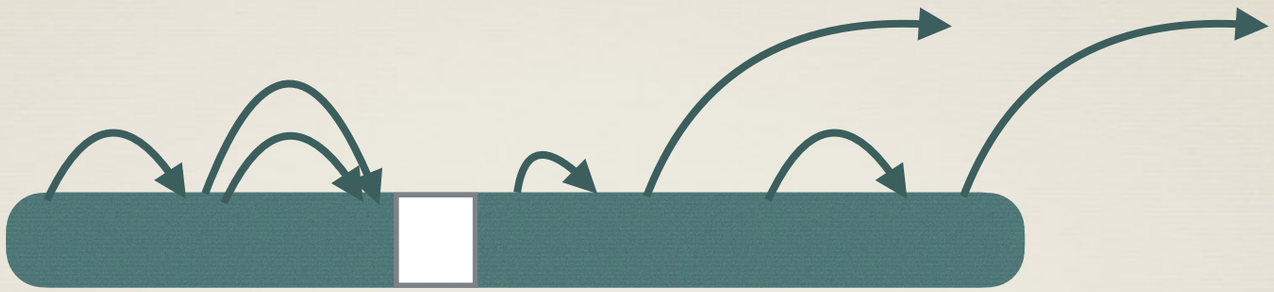


Case 1: The return is within the same context.

Cut this call-return pair. Remove.

Bounded Scope Behaviours

* Between any call and the corresponding return there are at most k context-switches

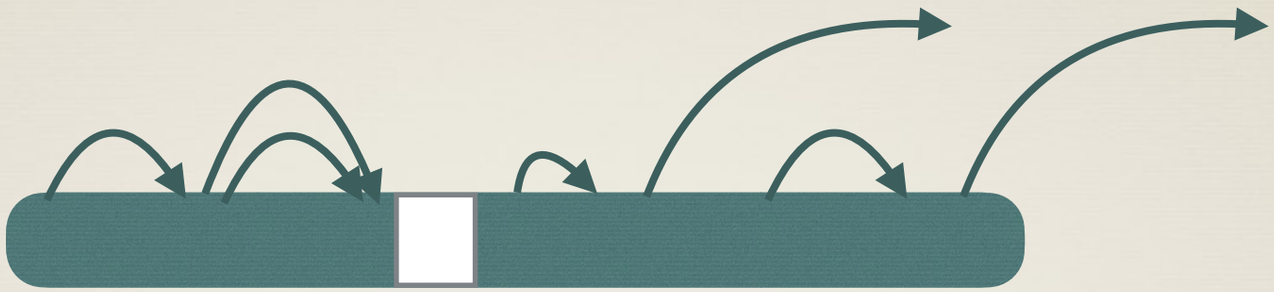


Case 1: The return is within the same context.

First context is left with a hole

Bounded Scope Behaviours

* Between any call and the corresponding return there are at most k context-switches



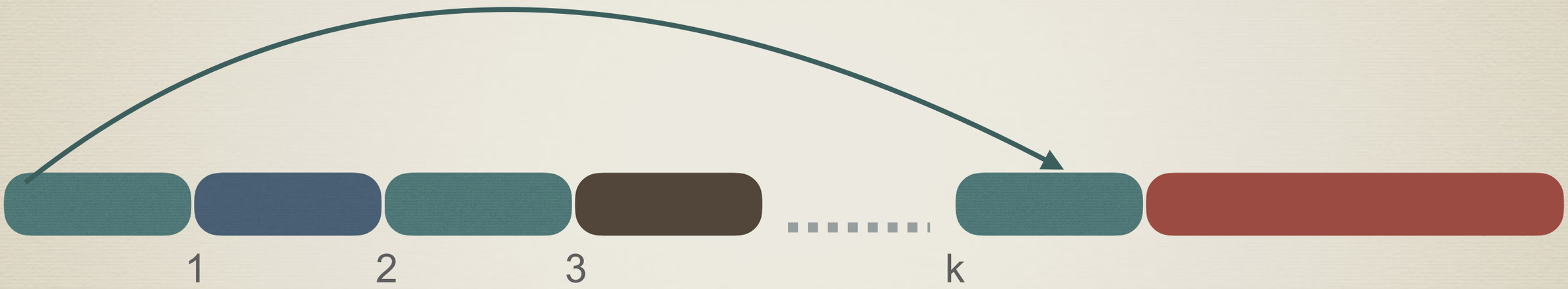
No green nesting edge crosses the hole

Case 1: The return is within the same context.

First context is left with a hole

Bounded Scope Behaviours

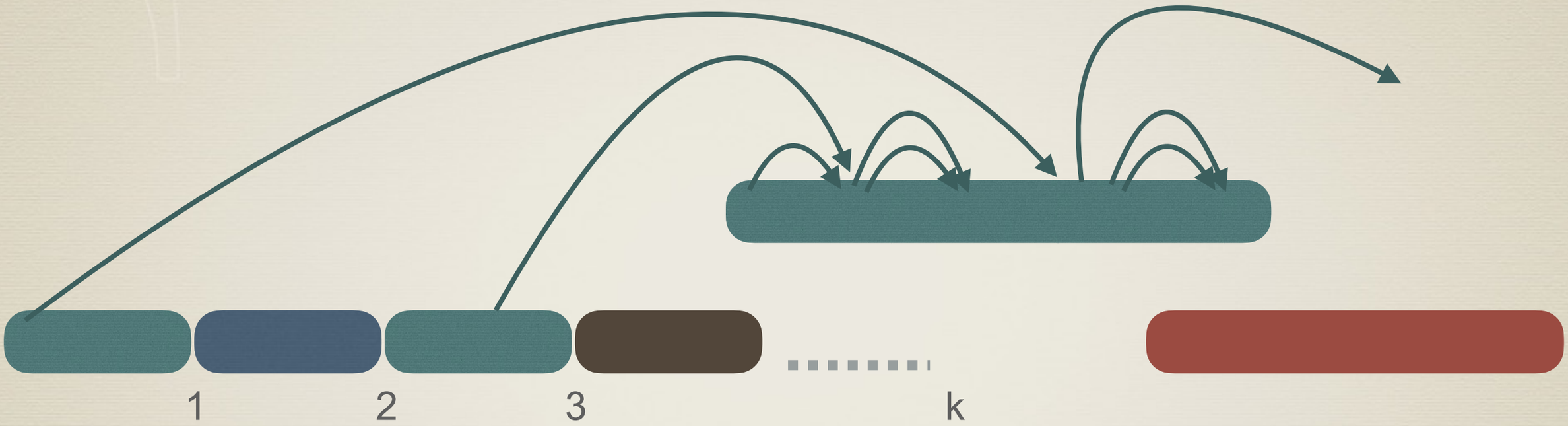
* Between any call and the corresponding return there are at most k context-switches



Case 2: The return is in a different context.

Bounded Scope Behaviours

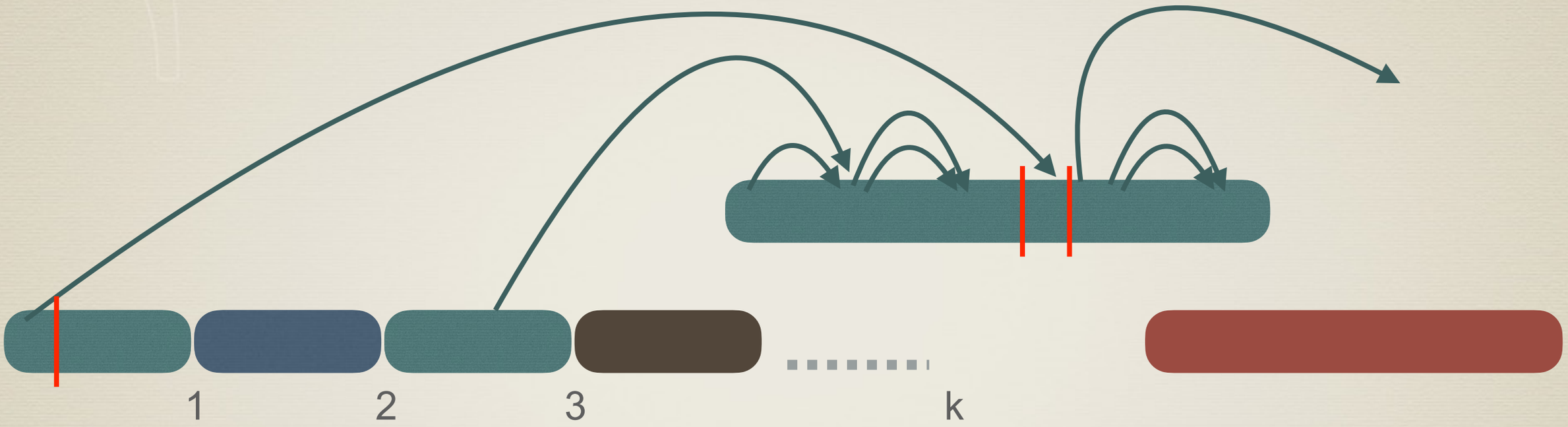
* Between any call and the corresponding return there are at most k context-switches



Case 2: The return is in a different context.

Bounded Scope Behaviours

* Between any call and the corresponding return there are at most k context-switches

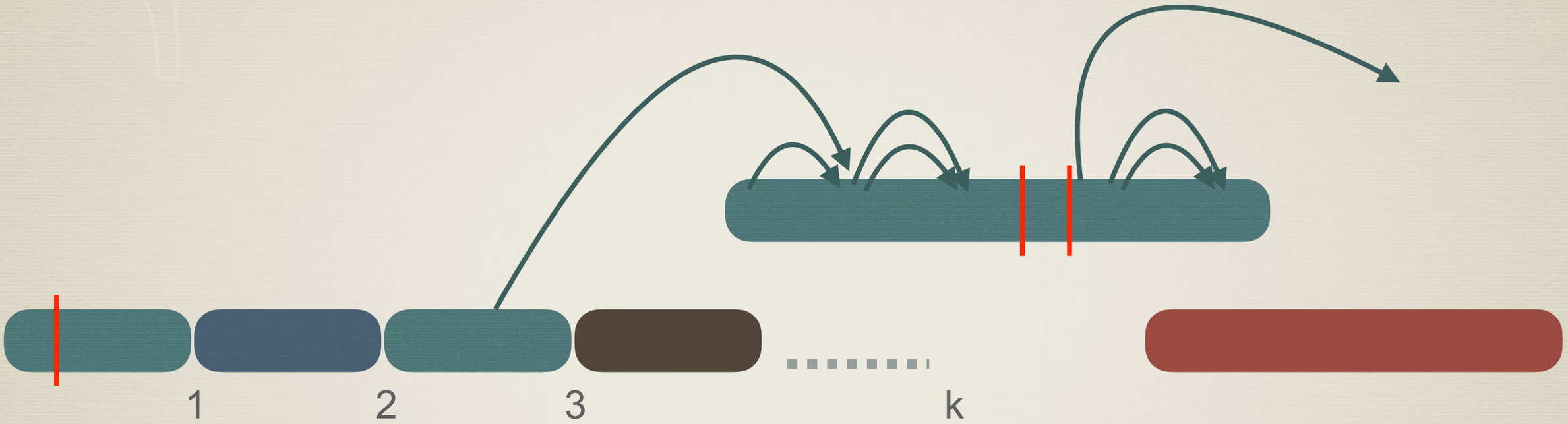


Case 2: The return is in a different context.

Cut this call-return pair. Remove.

Bounded Scope Behaviours

* Between any call and the corresponding return there are at most k context-switches

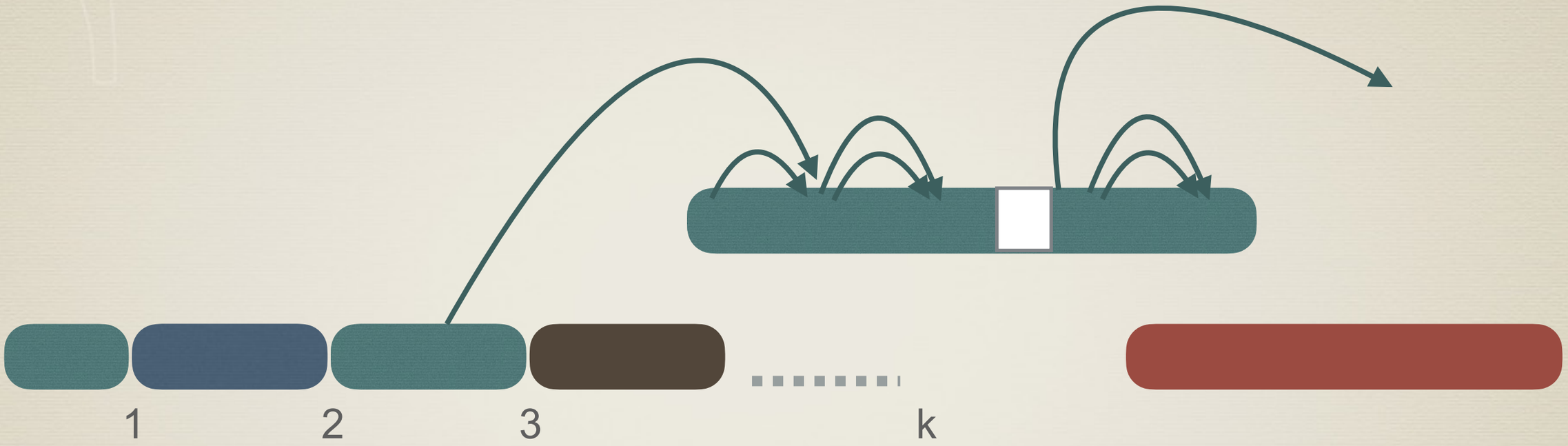


Case 2: The return is in a different context.

Cut this call-return pair. Remove.

Bounded Scope Behaviours

* Between any call and the corresponding return there are at most k context-switches

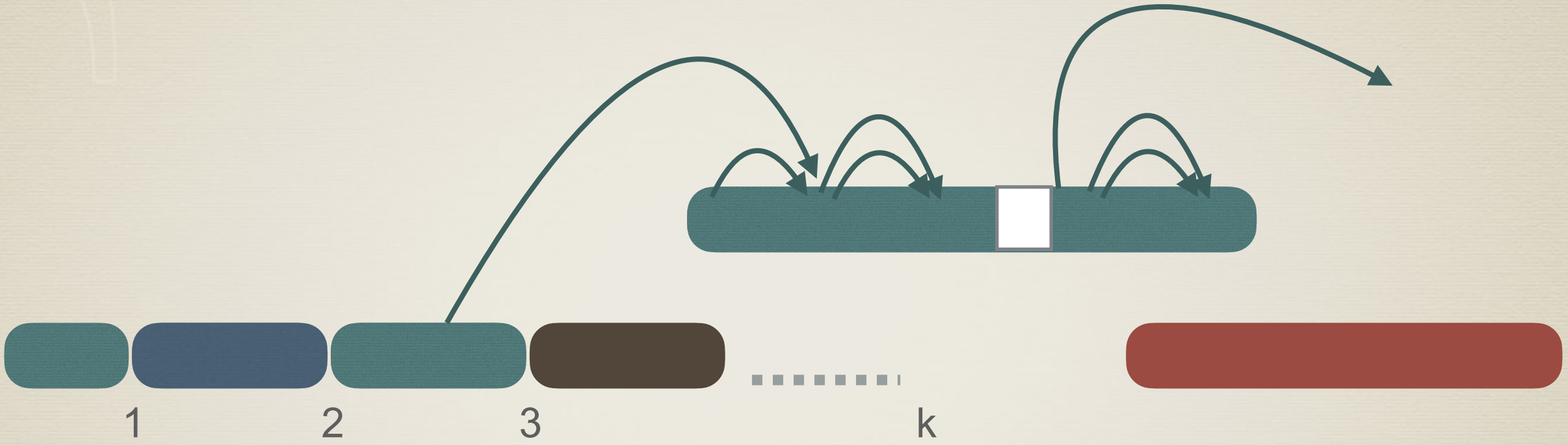


Case 2: The return is in a different context.

The called context is left with a hole.

Bounded Scope Behaviours

* Between any call and the corresponding return there are at most k context-switches



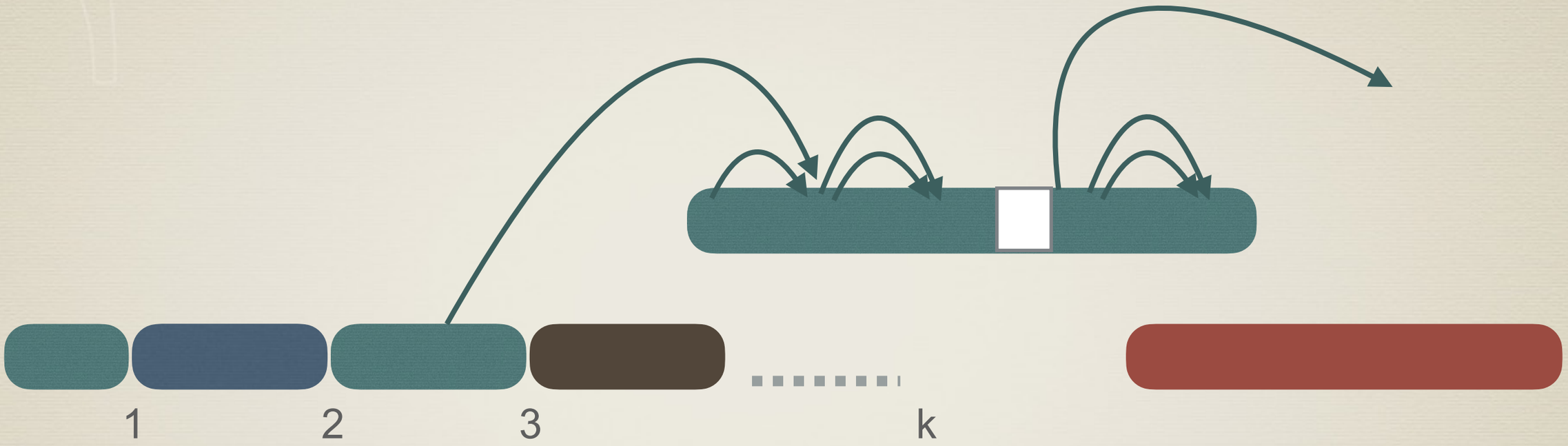
No green nesting edge crosses the hole

Case 2: The return is in a different context.

The called context is left with a hole.

Bounded Scope Behaviours

* Between any call and the corresponding return there are at most k context-switches



Case 2: The return is in a different context.

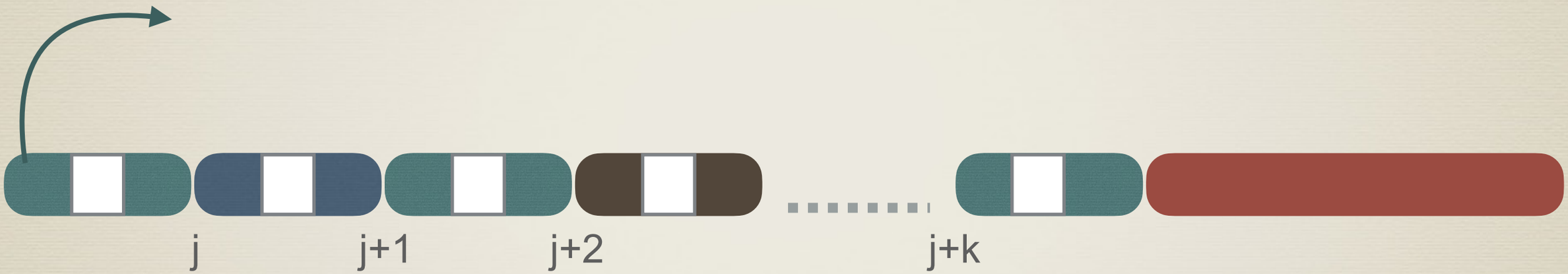
The called context is left with a hole.

Bounded Scope Behaviours

Maintaining invariantly that

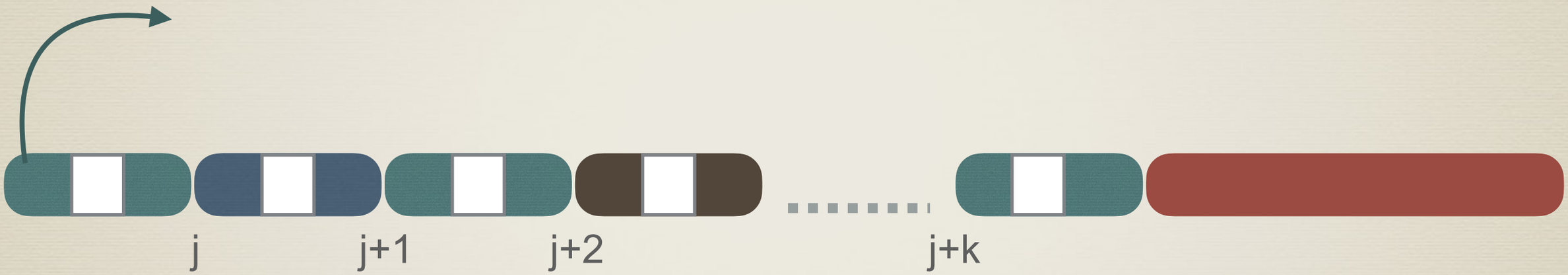
* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context ...

we will show that we can remove one more edge.



Bounded Scope Behaviours

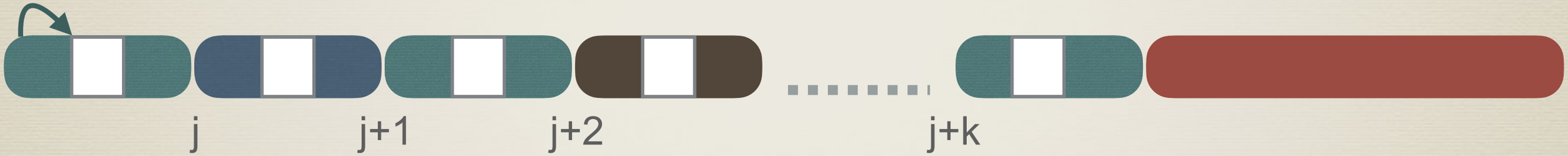
* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context



Case 1: The return is within the same context.

Bounded Scope Behaviours

* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context

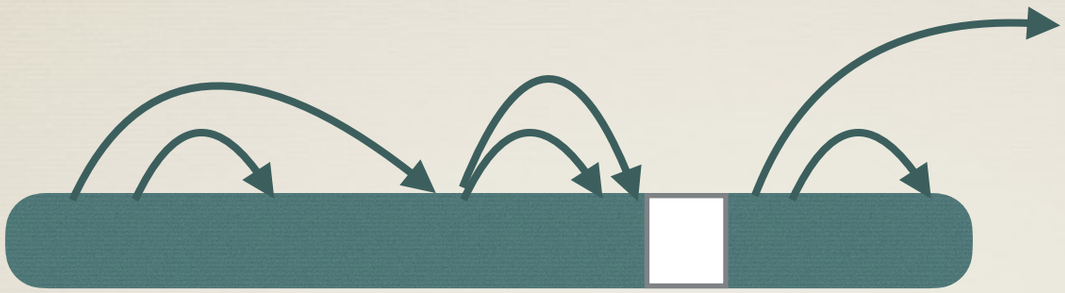


Case 1: The return is within the same context.

The edge cannot cross the hole.

Bounded Scope Behaviours

* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context

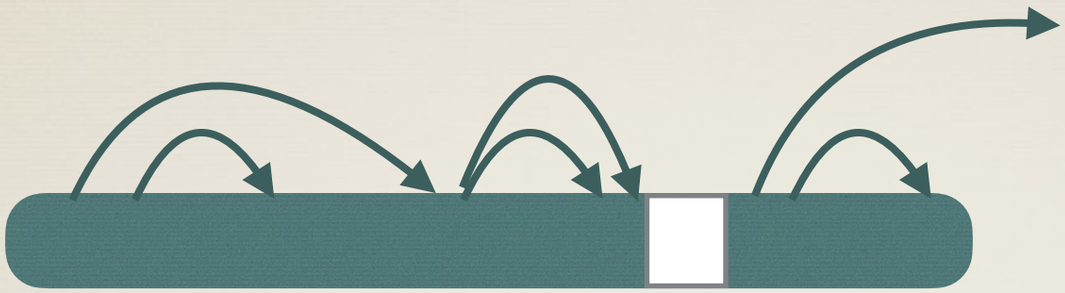


Case 1: The return is within the same context.

The edge cannot cross the hole.

Bounded Scope Behaviours

* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context



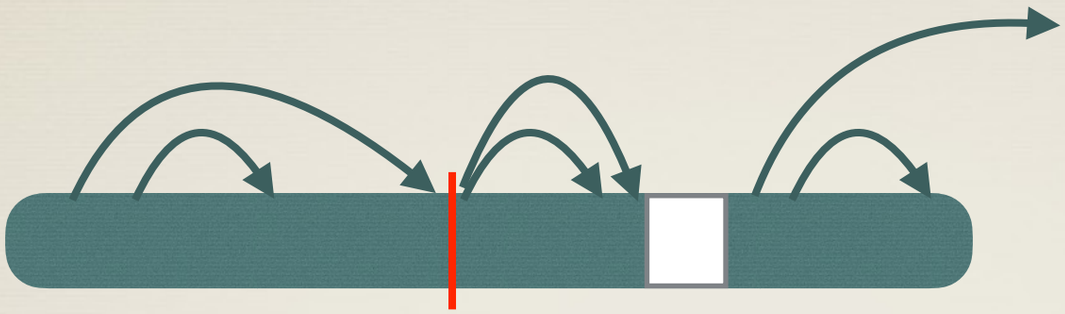
Case 1: The return is within the same context.

The edge cannot cross the hole.

Between target and holethere is a complete nested word.

Bounded Scope Behaviours

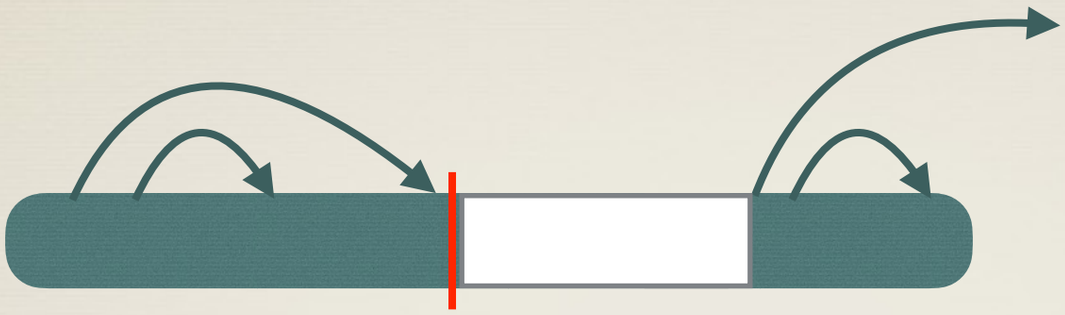
* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context



Case 1: The return is within the same context.
Cut and remove the nested word.

Bounded Scope Behaviours

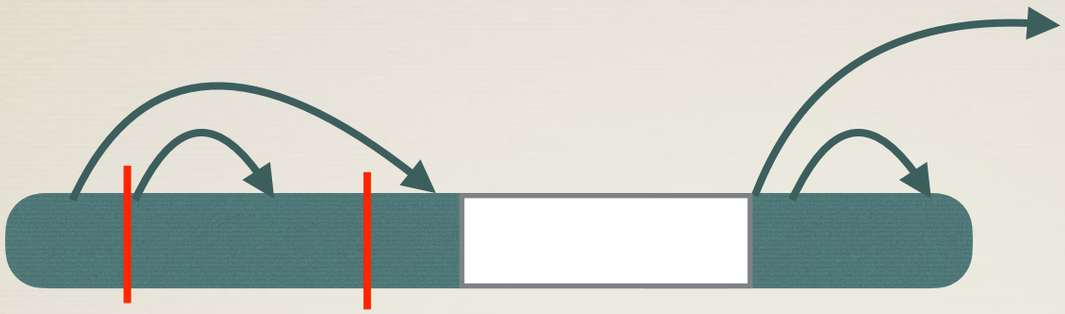
* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context



Case 1: The return is within the same context.
Cut and remove the nested word.

Bounded Scope Behaviours

* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context

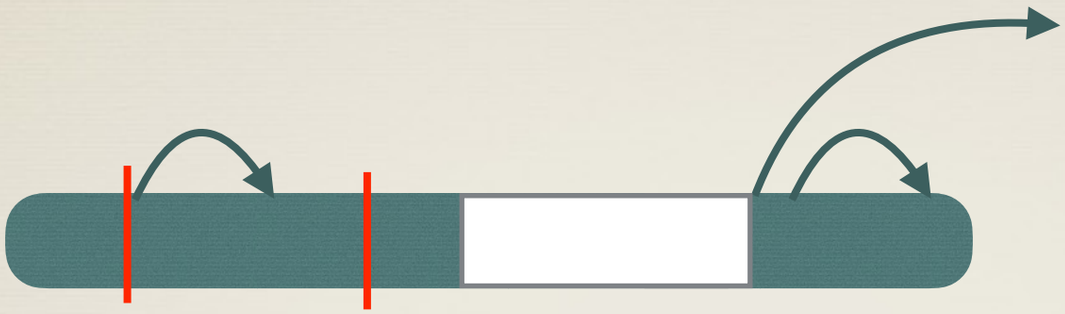


Case 1: The return is within the same context.

Cut this call-return pair. Remove.

Bounded Scope Behaviours

* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context

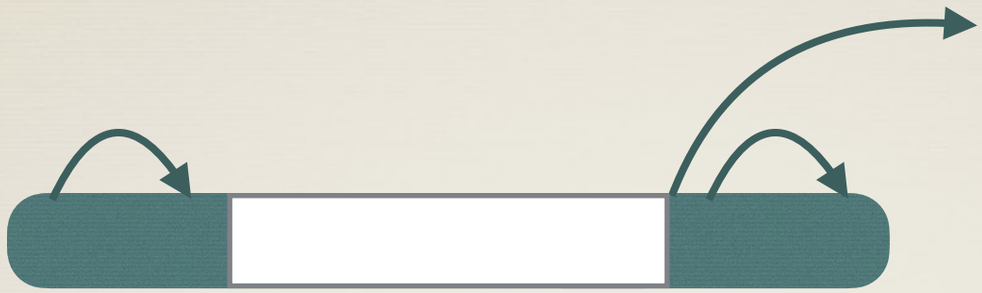


Case 1: The return is within the same context.

Cut this call-return pair. Remove.

Bounded Scope Behaviours

* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context

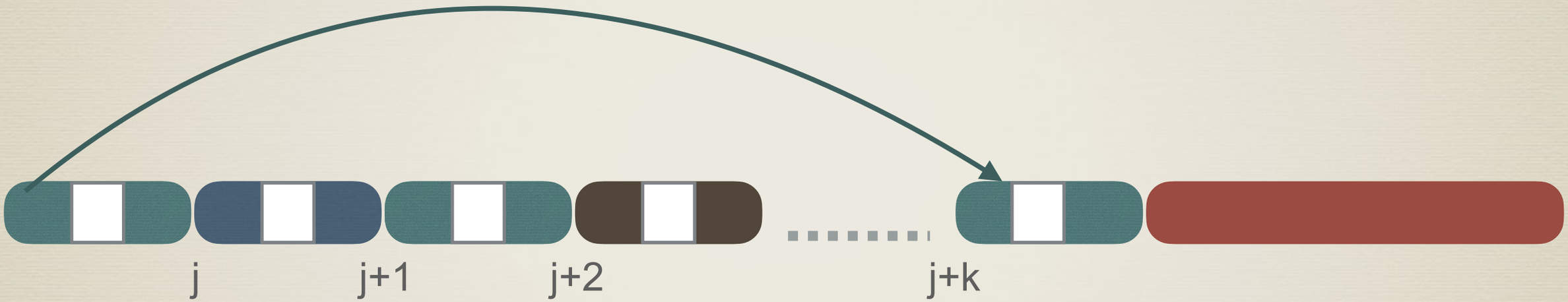


Case 1: The return is within the same context.

Nesting edge removed maintaining invariant.

Bounded Scope Behaviours

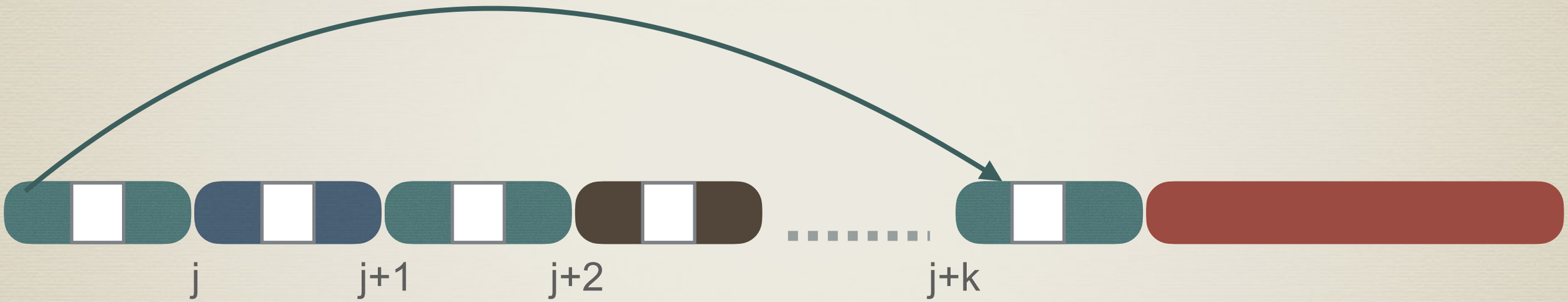
* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context



Case 2: The return is in a different context.

Bounded Scope Behaviours

* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context

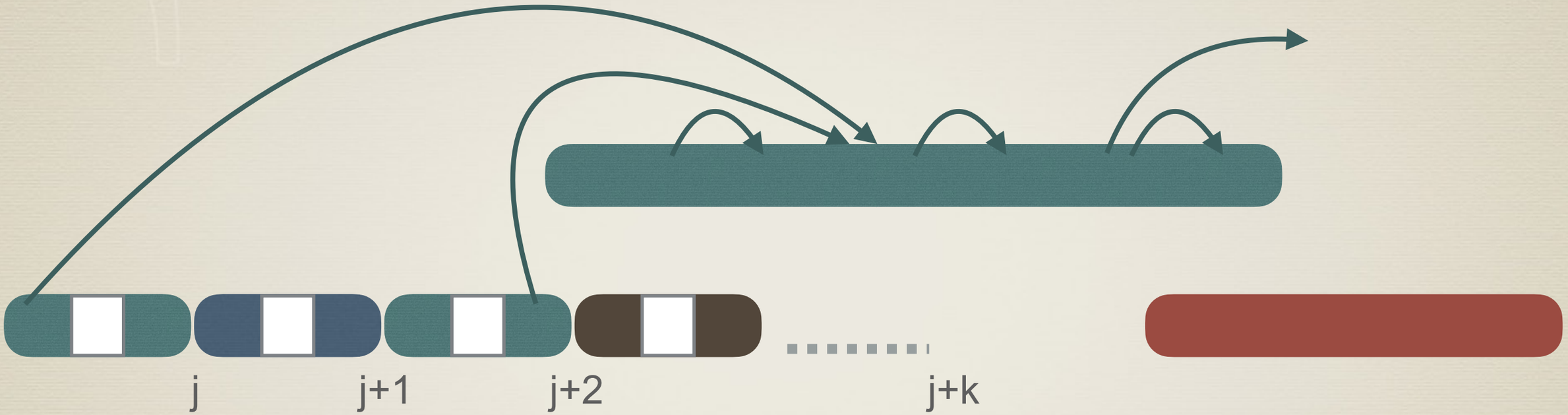


Case 2: The return is in a different context.

Target is before the hole (if any) in that context.

Bounded Scope Behaviours

* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context

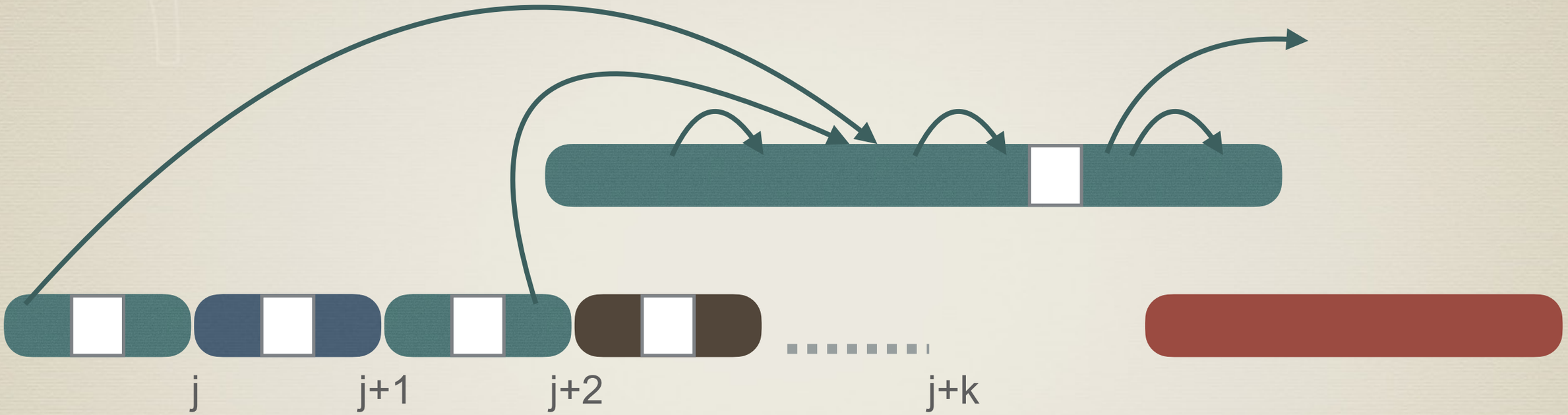


Case 2: The return is in a different context.

If there is no hole, do as in the first case, introduce hole and remove edge.

Bounded Scope Behaviours

* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context

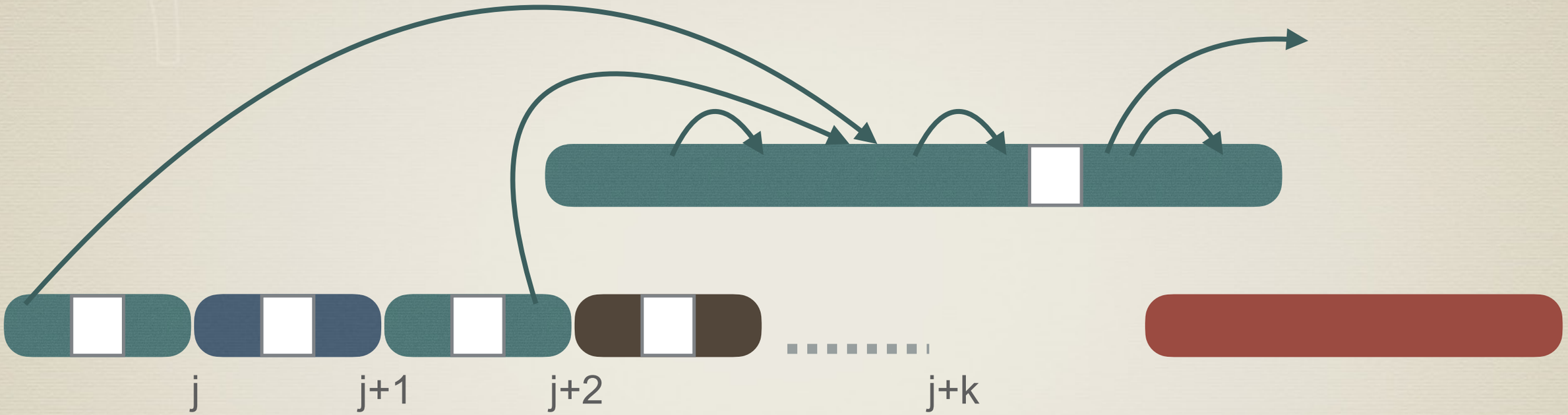


Case 2: The return is in a different context.

If there is a hole: Target is before the hole.

Bounded Scope Behaviours

* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context

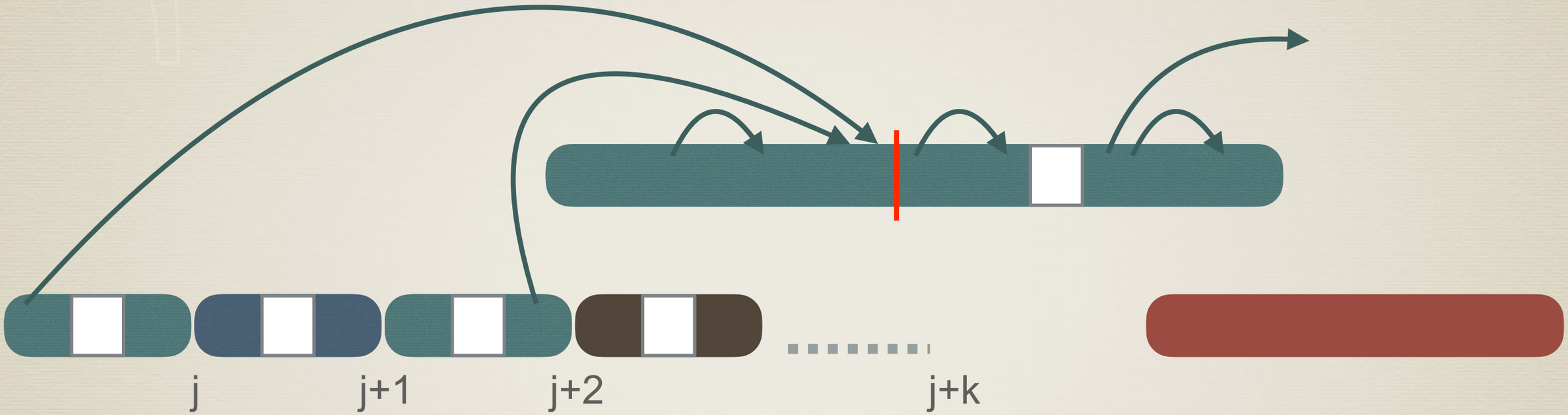


Case 2: The return is in a different context.

If there is a hole: between the target and the hole is a nested word

Bounded Scope Behaviours

* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context

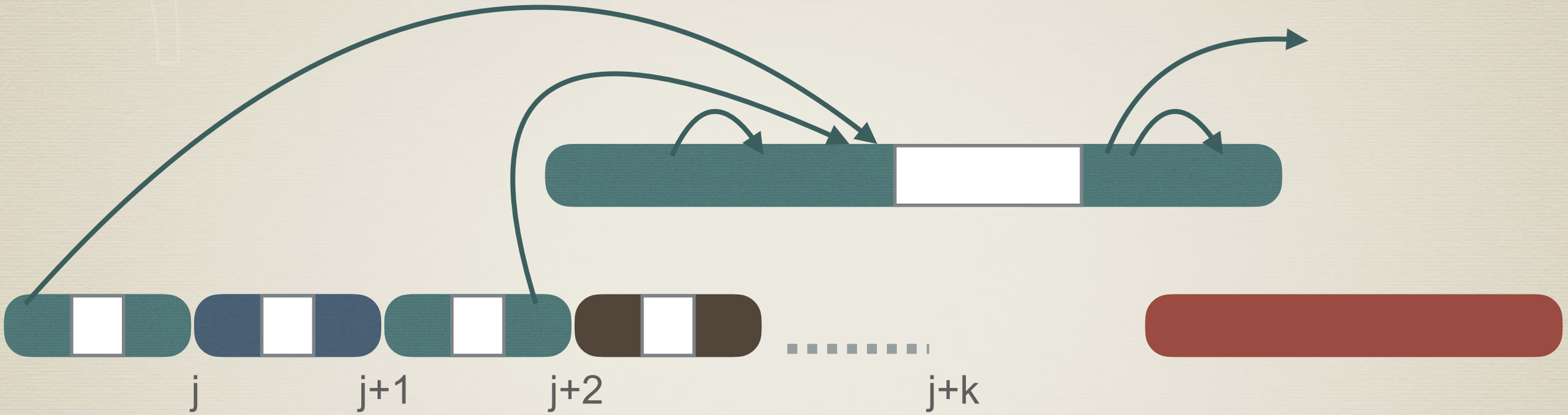


Case 2: The return is in a different context.

If there is a hole: Cut and remove nested word to expand the hole

Bounded Scope Behaviours

* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context

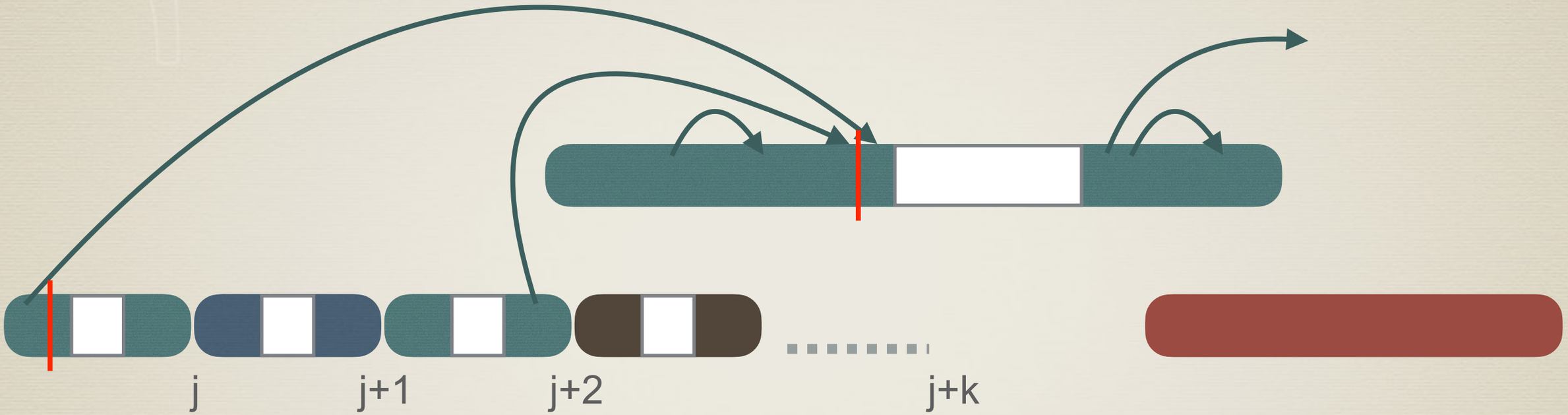


Case 2: The return is in a different context.

If there is a hole: Cut and remove nested word to expand the hole

Bounded Scope Behaviours

* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context

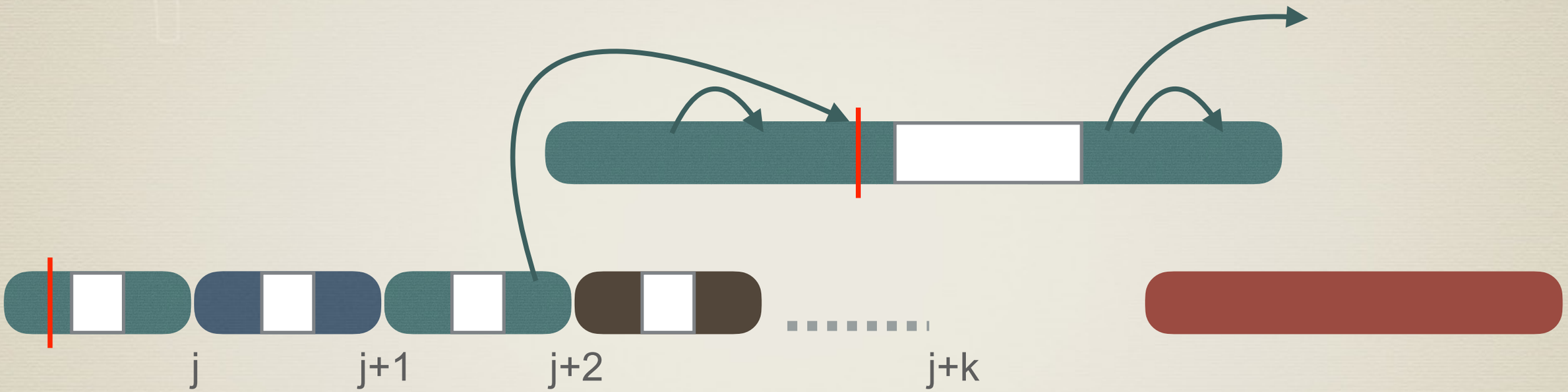


Case 2: The return is in a different context.

If there is a hole: Cut and remove the nesting edge maintaining the invariant

Bounded Scope Behaviours

* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context

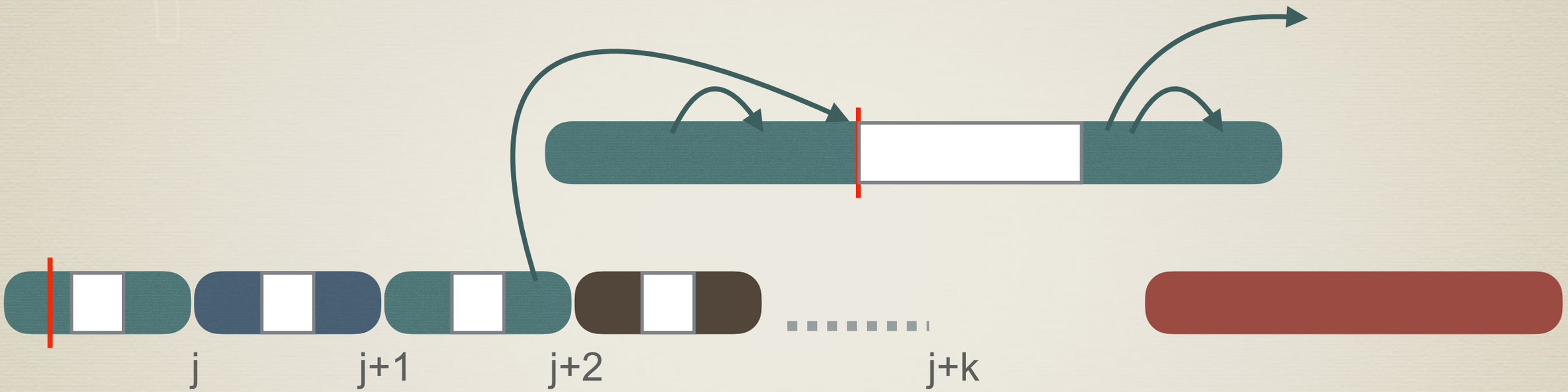


Case 2: The return is in a different context.

If there is a hole: Cut and remove the nesting edge maintaining the invariant

Bounded Scope Behaviours

* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context

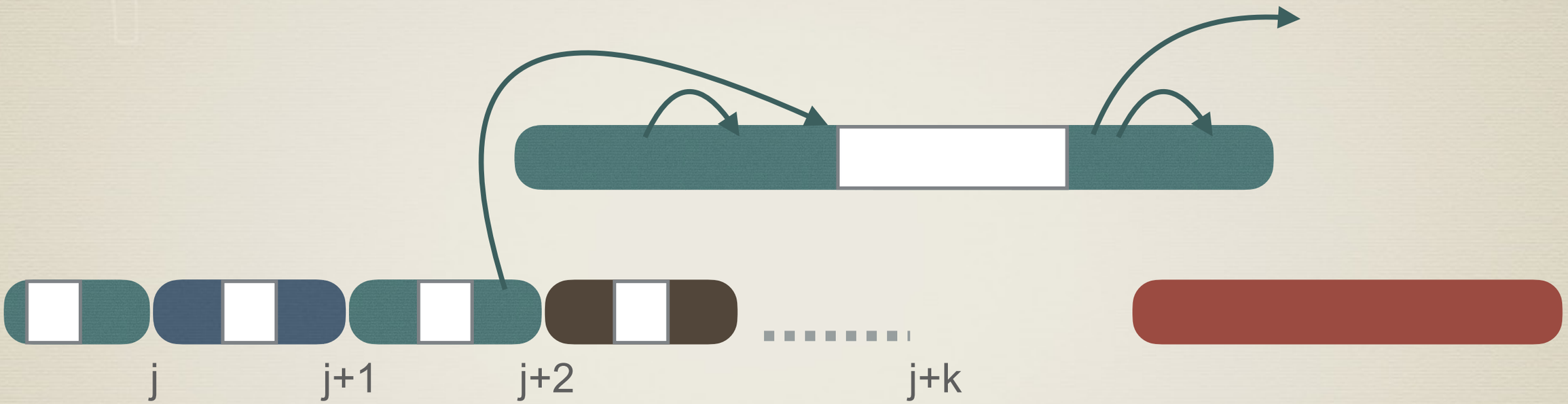


Case 2: The return is in a different context.

If there is a hole: Cut and remove the nesting edge maintaining the invariant

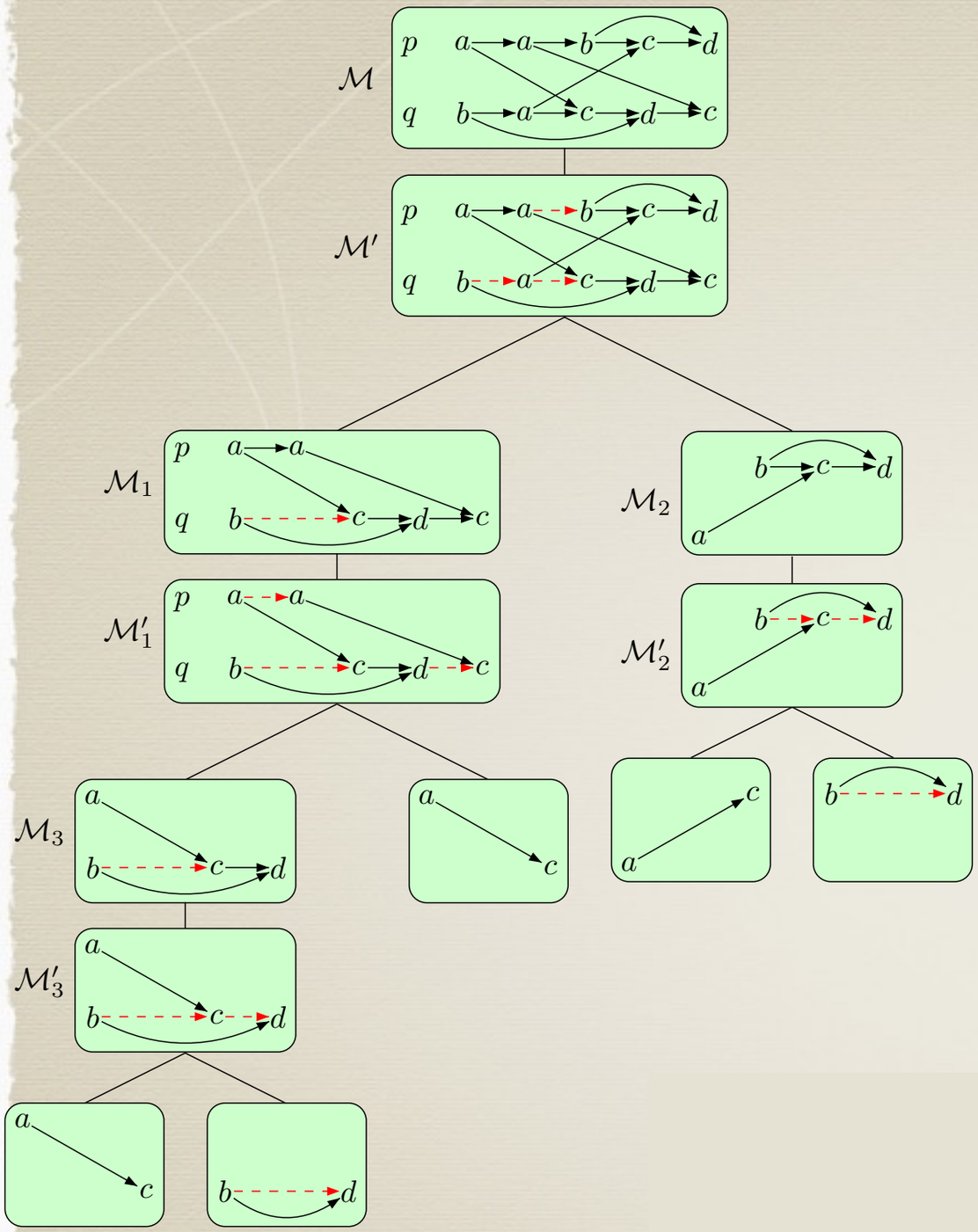
Bounded Scope Behaviours

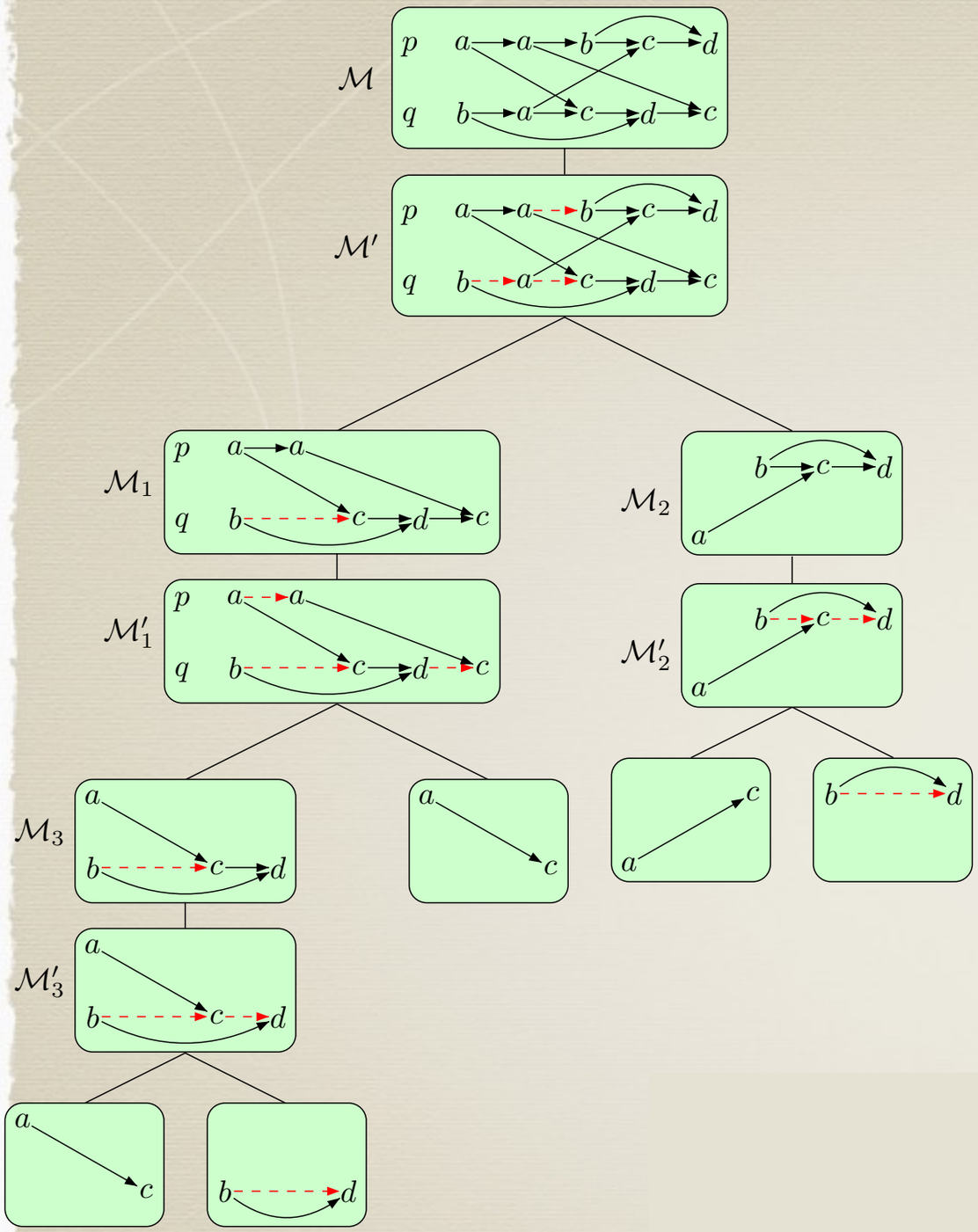
* we have at most 1 hole each in the first k contexts
* no green edge crosses a hole in a green context



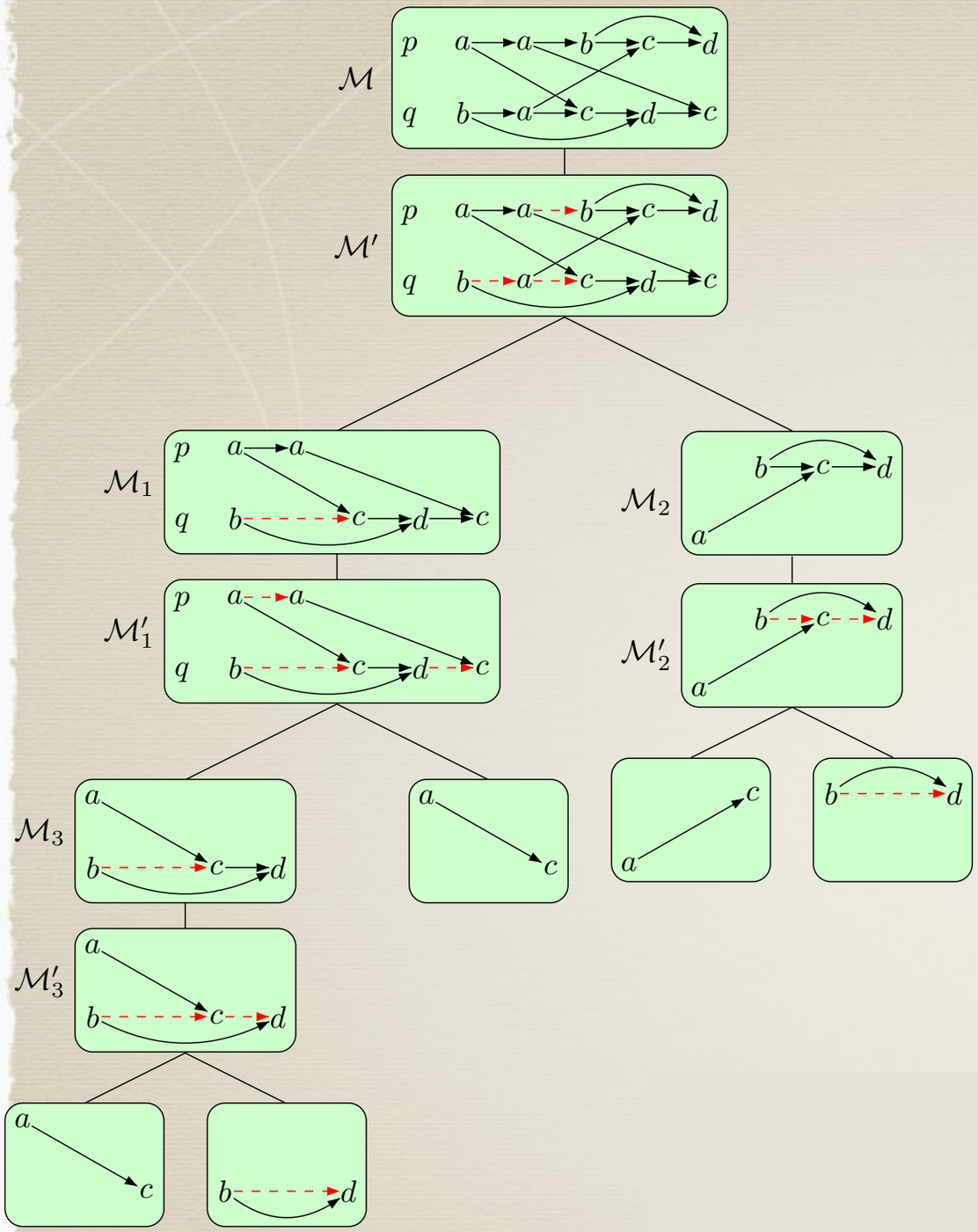
Case 2: The return is in a different context.

If there is a hole: Cut and remove the nesting edge maintaining the invariant


$\mathcal{M}$
$\mathcal{M}'$
$\mathcal{M}_1$
$\mathcal{M}_2$
$\mathcal{M}'_1$
$\mathcal{M}'_2$
$\mathcal{M}_3$
$\mathcal{M}'_3$

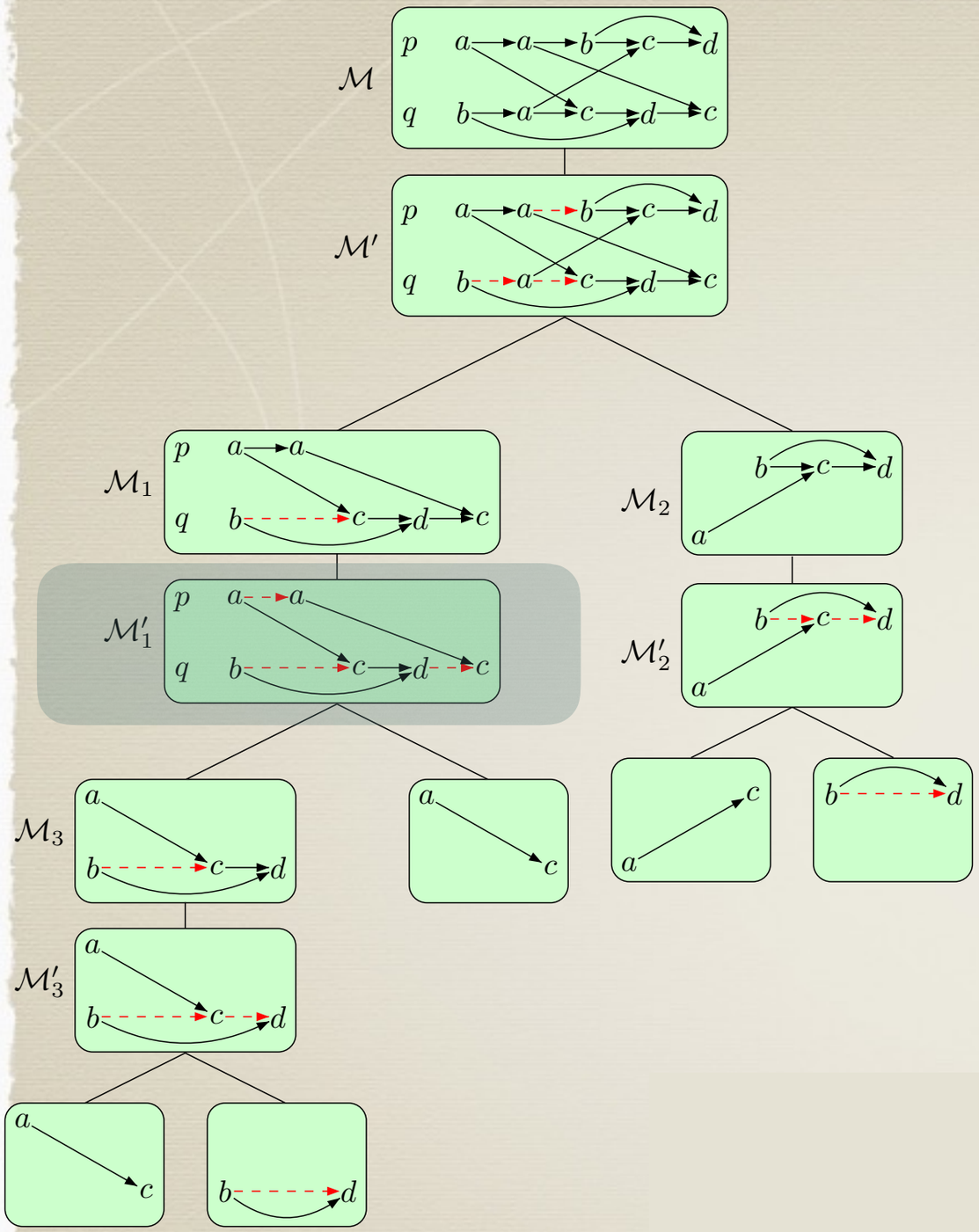


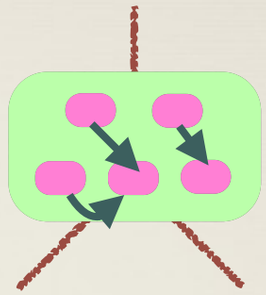
Decomposition trees are over an infinite alphabet



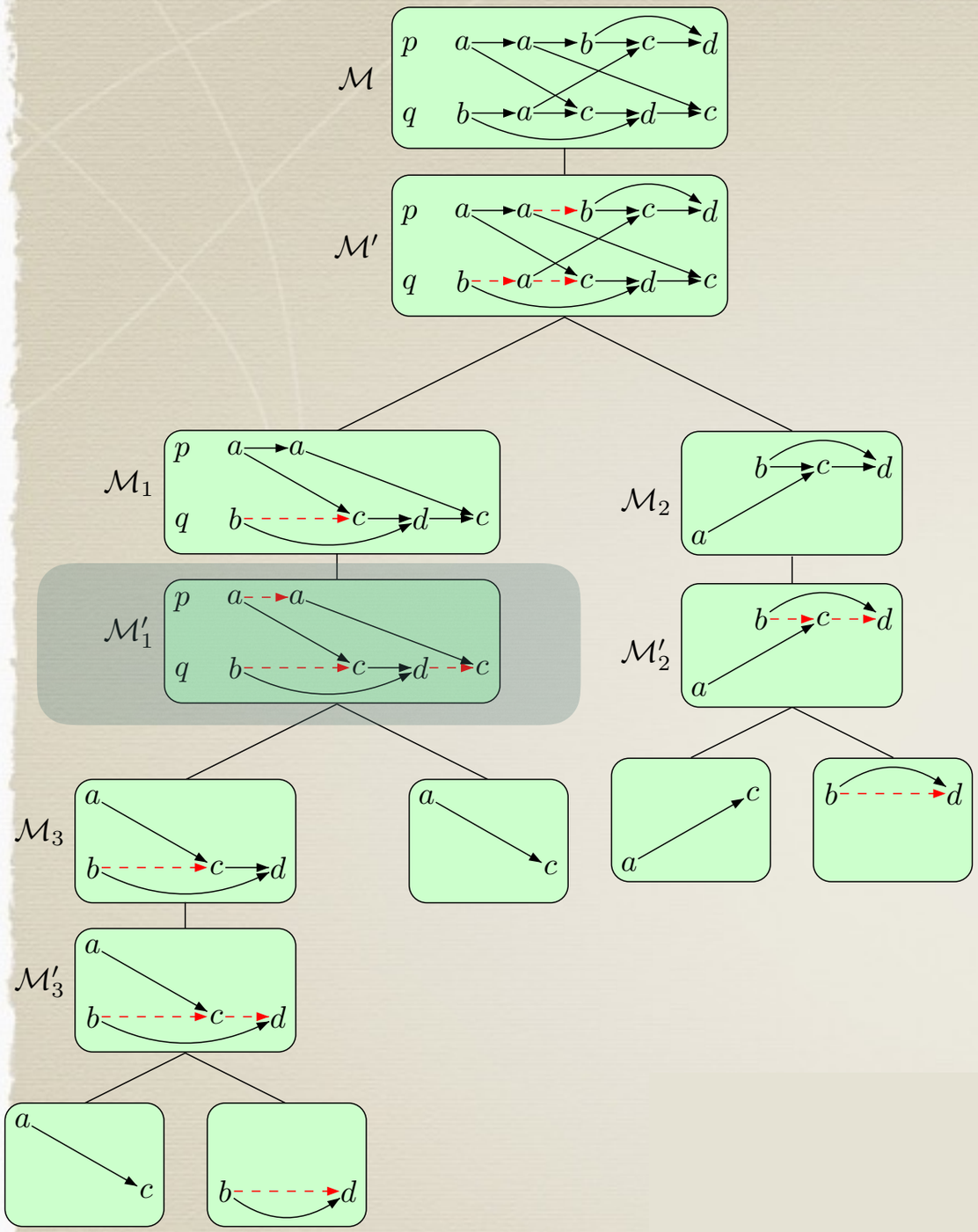
Decomposition trees are over an infinite alphabet

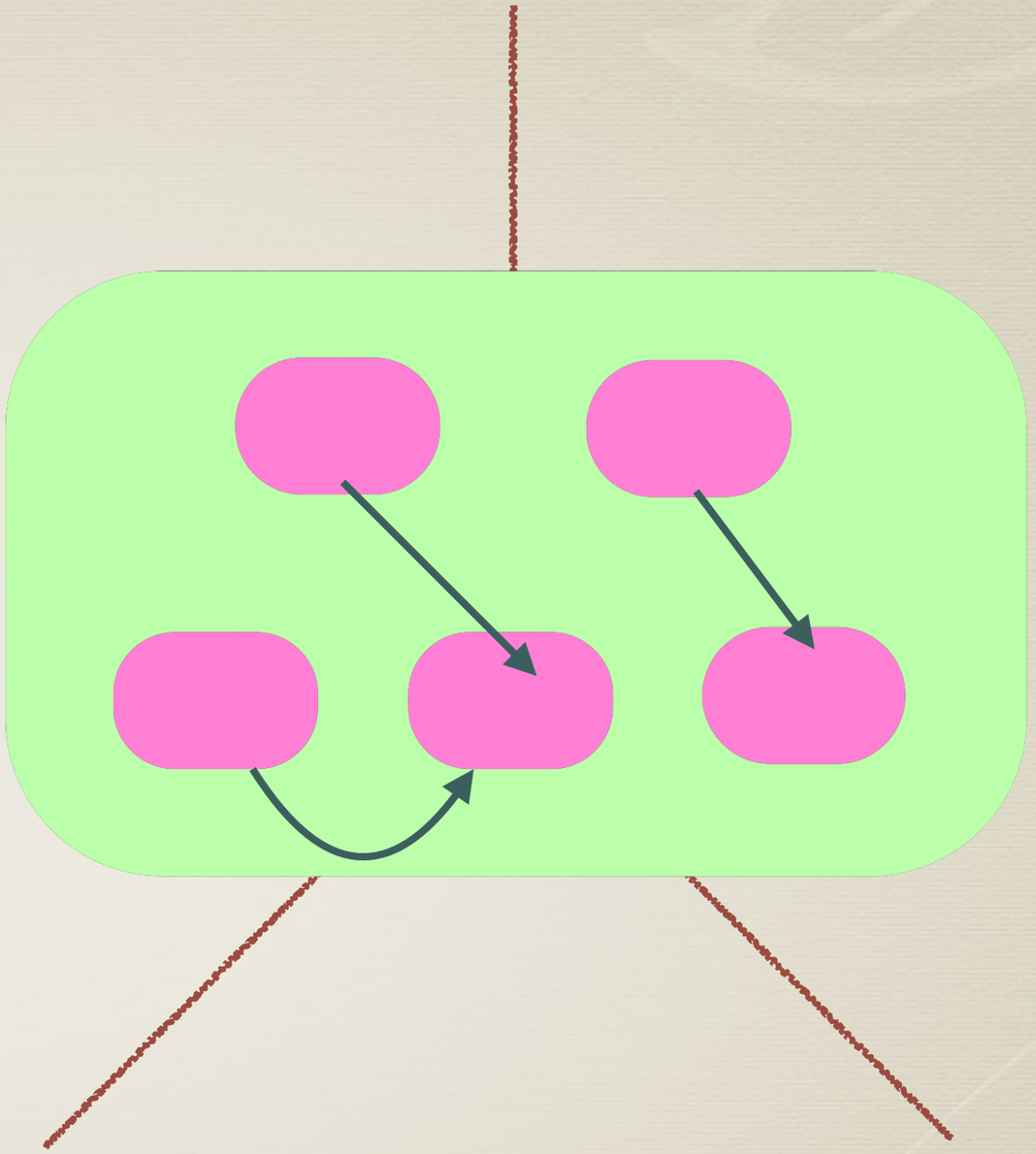
How do we convert this to a finitely labelled tree? (from which the original can be reconstructed)





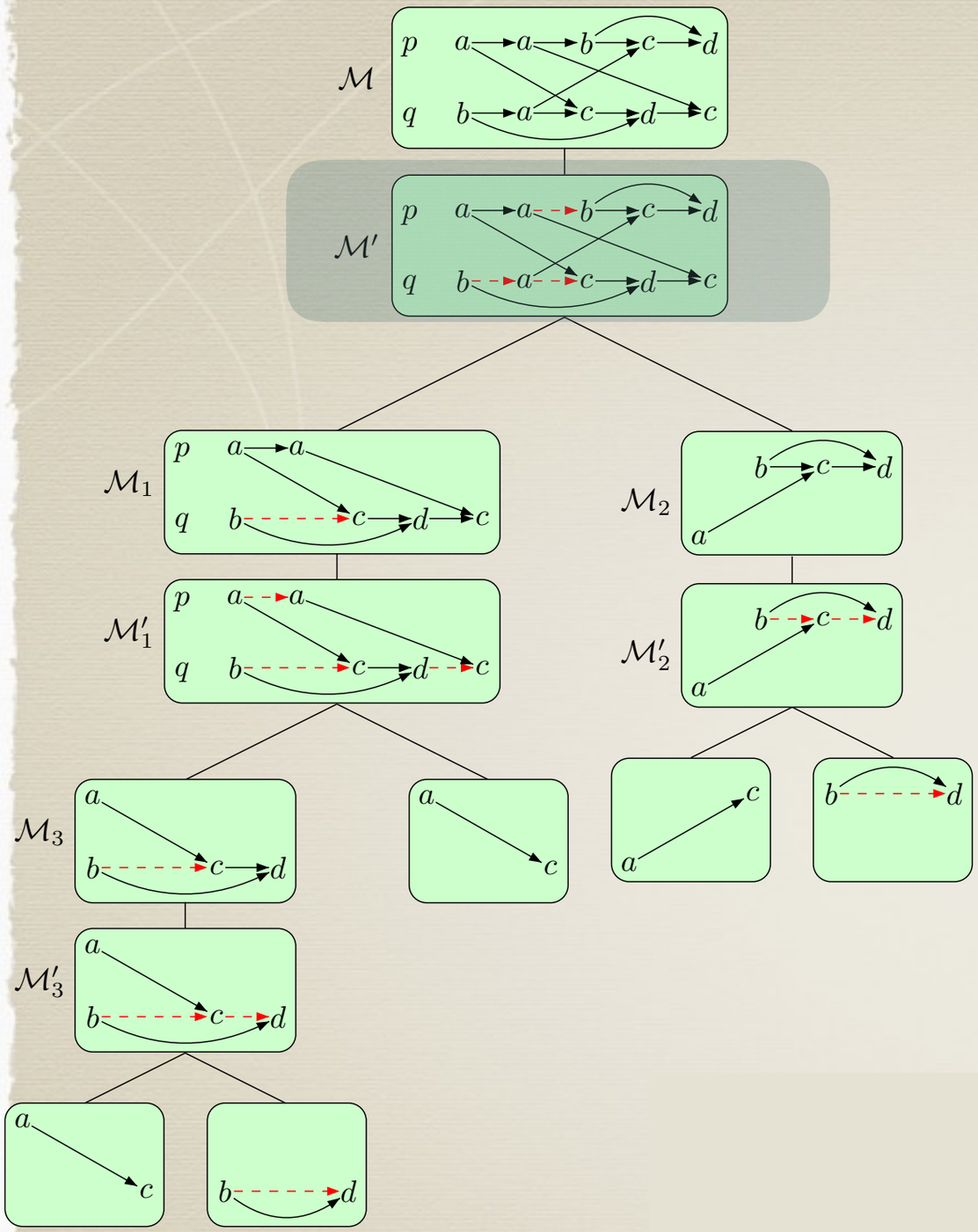
How do we convert this to a finitely labelled tree? (from which the original can be reconstructed)

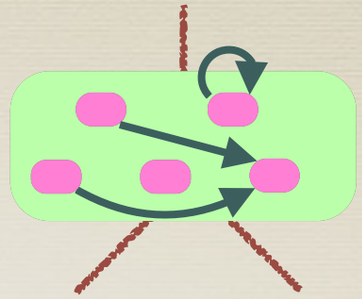




One node for each component on each process.

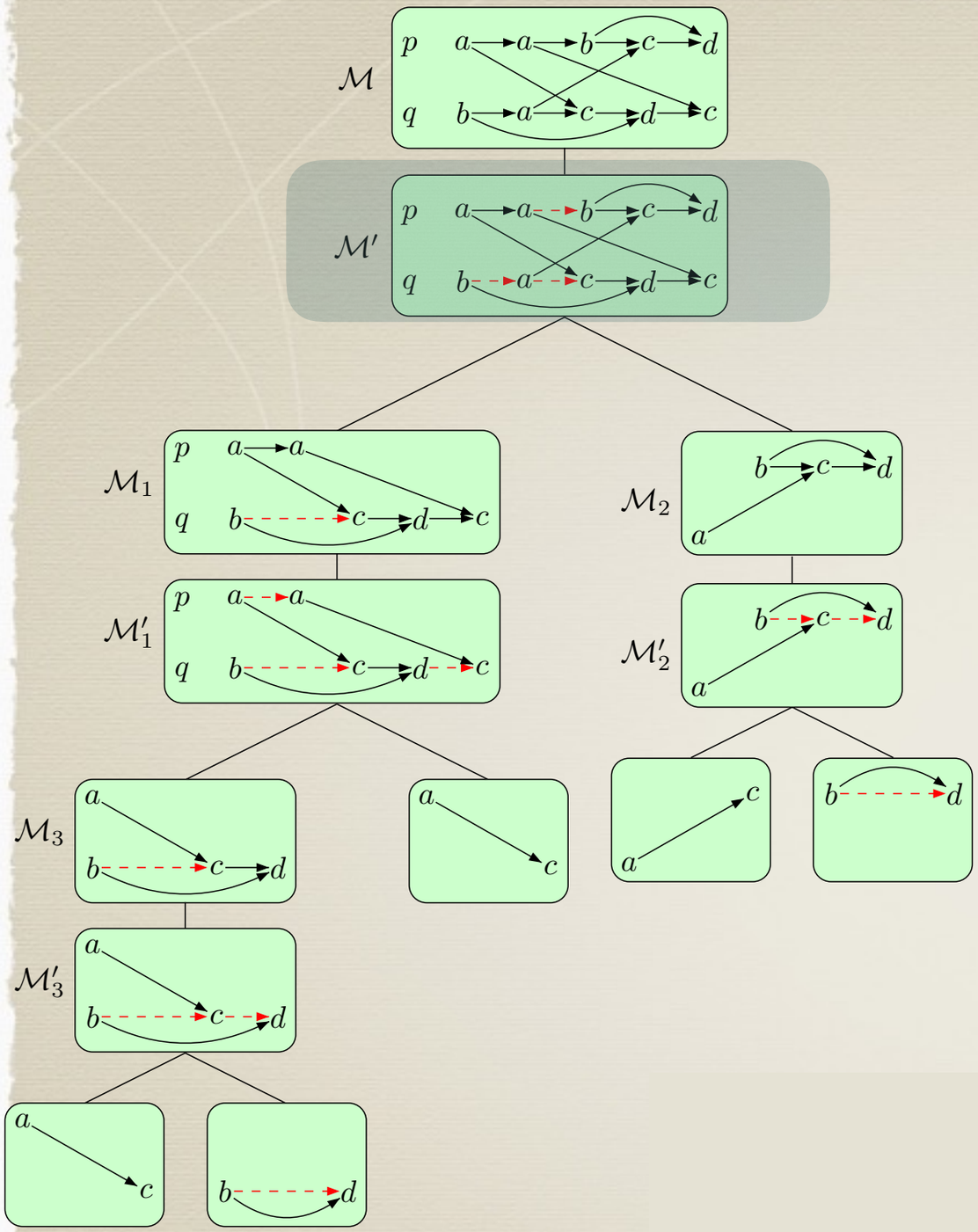
One abstract edge for each data-structure

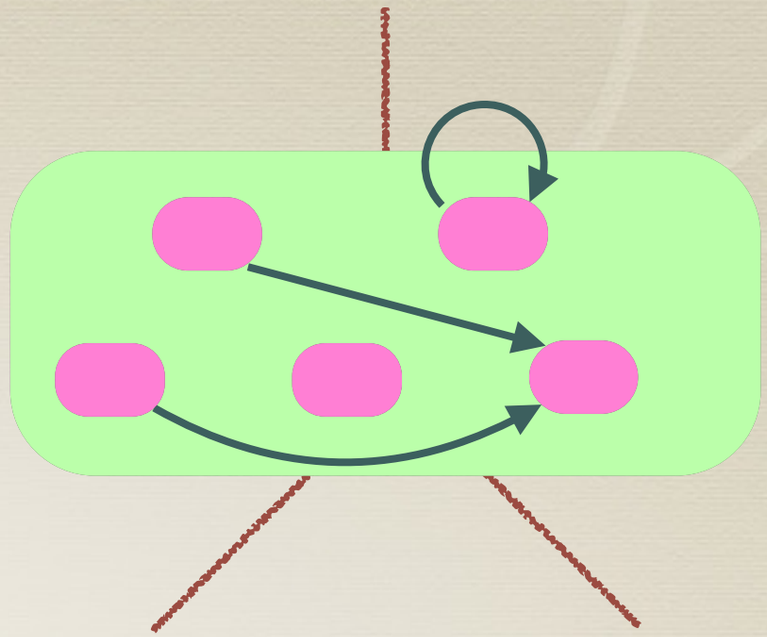




One node for each component on each process.

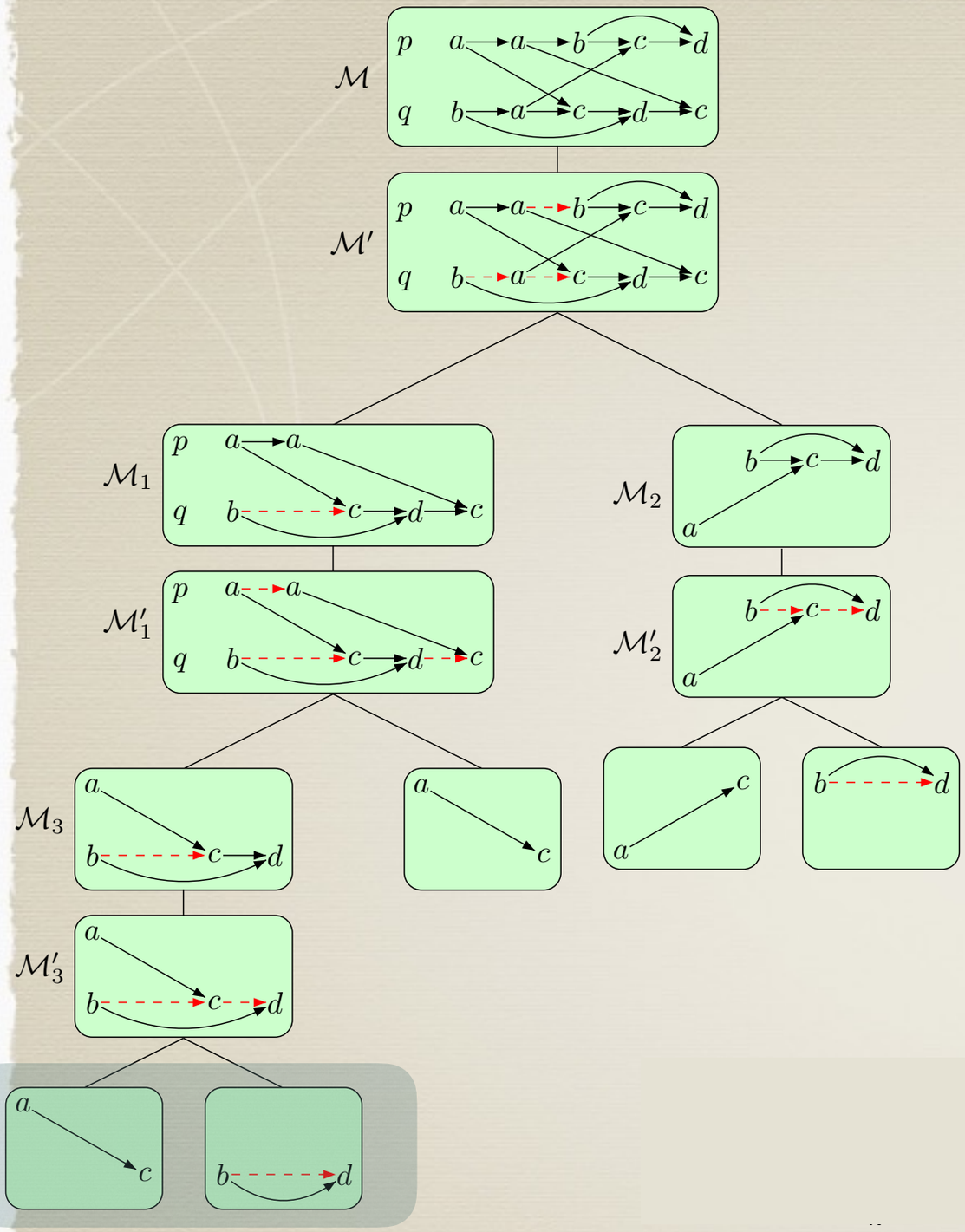
One abstract edge for each data-structure

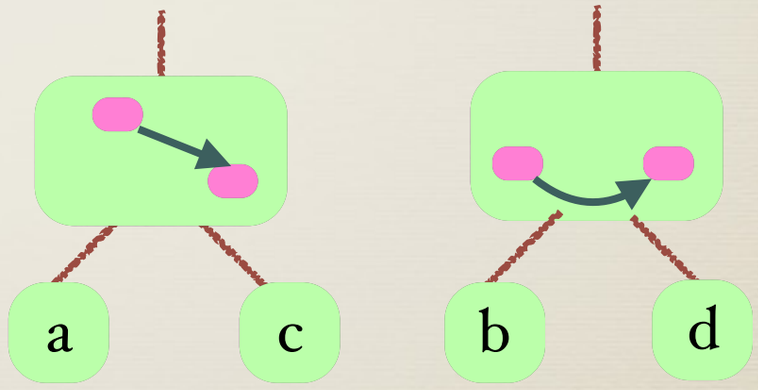




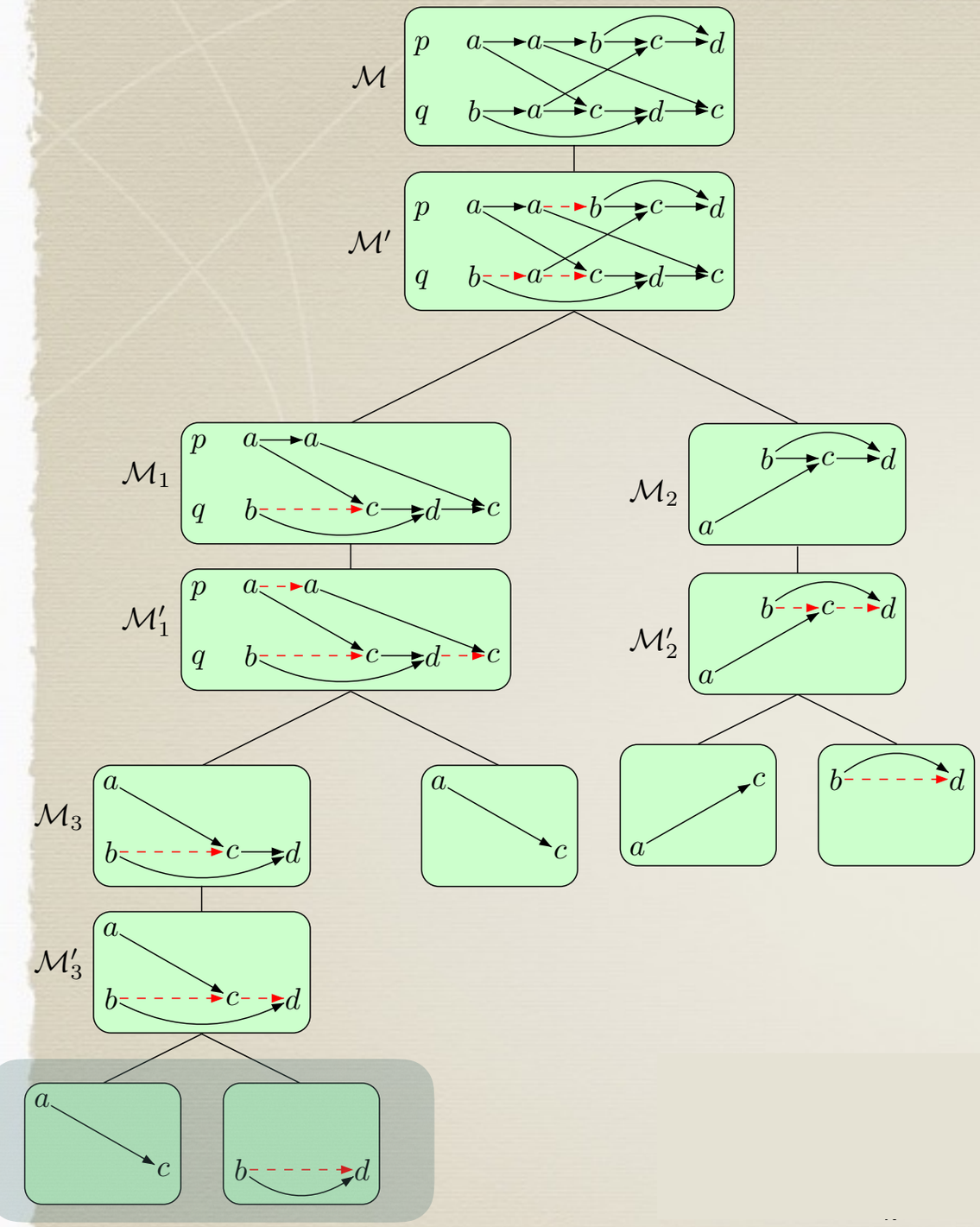
One node for each component on each process.

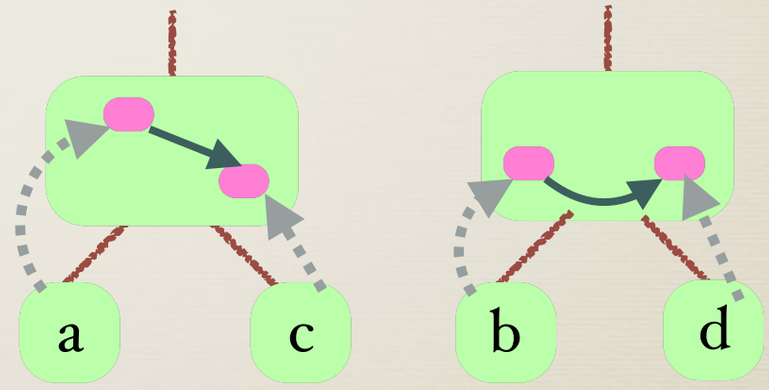
One abstract edge for each data-structure





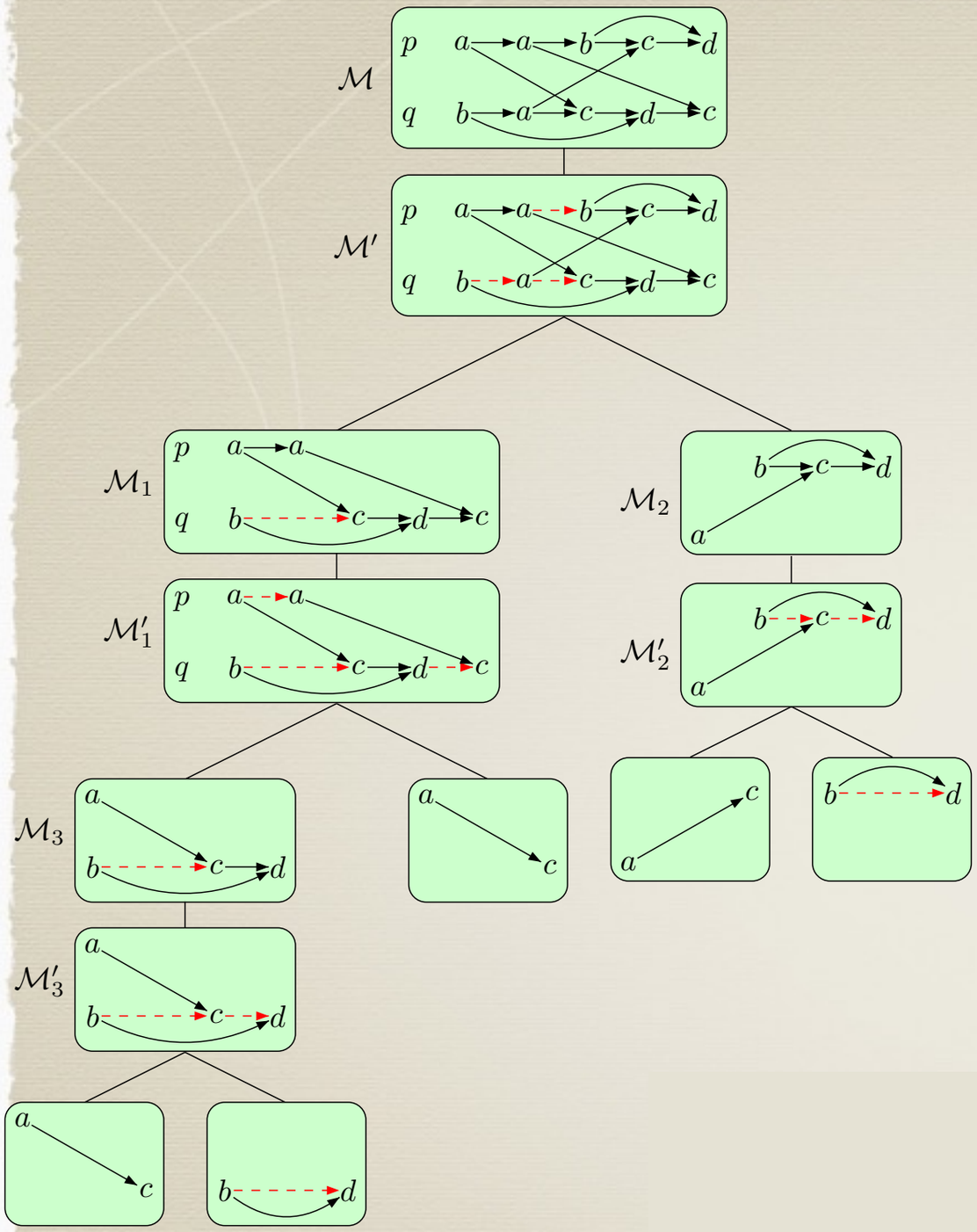
Basic Splits translate to nodes with two leaves attached to them.

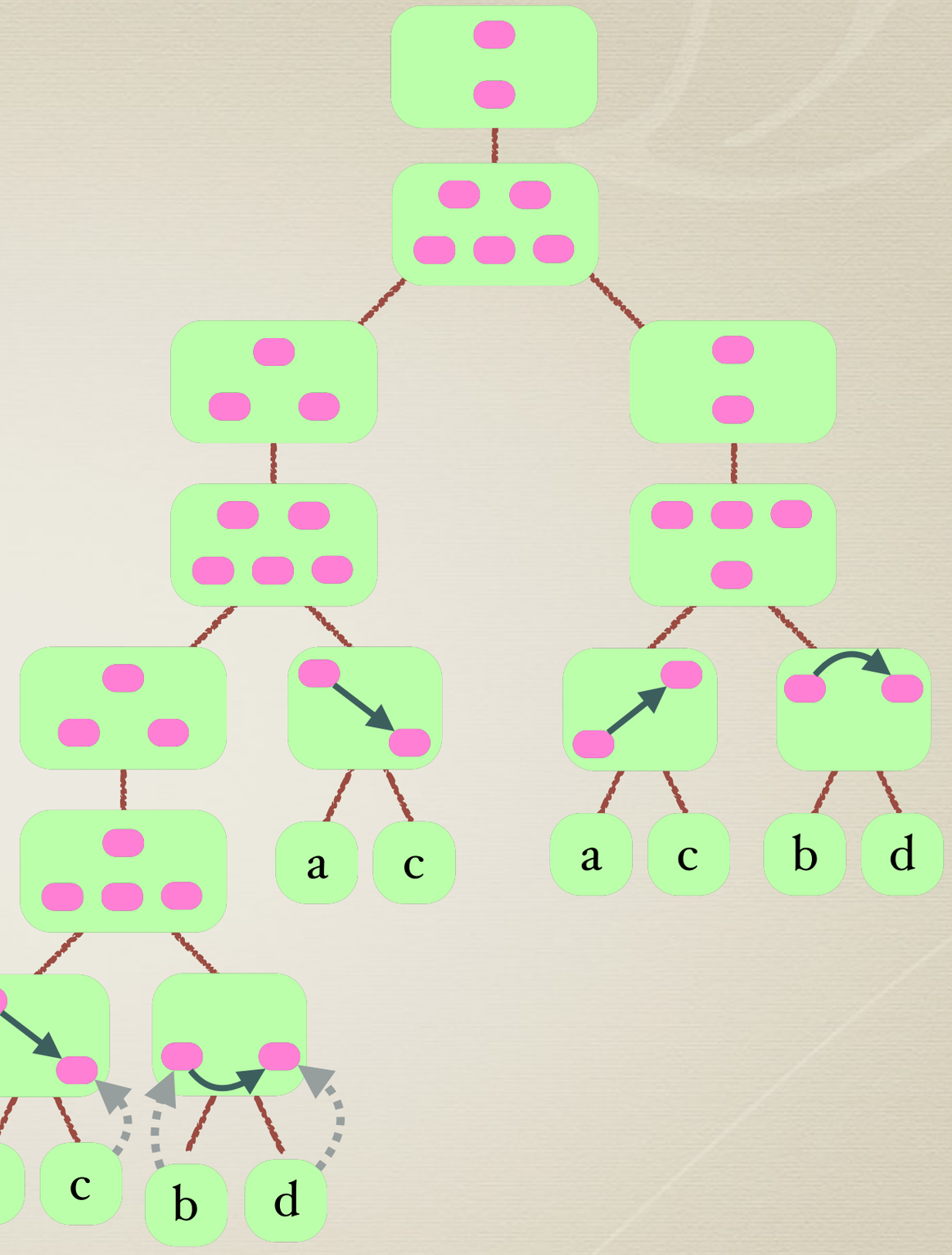




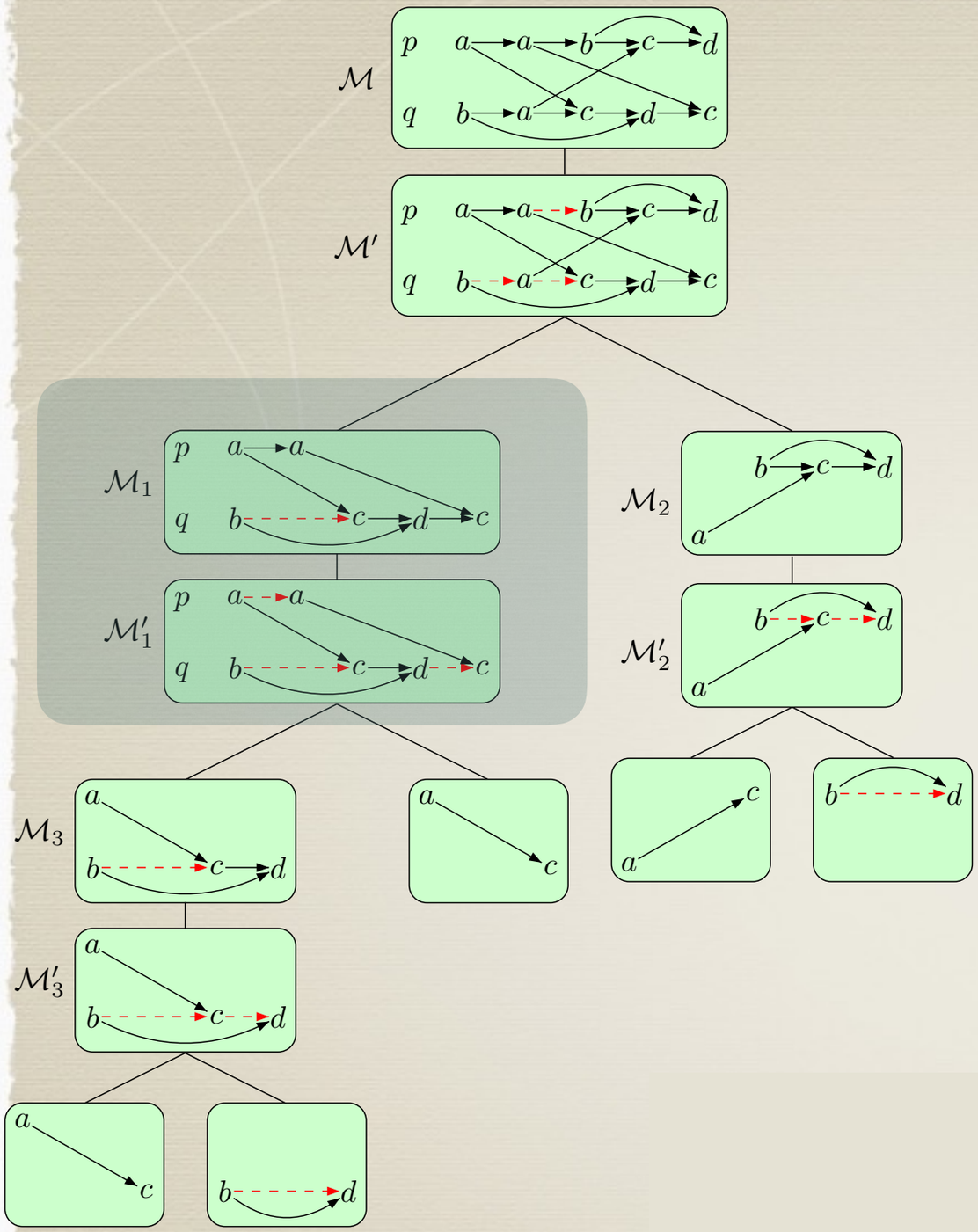
Basic Splits translate to nodes with two leaves attached to them.

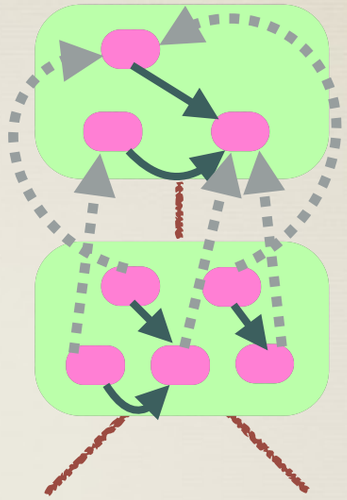
Map leaves to corresponding nodes.



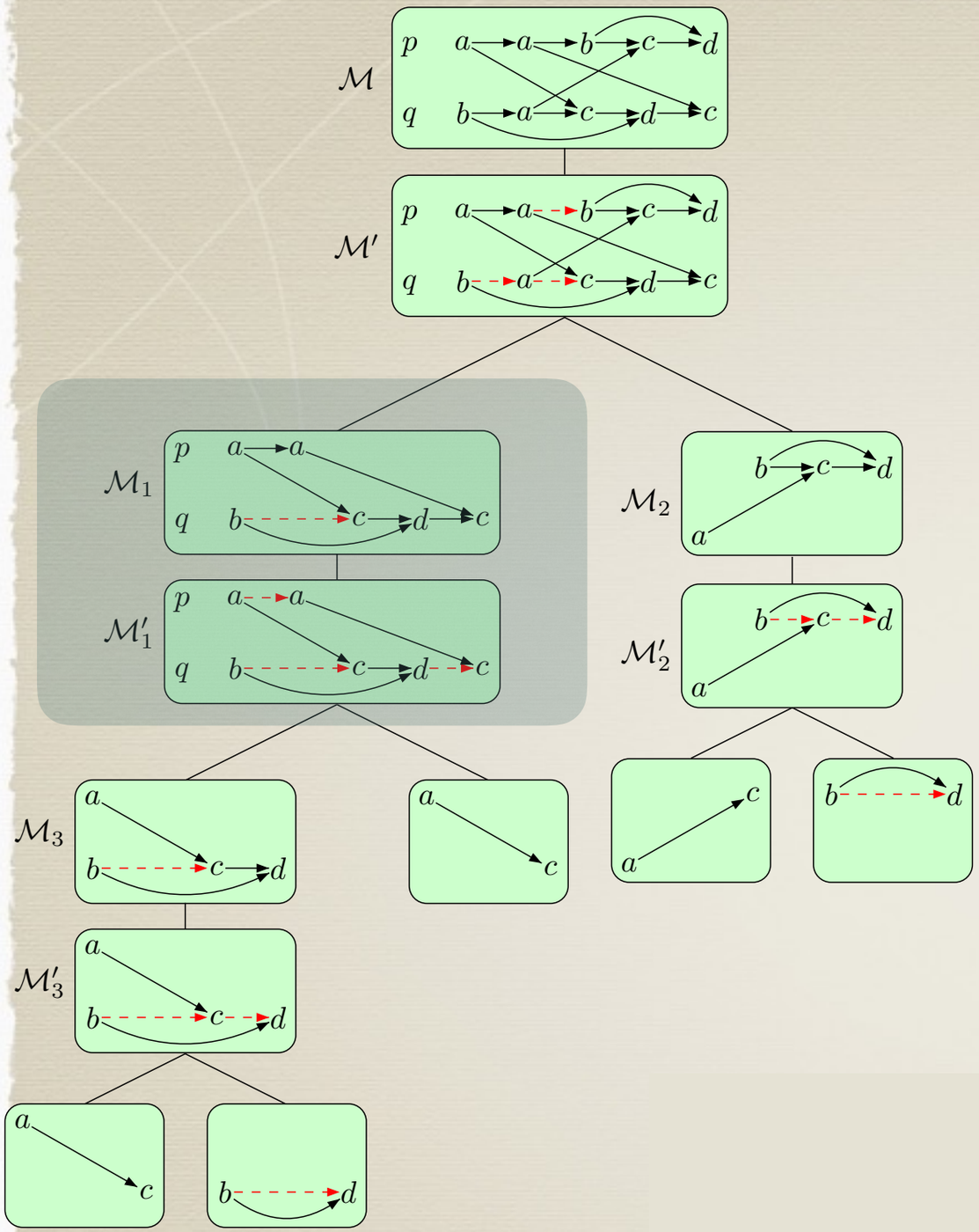


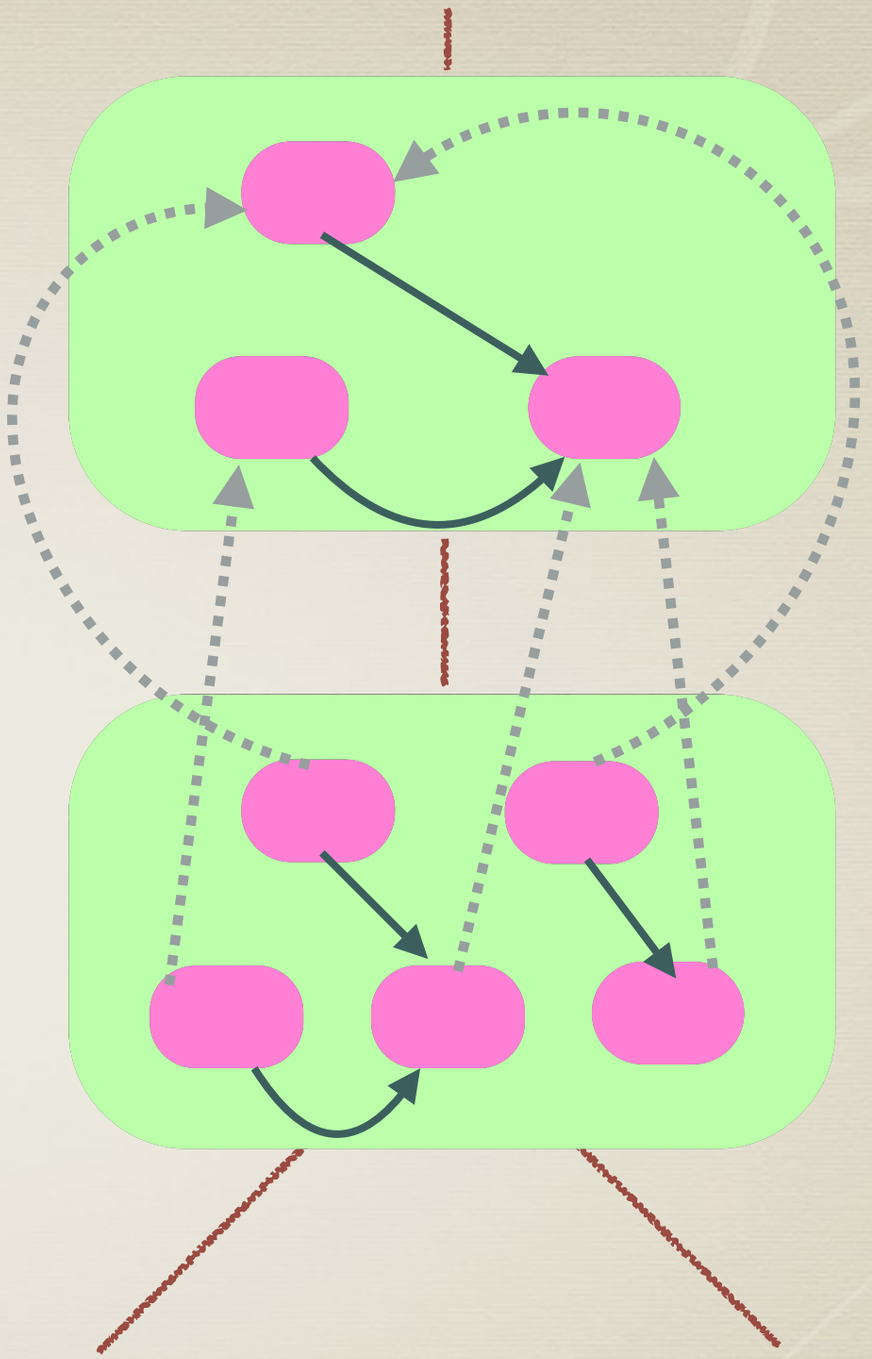
Map leaves to nodes in the label above.



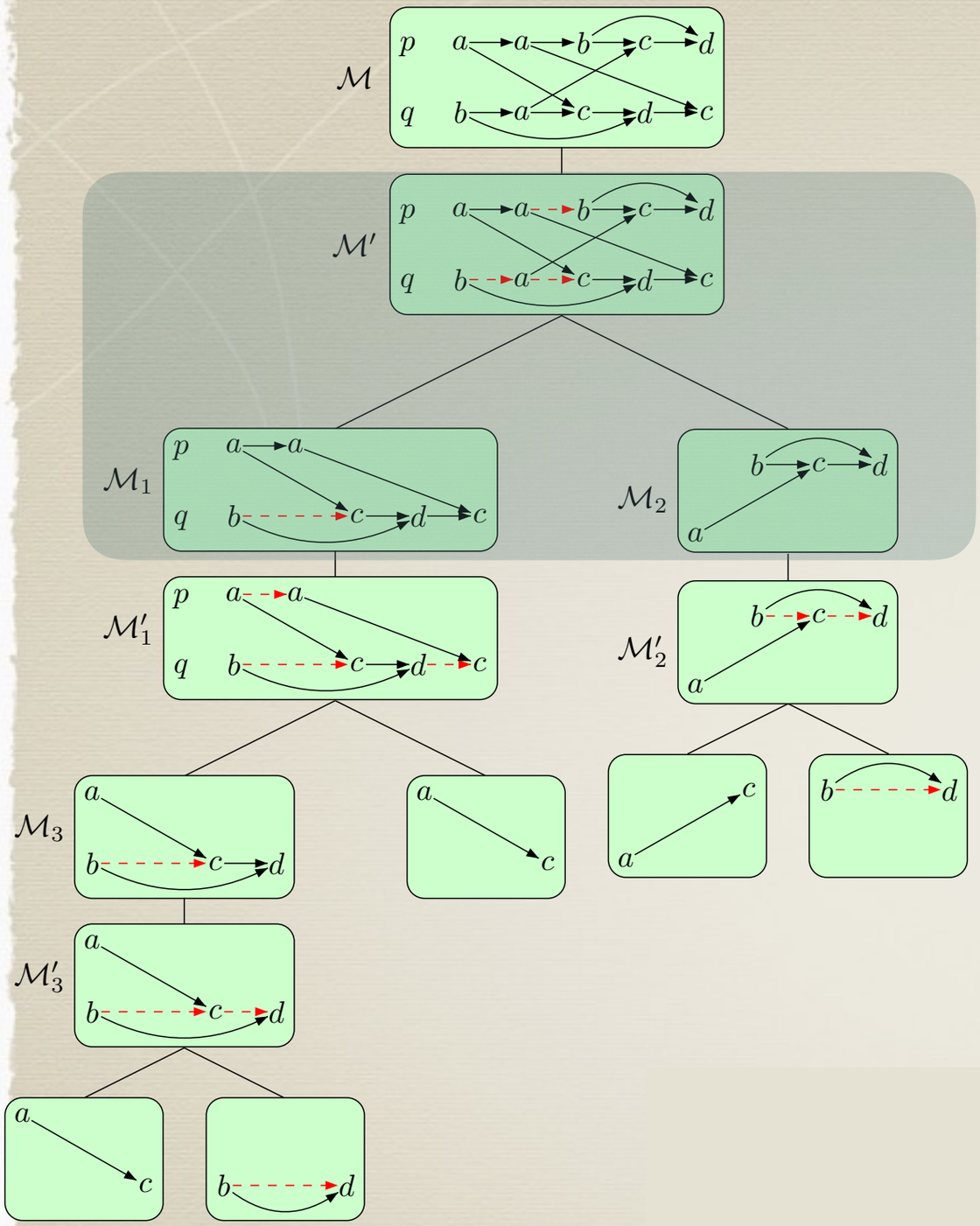


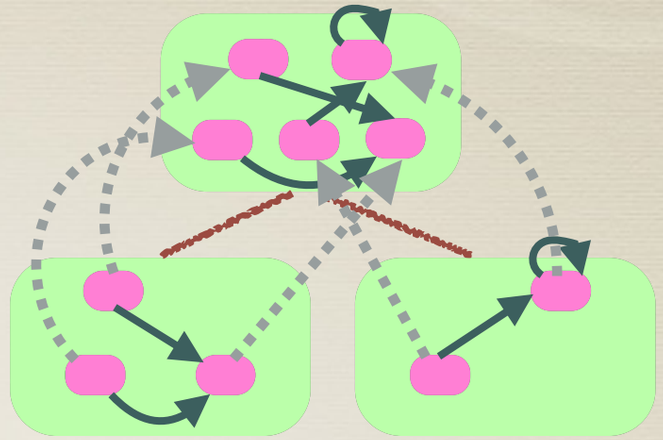
Map nodes to nodes in the parent: Cut operation.



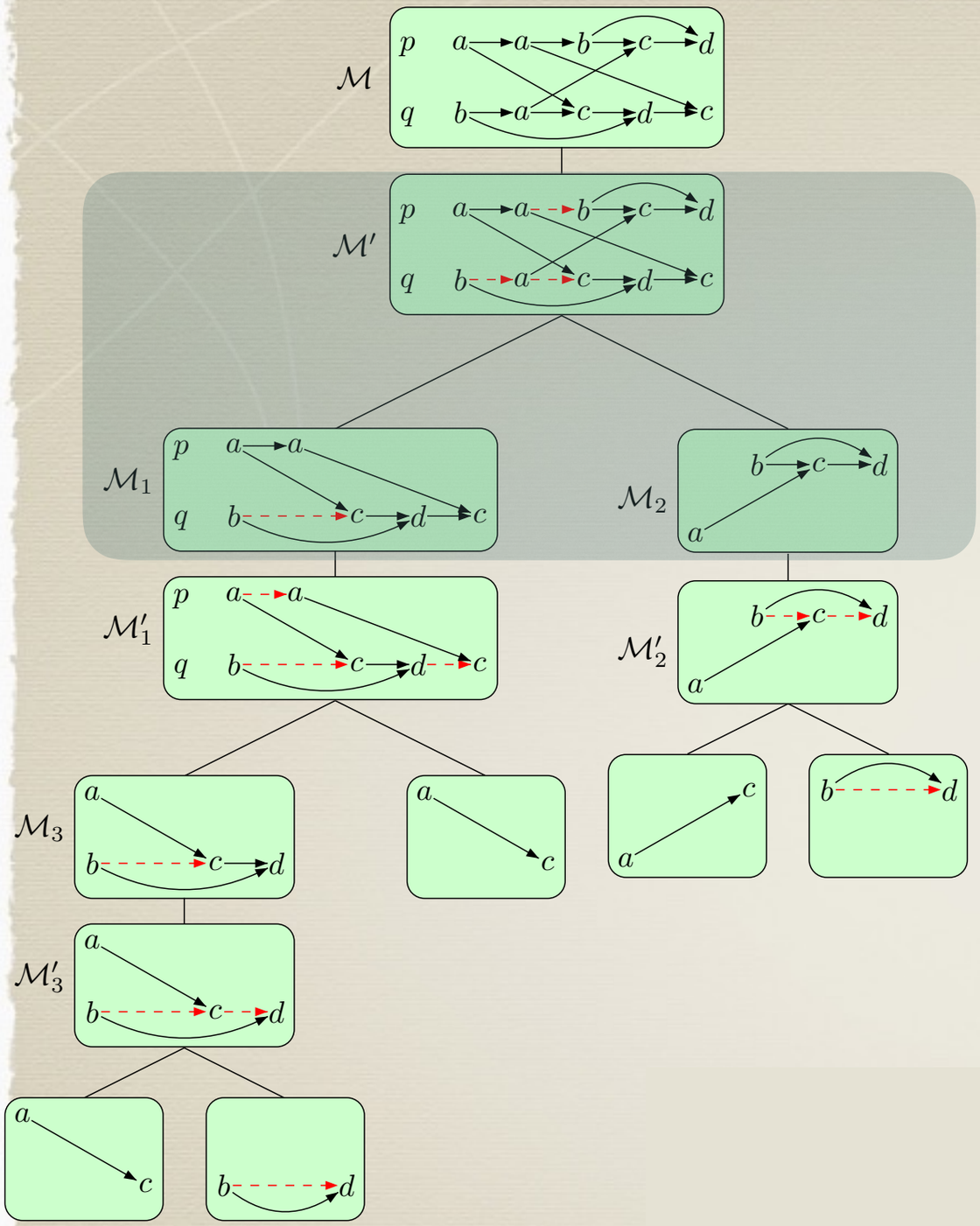


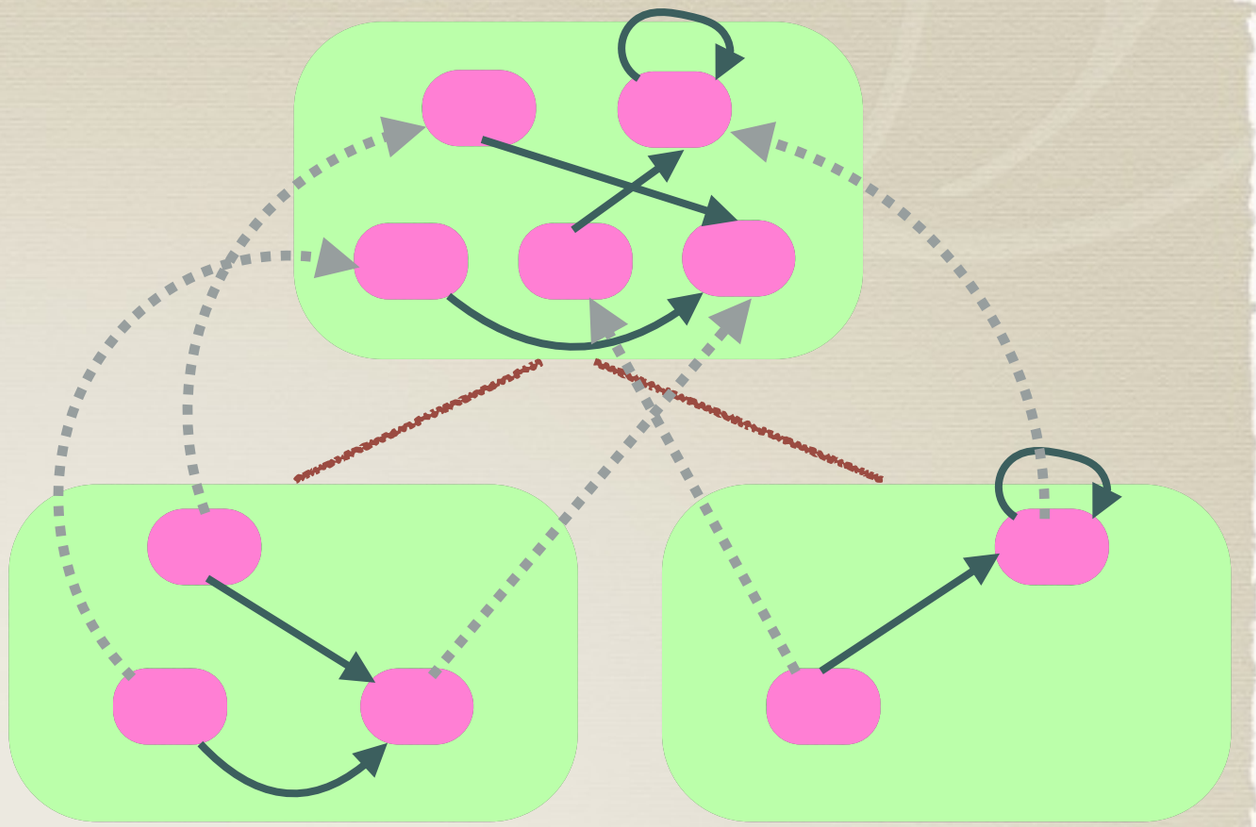
Map nodes to nodes in the parent: Cut operation.





Map nodes to nodes in the parent: Split operation.





Map nodes to nodes in the parent: Split operation.

Conditions on the labels

The labels constitute a finite alphabet

* Leaves occur in pairs.
* Node above leaf pairs has a 2 components and one data-structure edge.
* Binary nodes: number of components in each process is the sum of the numbers in the children.
* Unary nodes: number of components in each process is <= number in the child and one process has one fewer.
* Mapping between parents and children is injective
* Data-structure edges are correctly propagated
* Data-structure nesting/ordering rules are followed.
* The root has only one component per process.
* ...

Tree-automata for split-width k

Theorem:

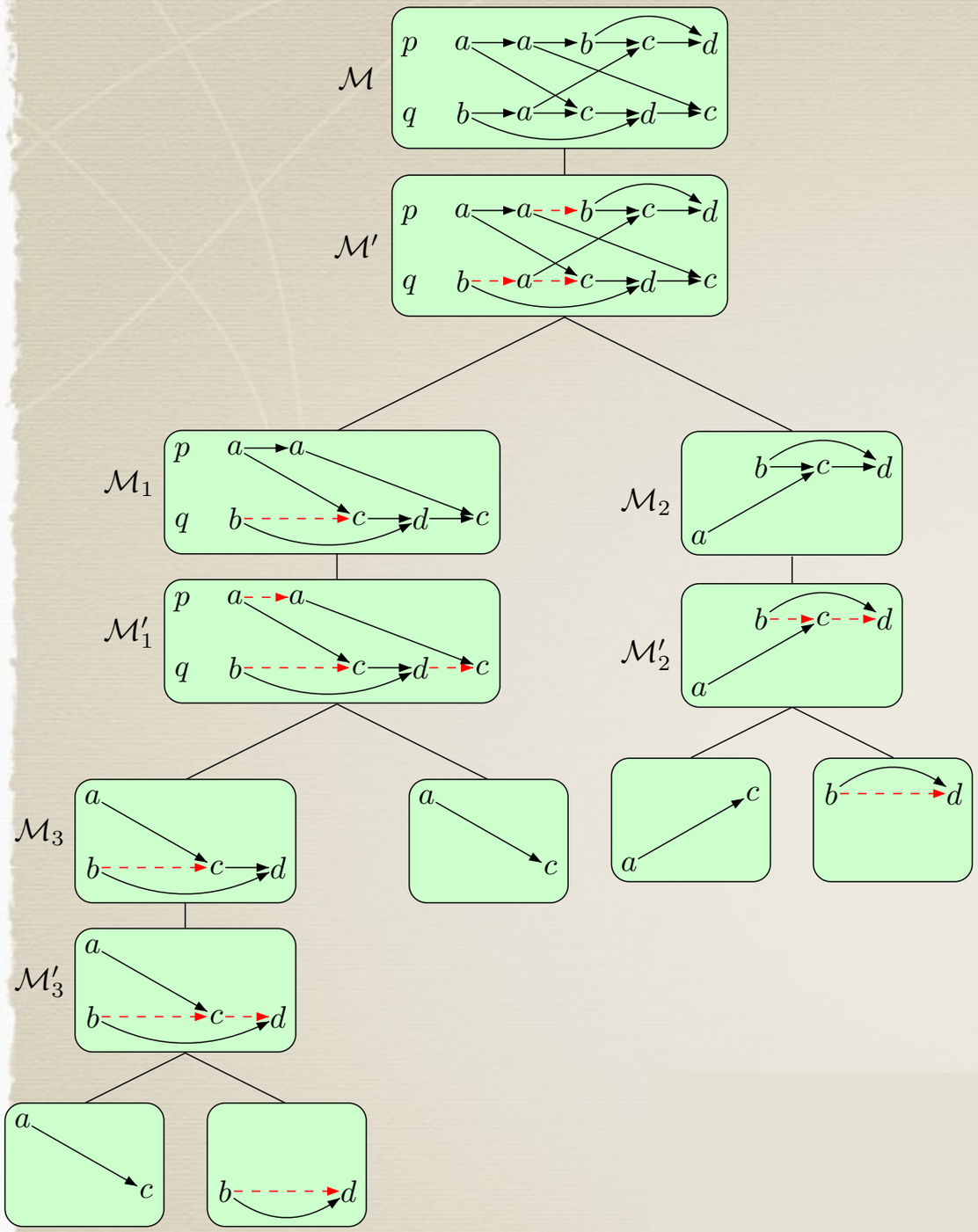
1. Any labelled tree satisfying the consistency conditions identifies a unique split-width k decomposition tree.
2. Every split-width k decomposition tree can be identified by a labelled tree satisfying the consistency conditions.
3. The set of all labelled trees satisfying the consistency conditions forms a regular tree language recognised by a tree-automaton whose size is exponential in k.

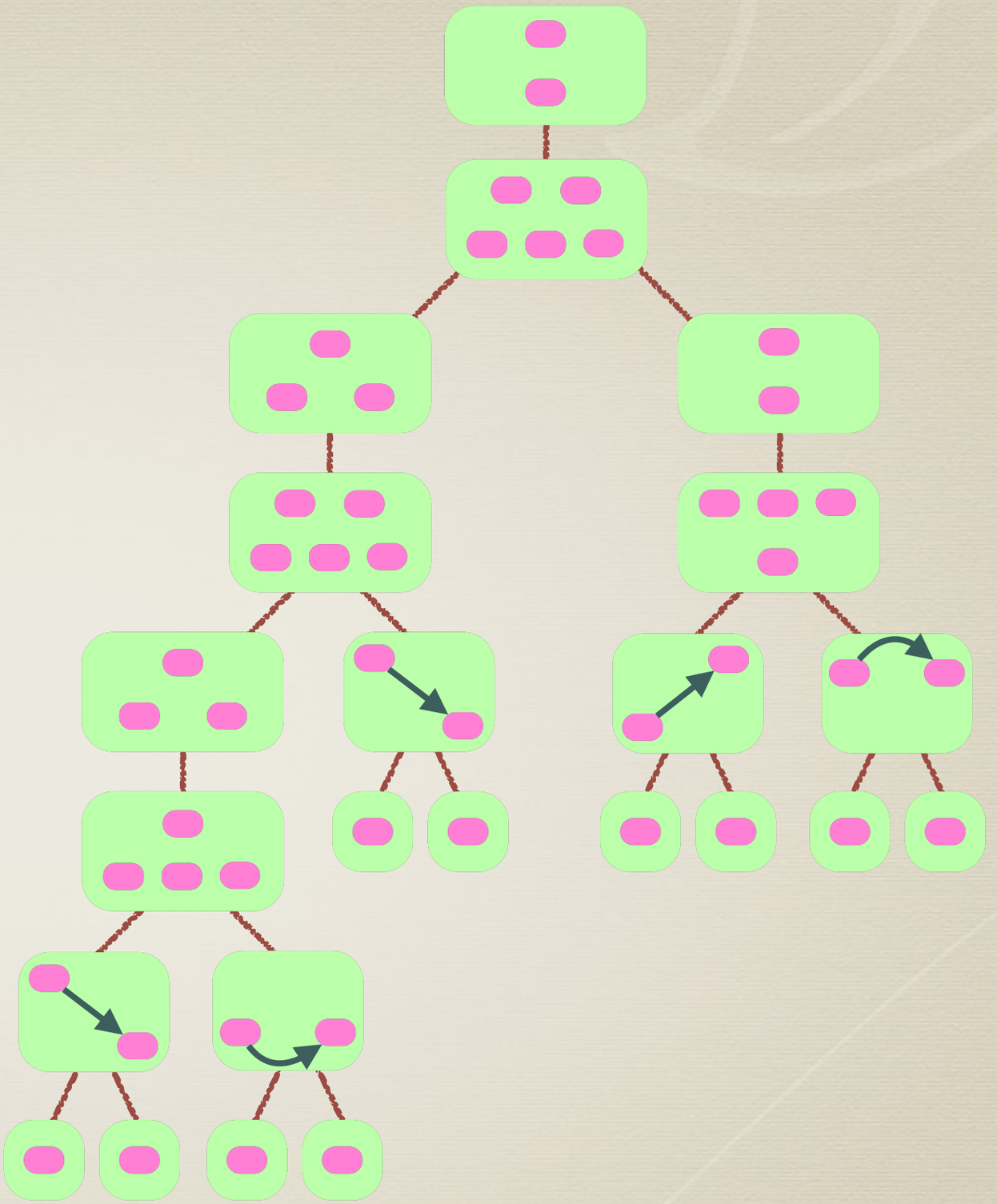
Tree-automata for split-width k

Theorem:

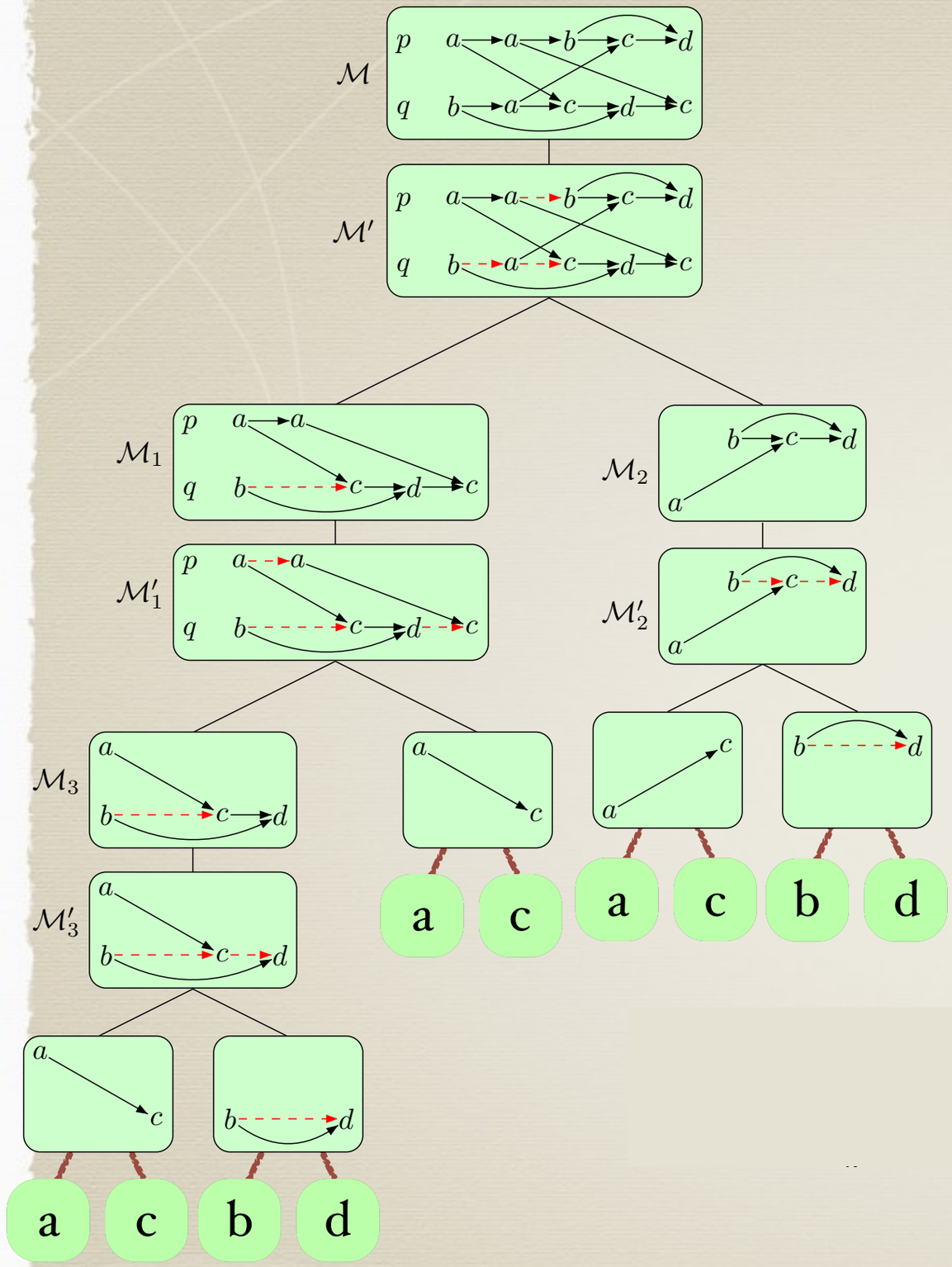
1. Any labelled tree satisfying the consistency conditions identifies a unique split-width k decomposition tree.
2. Every split-width k decomposition tree can be identified by a labelled tree satisfying the consistency conditions.
3. The set of all labelled trees satisfying the consistency conditions forms a regular tree language recognised by a tree-automaton whose size is exponential in k.

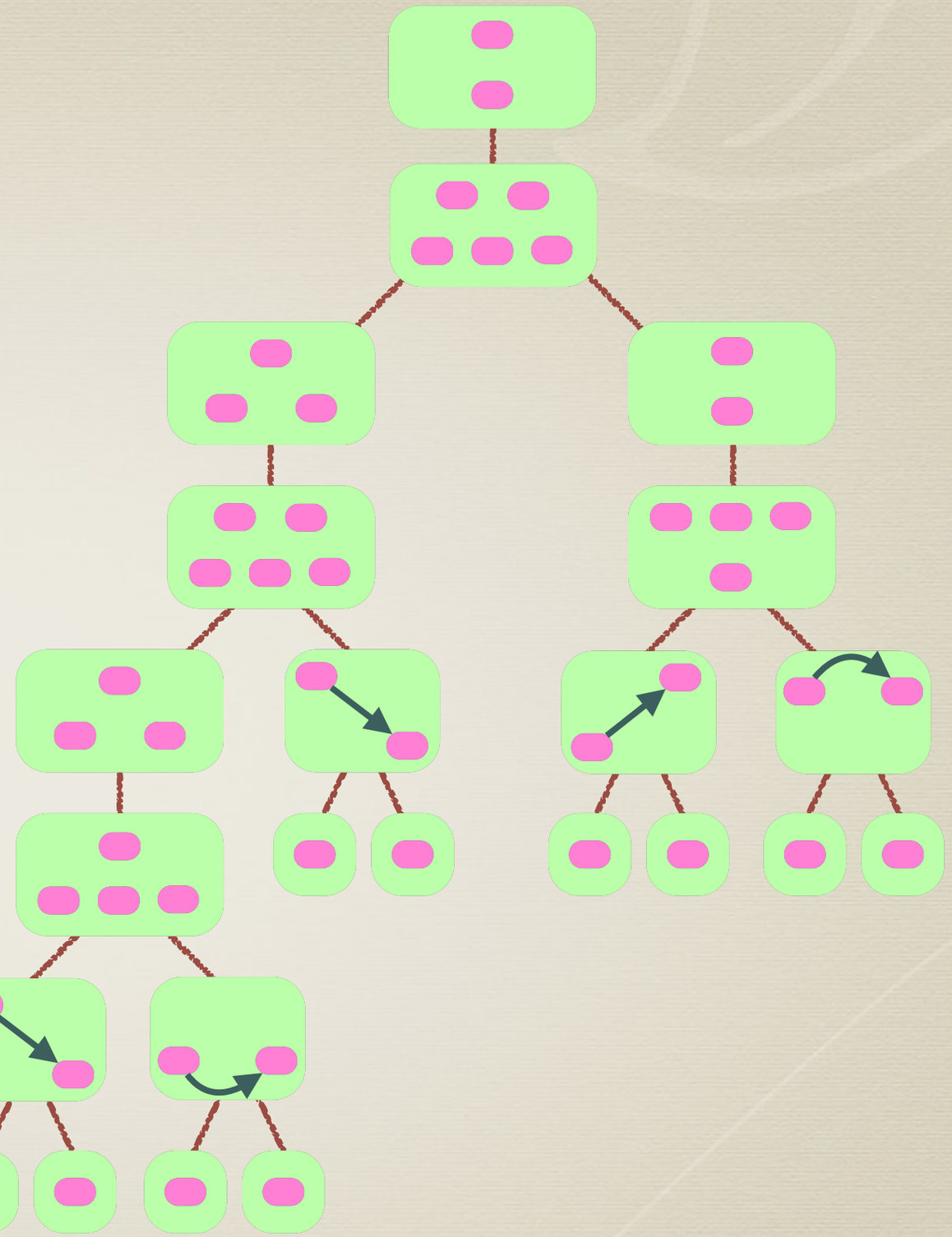
Interpret split-width k CBMs on these trees!



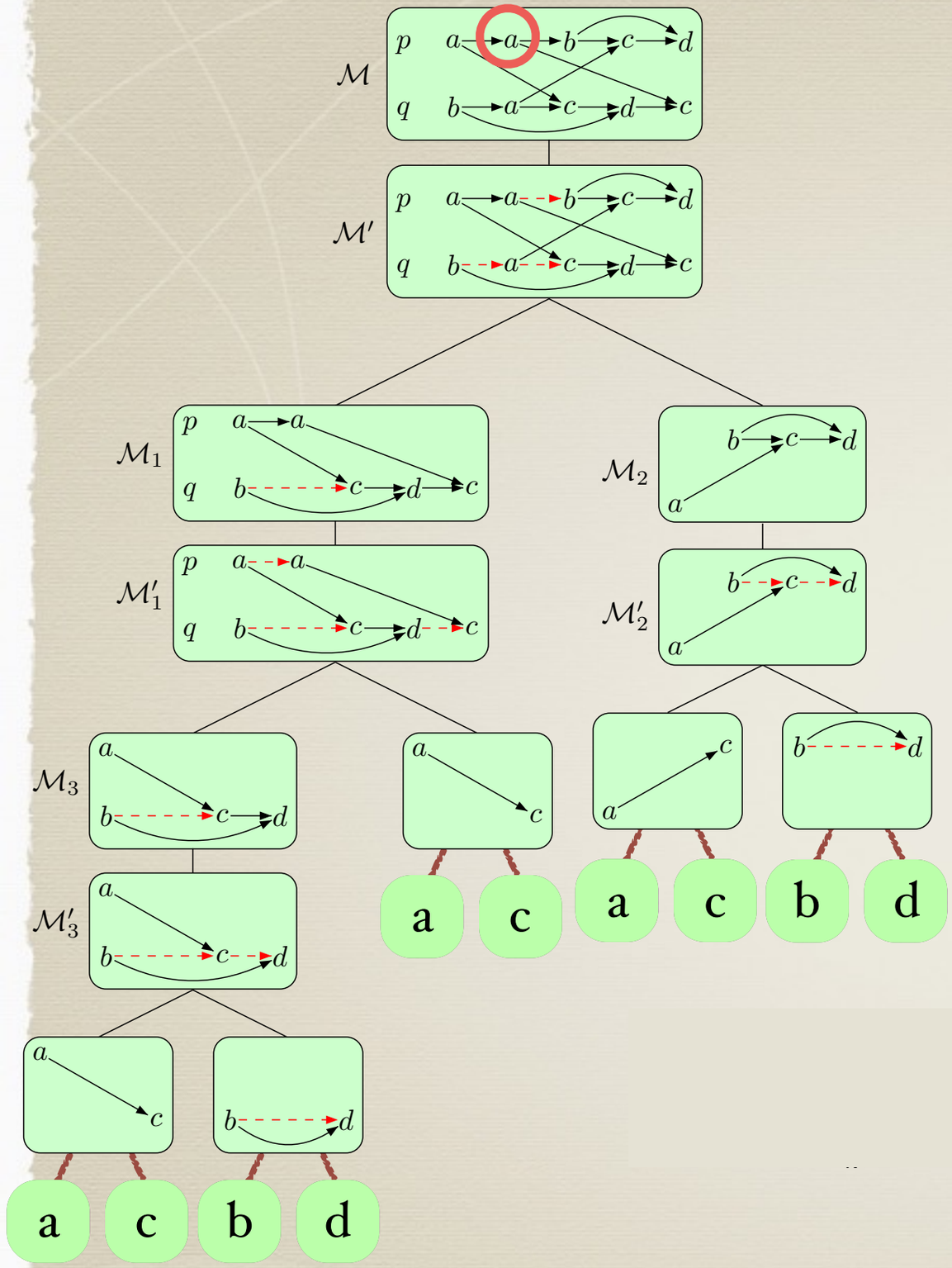


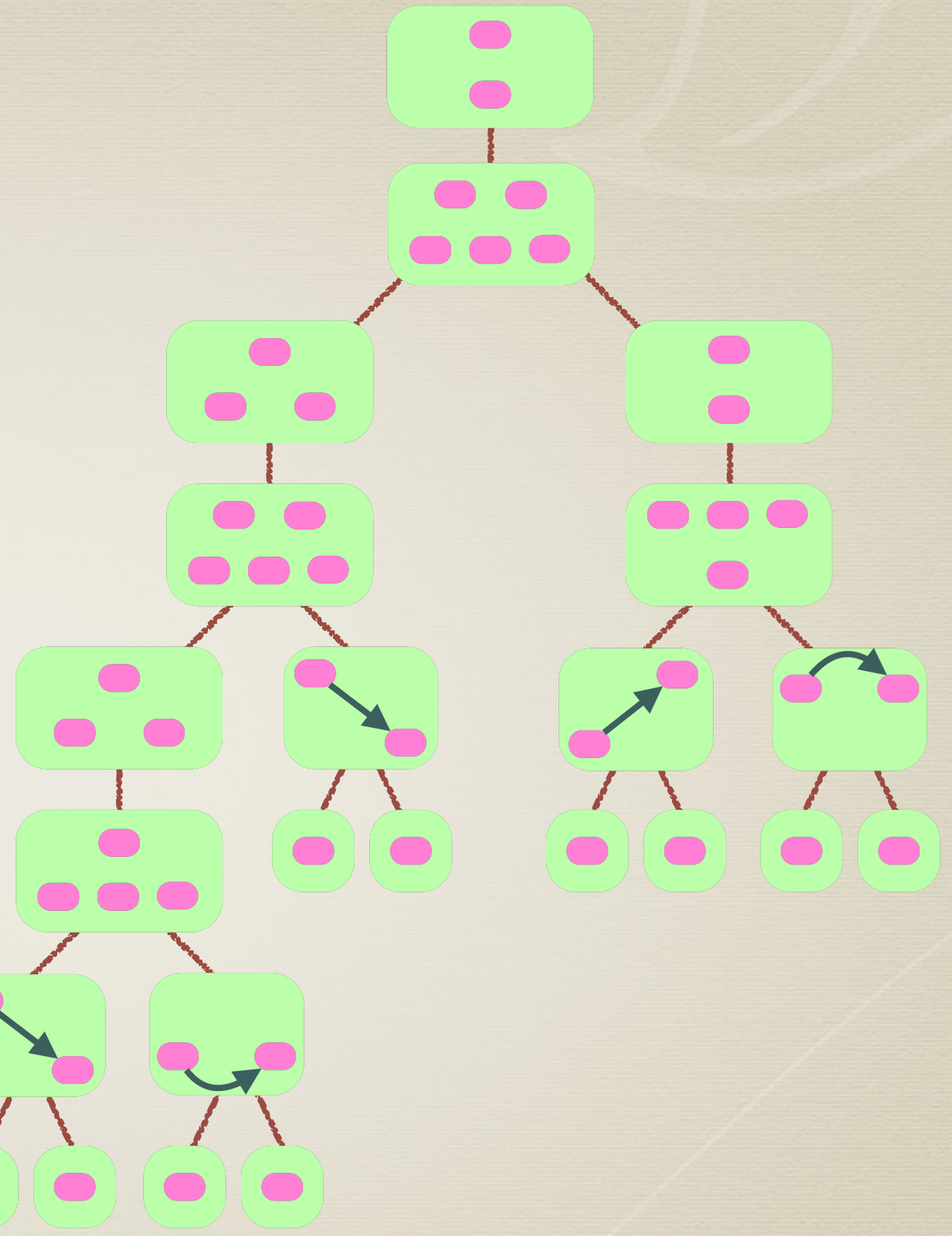
Tree interpretation





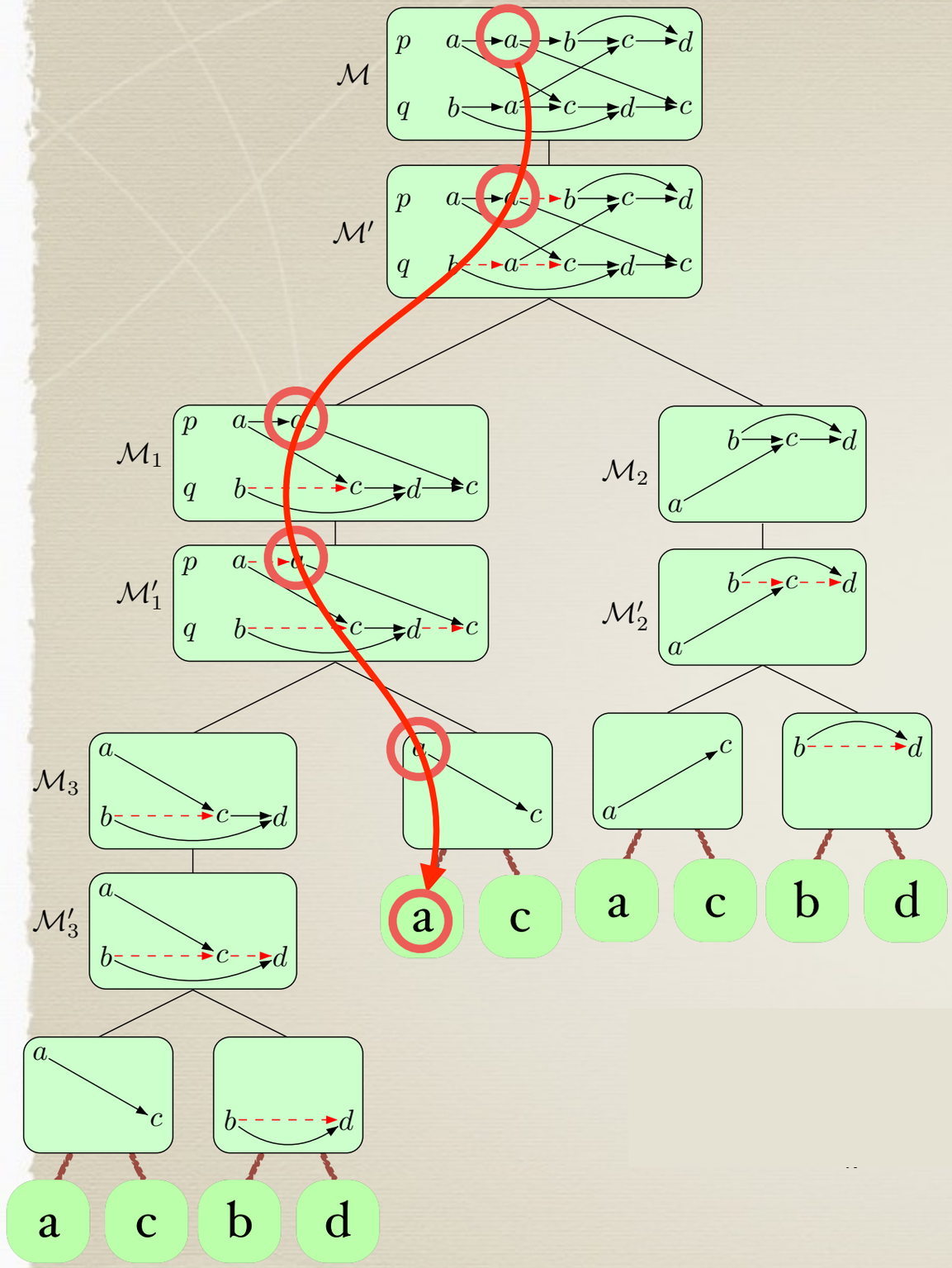
Tree interpretation

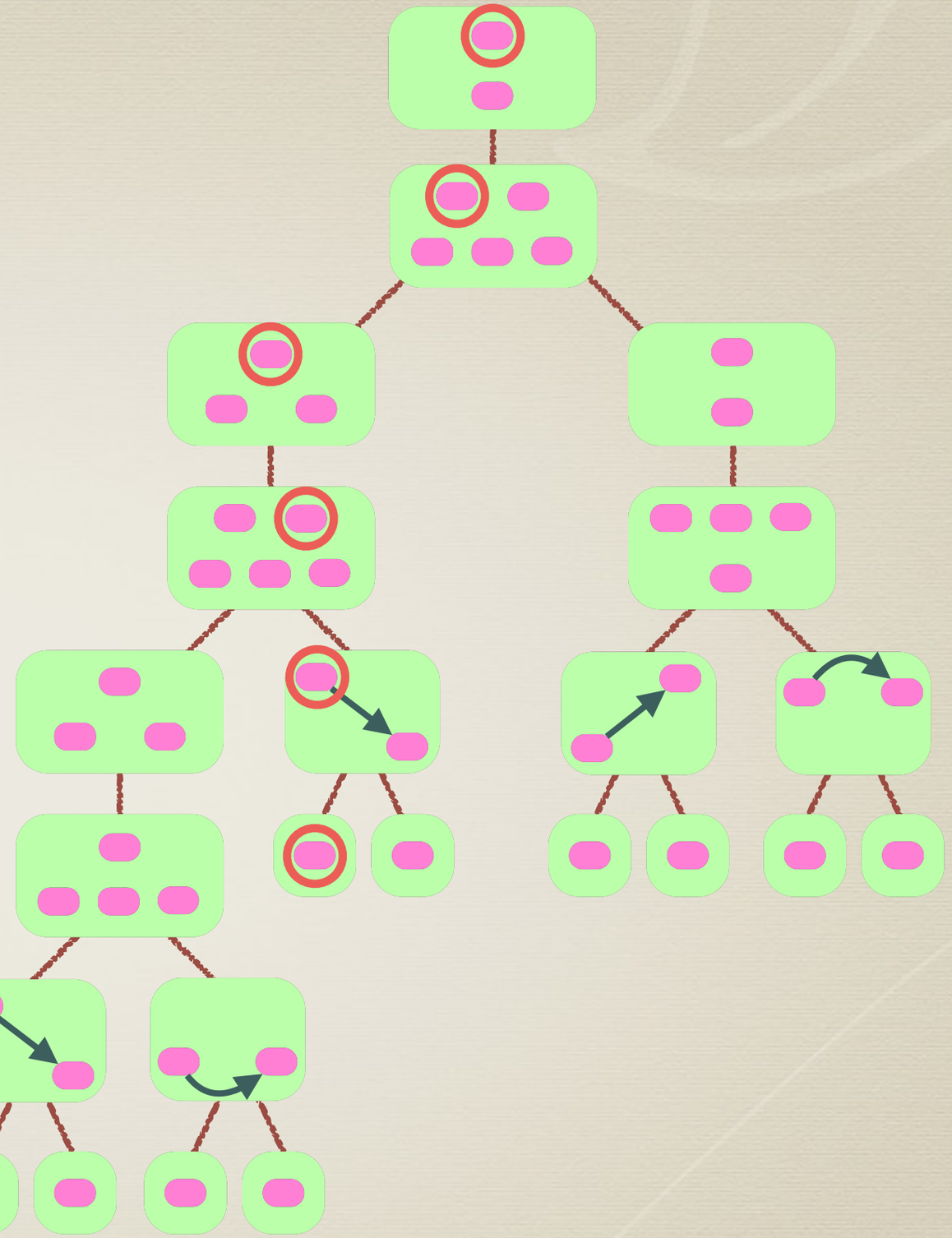




Vertices = Leaves

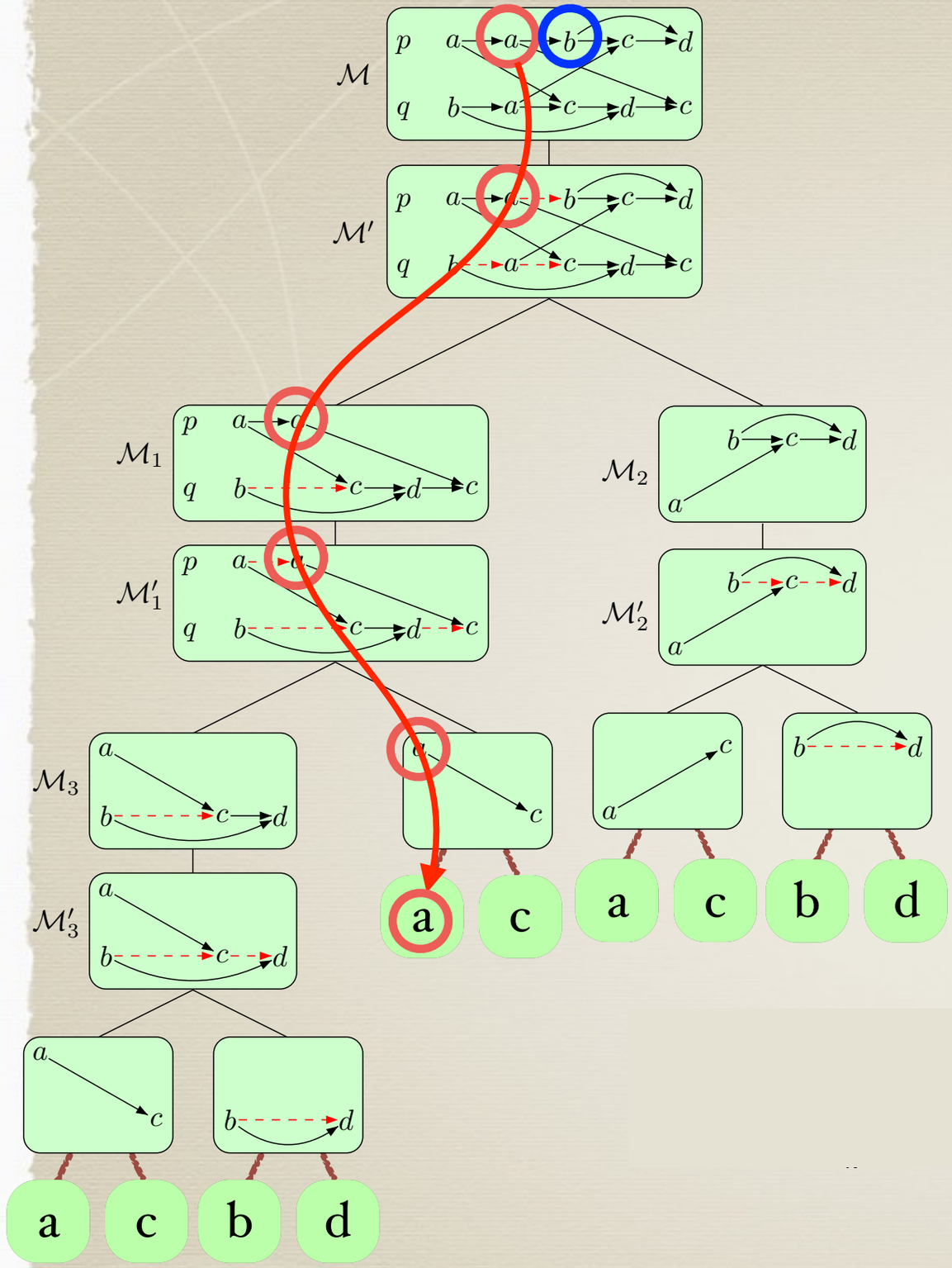
Tree interpretation

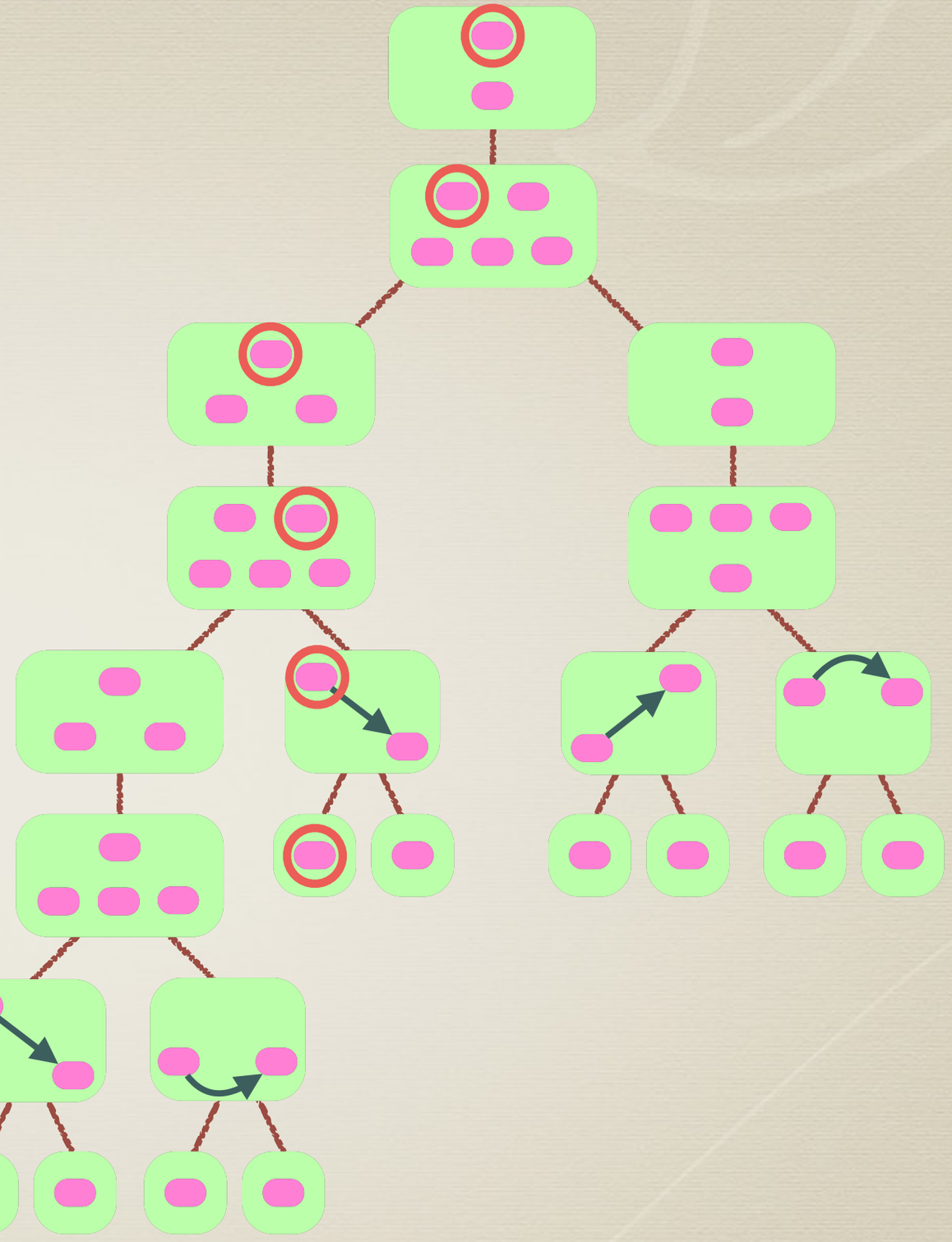




Vertices = Leaves

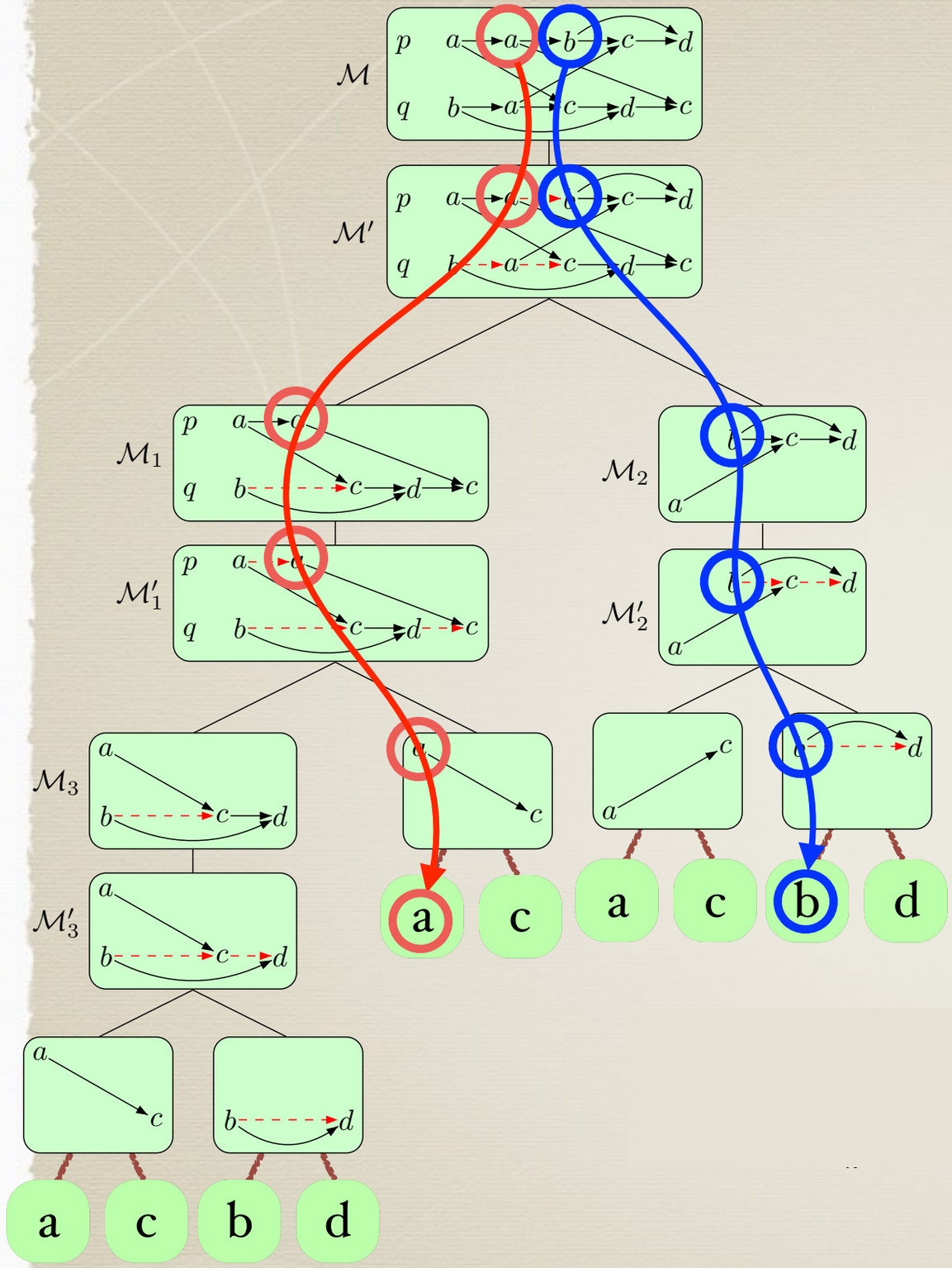
Tree interpretation

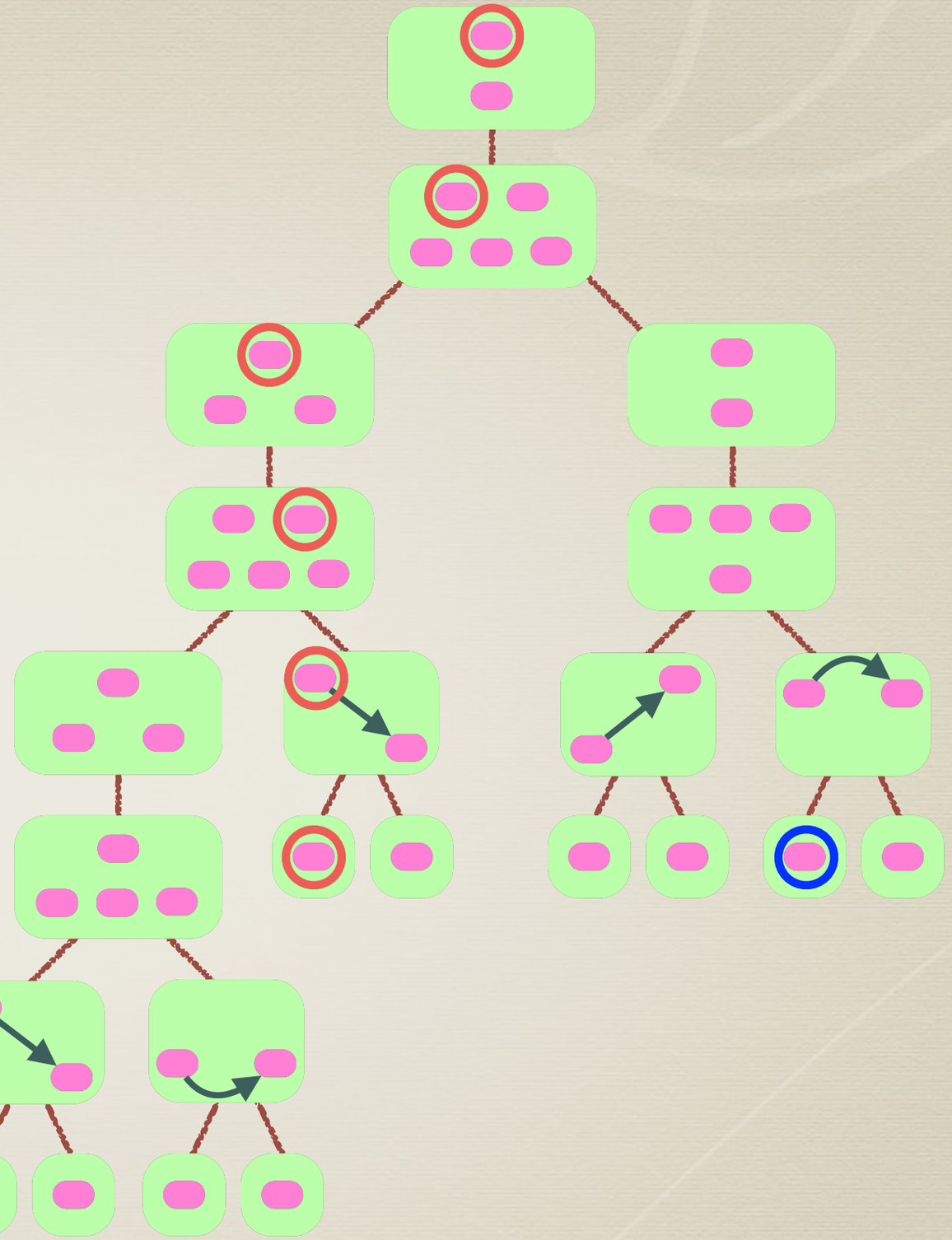




Vertices = Leaves

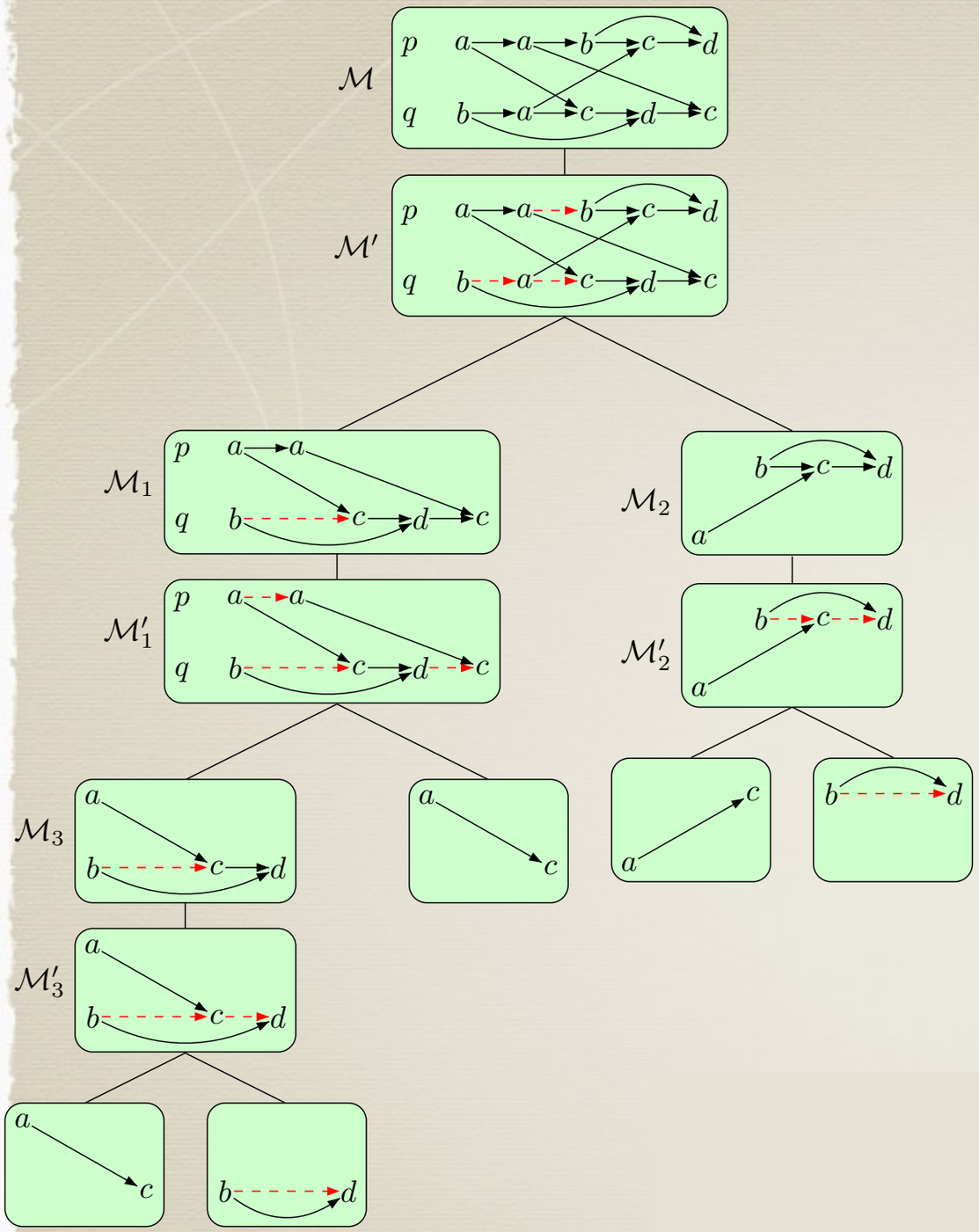
Tree interpretation

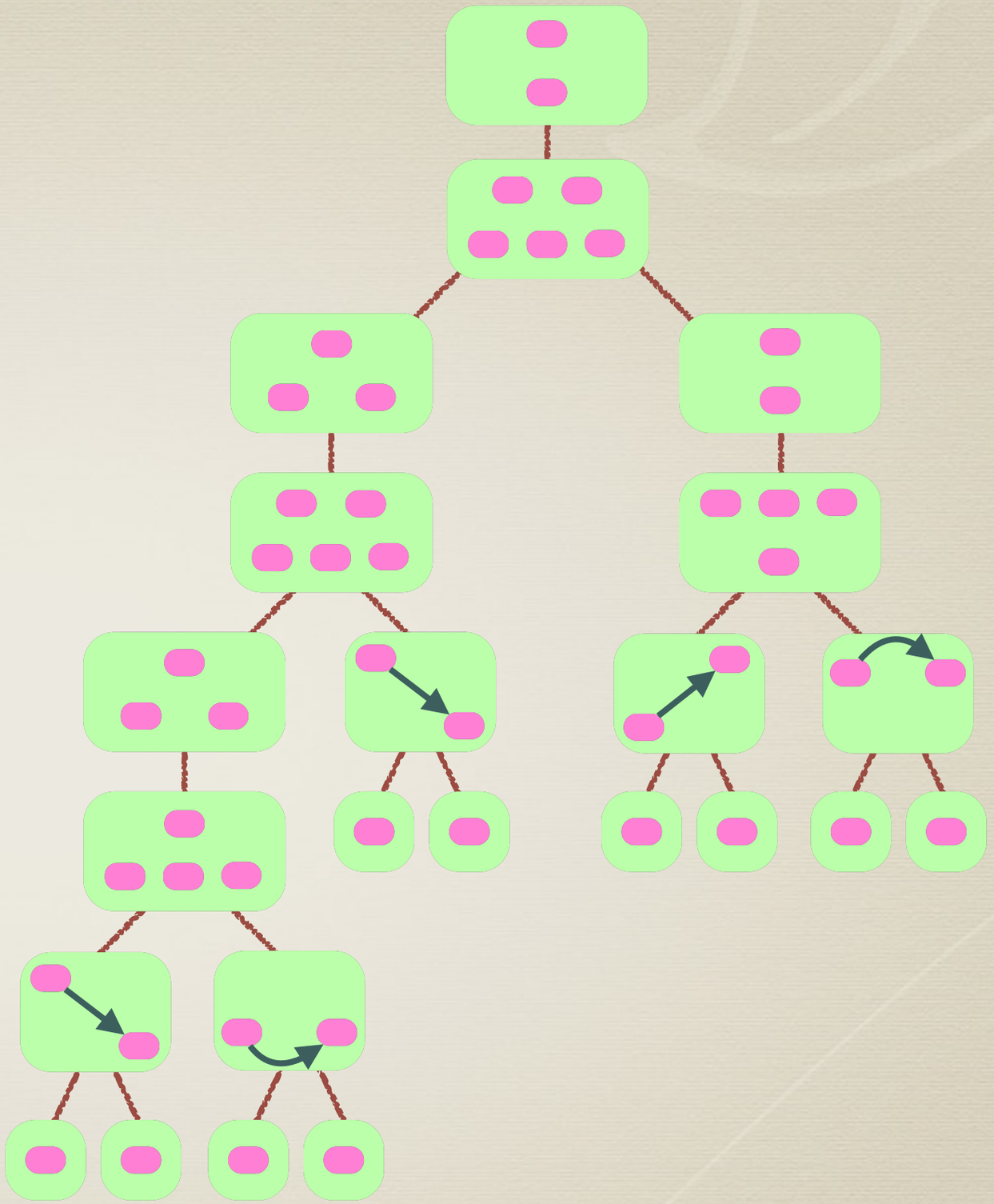




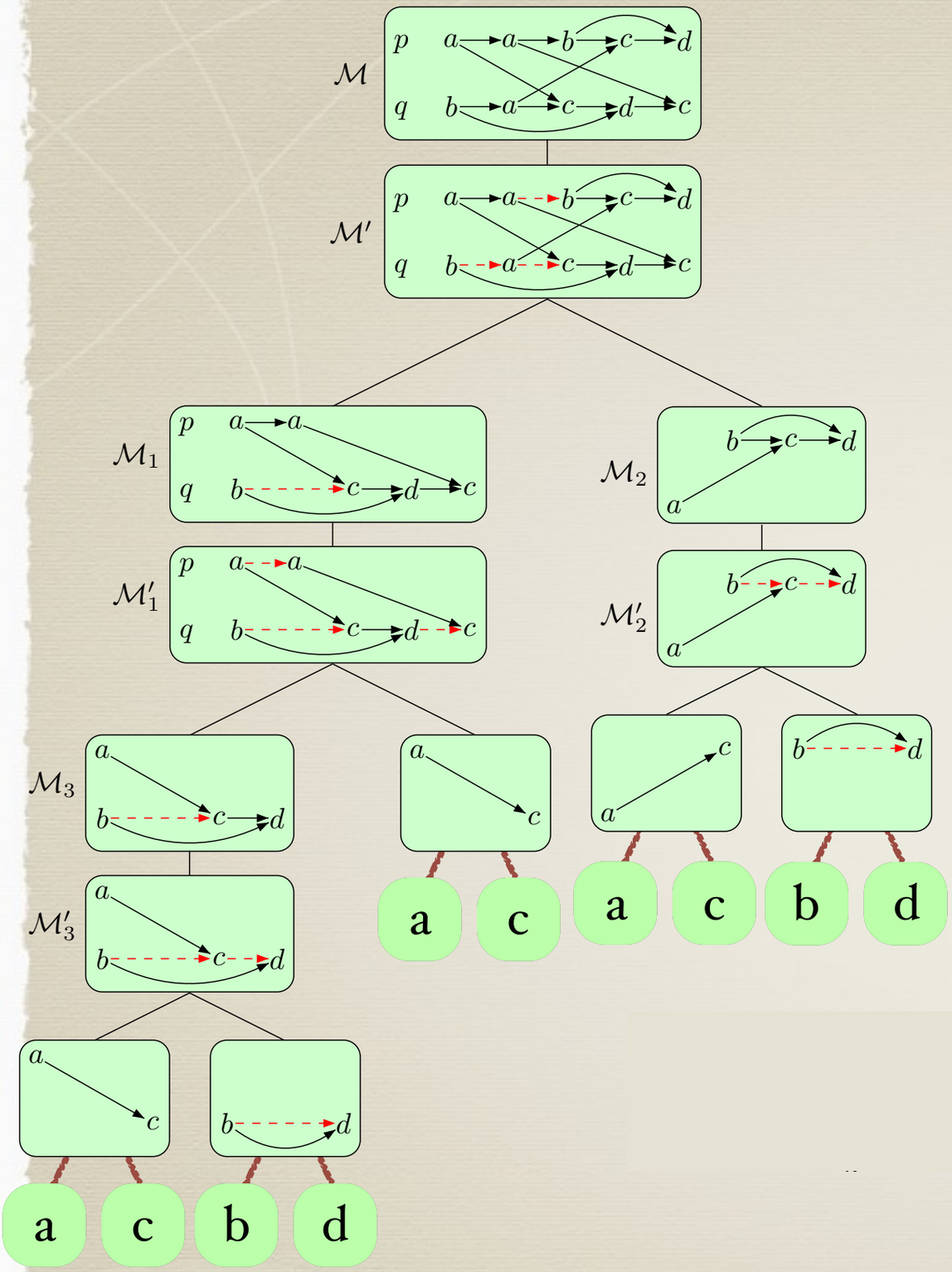
Vertices = Leaves

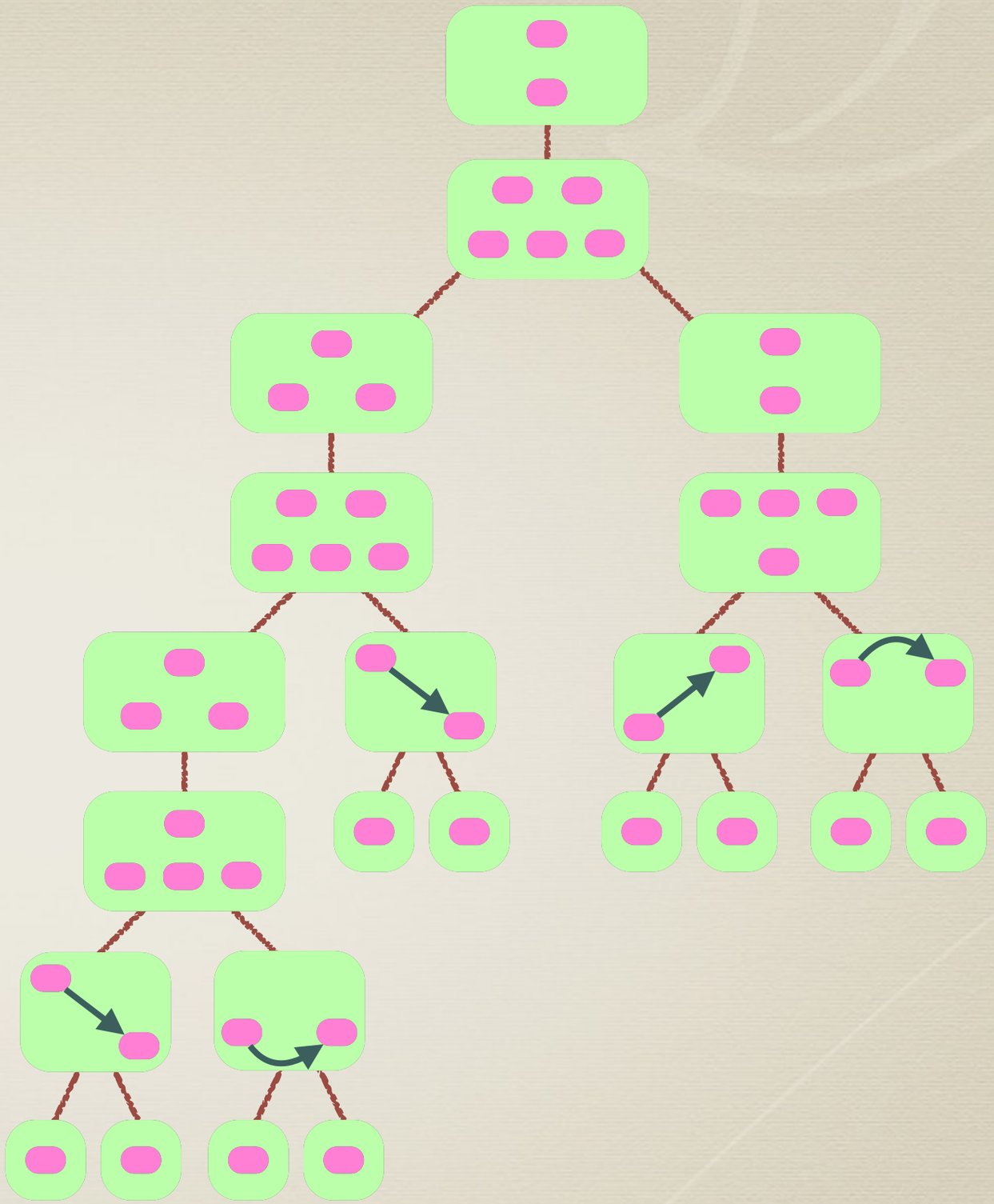
Tree interpretation



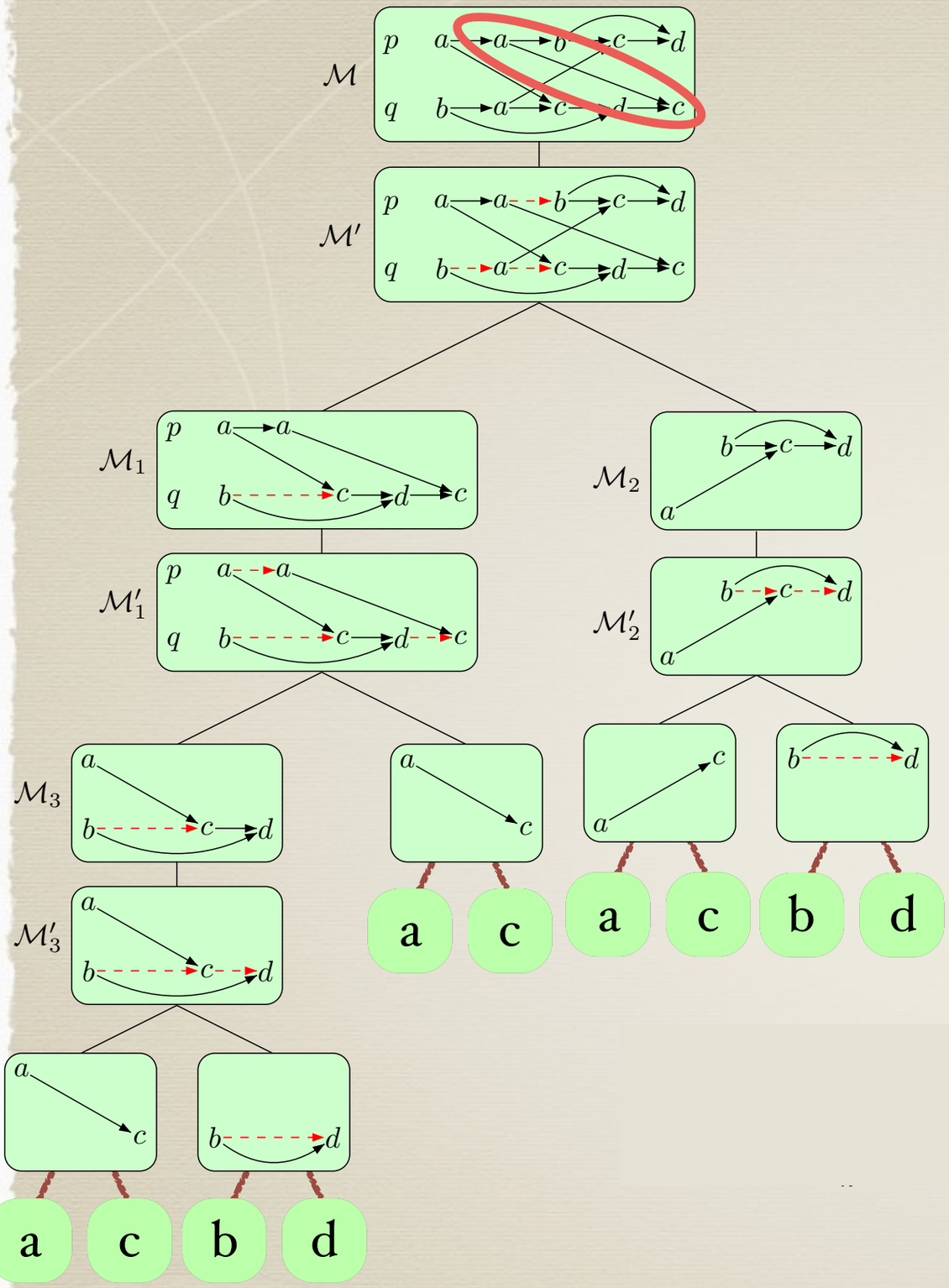


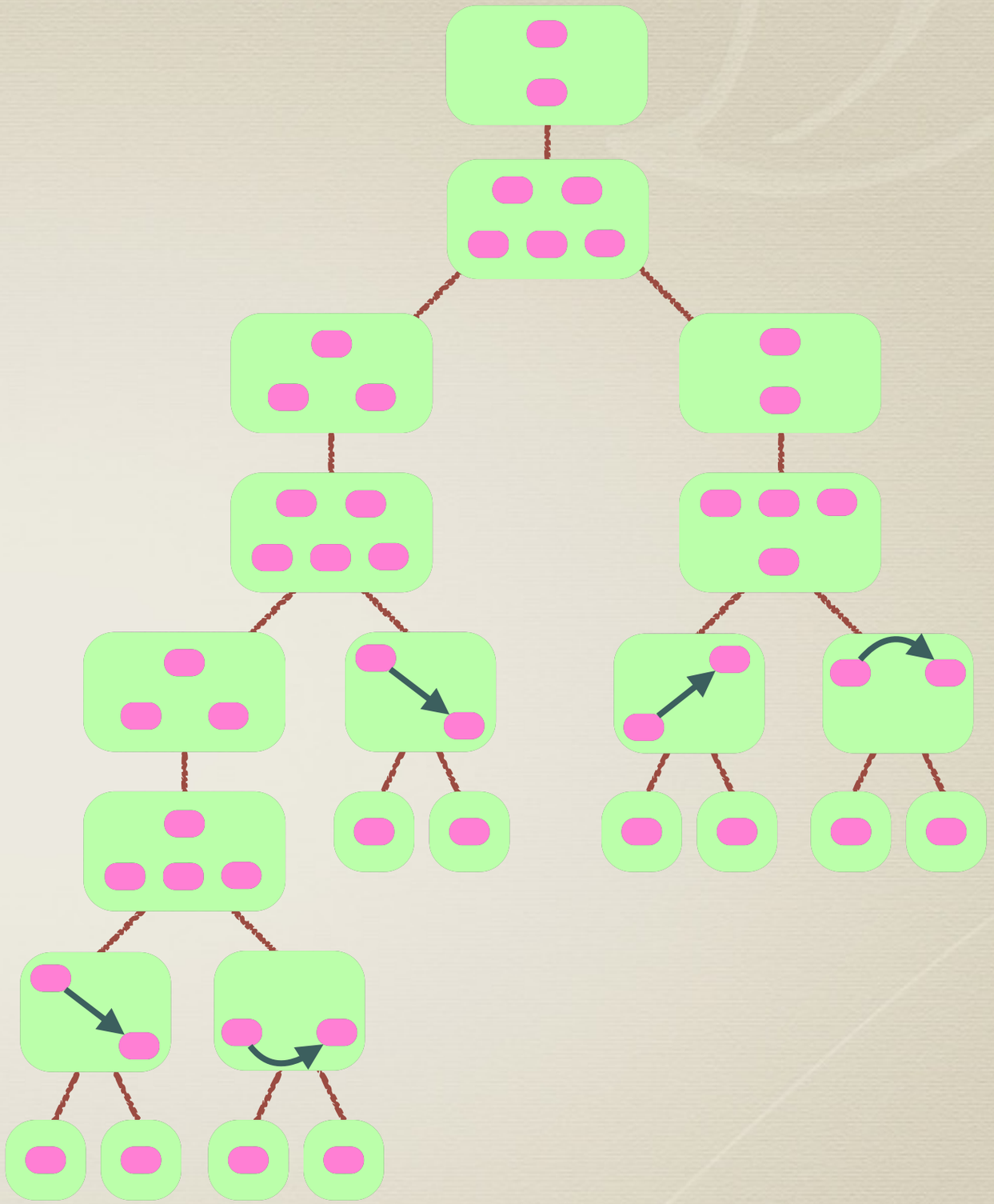
Tree interpretation





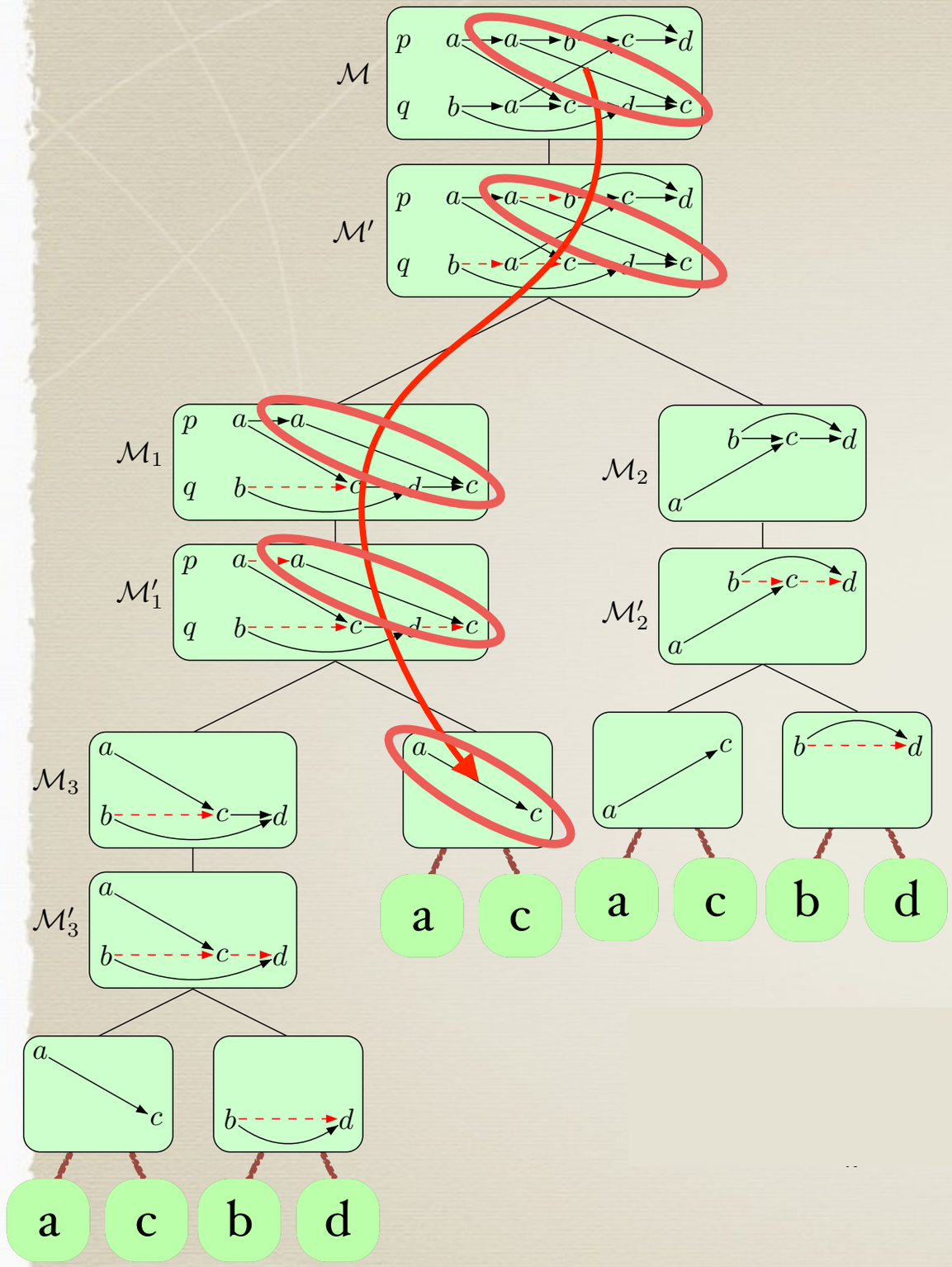
Tree interpretation

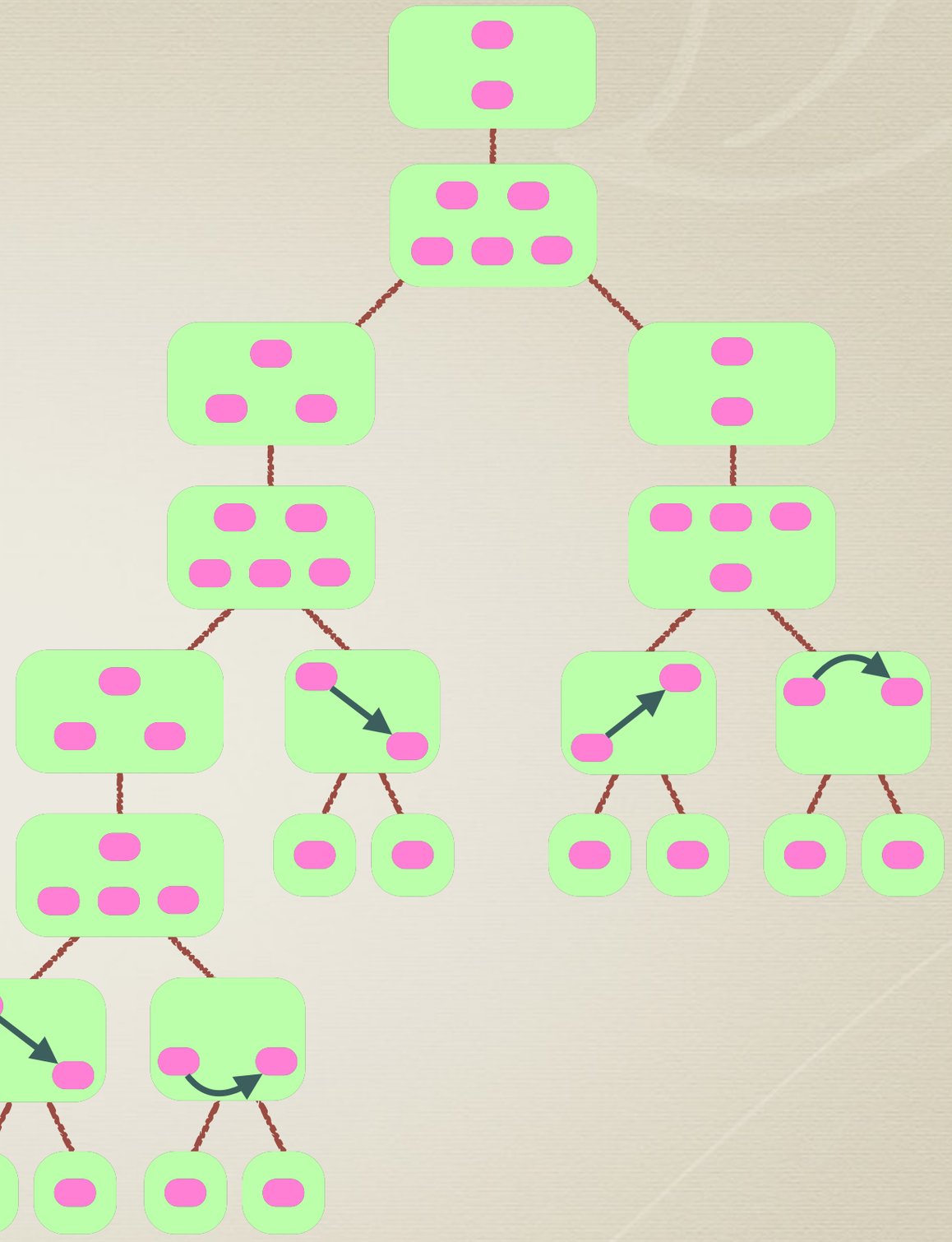




Data edges = Above Leaves

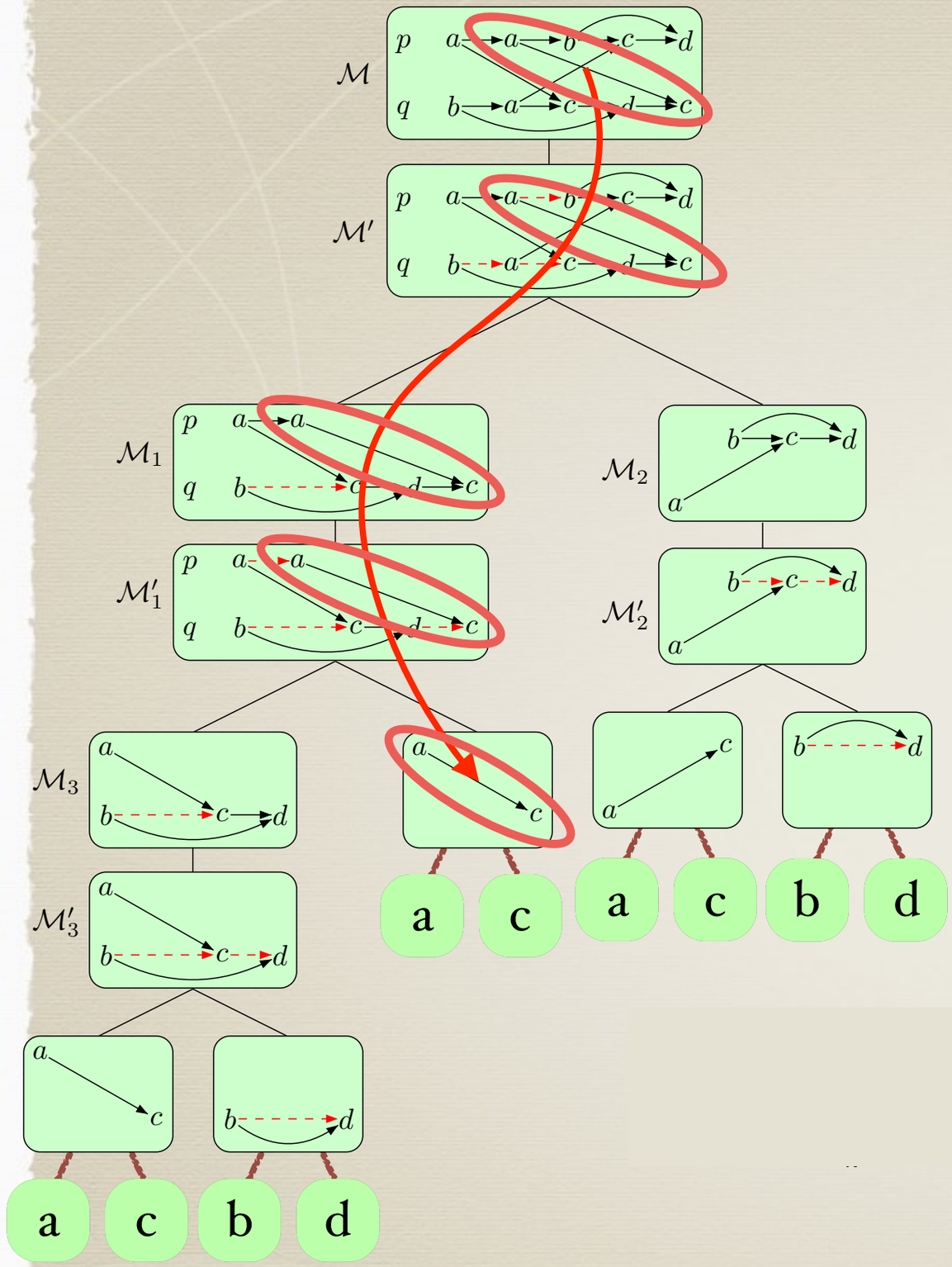
Tree interpretation

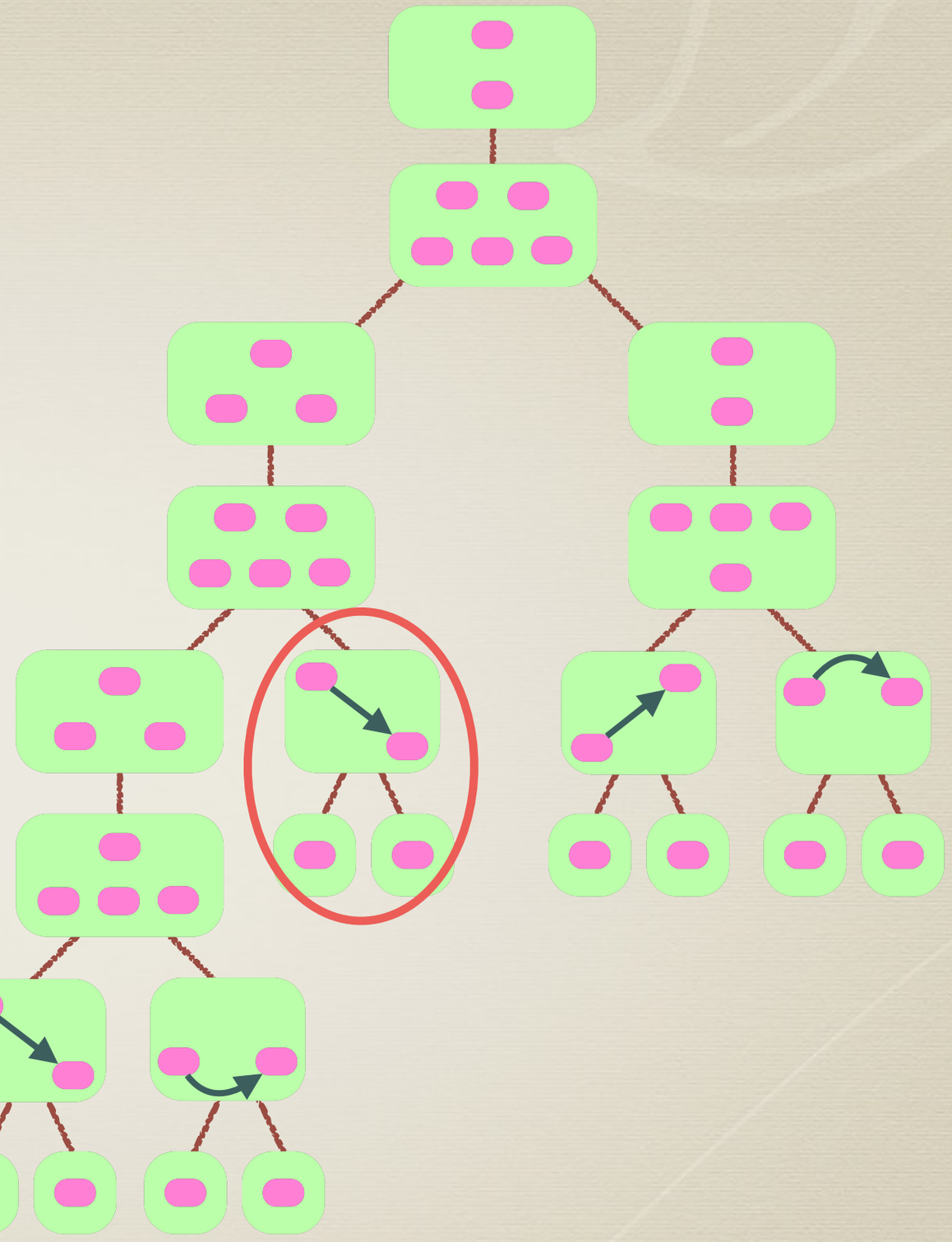




Data edges = Above Leaves

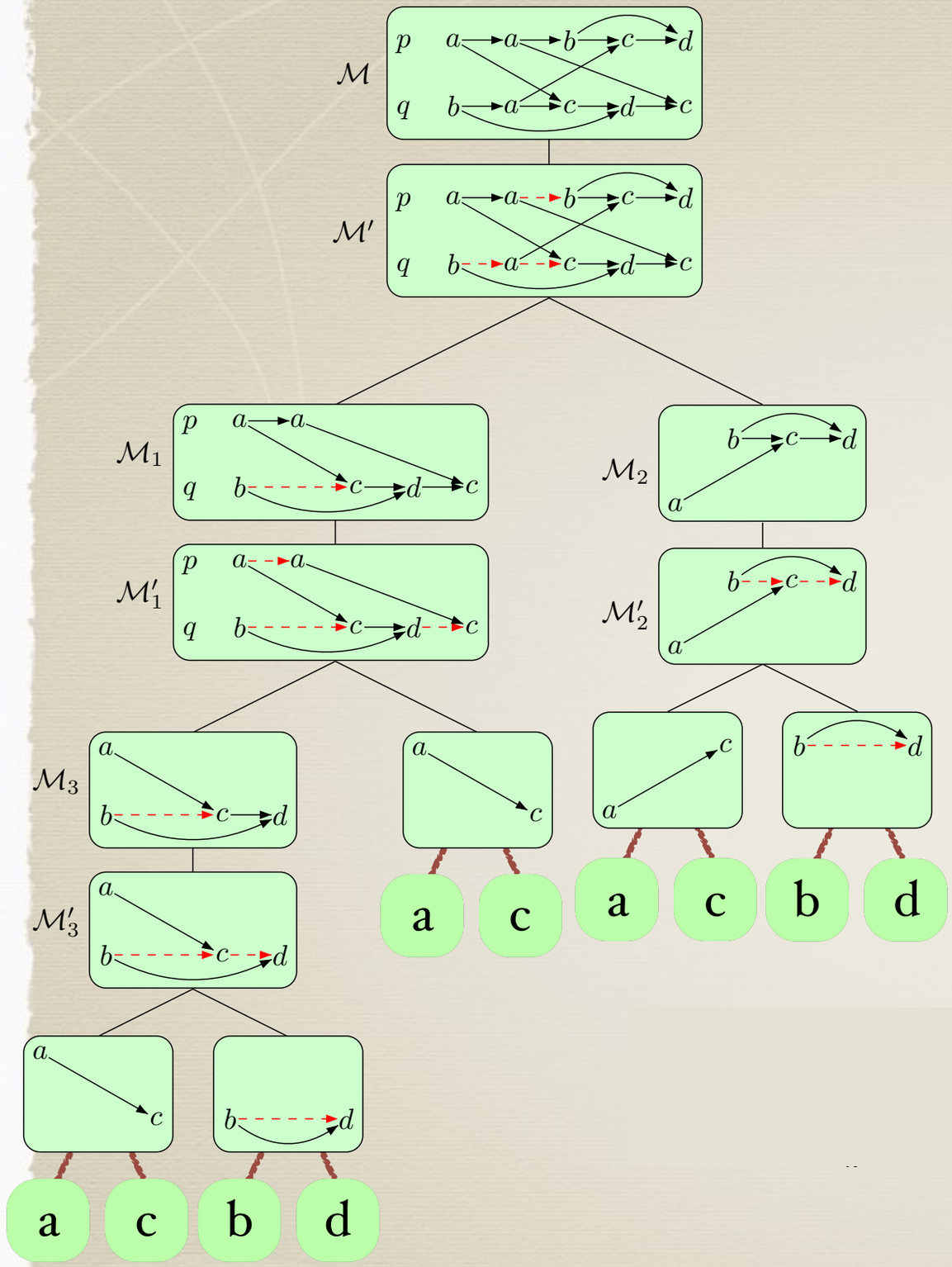
Tree interpretation

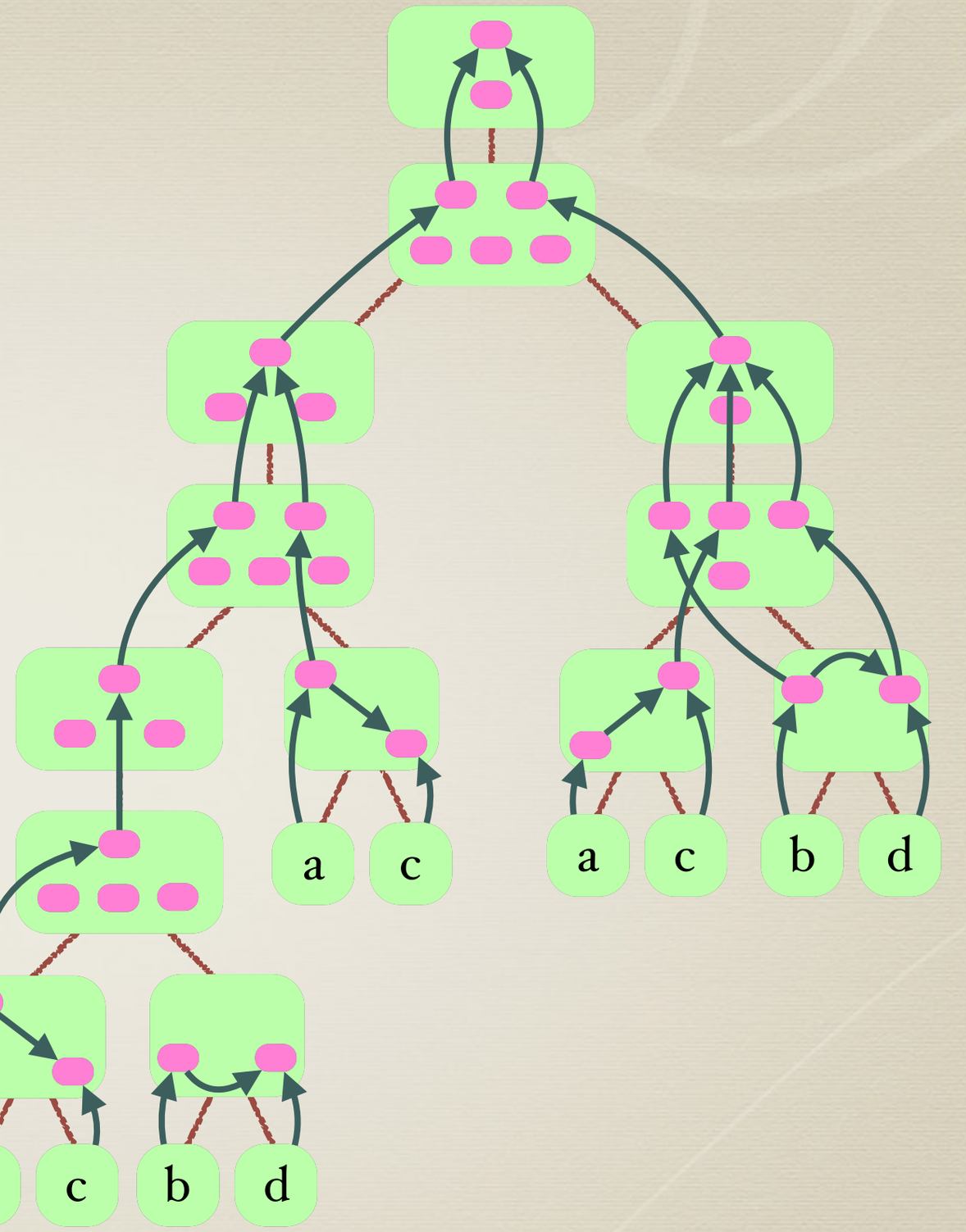




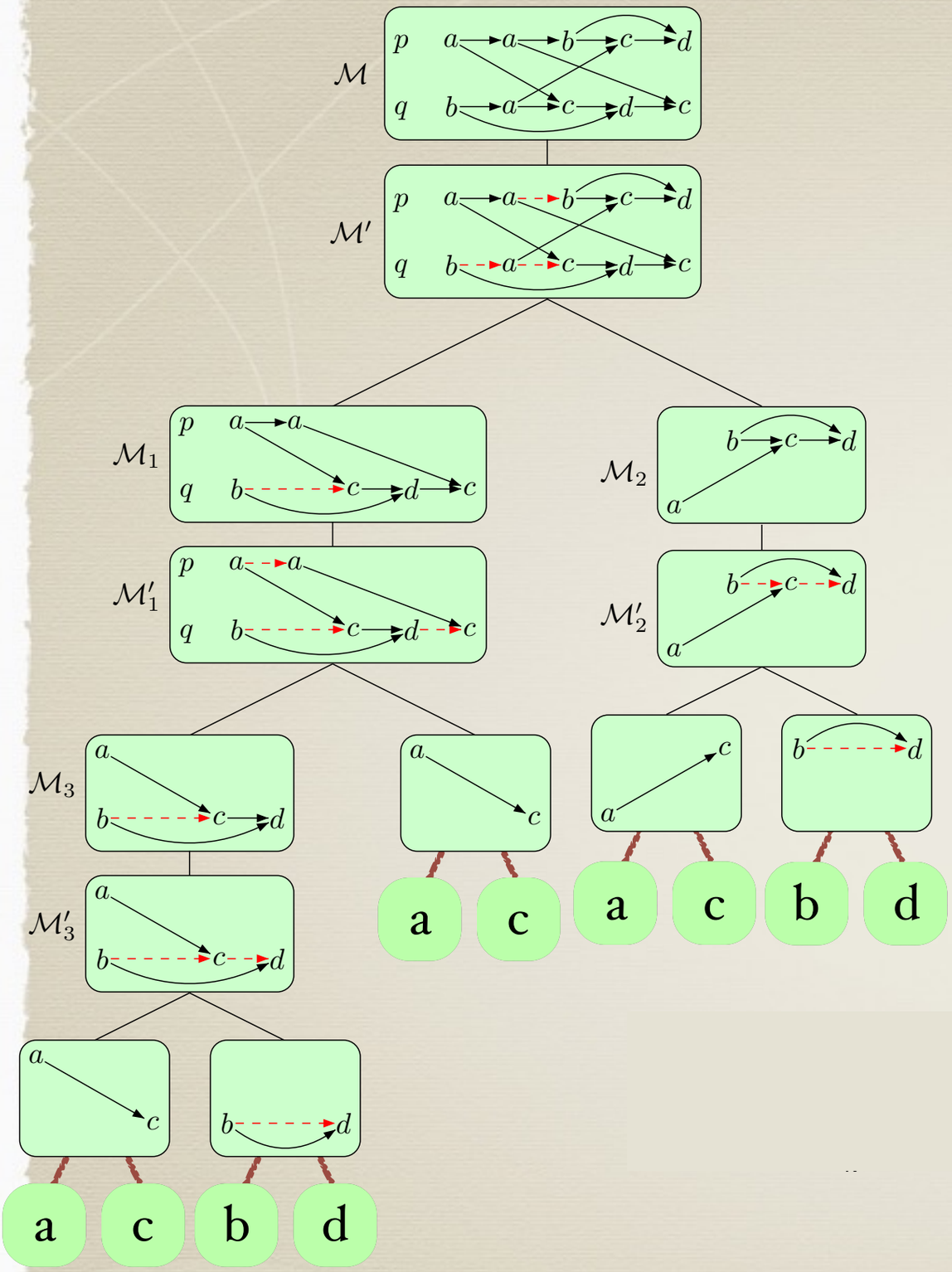
Data edges = Above Leaves

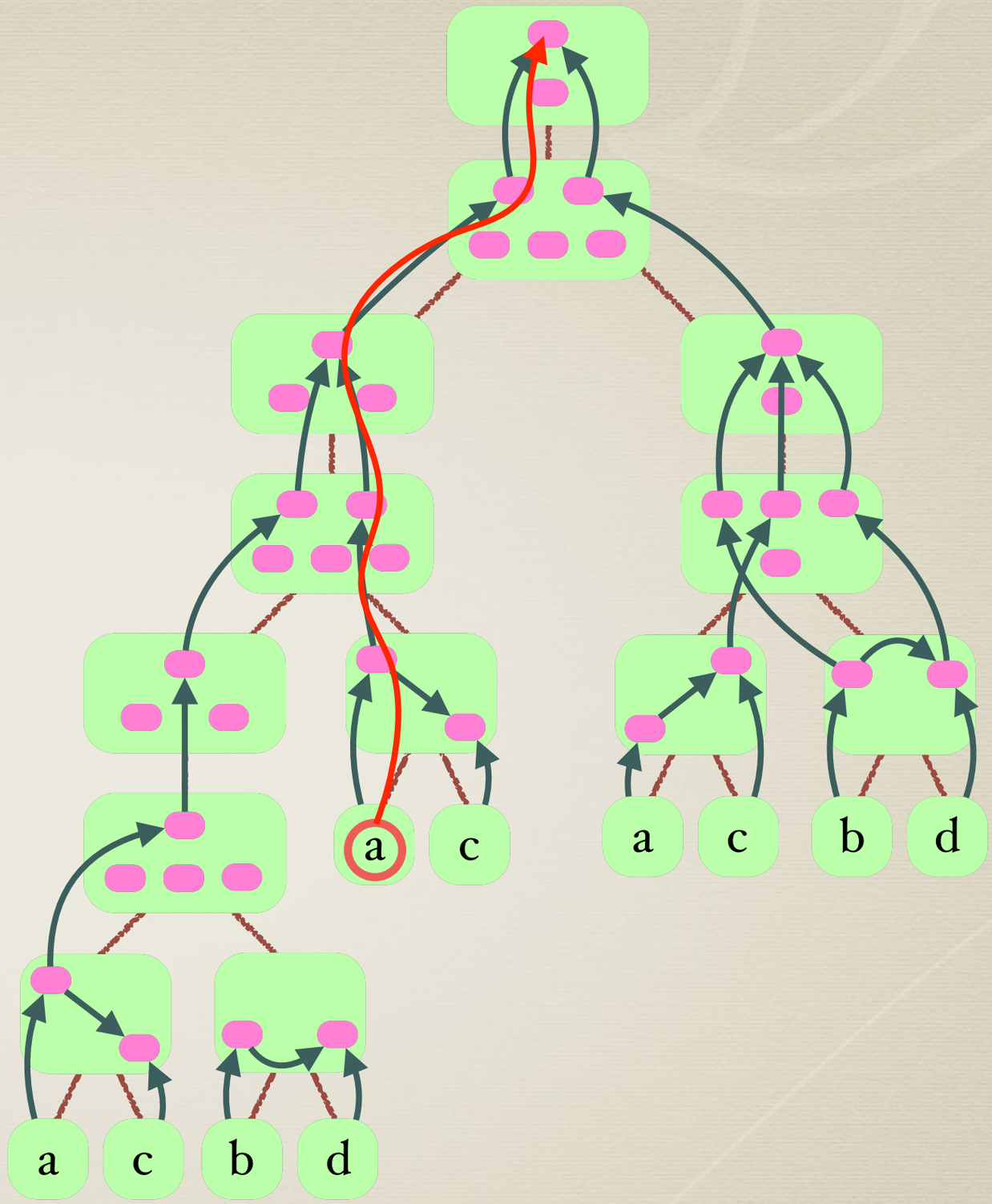
Tree interpretation



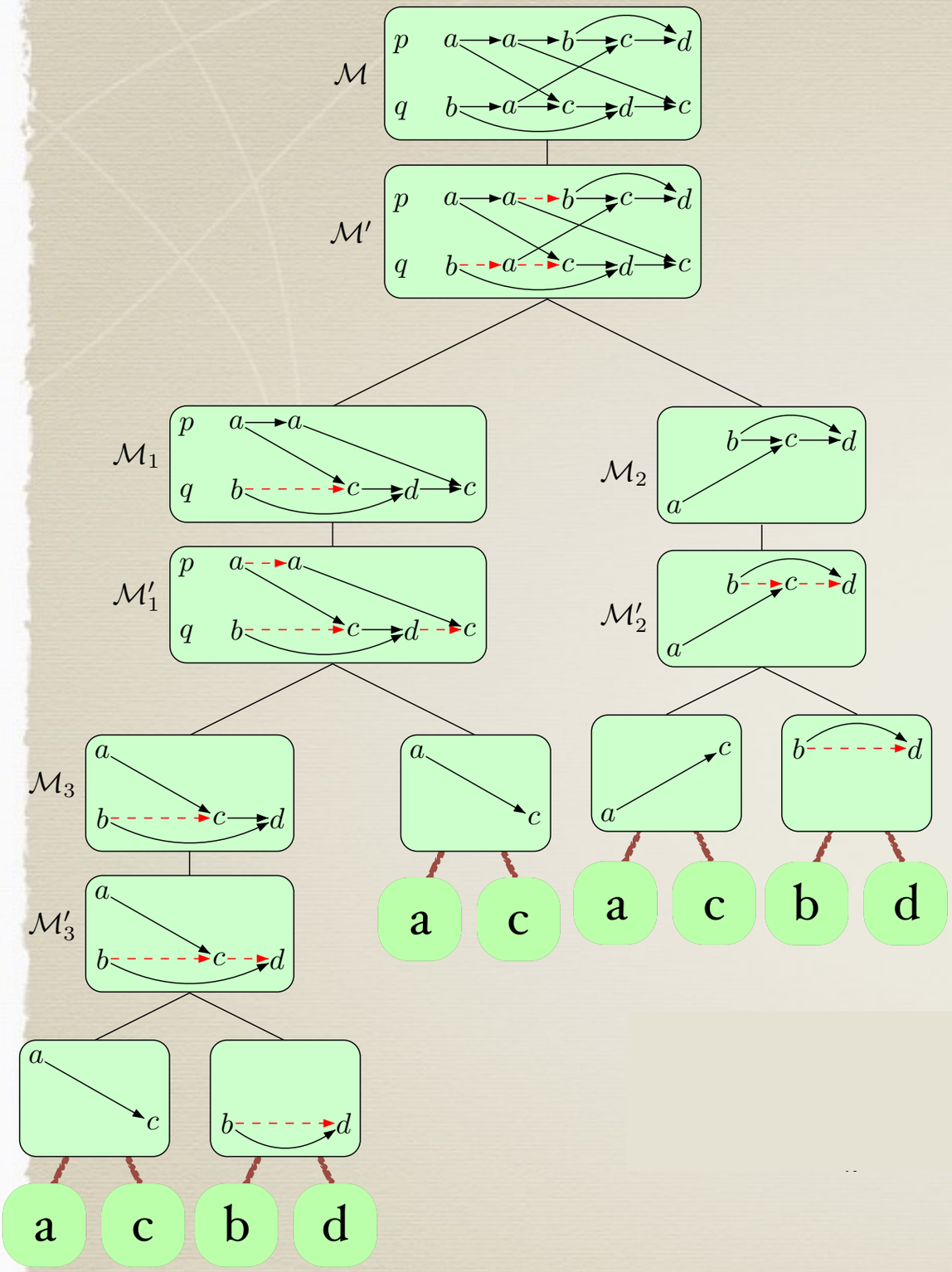


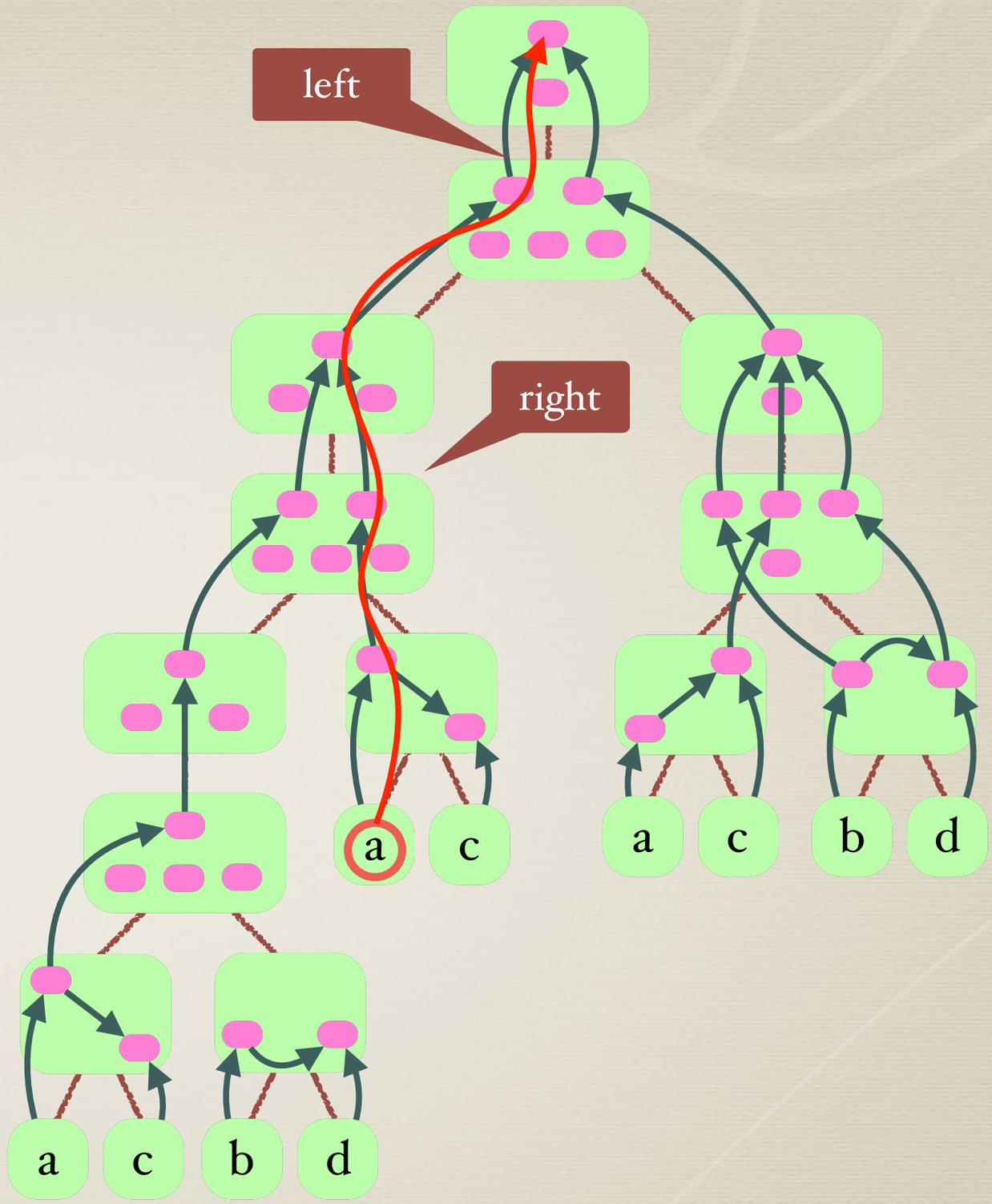
Tree interpretation



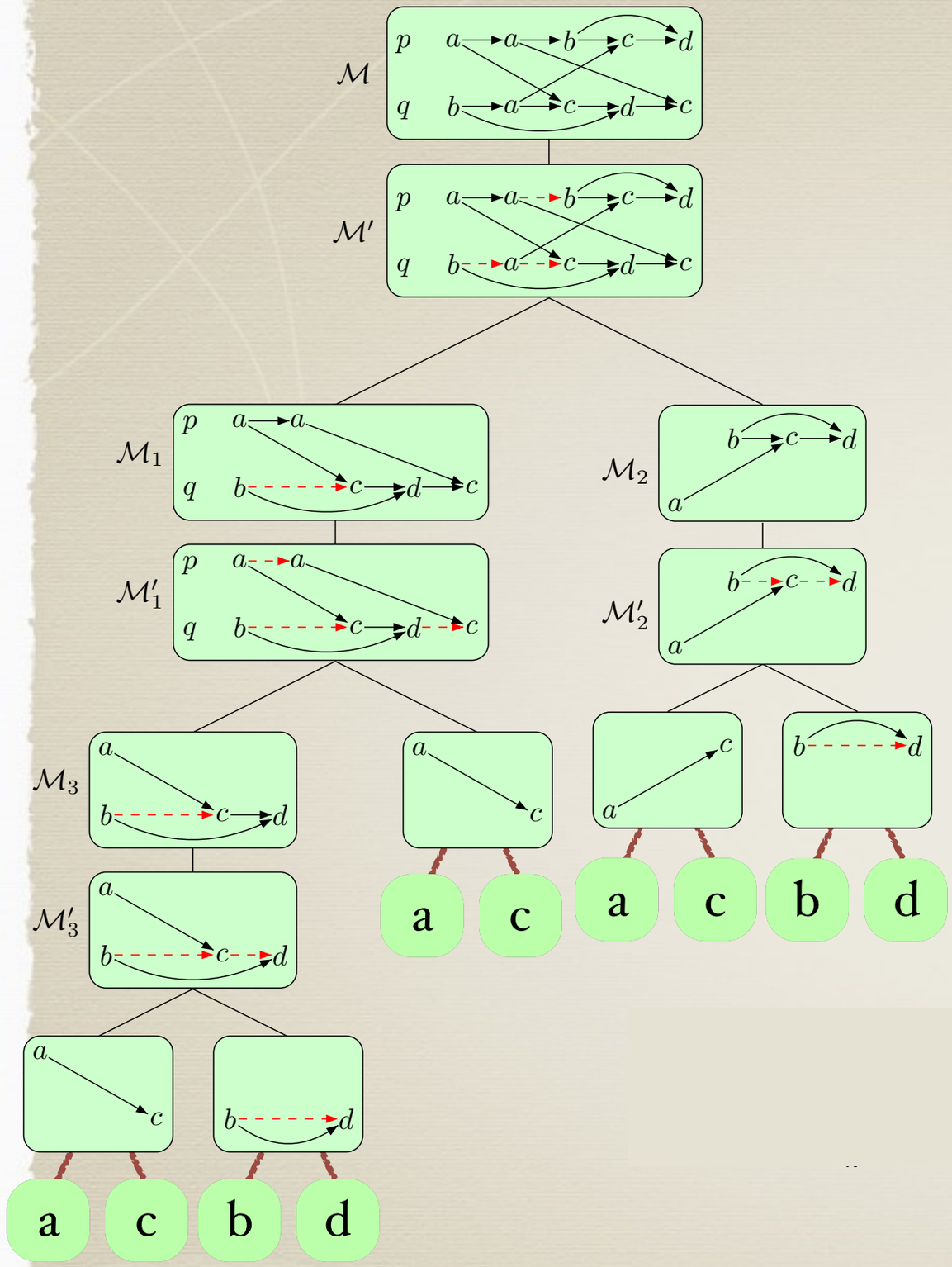


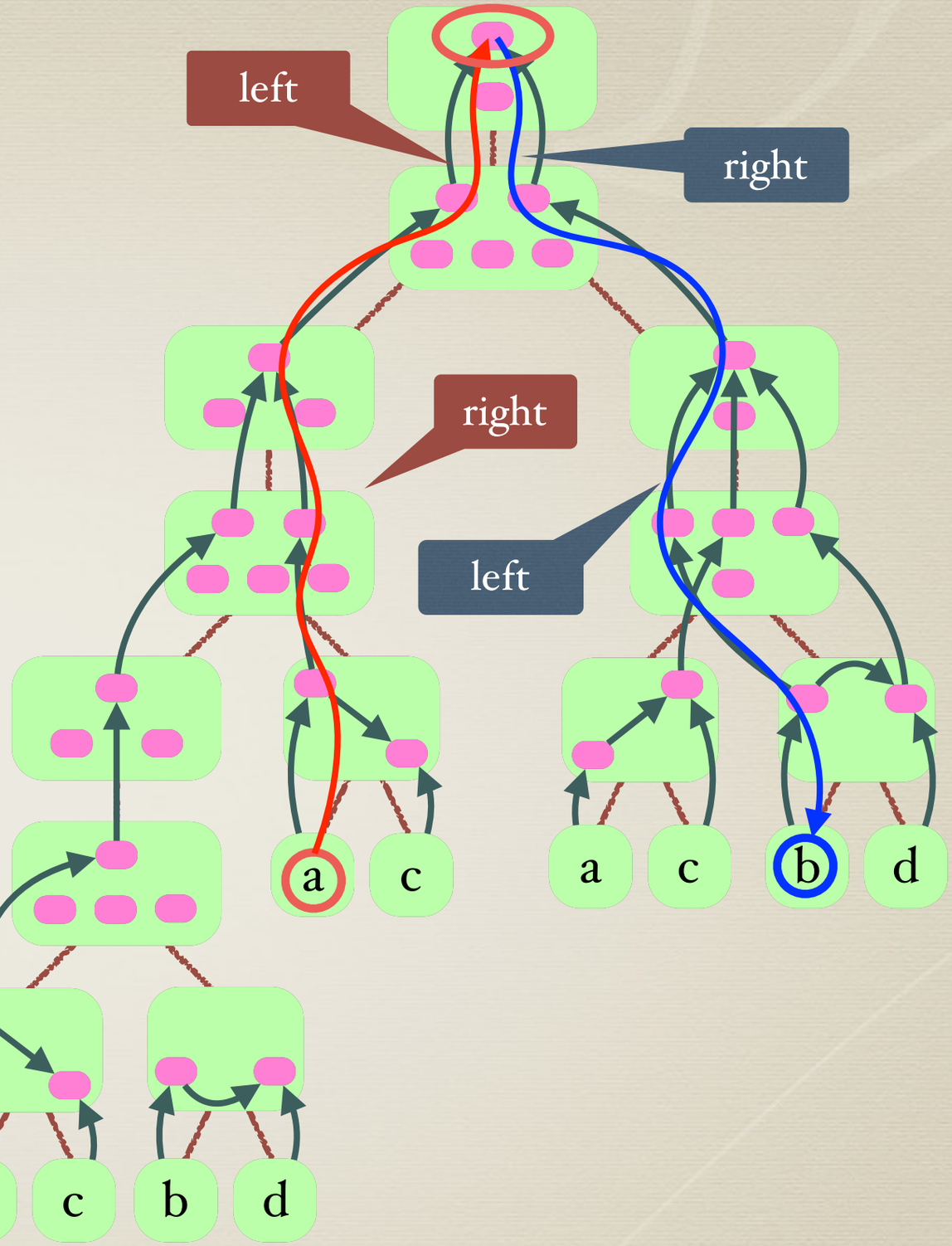
Tree interpretation



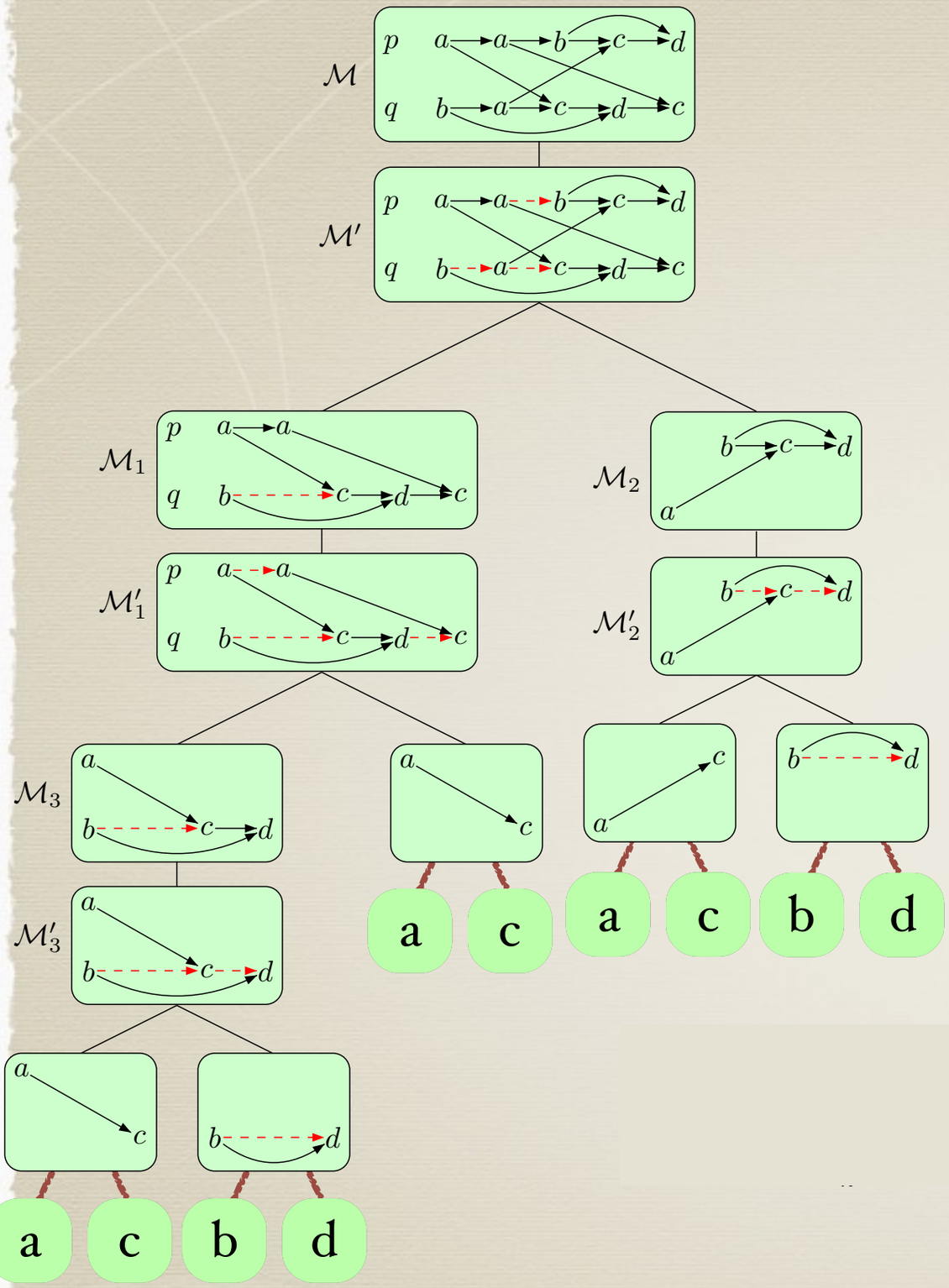


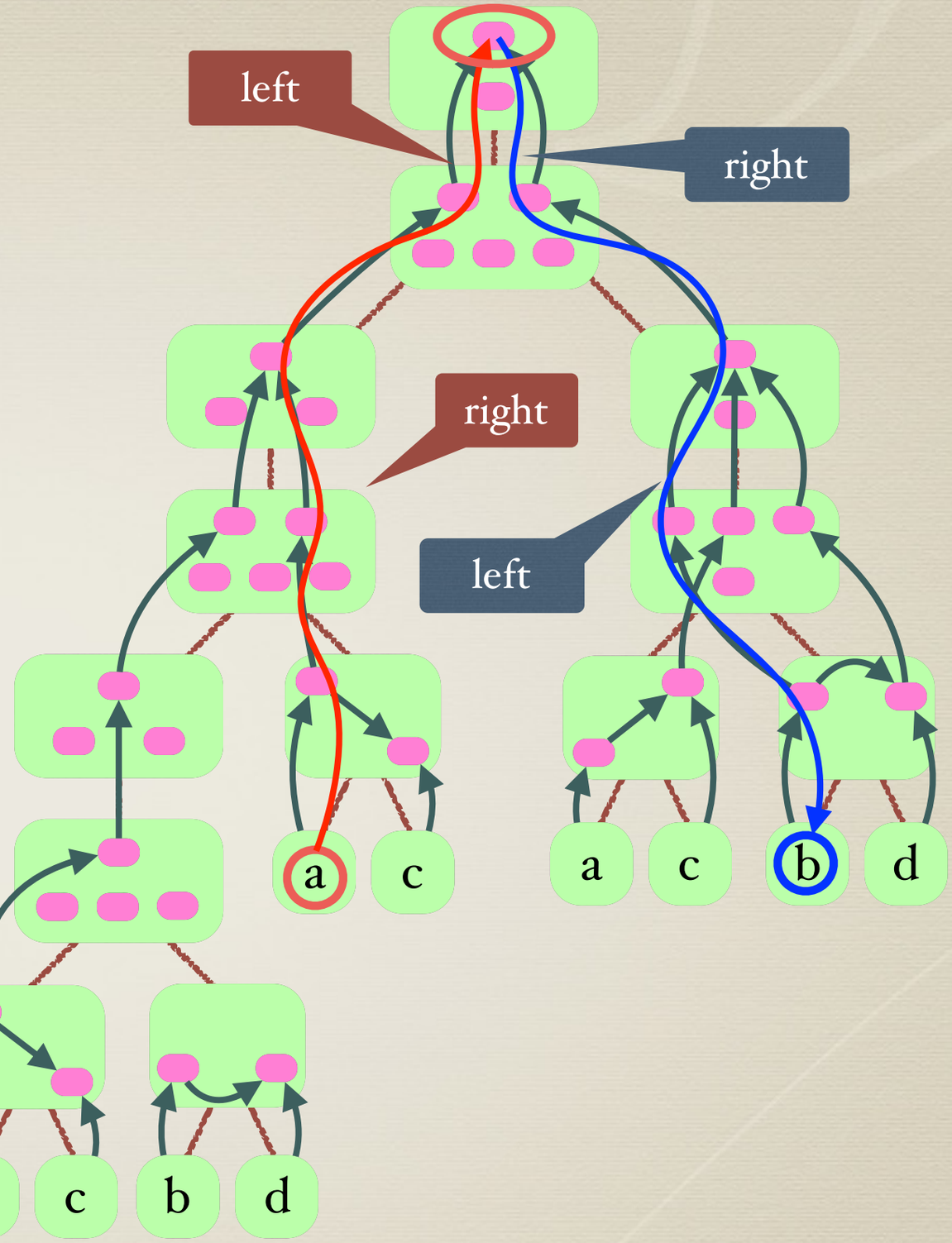
Tree interpretation





Tree interpretation





Process edges = Work!!

Tree interpretation

Existentially Bounded MSCs

* The class of MSCs that are existentially k bounded have split-width k+1
* Existentially k-bounded MSCs form an MSO definable class (GenKusMus07)

Theorem: (GenKusMus07)

1. MSO theory of existentially k bounded MSCs is decidable.
2. MSO model checking for Message Passing Automata w.r.t existentially k-bounded behaviours is decidable.

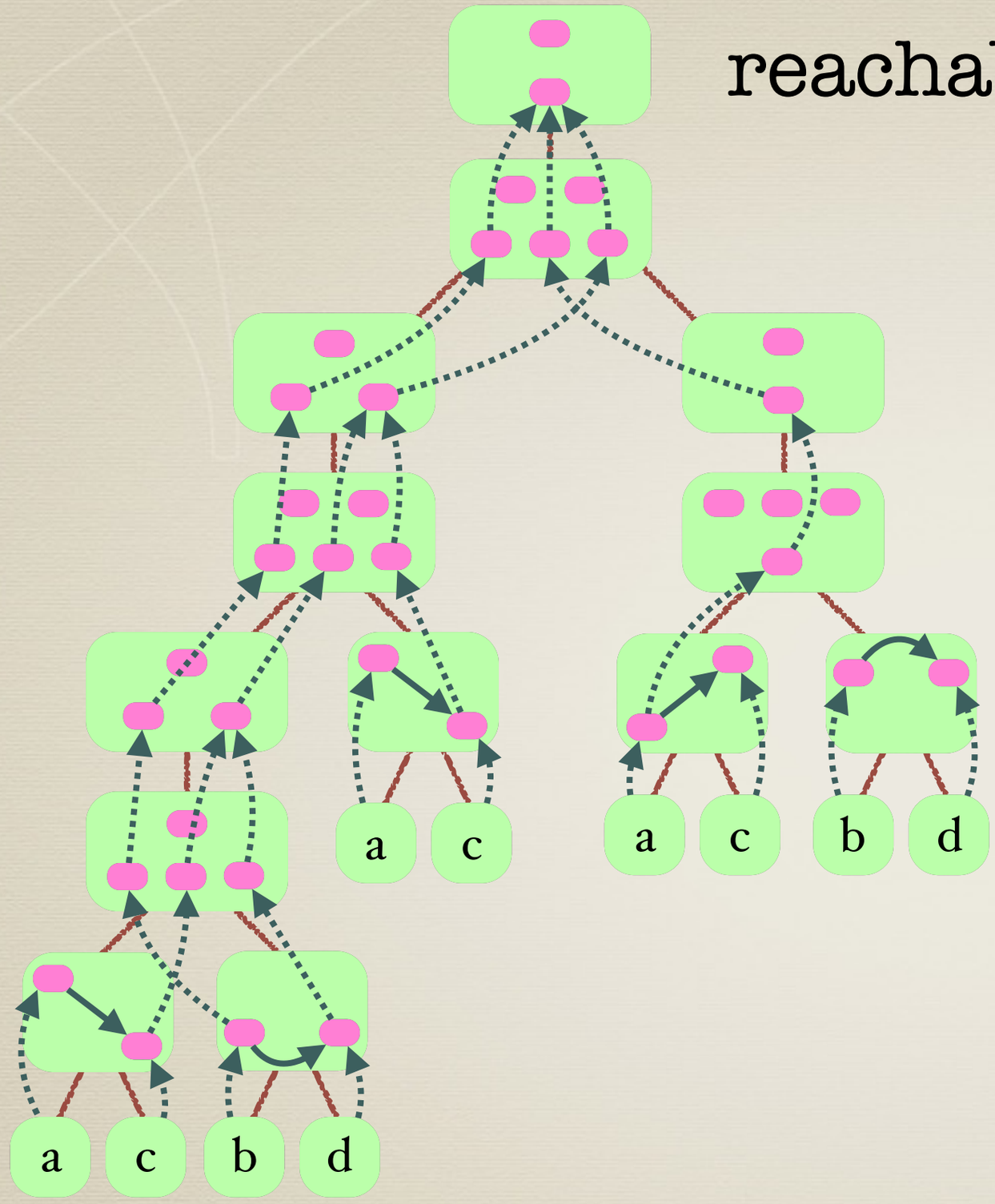
Bounded Scope Behaviours

* The class of MNWs with scope bound k has split-width k+1
* k bounded-scope MNWs is MSO expressible.

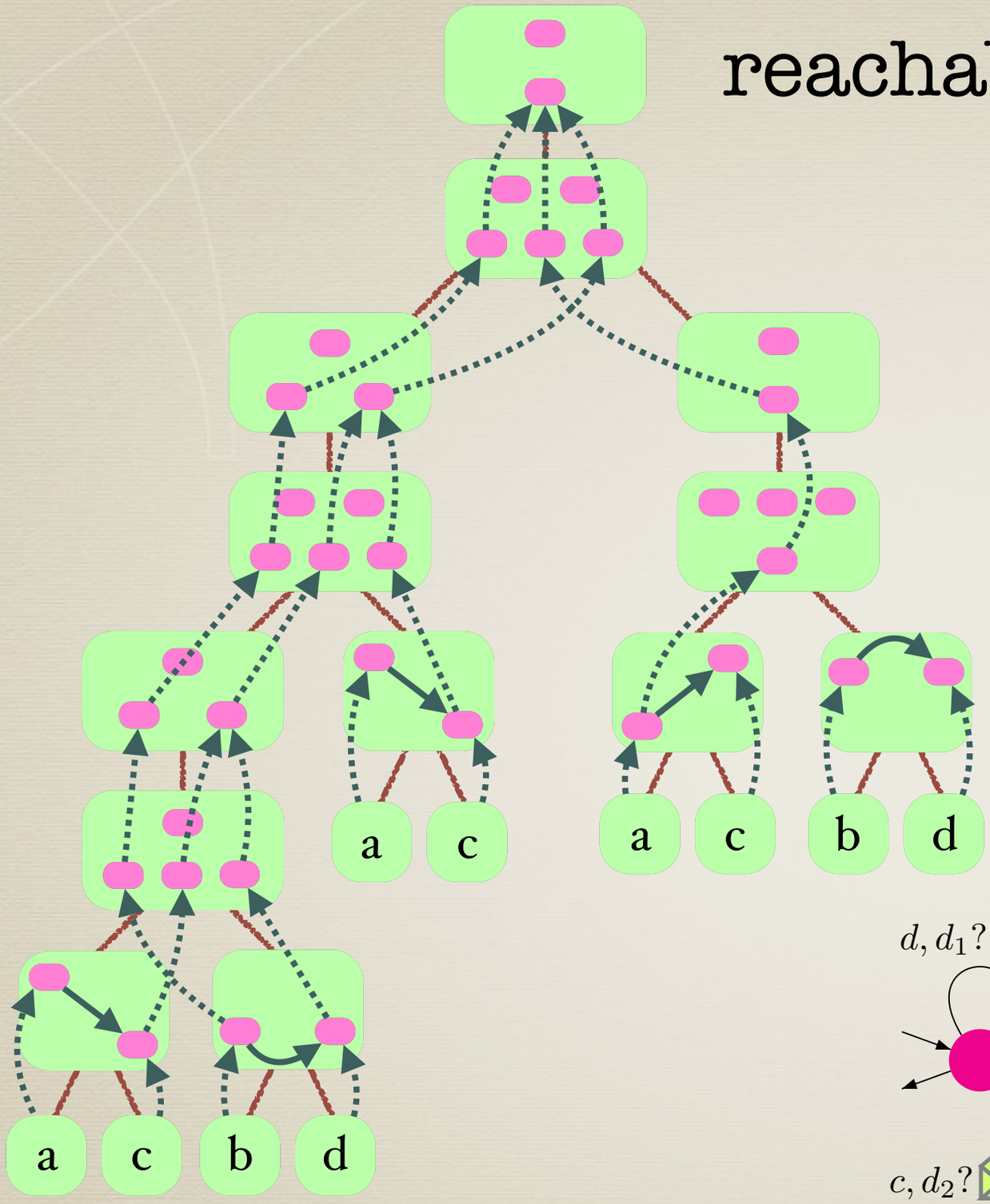
Theorem:

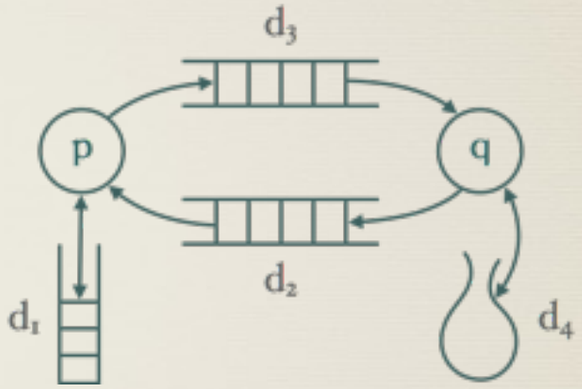
1. MSO theory of k scope-bound MNWs is decidable.
2. MSO model checking for MPDS w.r.t k scope-bounded behaviours is decidable.

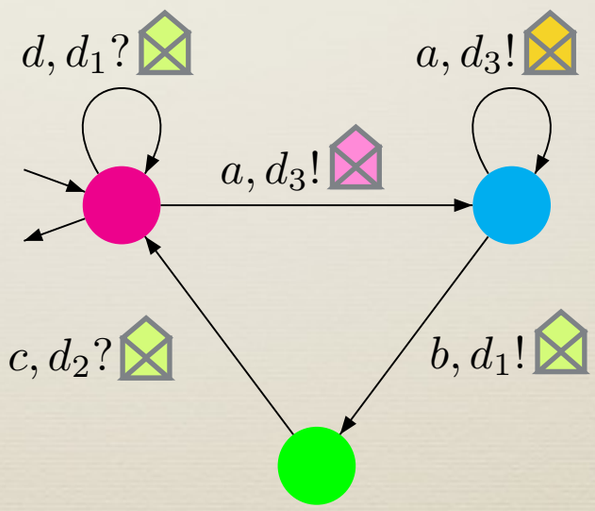
reachability/emptiness

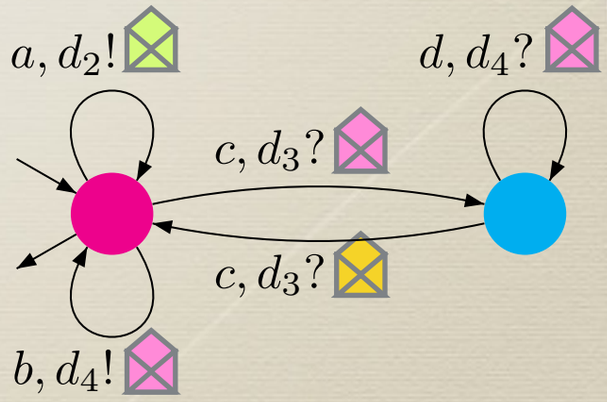


reachability/emptiness

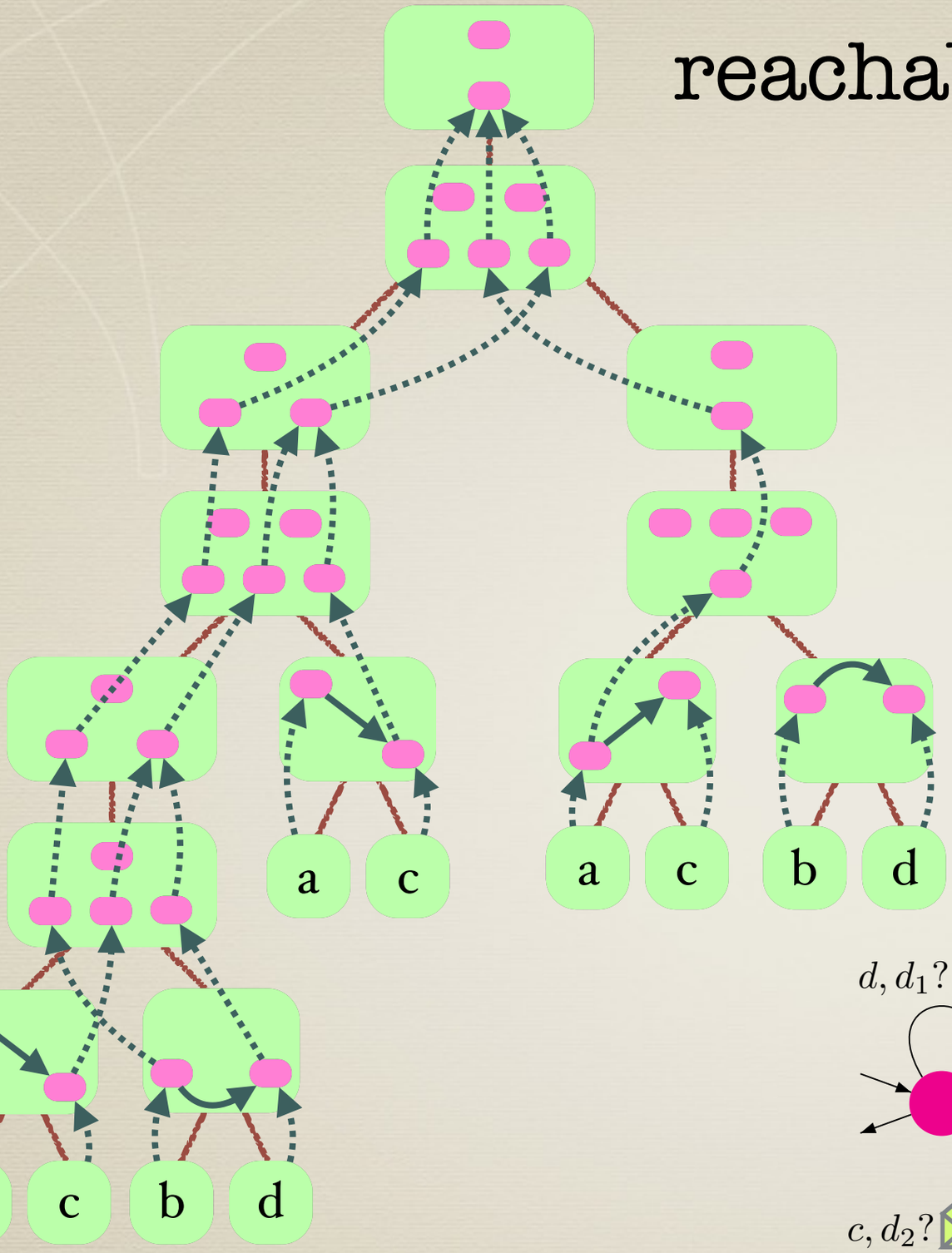


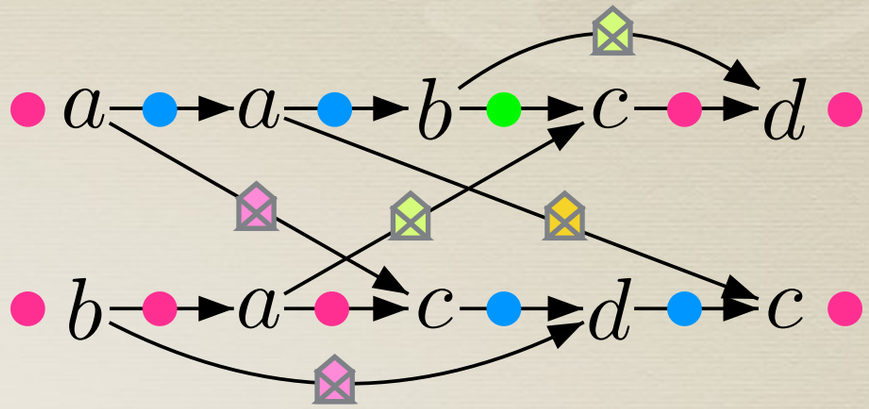


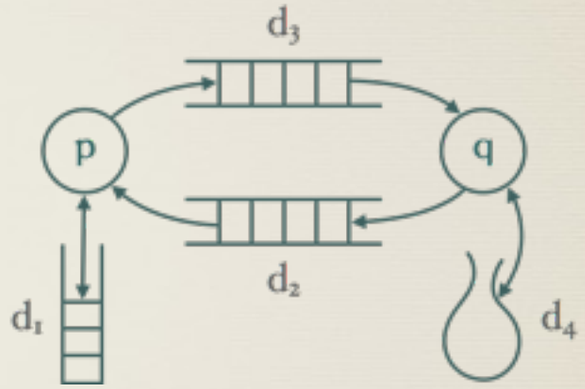


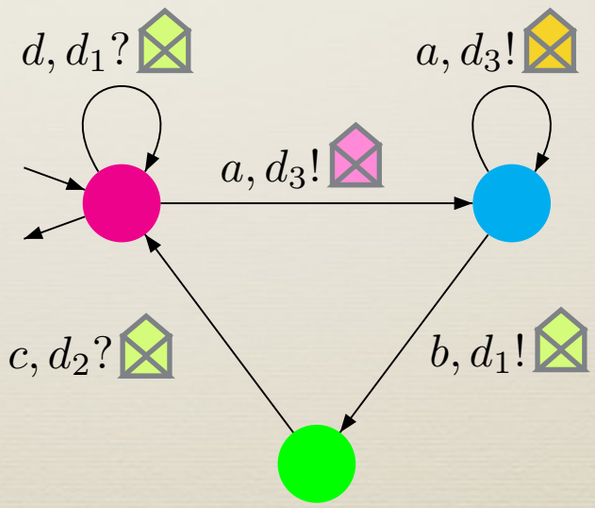


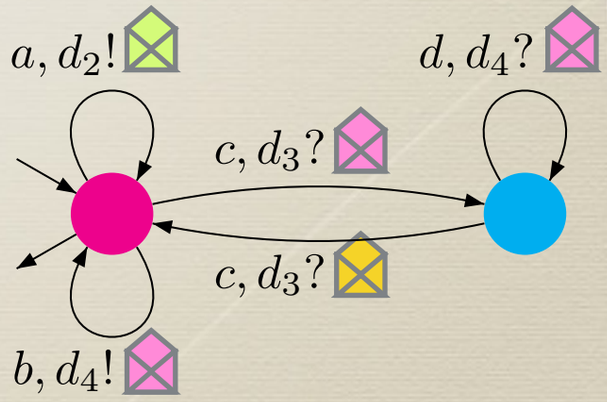
reachability/emptiness



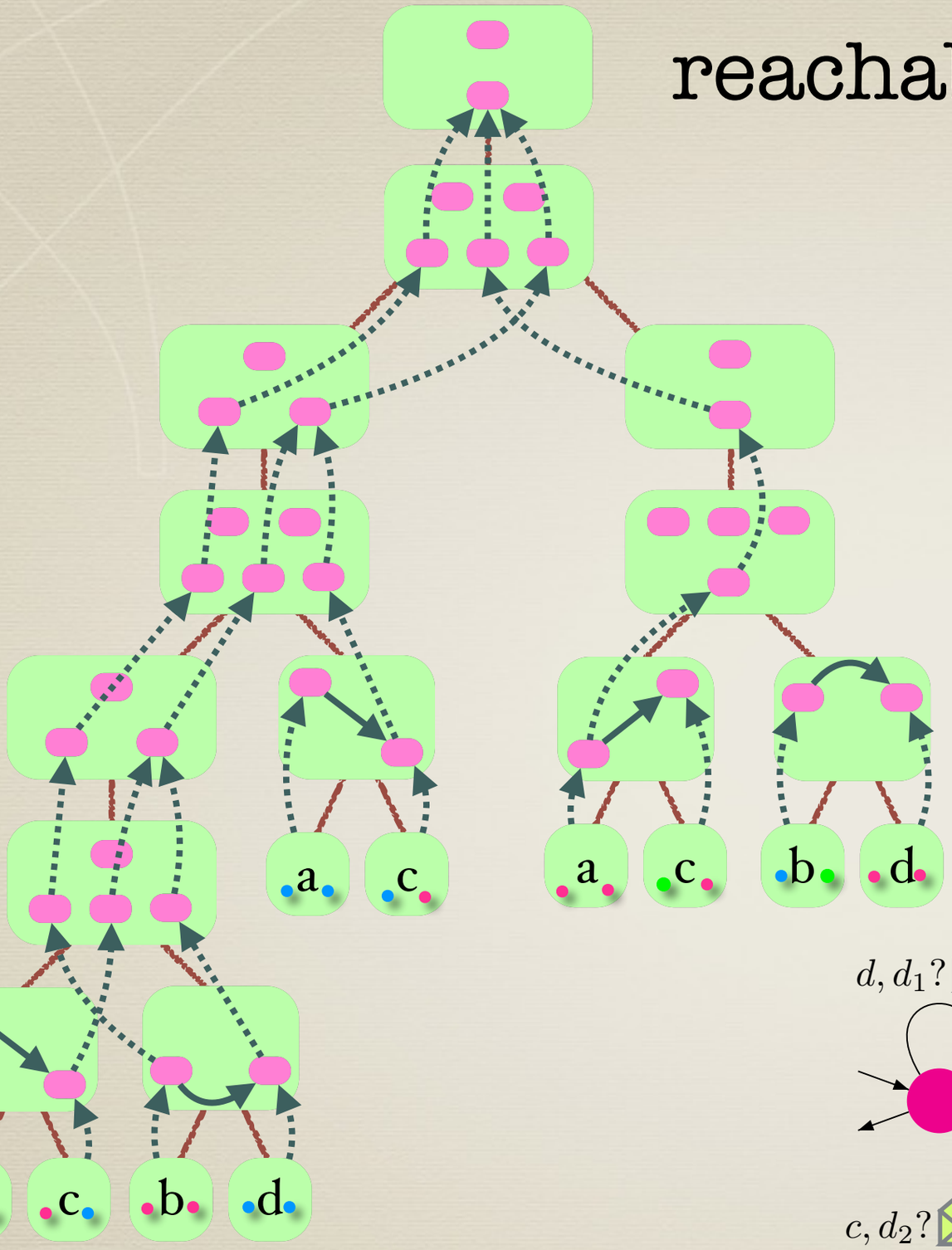


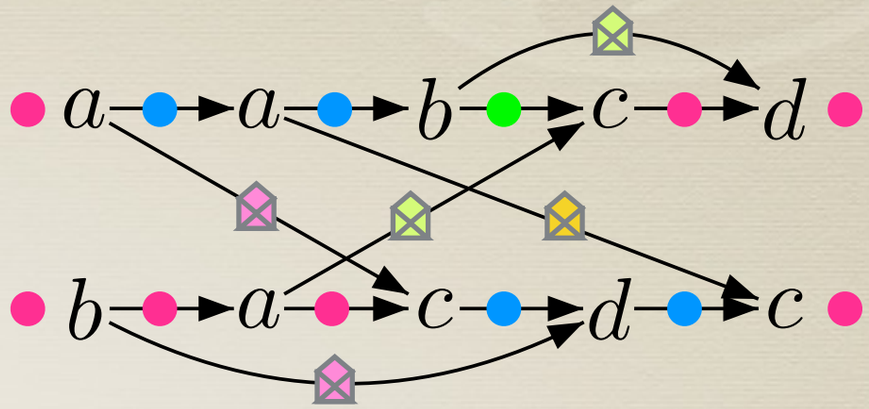


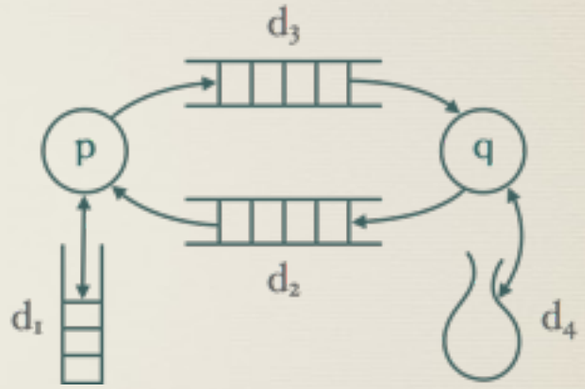


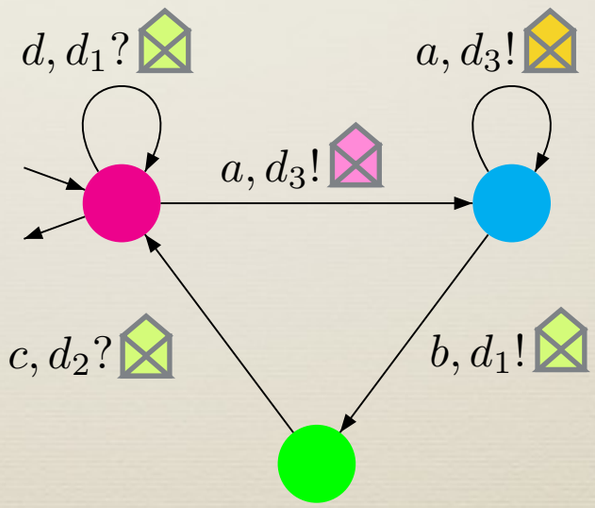


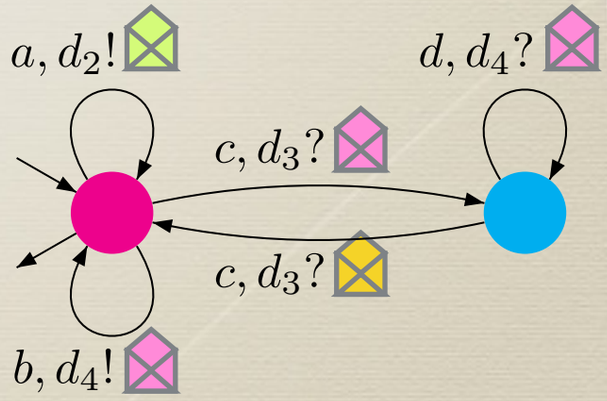
reachability/emptiness



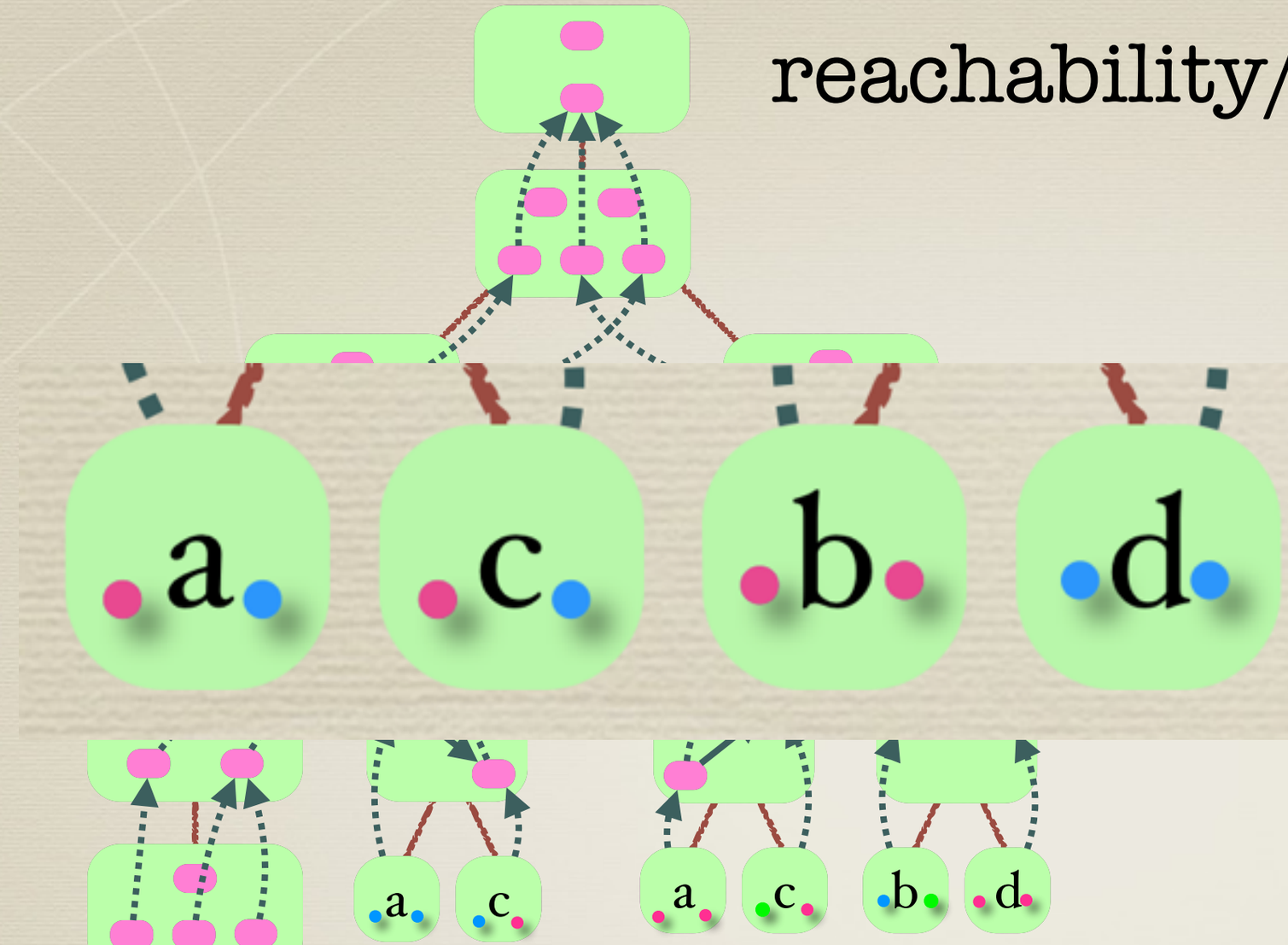


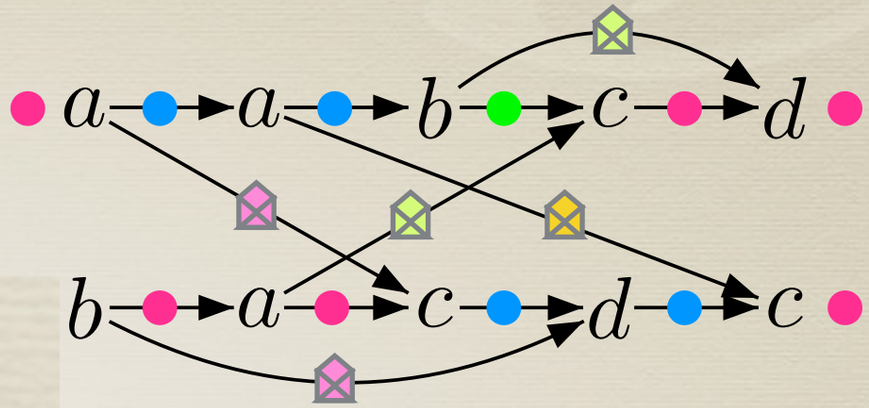


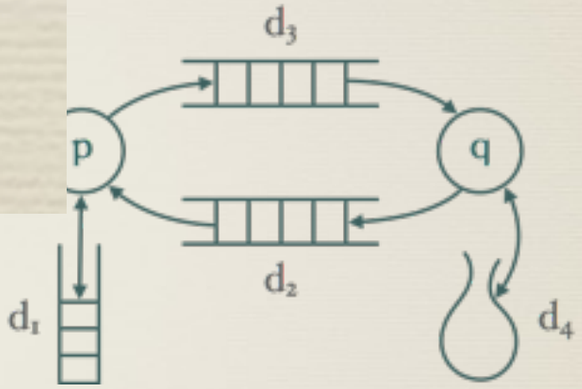


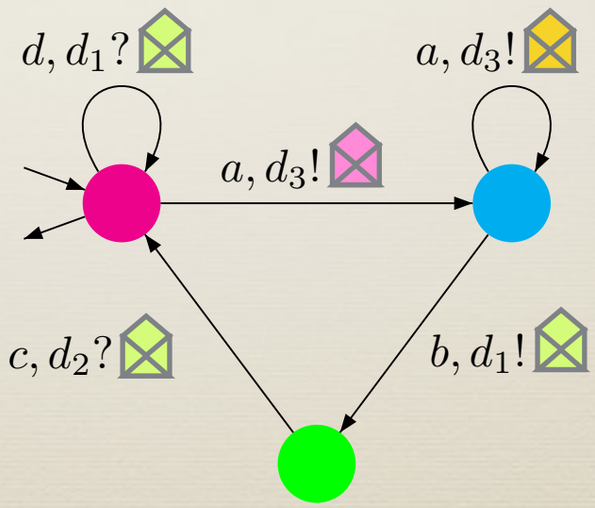


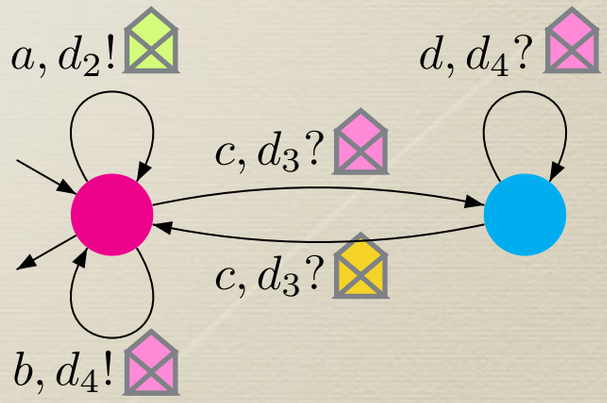
reachability/emptiness



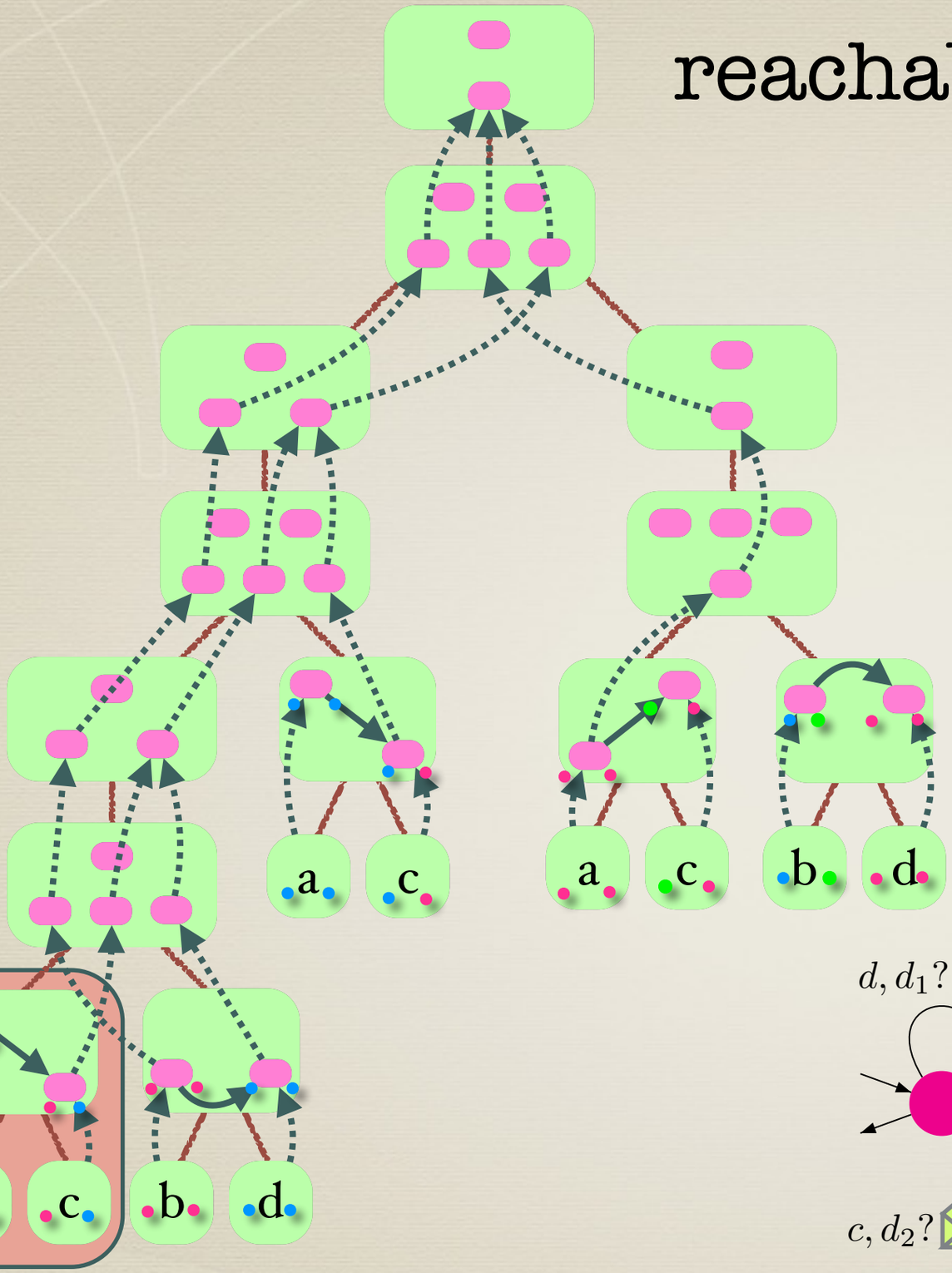


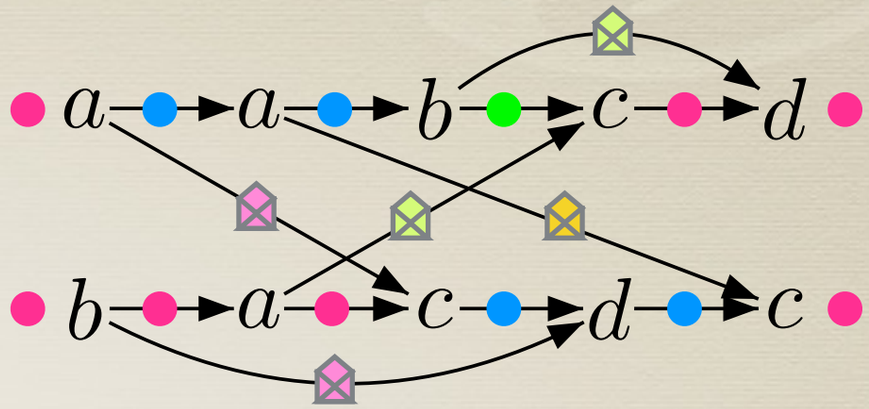


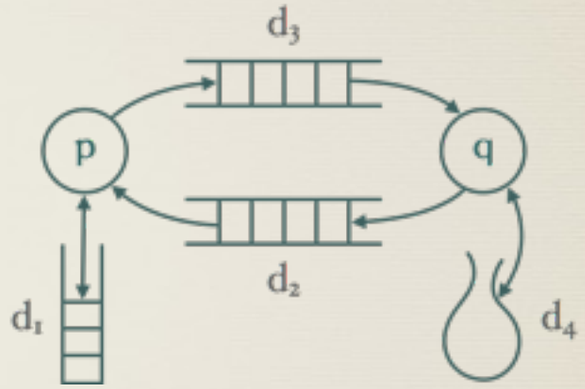


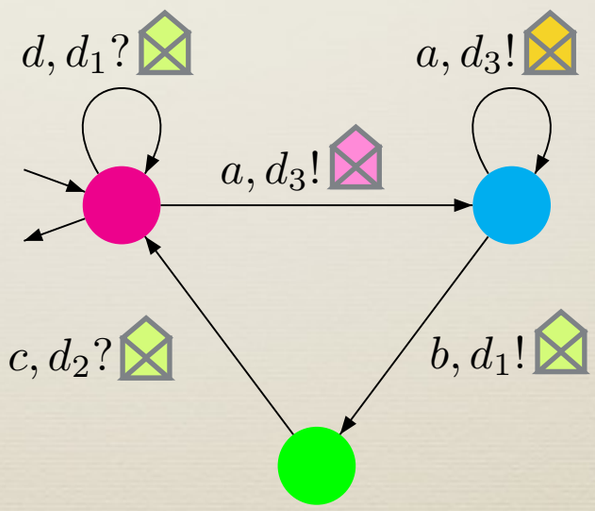


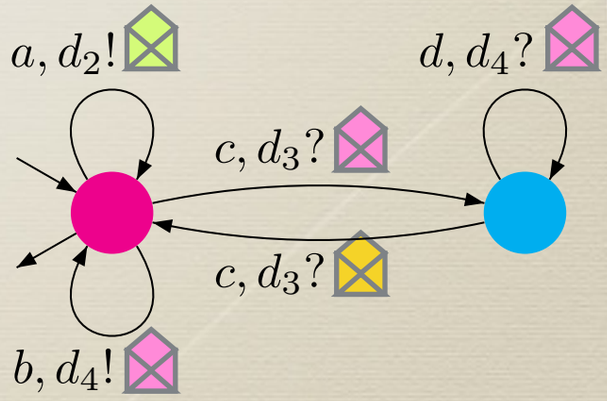
reachability/emptiness



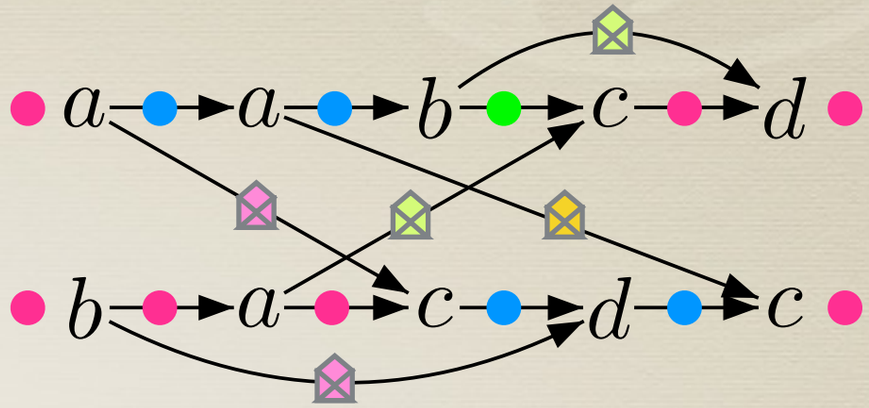


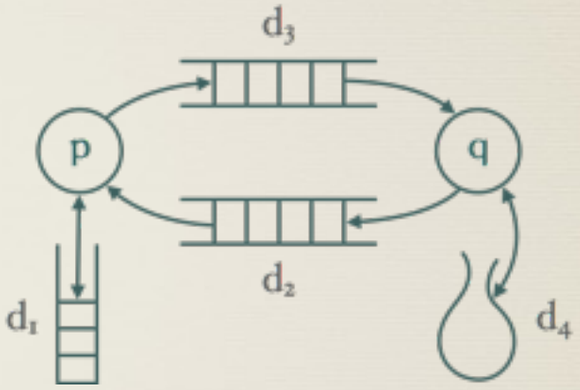


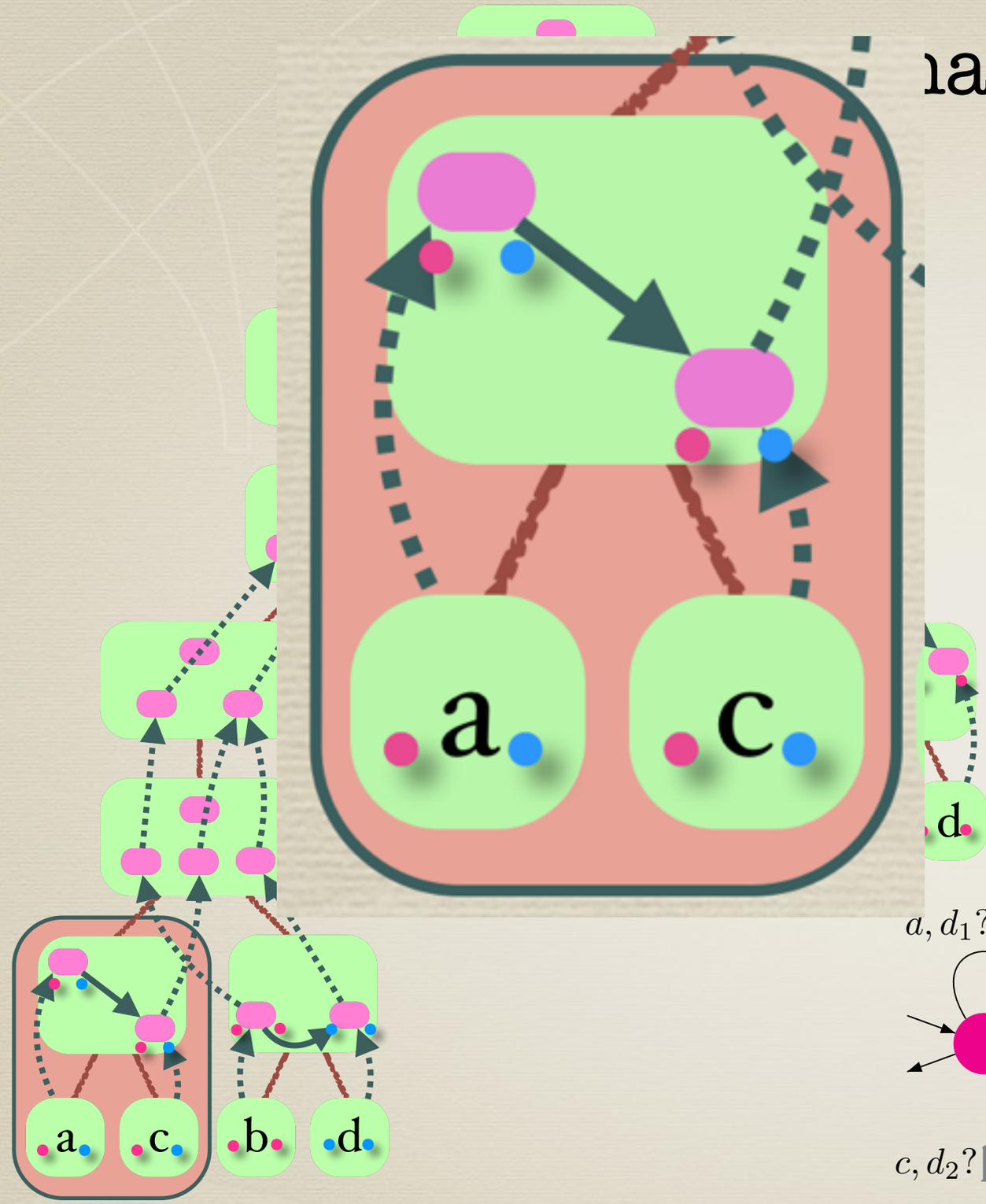


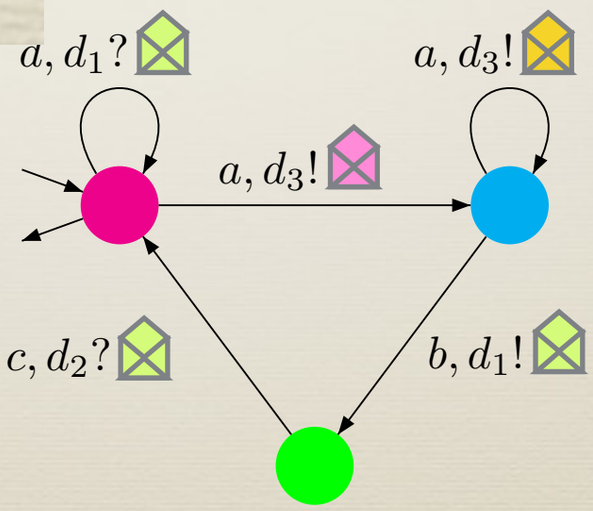


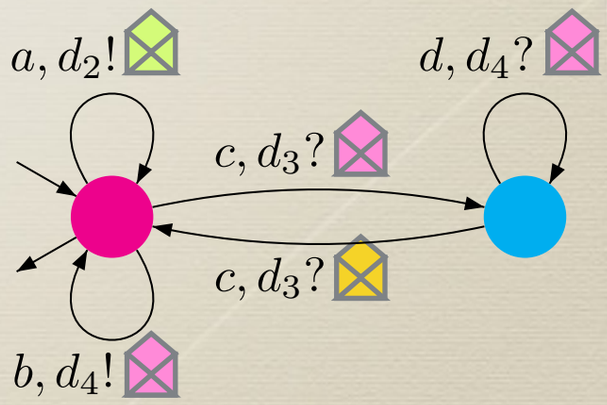
ıability/emptiness



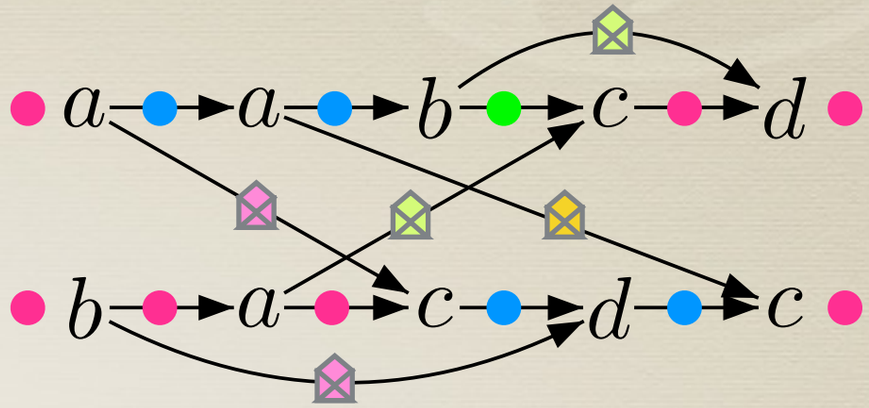


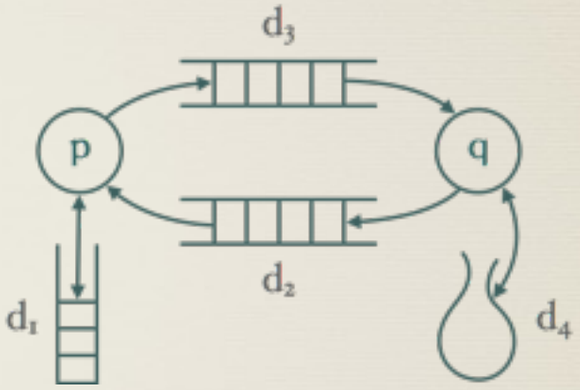


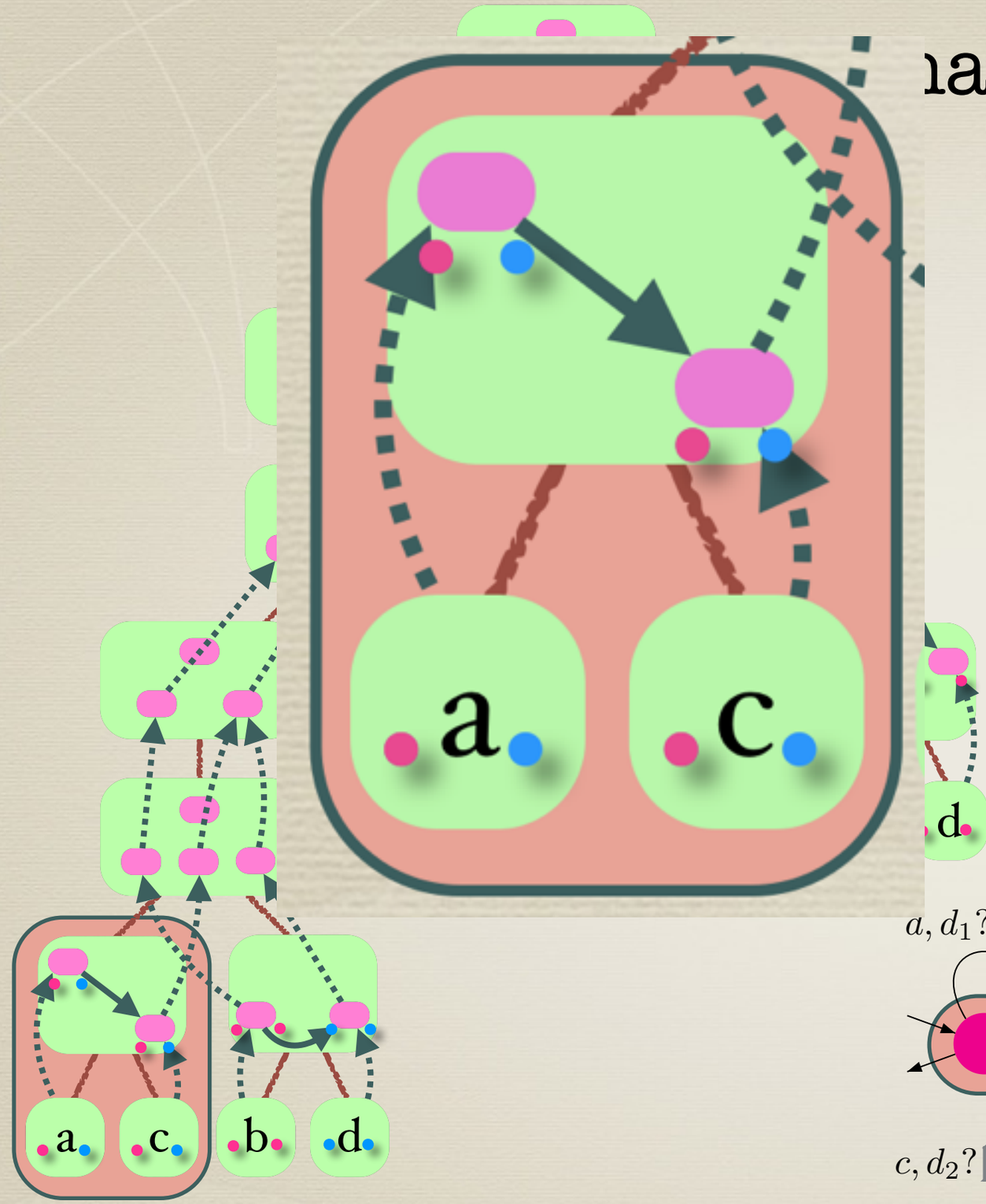


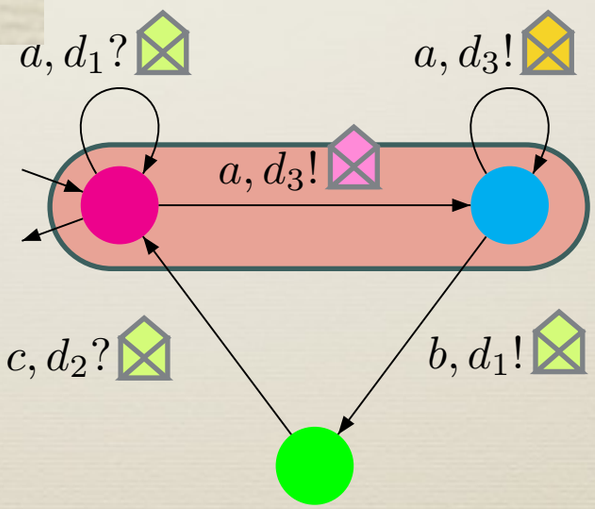


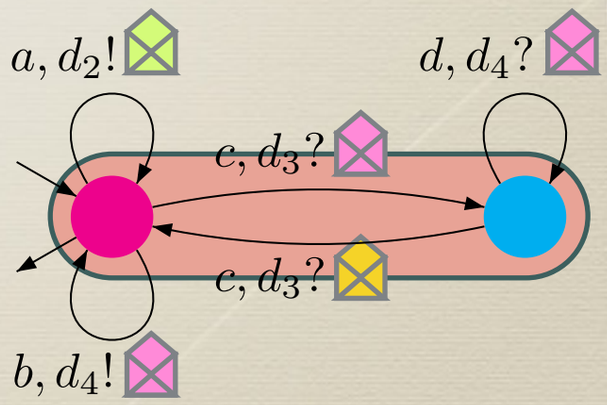
ıability/emptiness



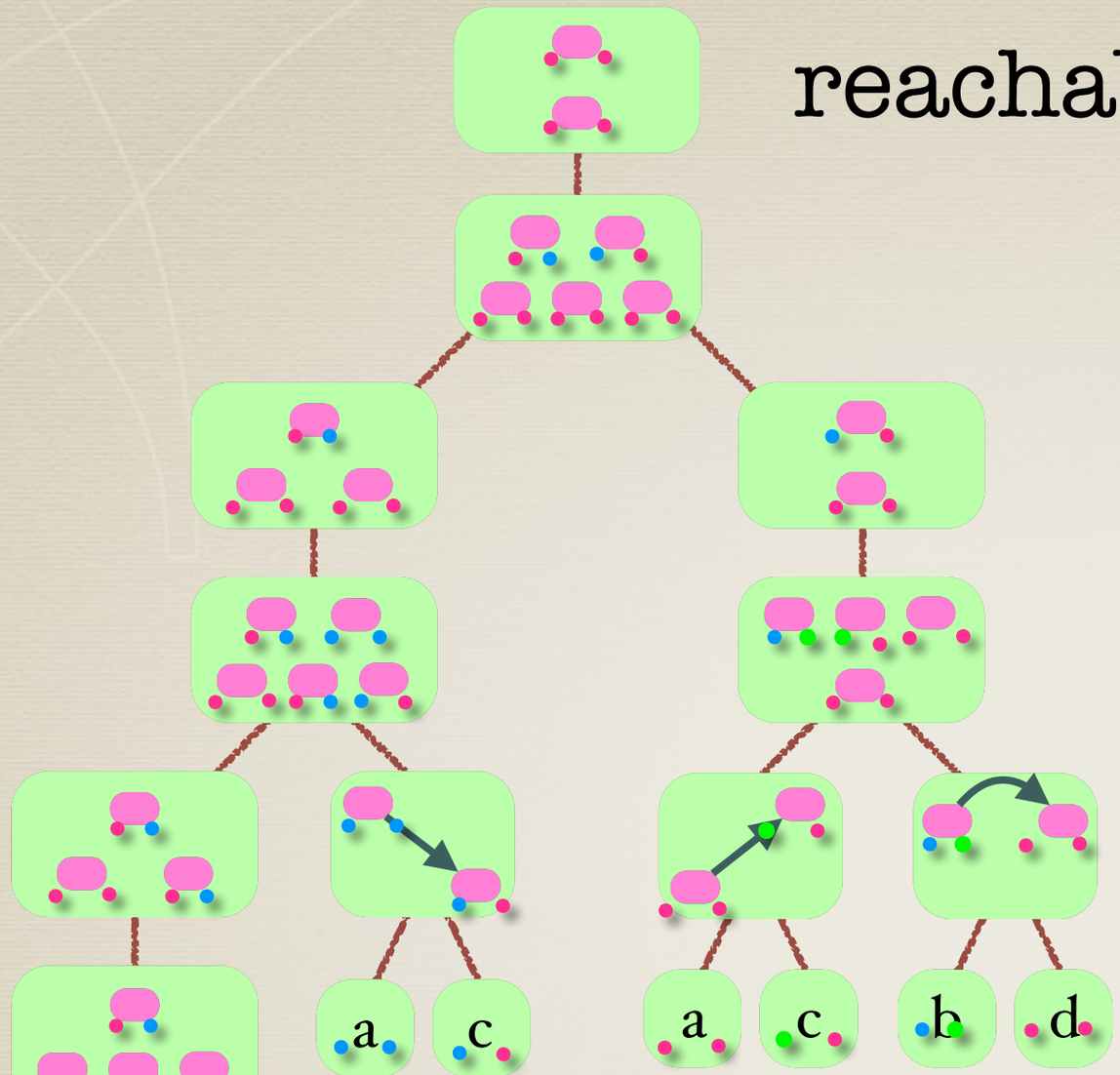


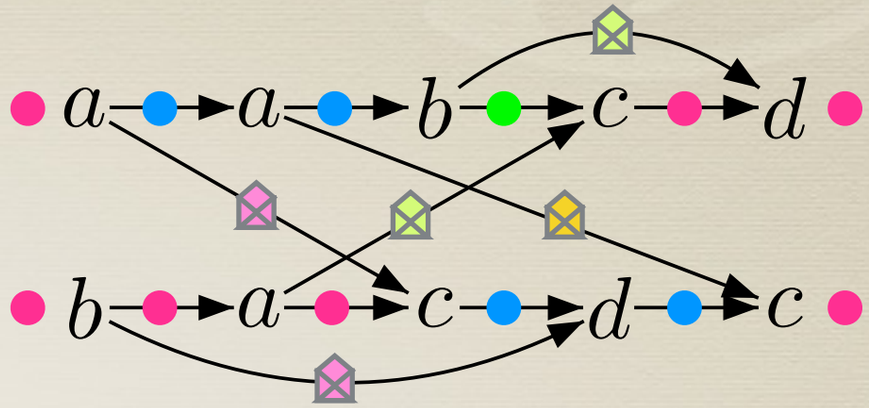


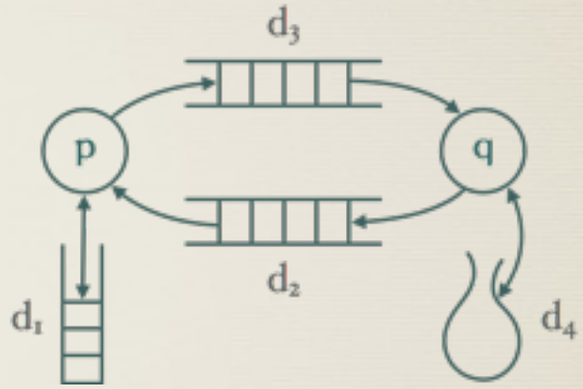


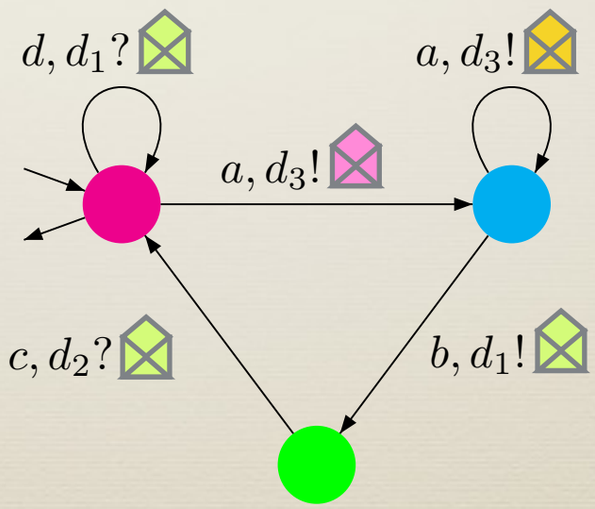


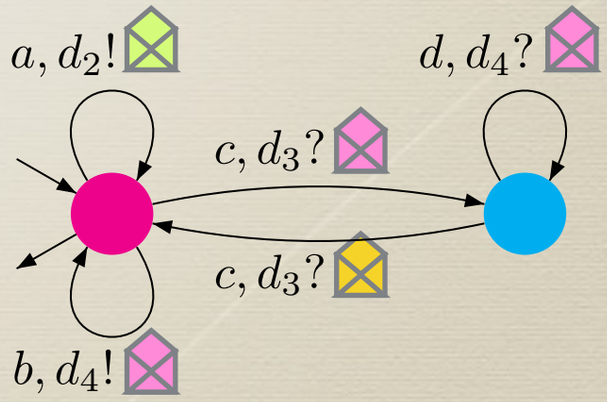
reachability



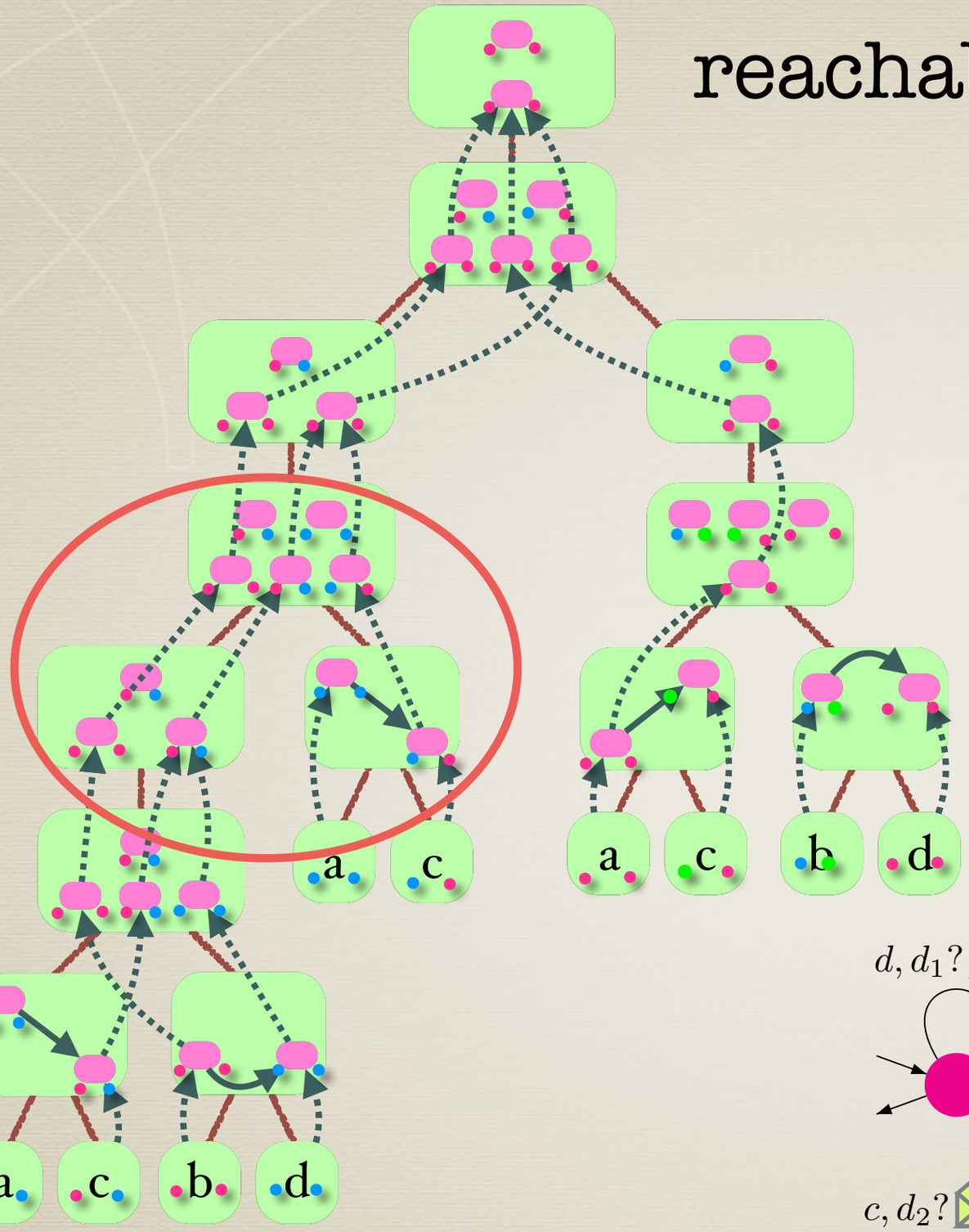


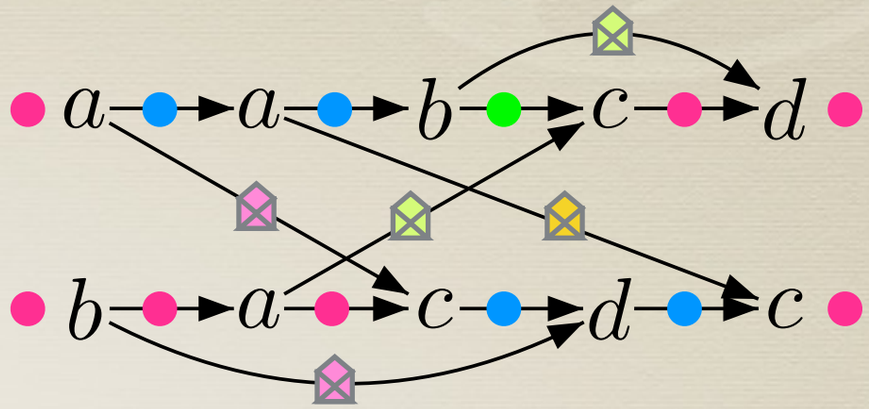


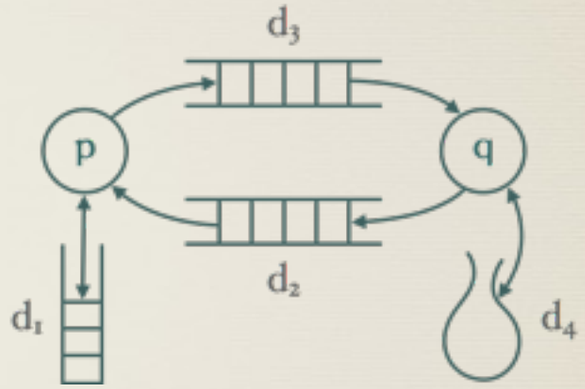


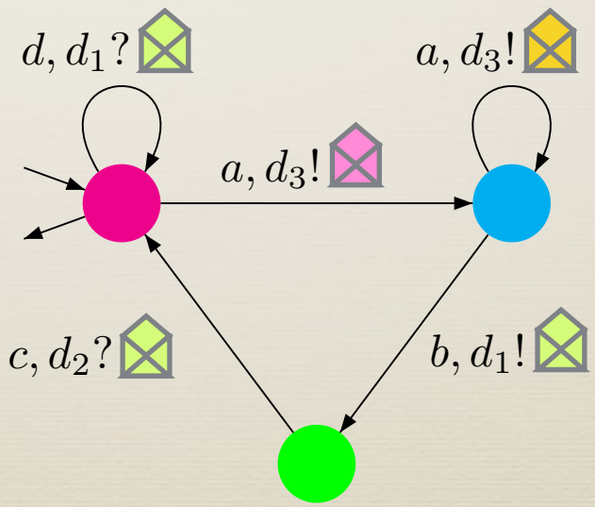


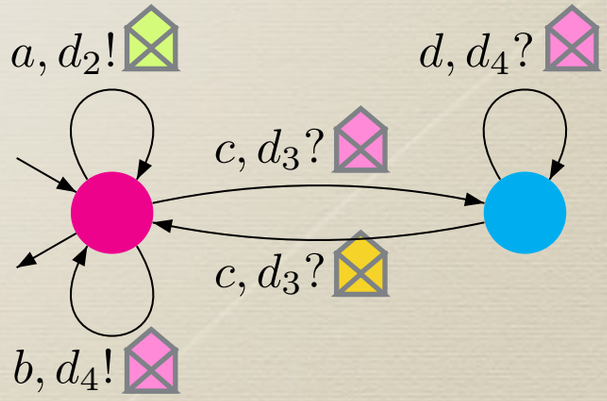
reachability/emptiness



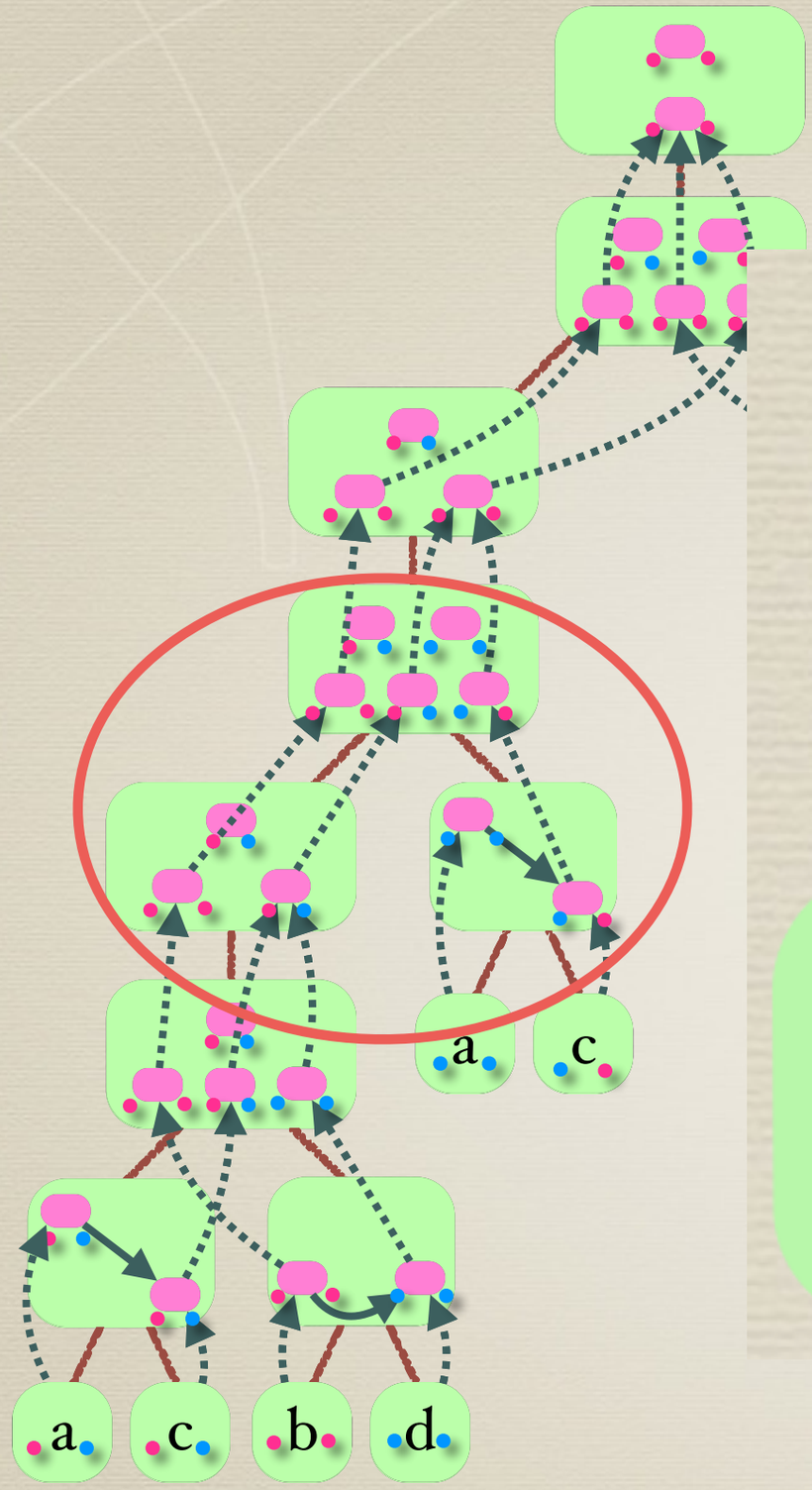


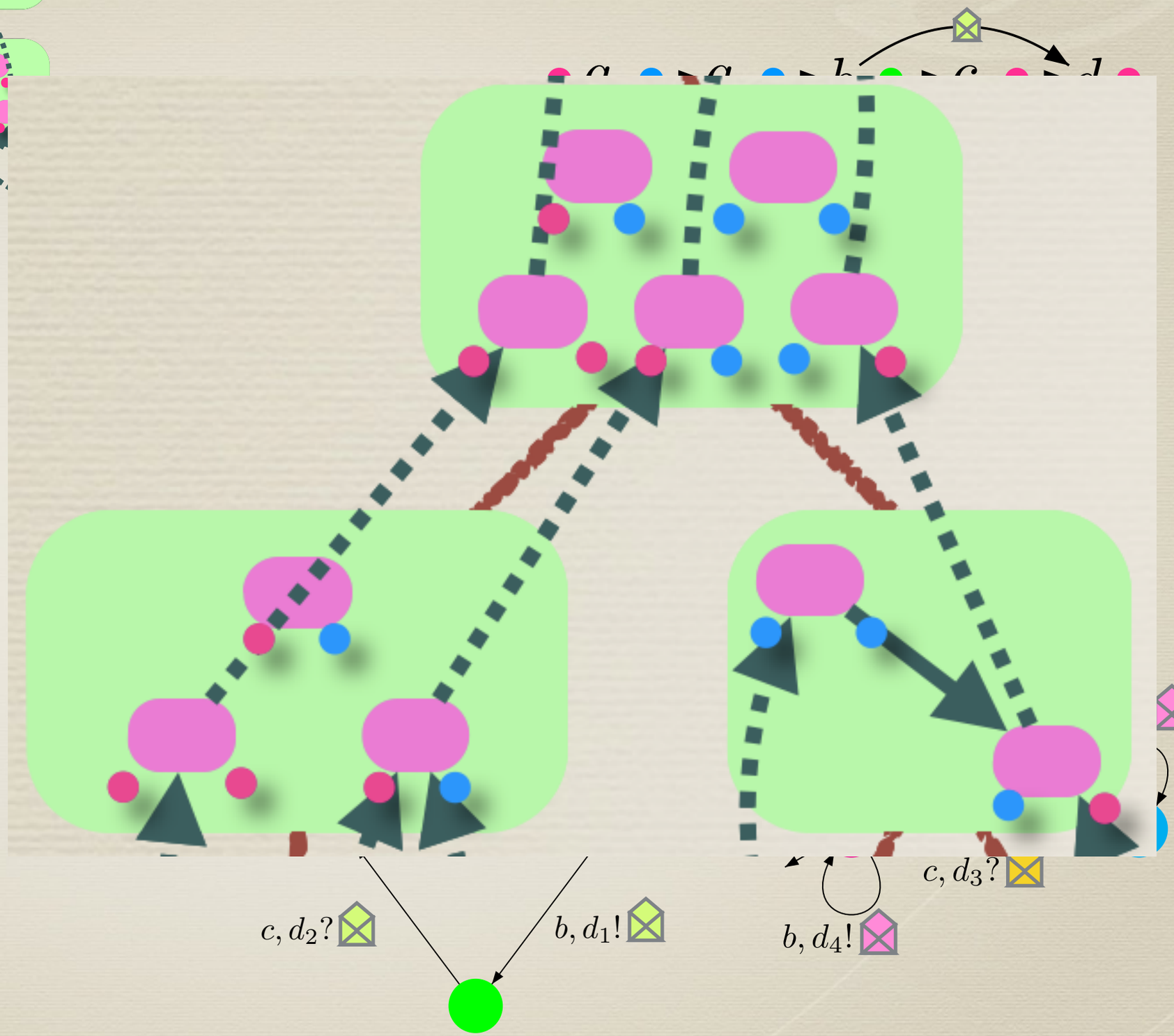




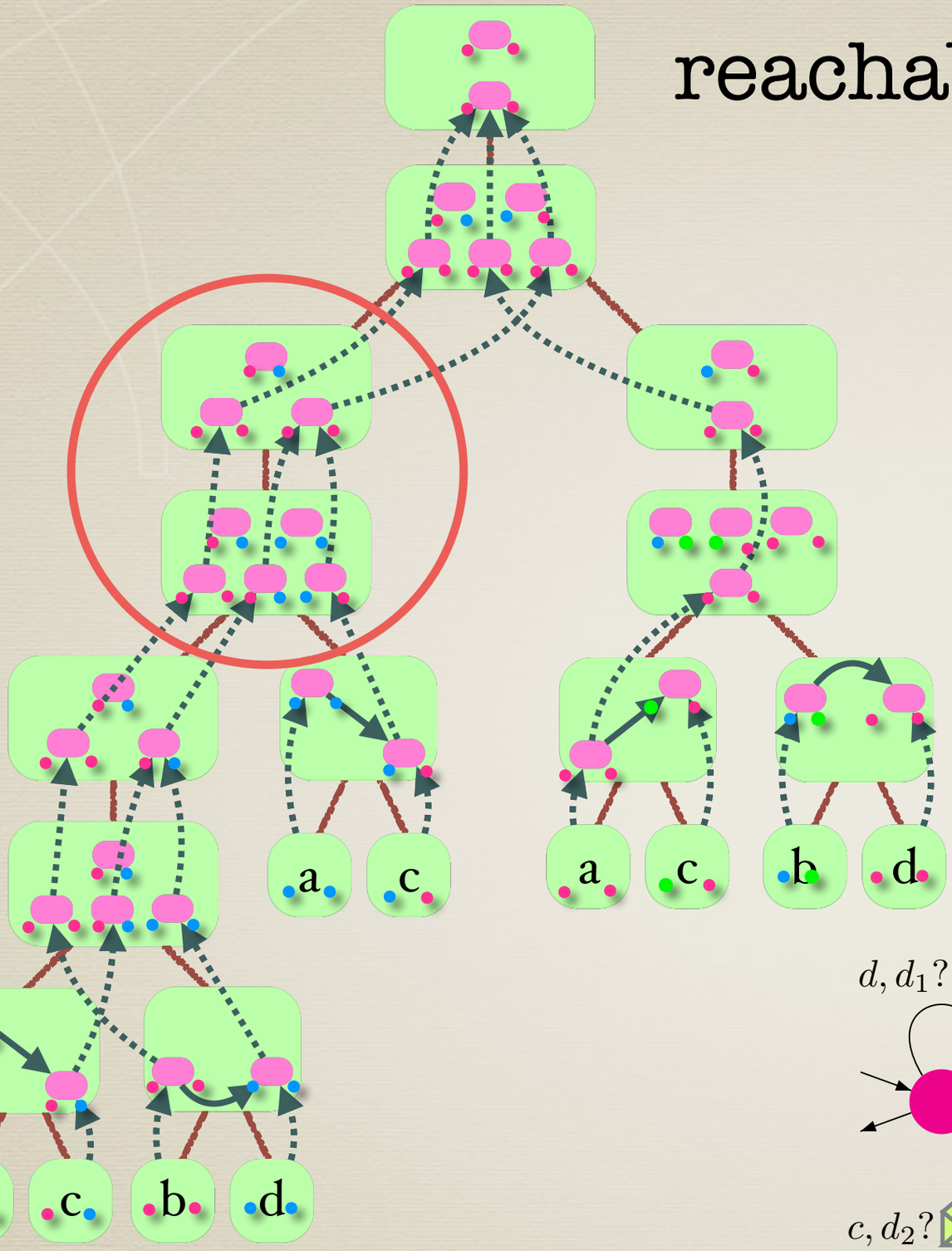


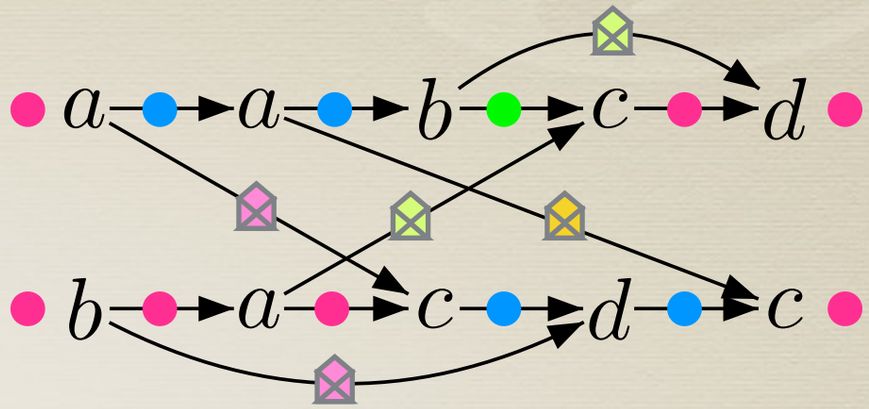
reachability/emptiness

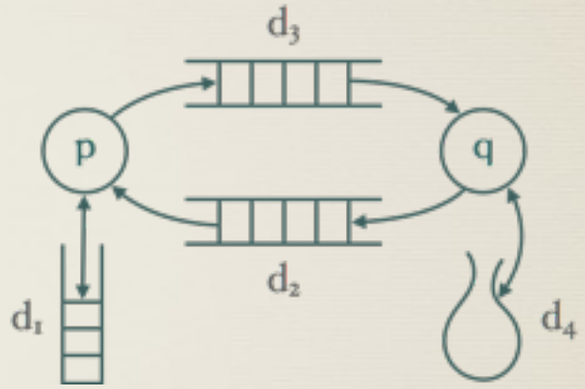


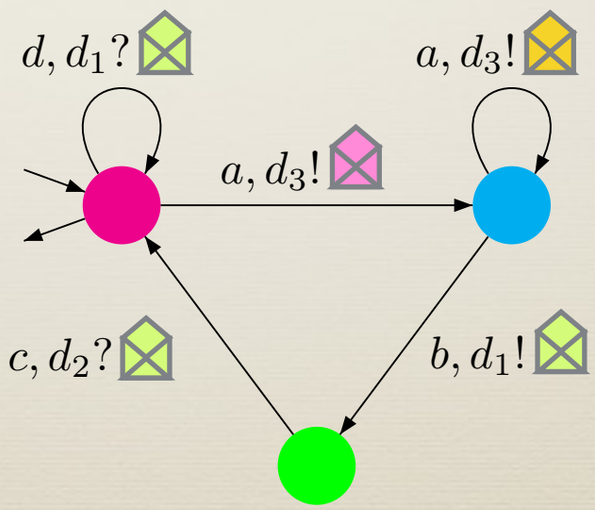


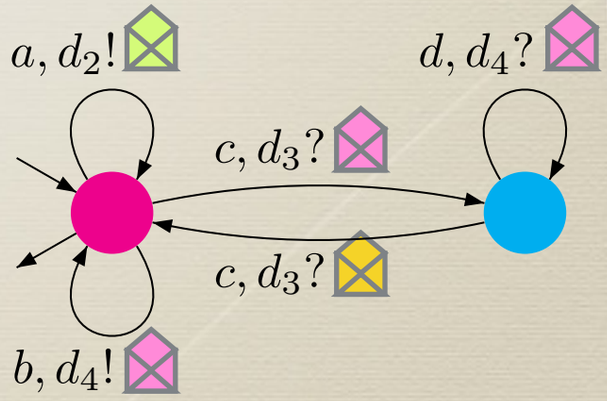
reachability/emptiness



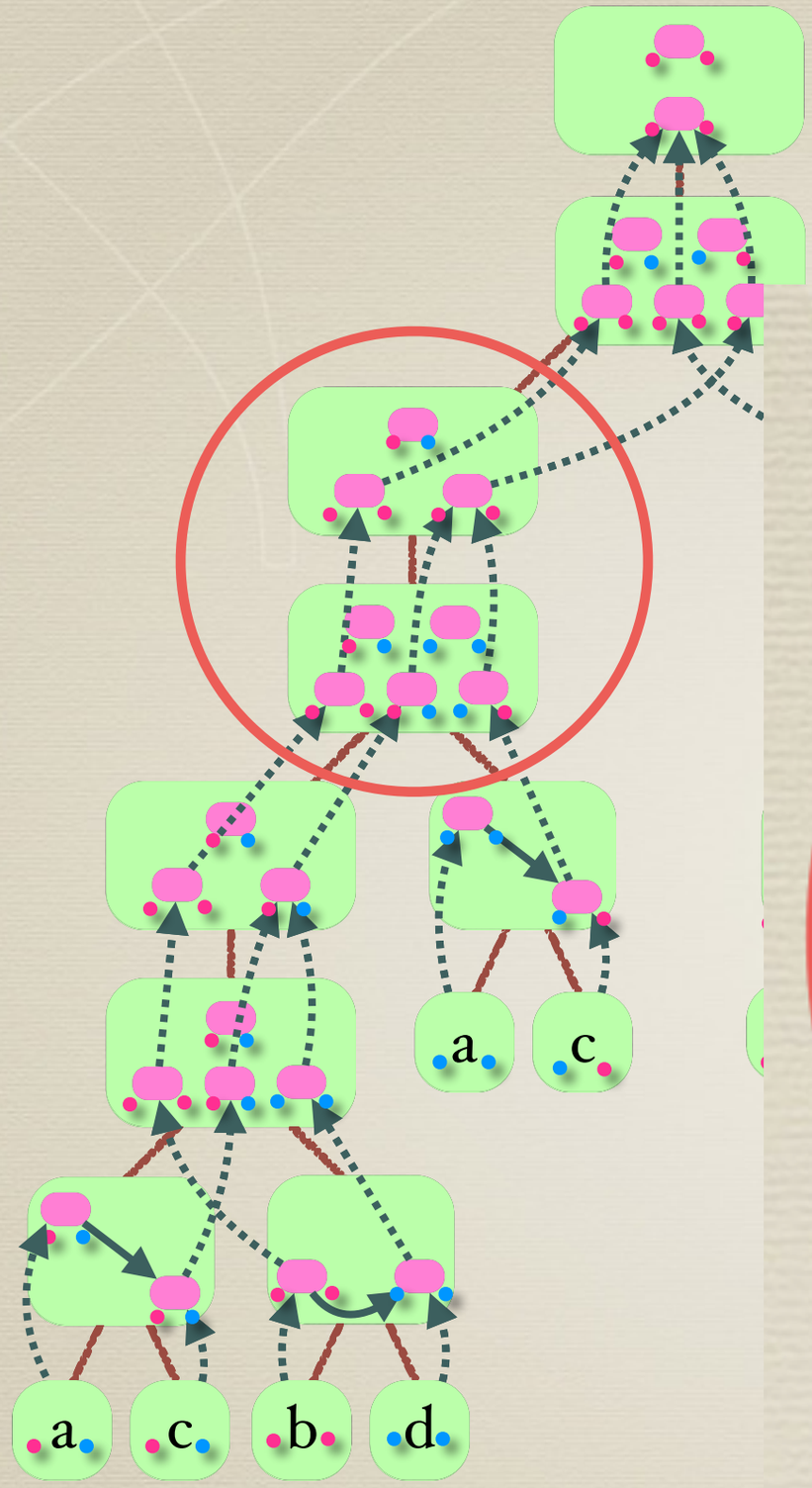


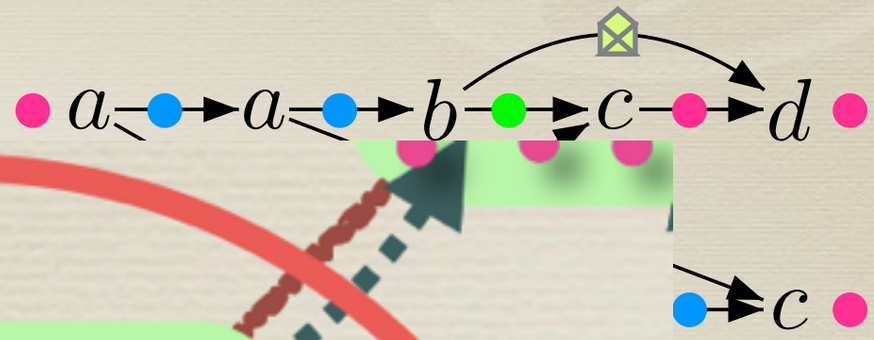


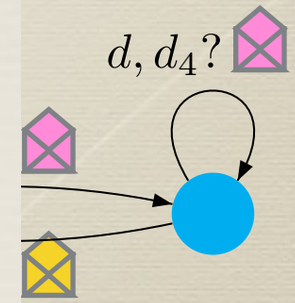




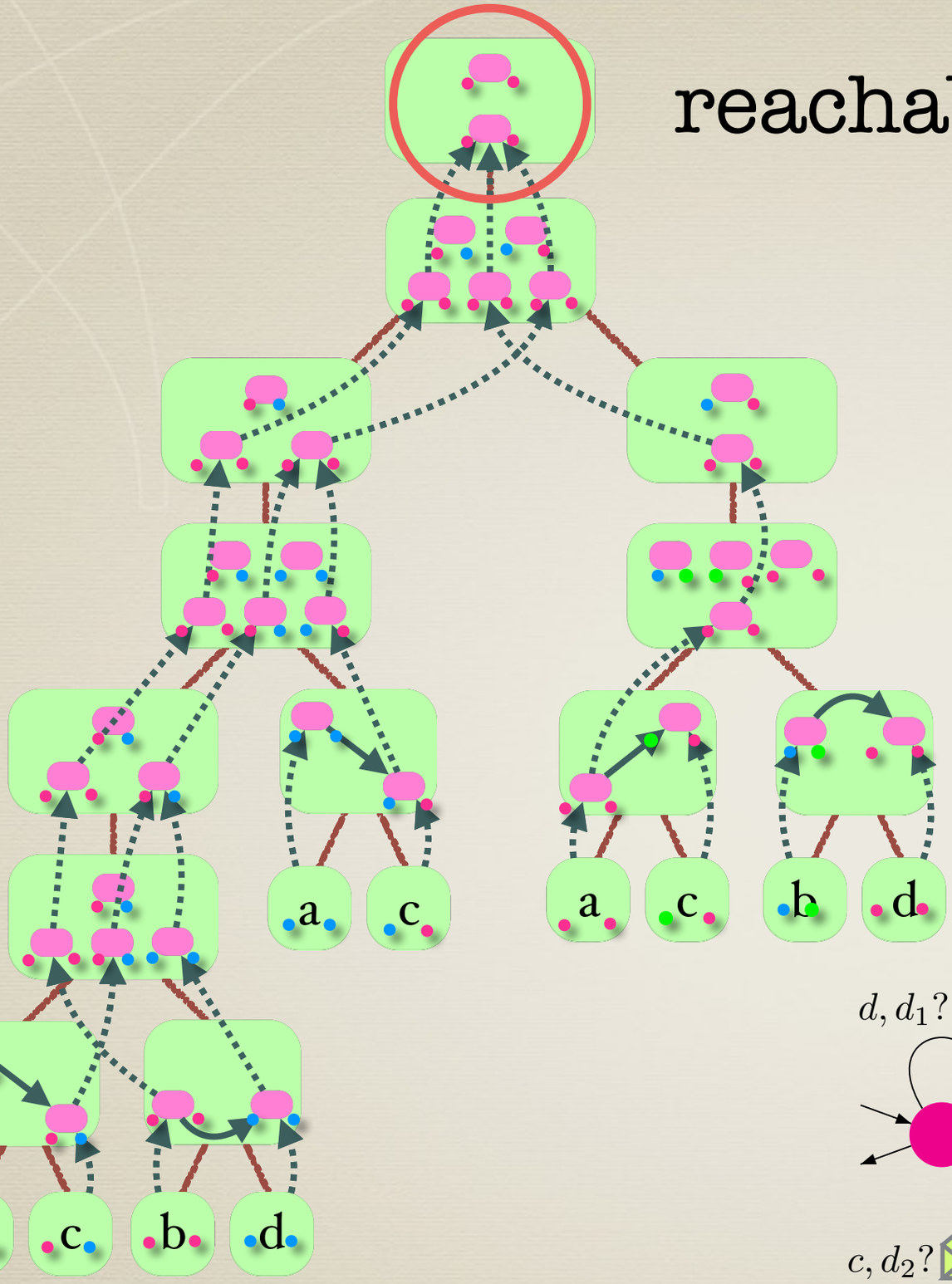
reachability/emptiness

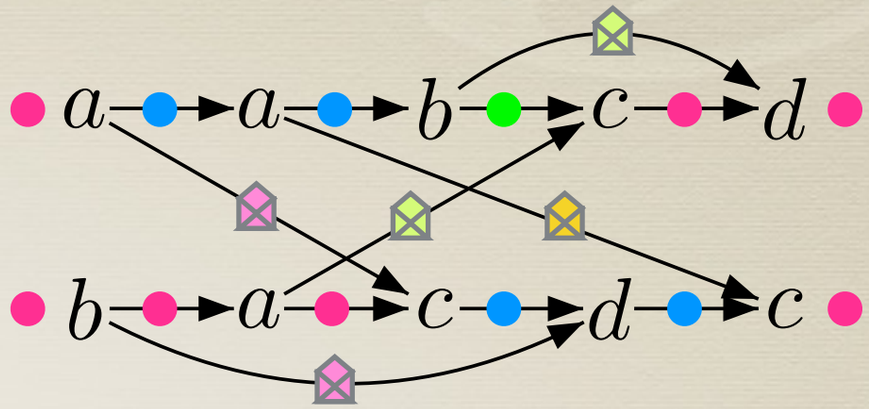


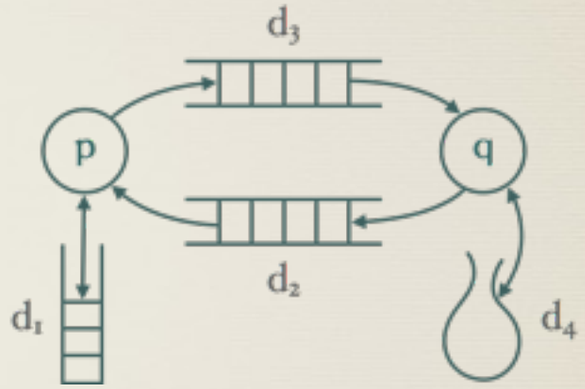


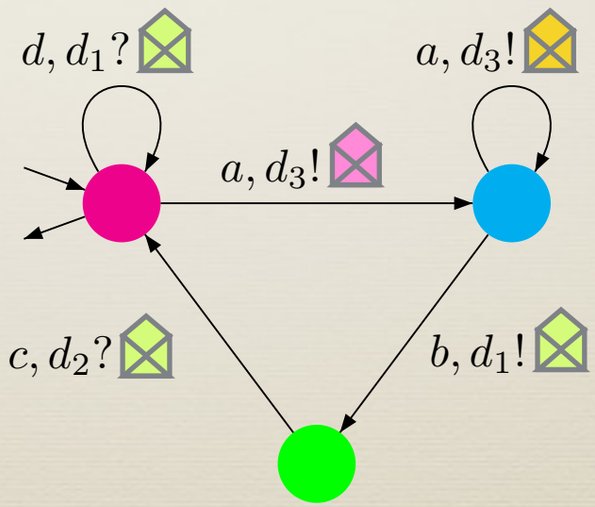


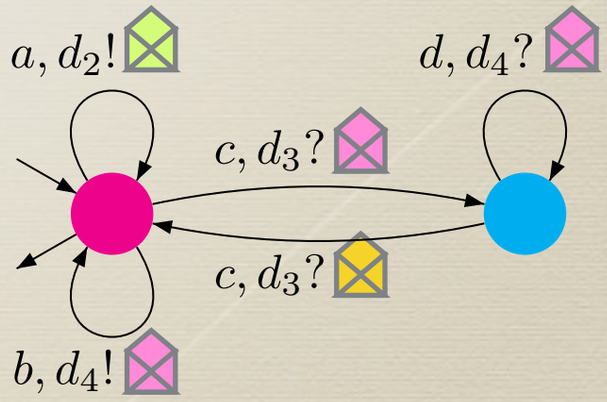
reachability/emptiness



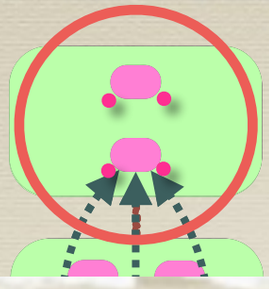


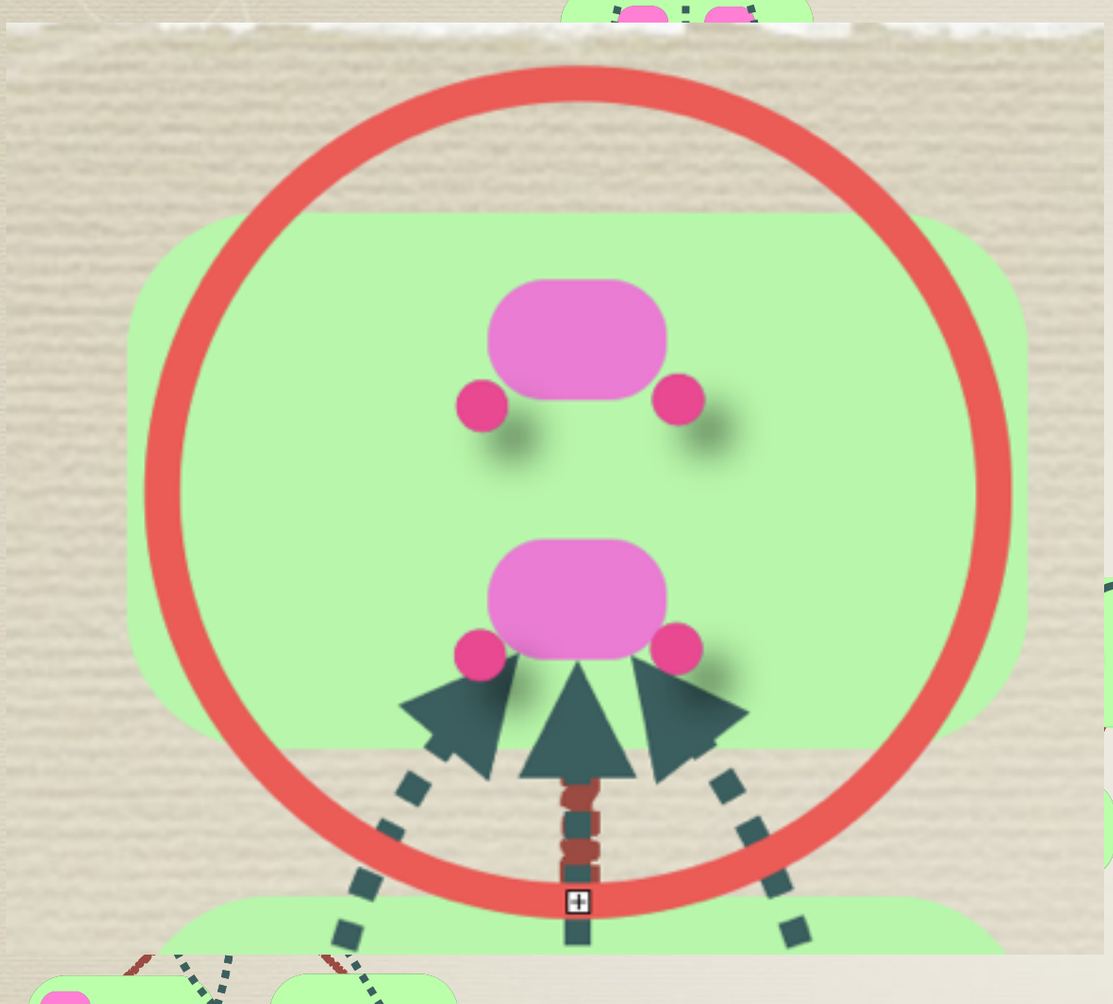


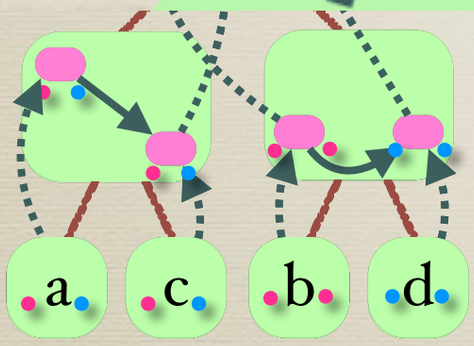


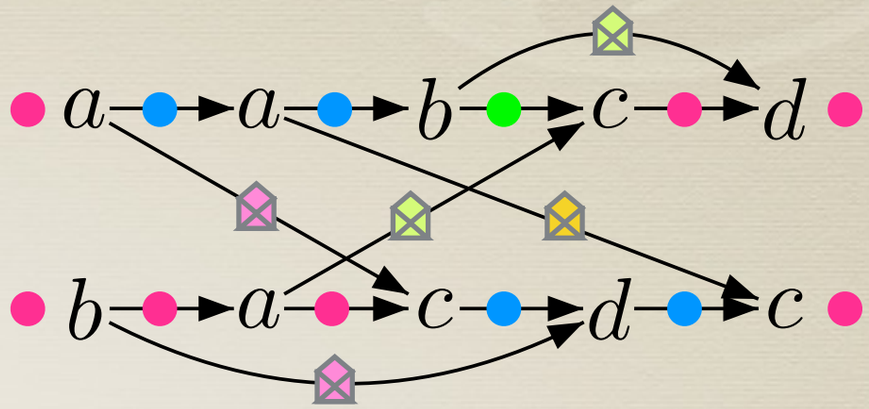


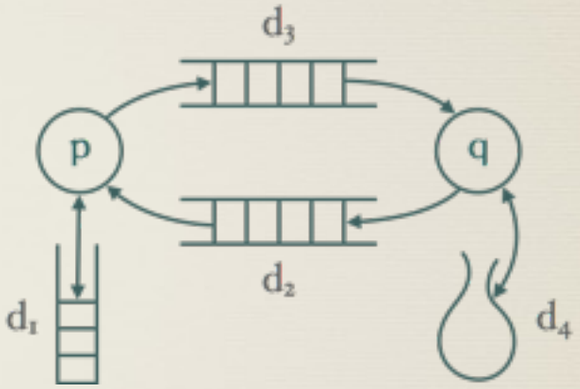
reachability/emptiness

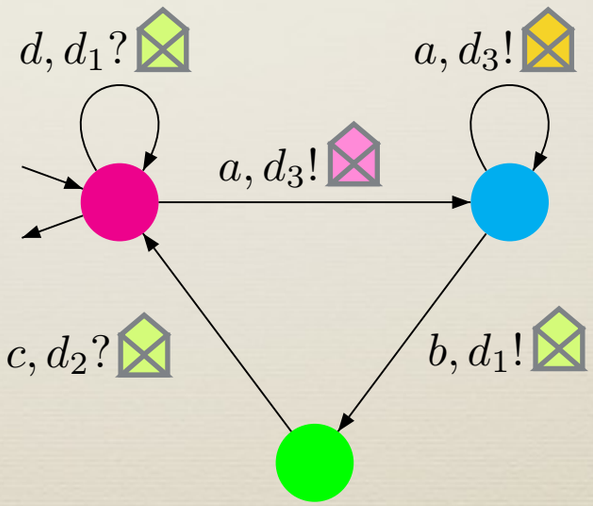


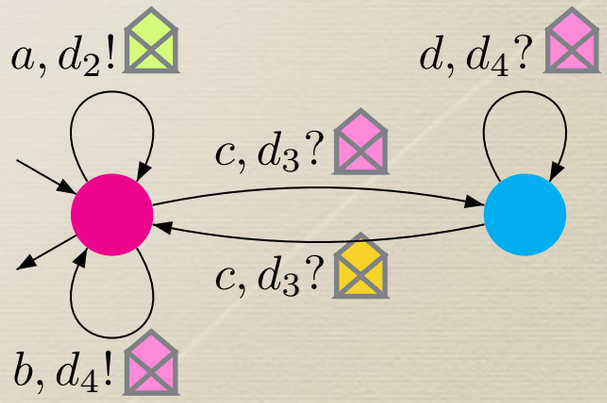












Split-width: parametrized verification

Problem	Complexity	
	bound on split-width part of the input (in unary)	bound on split-width fixed
CPDS emptiness	ExpTime-Complete	PTime-Complete
CPDS inclusion or universality	2ExpTime	ExpTime-Complete
LTL / CPDL satisfiability or model checking	ExpTime-Complete	
ICPDL satisfiability or model checking	2ExpTime -Complete	
MSO satisfiability or model checking	Non-elementary	

Width: split vs tree vs clique

Split-Width k

Tree-Width t

Clique-Width c

Let C be a class of bounded degree MSO definable graphs.
TFAE

1. C has a decidable MSO theory
2. C can be interpreted in binary trees
3. C has bounded tree-width
4. C has bounded clique-width
5. C has bounded split-width (for CBMs)

Width: split vs tree vs clique

Let C be a class of bounded degree MSO definable graphs.
TFAE

1. C has a decidable MSO theory
2. C can be interpreted in binary trees
3. C has bounded tree-width
4. C has bounded clique-width
5. C has bounded split-width (for CBMs)

Width: split vs tree vs clique

Let C be a class of bounded degree MSO definable graphs.
TFAE

1. C has a decidable MSO theory
2. C can be interpreted in binary trees
3. C has bounded tree-width
4. C has bounded clique-width
5. C has bounded split-width (for CBMs)

Outline

- Models
- Behaviours
- Specifications
- Verification via graph decompositions
  - Split-width
- Conclusion

Conclusion

* Use graphs to reason about behaviors of systems
  distributed or sequential
* Exploit graph theory
  * Look for Tree Interpretations
  * Take the “algebraic decompositions” way
  * Tailor algebra to the setting to find natural proofs for boundedness.
* Split-width: convenient decomposition technique
  as powerful as tree-width or clique-width for CBMs
  yields optimal algorithms

Conclusions…

- Extensions
  - Parameterized systems (size, topology)
    GasFor'16, FOSSACS'16
  - Timed systems
    AksGasKri'16, CONCUR'16
  - Higher-order PDA level 2
    AisGasSaivasan'16
  - Dynamic creation of processes
  - Read from many
  - Infinite behaviors
  - …

Main Sources

P. Madhusudan and G. Parlato
* Tree-width of Auxiliary Storage. In POPL 2011.

C. Aiswarya, P. Gastin, ..
* MSO decidability of multi-pushdown systems via split-width. In CONCUR 2012.
* Verifying Communicating Multi-pushdown Systems via Split-width. In ATVA 2014.

Aiswarya's PhD Thesis
* Many more classes with bounded split-width
* Many more results

C. Aiswarya, P. Gastin
* Reasoning About Distributed Systems: WYSIWYG. In FSTTCS 2014

B. Bollig, P. Gastin
* MPRI Lecture Notes on Non-sequential Theory of Distributed Systems

Main Sources

P. Madhusudan and G. Parlato
* Tree-width of Auxiliary Storage. In POPL 2011.

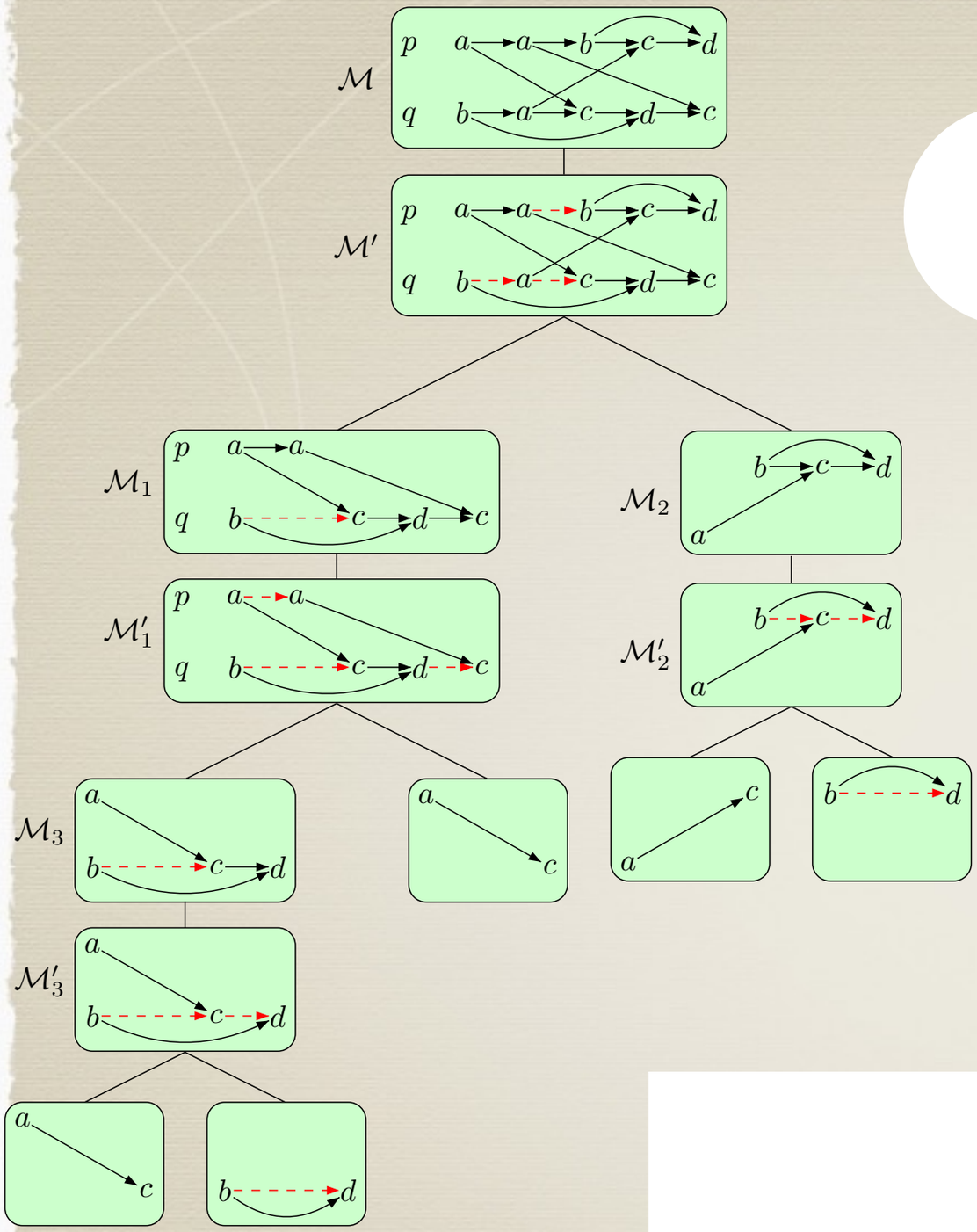
C. Aiswarya, P. Gastin, ..
* MSO decidability of multi-pushdown systems via split-width. In CONCUR 2012.
* Verifying Communicating Multi-pushdown Systems via Split-width. In ATVA 2014.

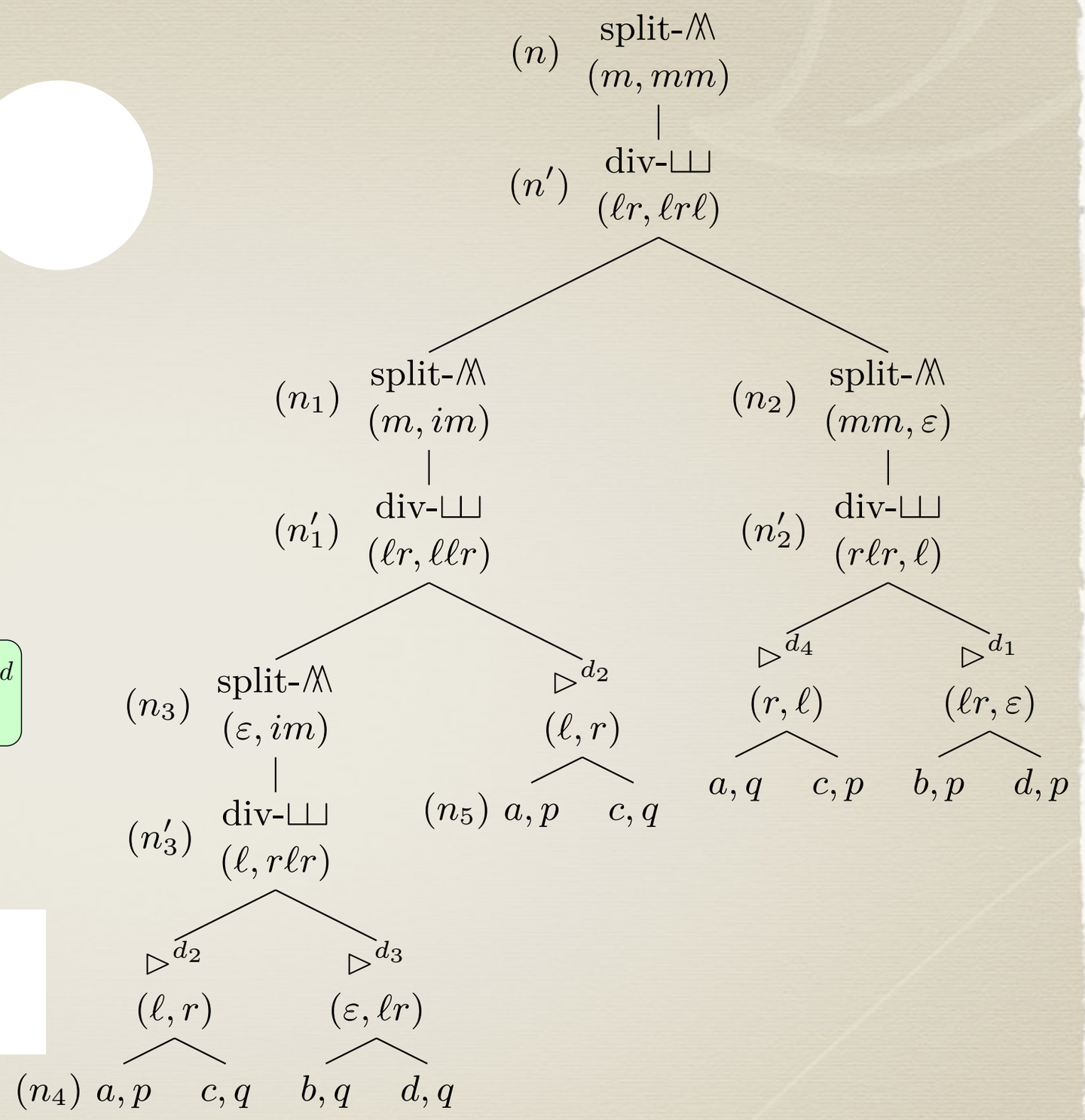
Aiswarya's PhD Thesis
* Many more classes with bounded split-width
* Many more results

C. Aiswarya, P. Gastin
* Reasoning About Distributed Systems:

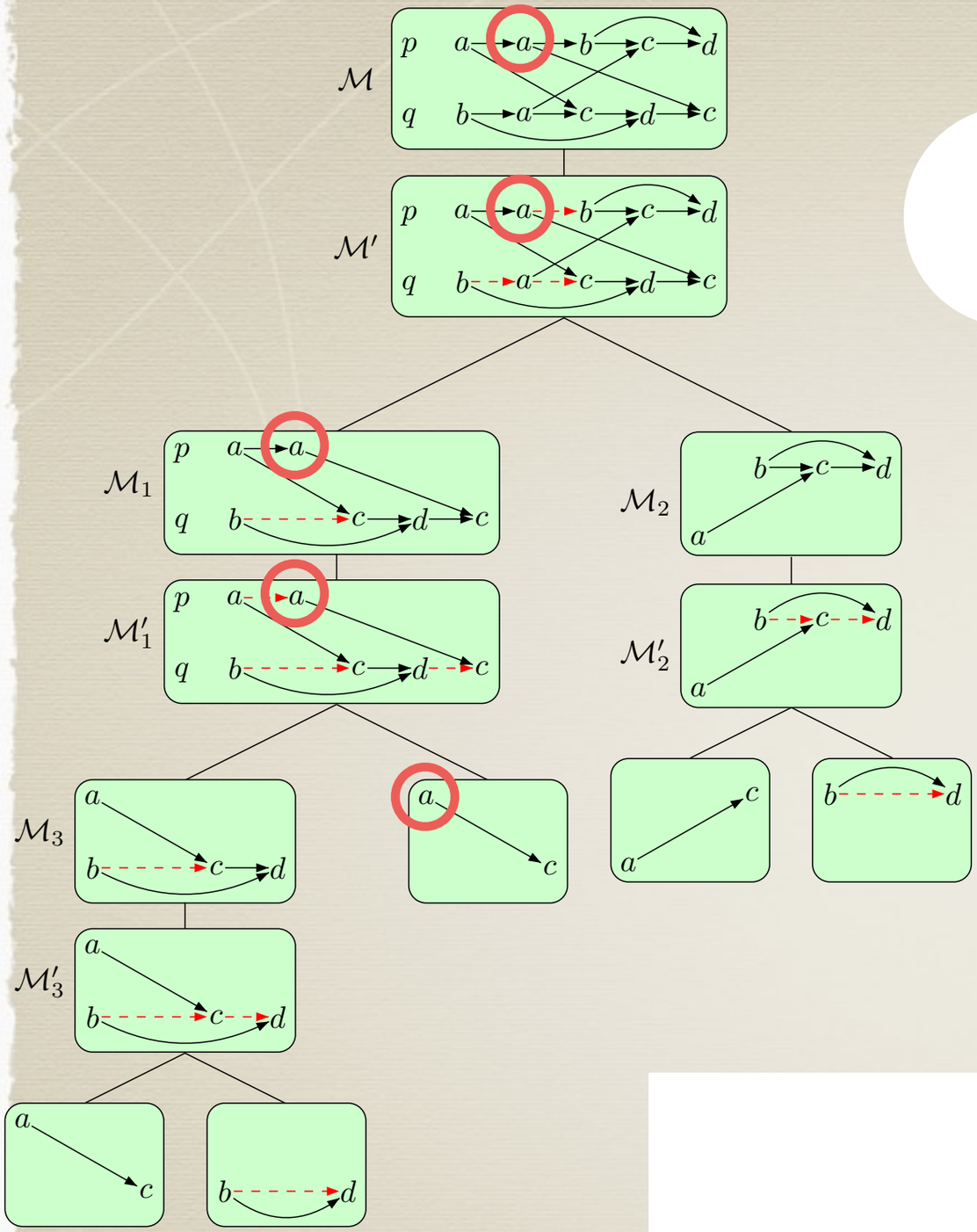
B. Bollig, P. Gastin
* MPRI Lecture Notes on Non-sequenti Distributed Systems

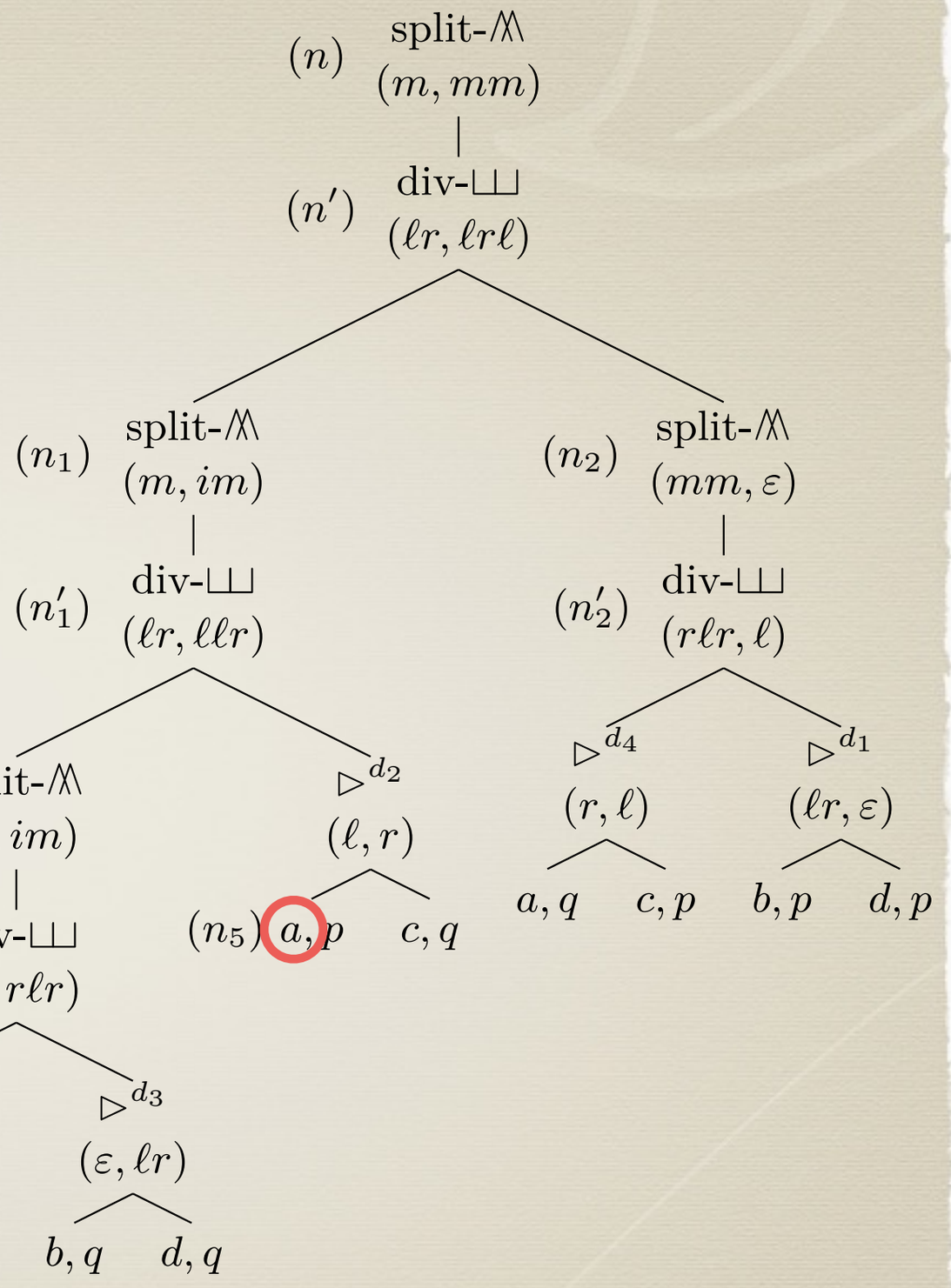
THANK YOU





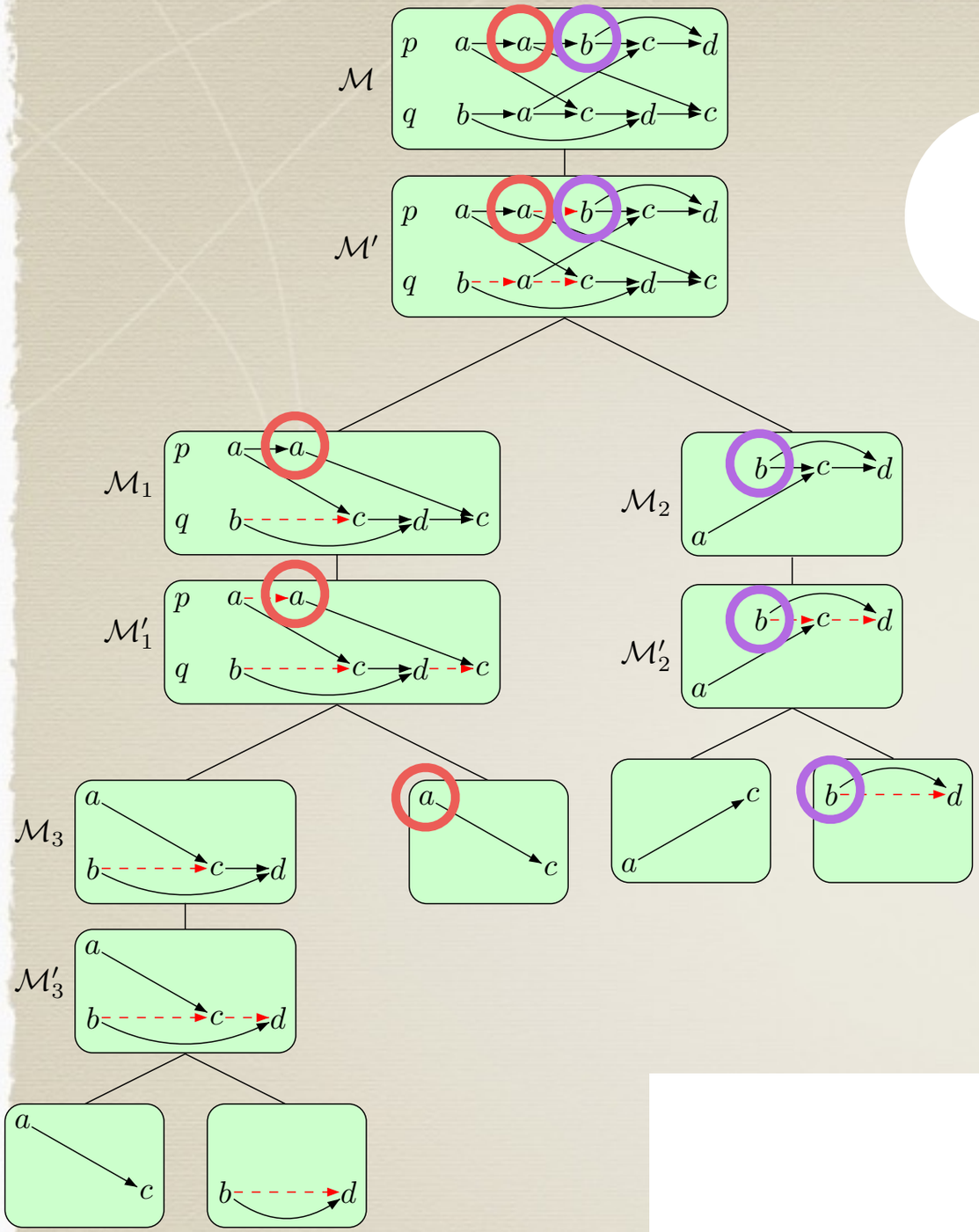
Tree interpretation in

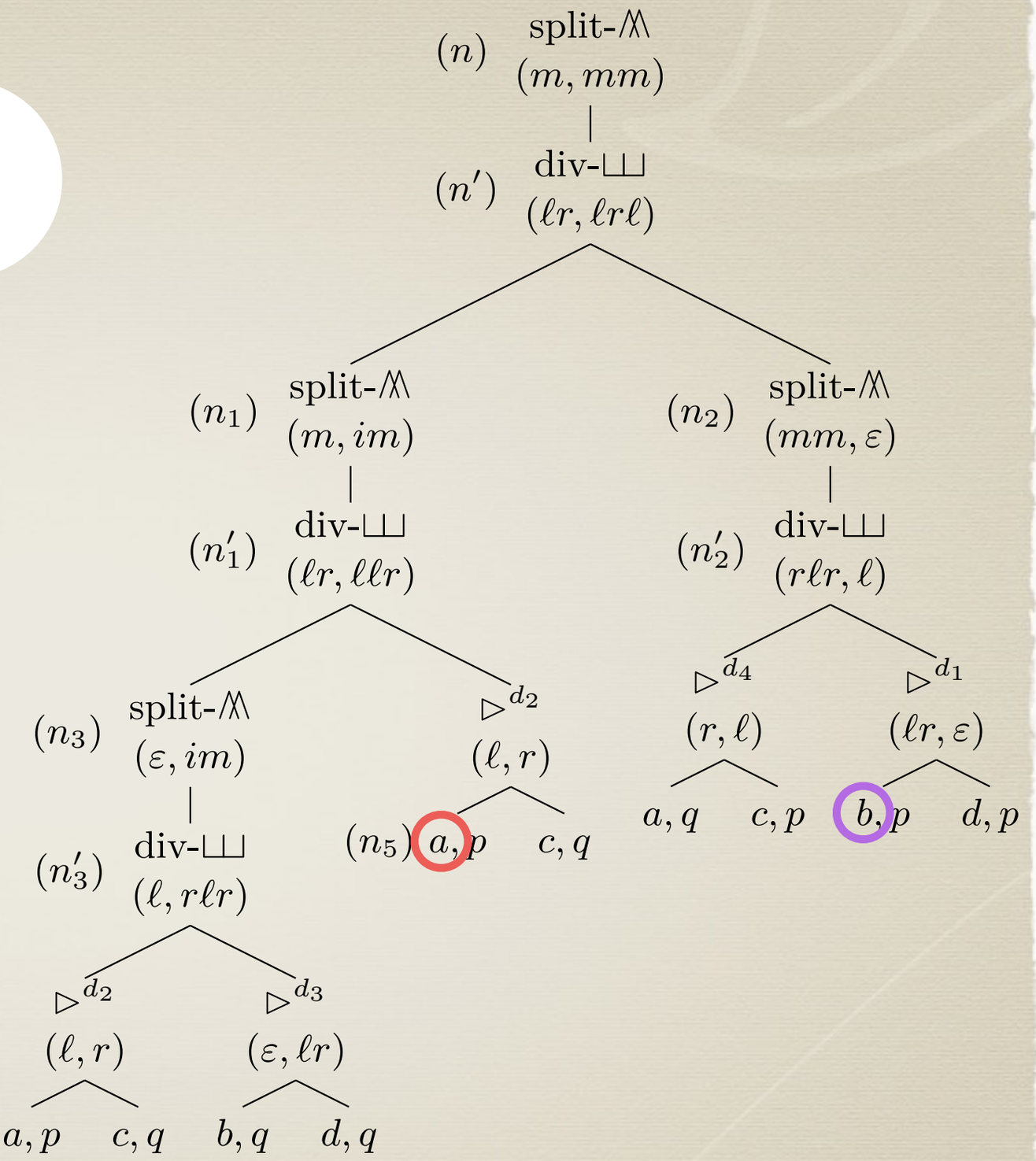




Vertices

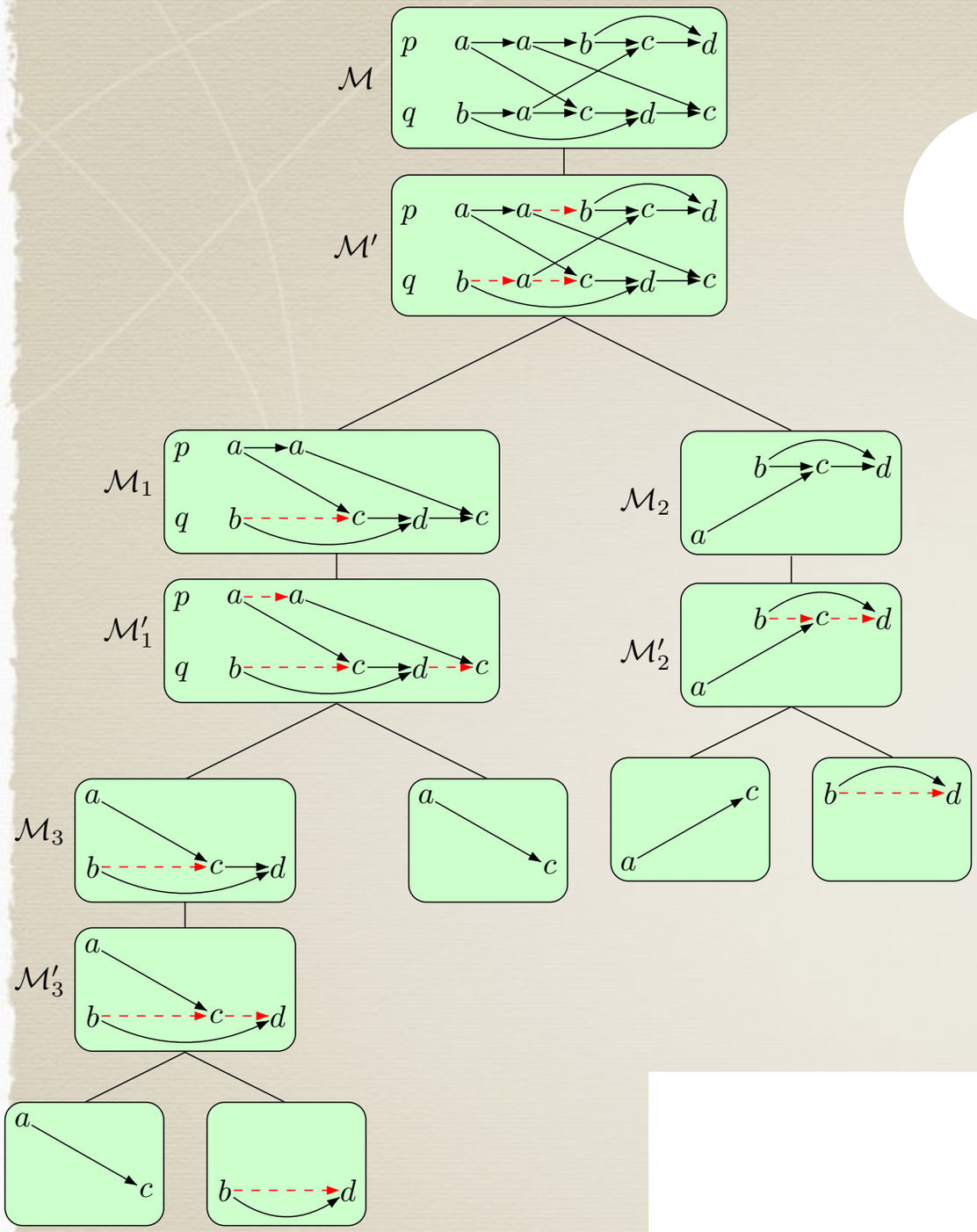
Tree interpretation in

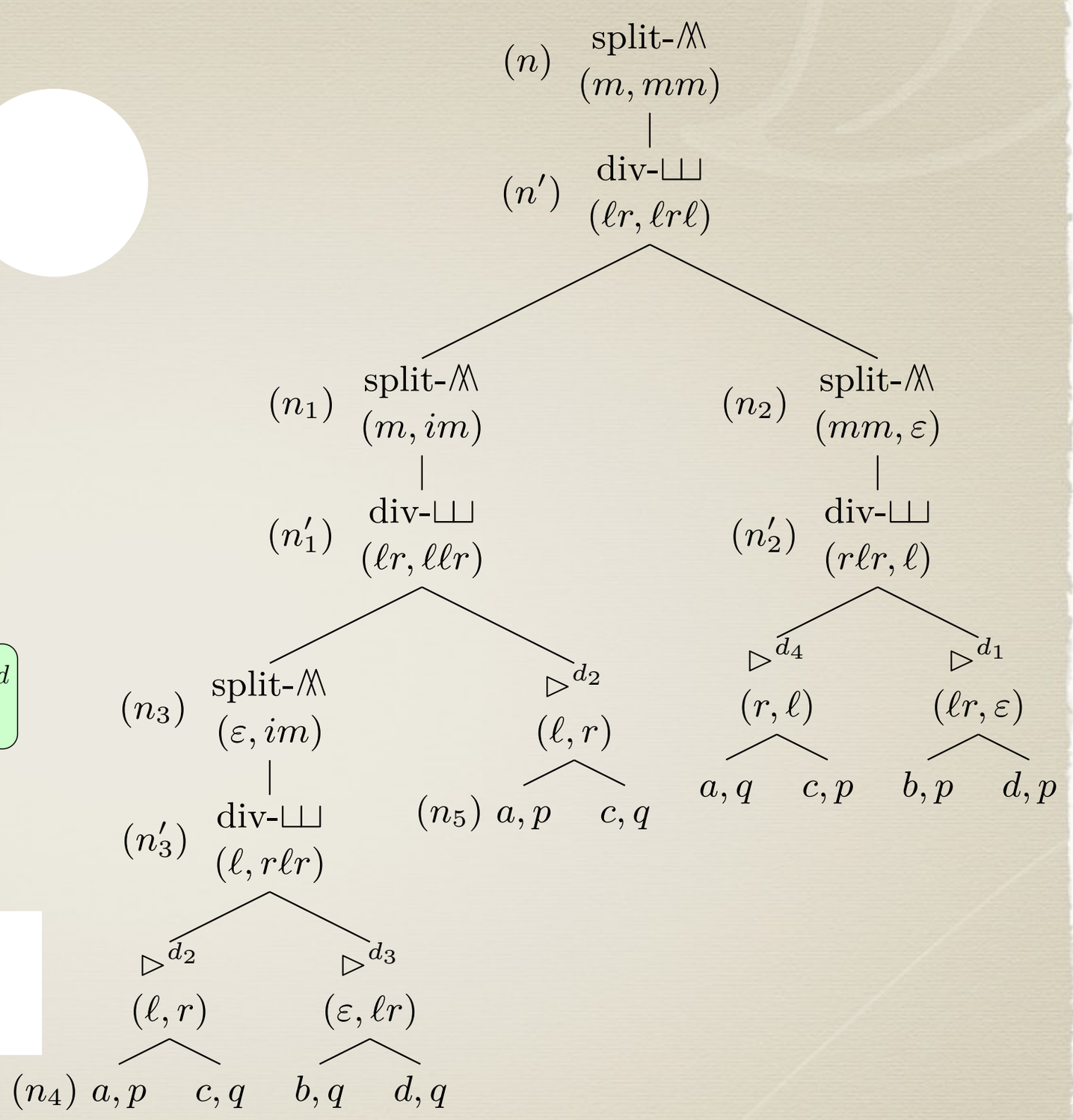




Vertices

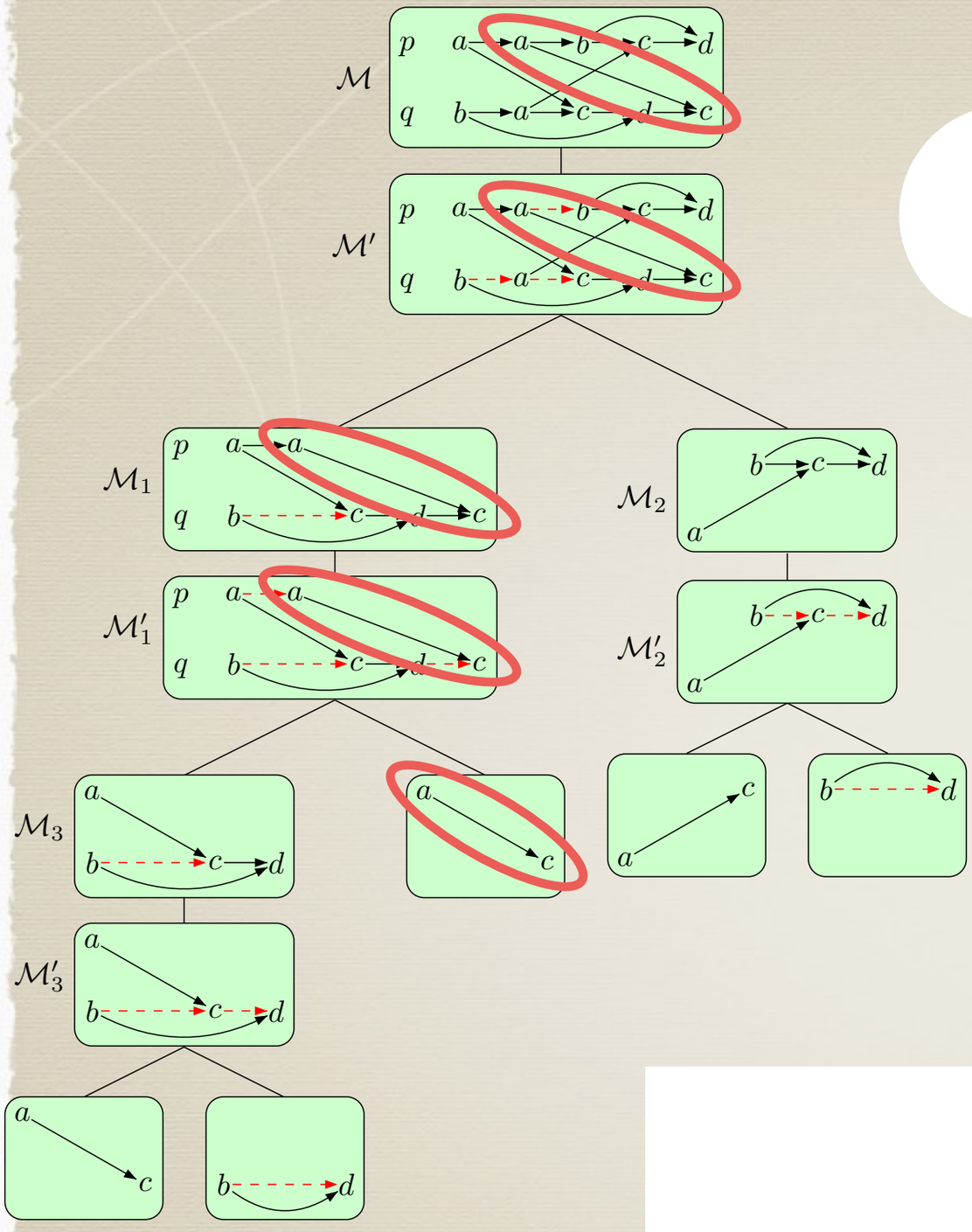
Tree interpretation in

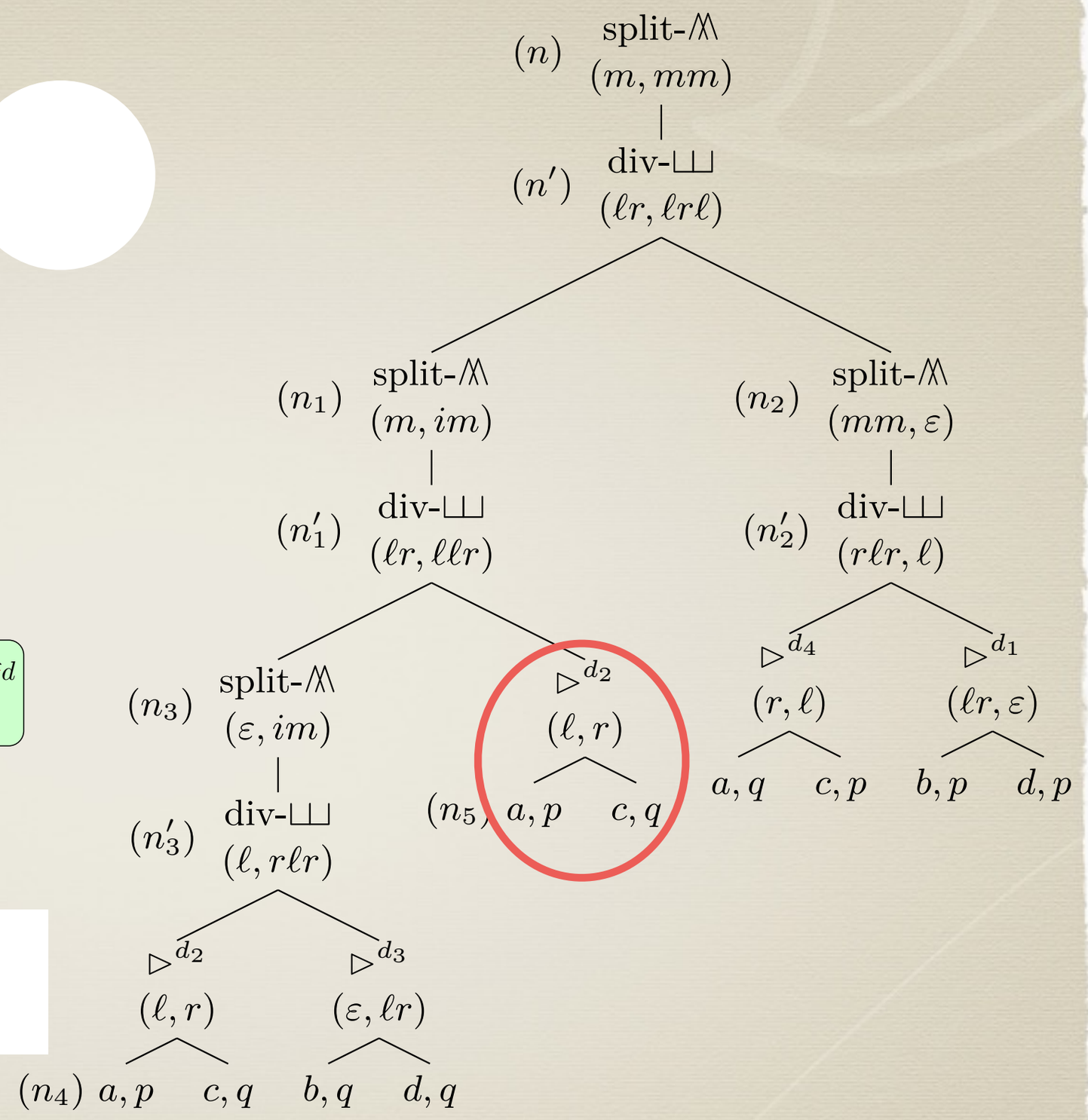




Vertices

Tree interpretation in

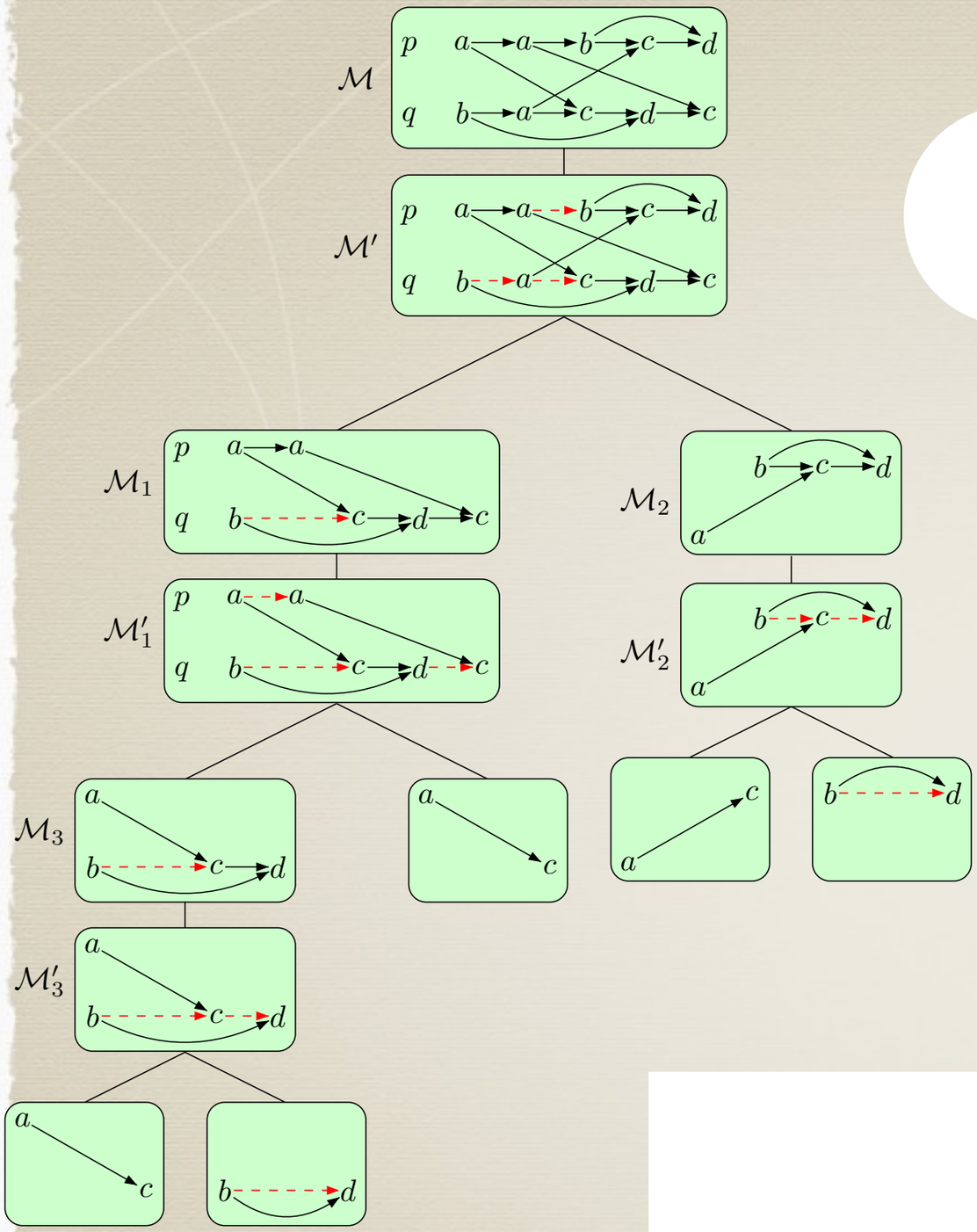


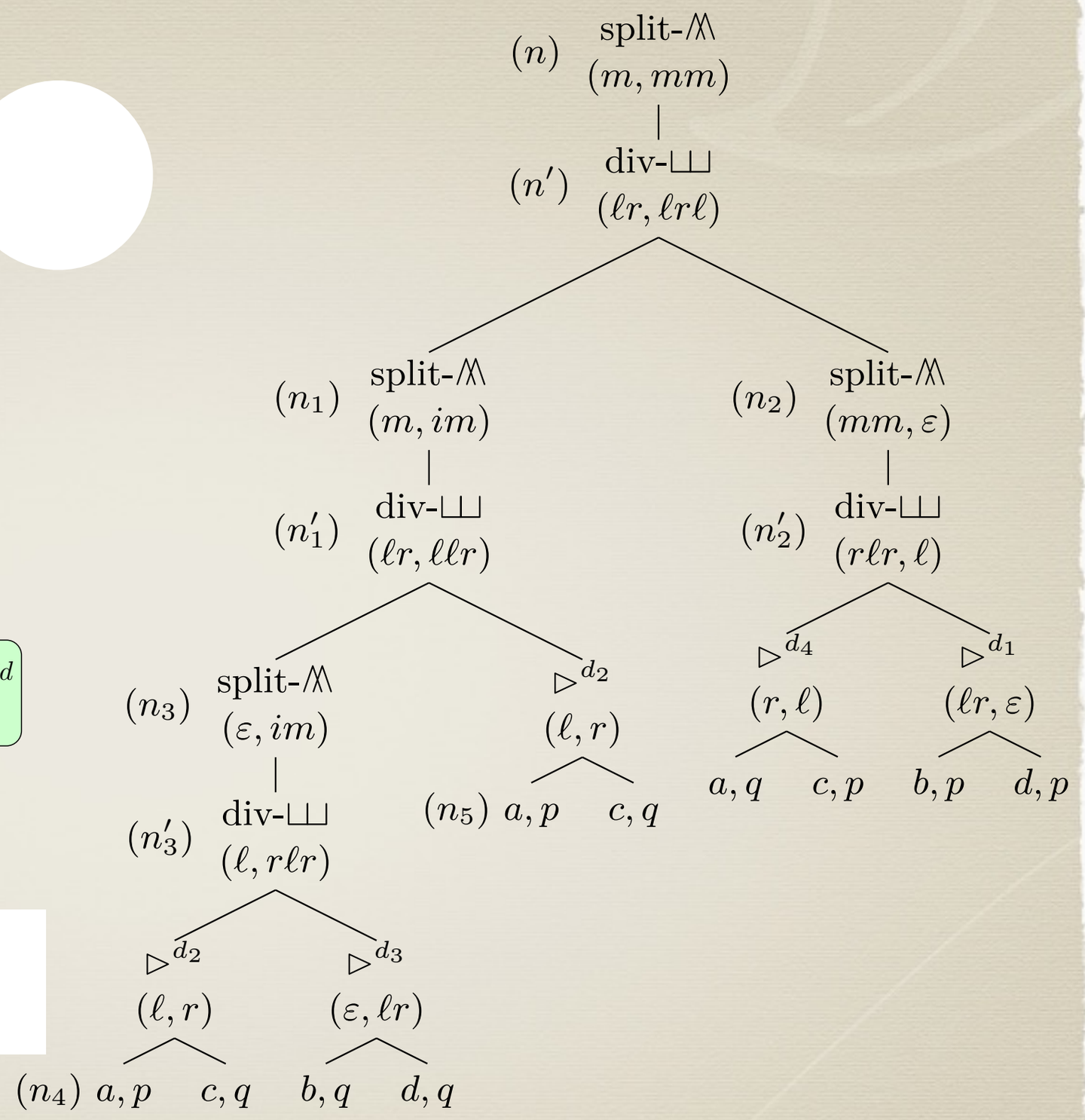


Vertices

Data edges

Tree interpretation in

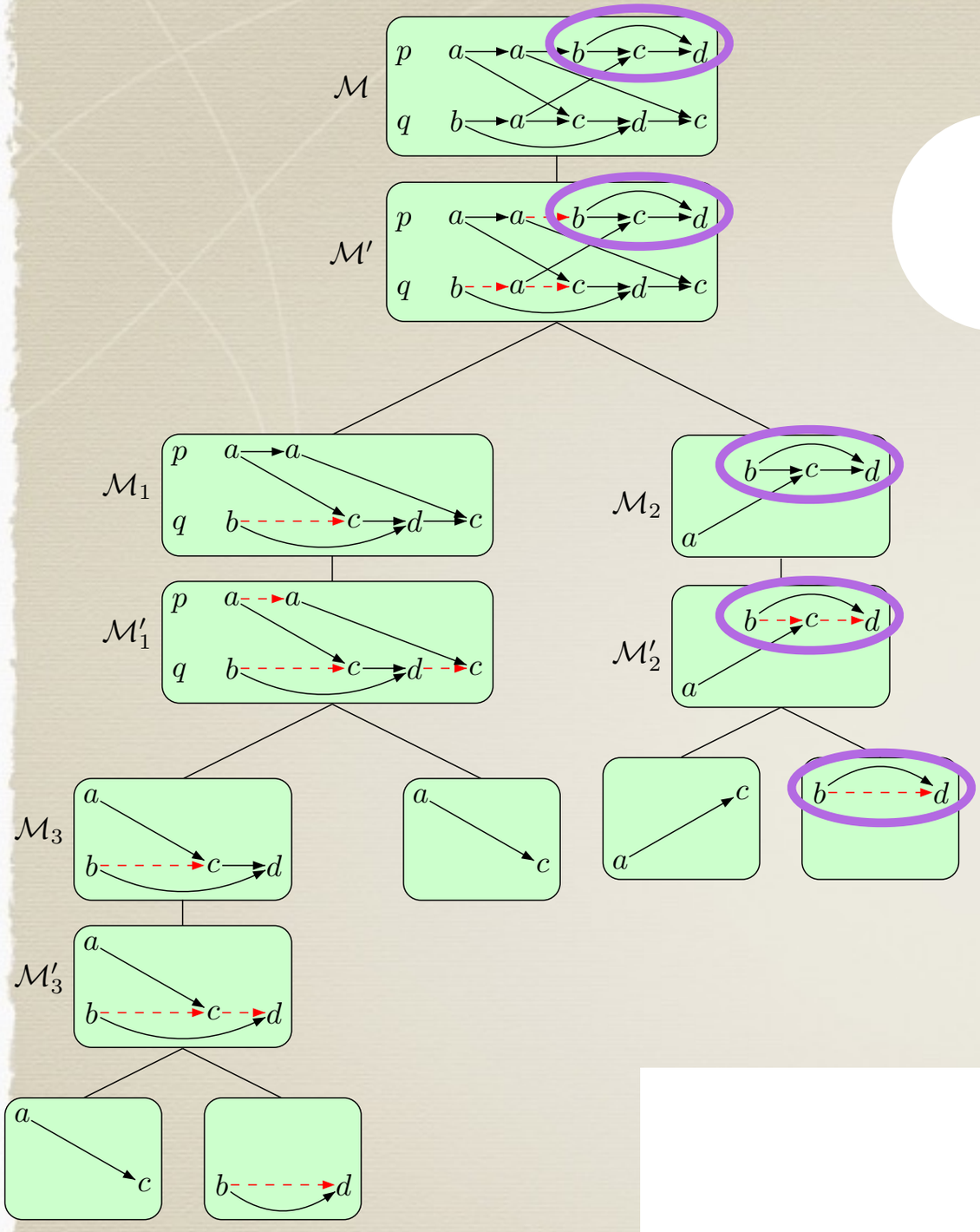


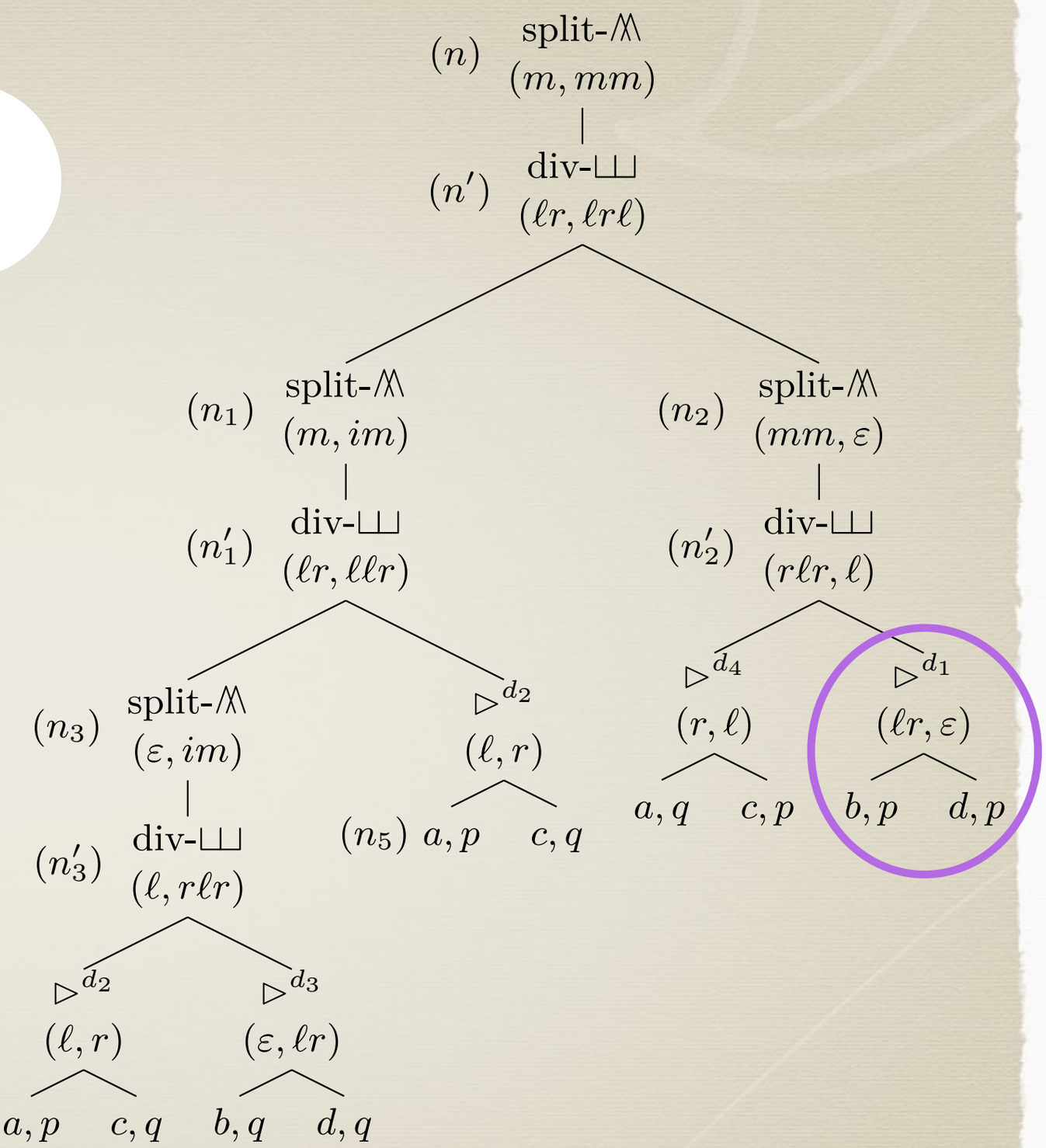


Vertices

Data edges

Tree interpretation in

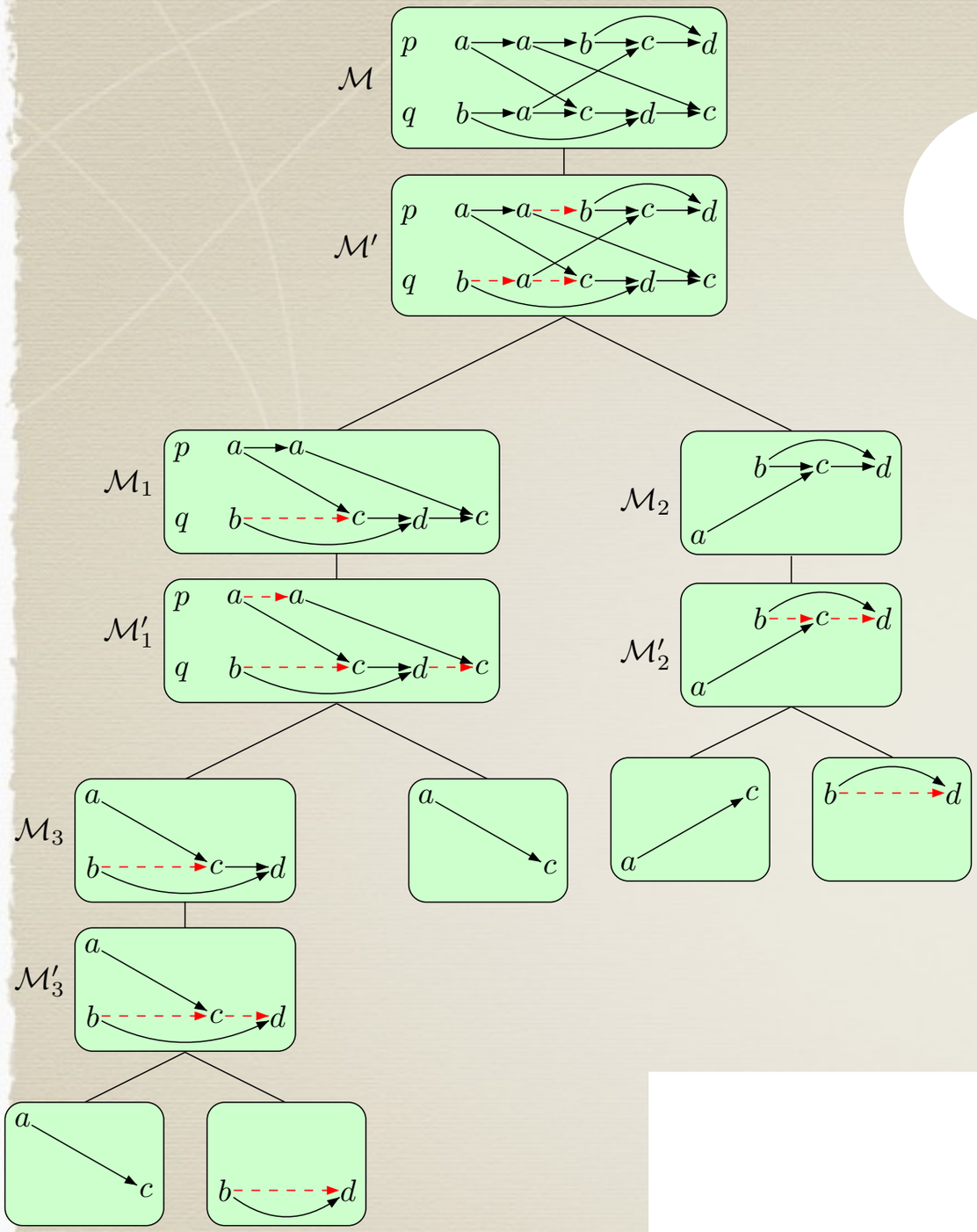


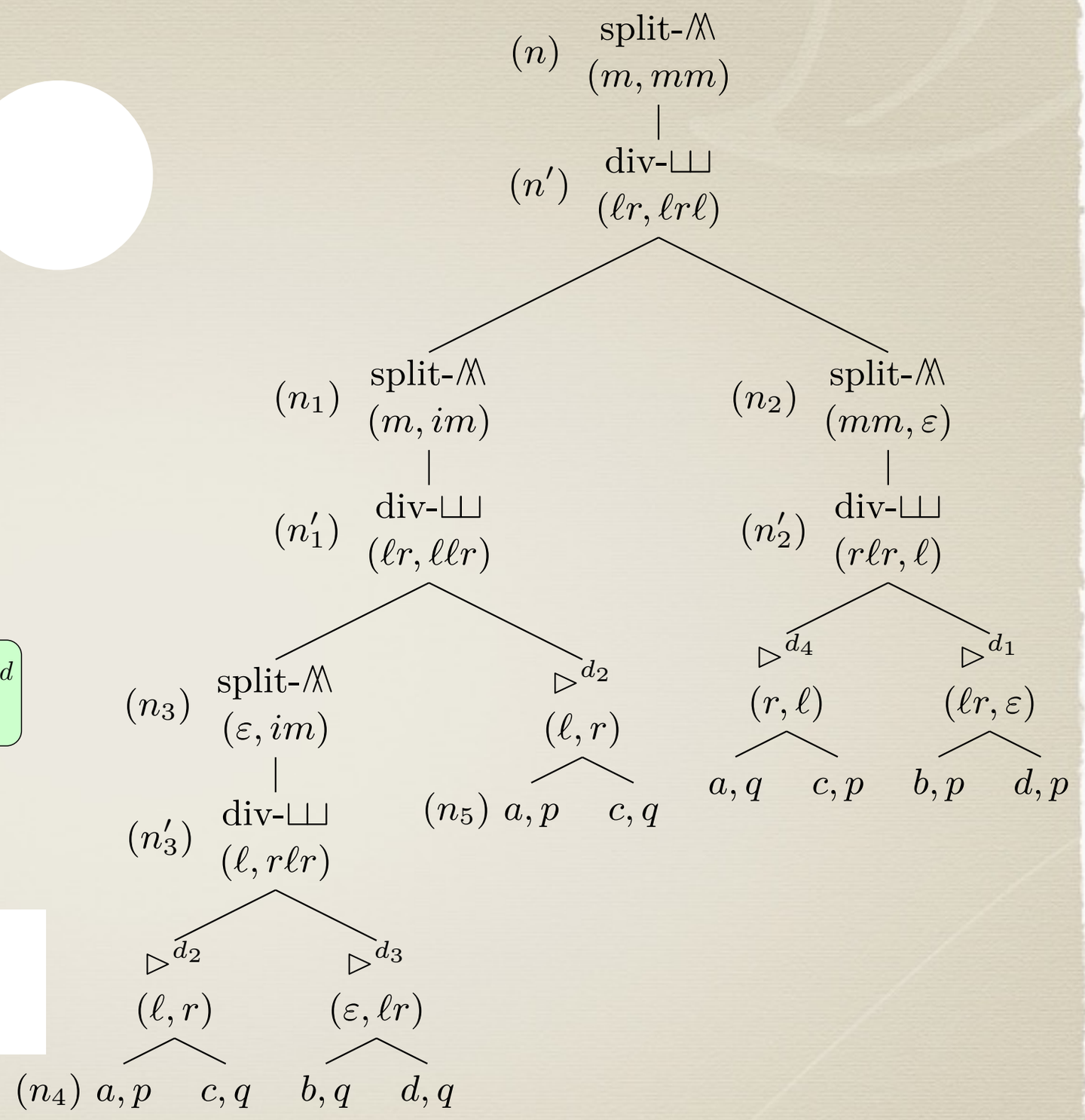


Vertices

Data edges

Tree interpretation in

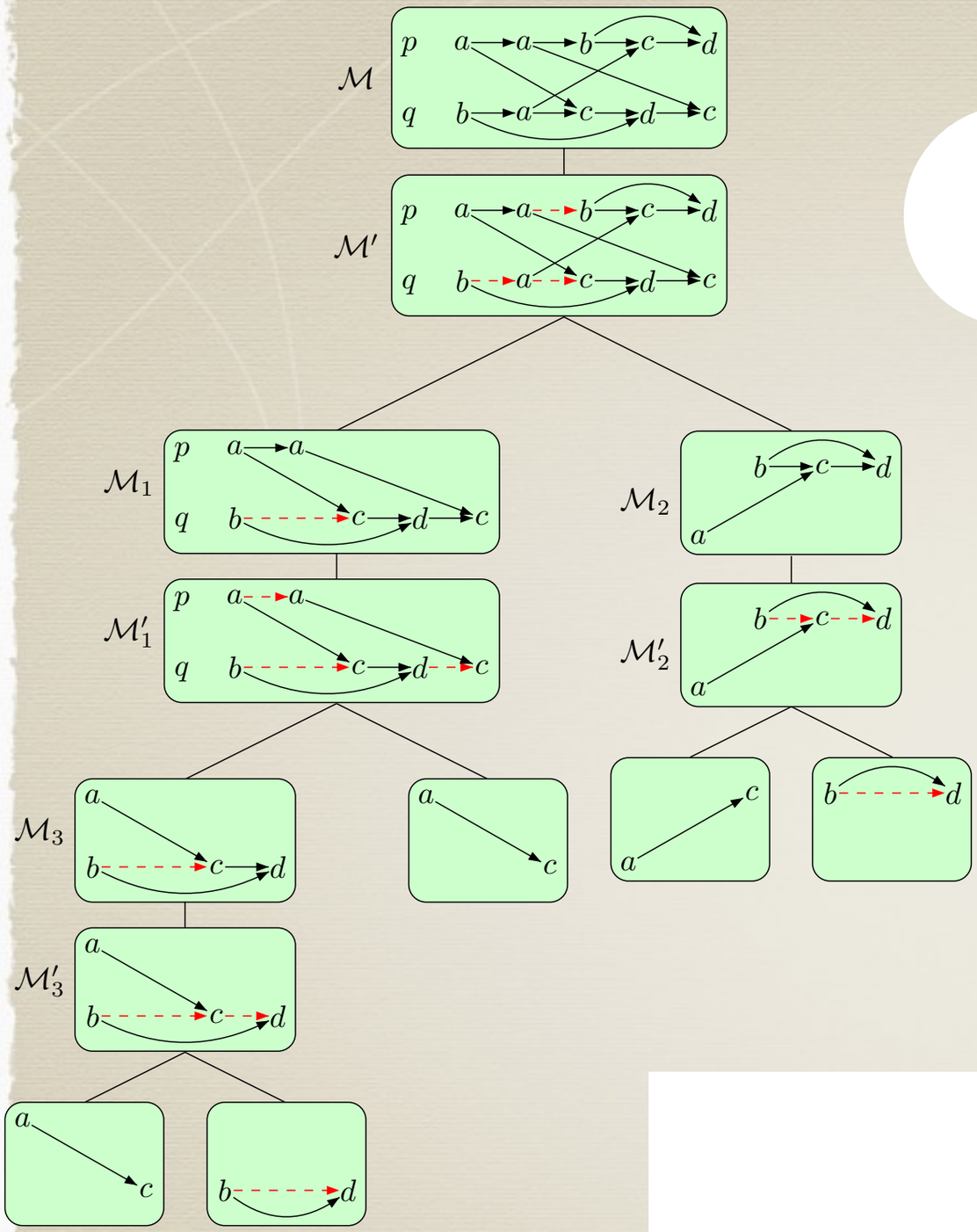


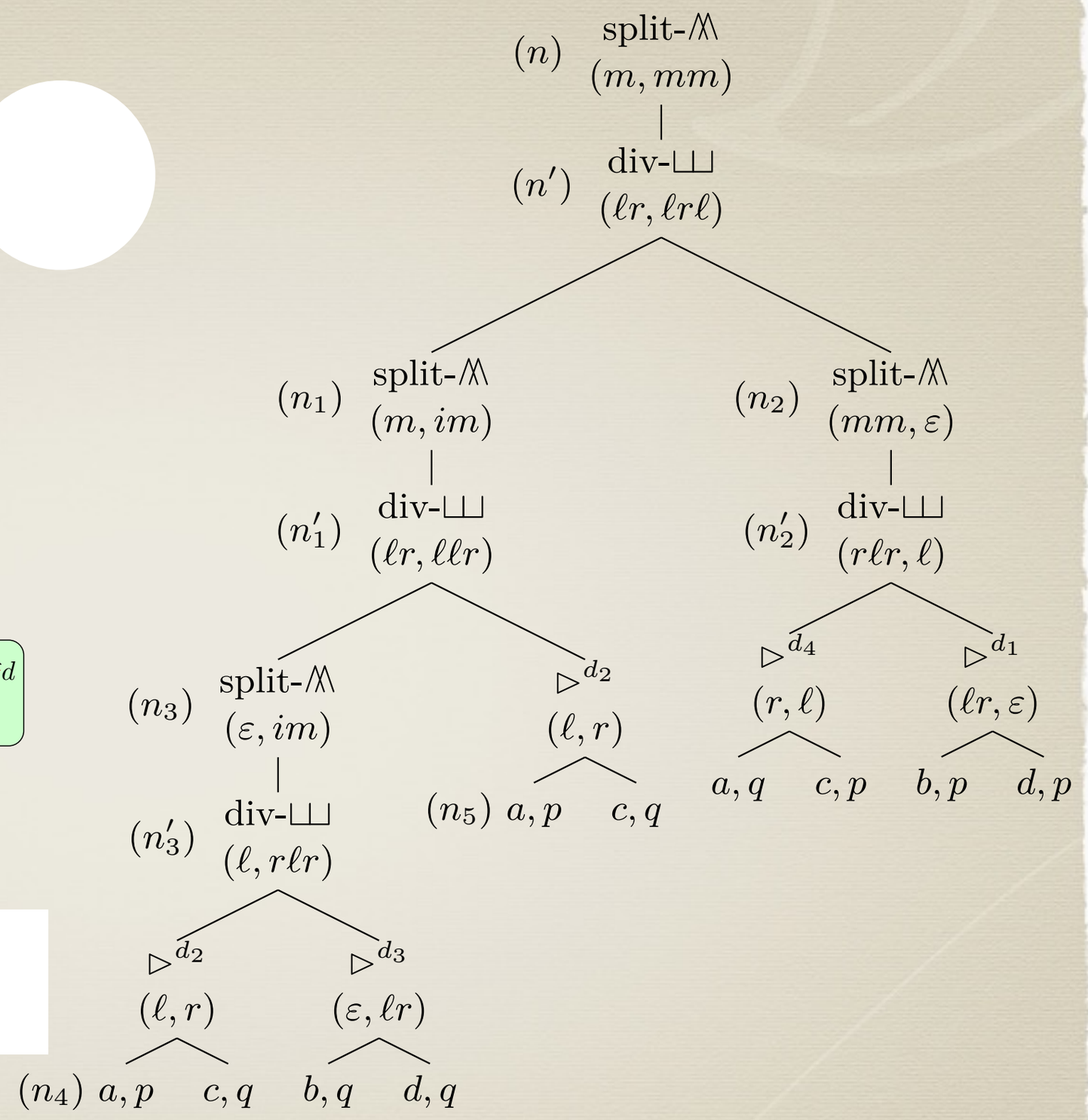


Vertices

Data edges

Tree interpretation in



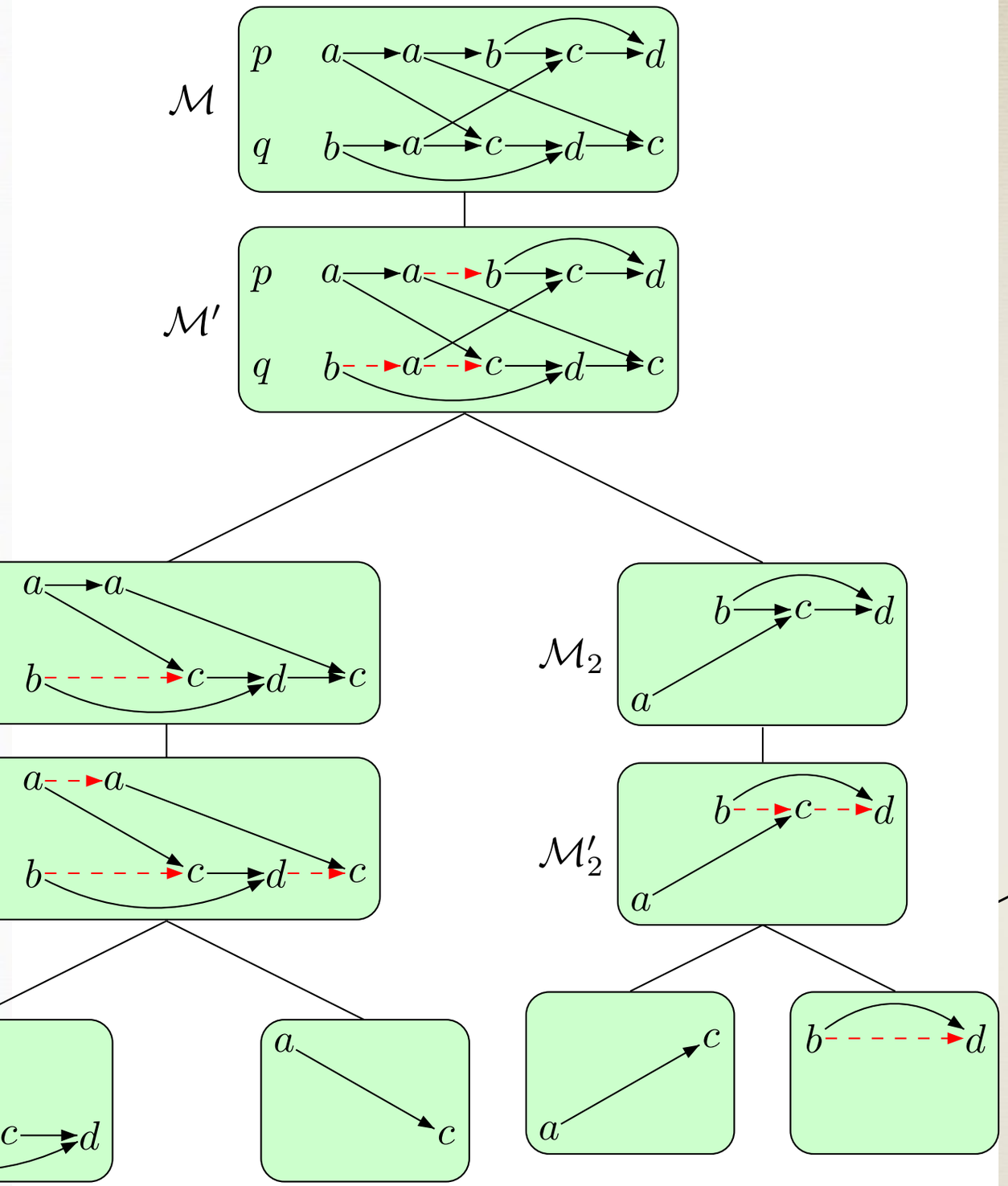


Vertices

Data edges

Process edges

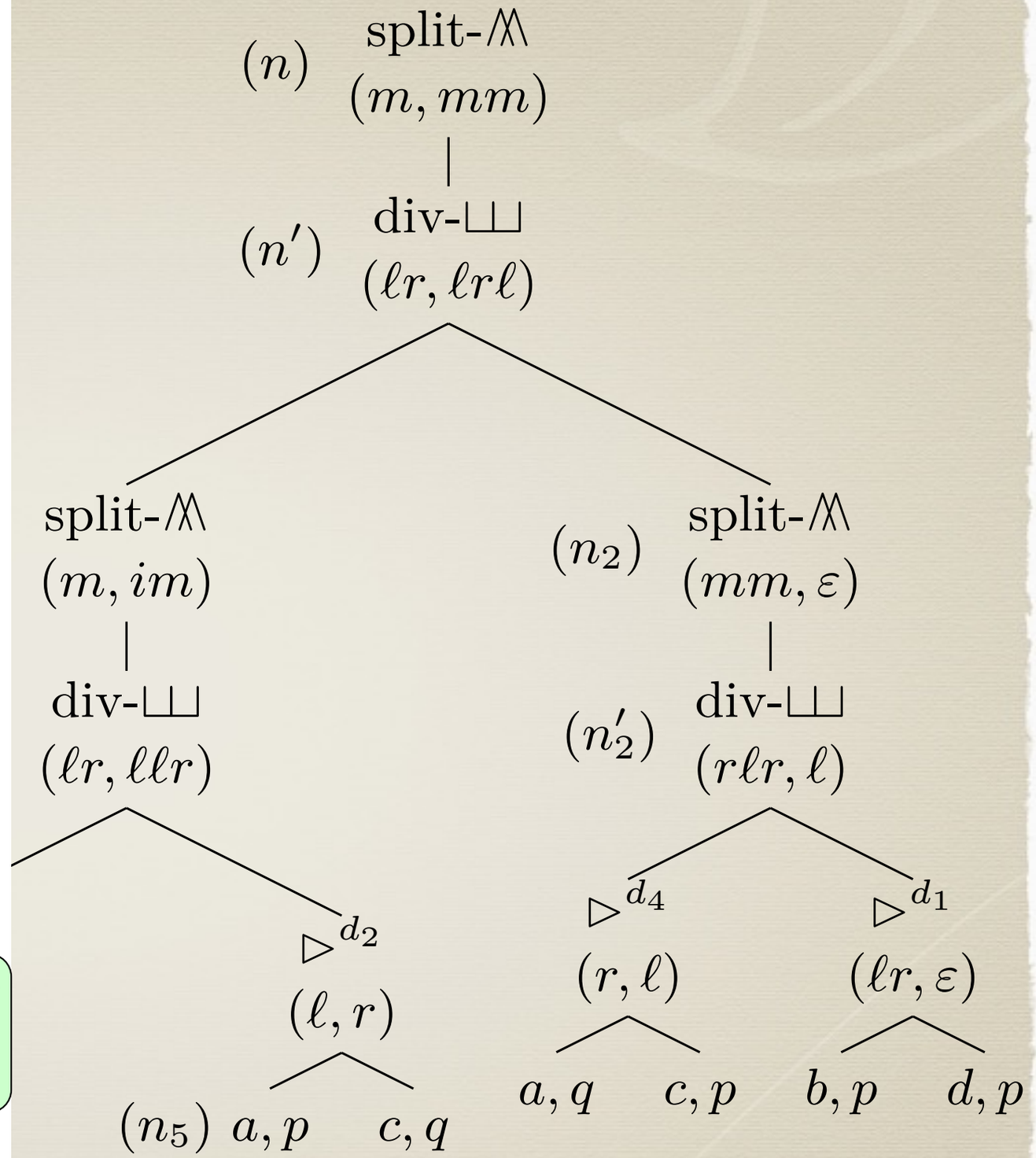
Tree interpretation in



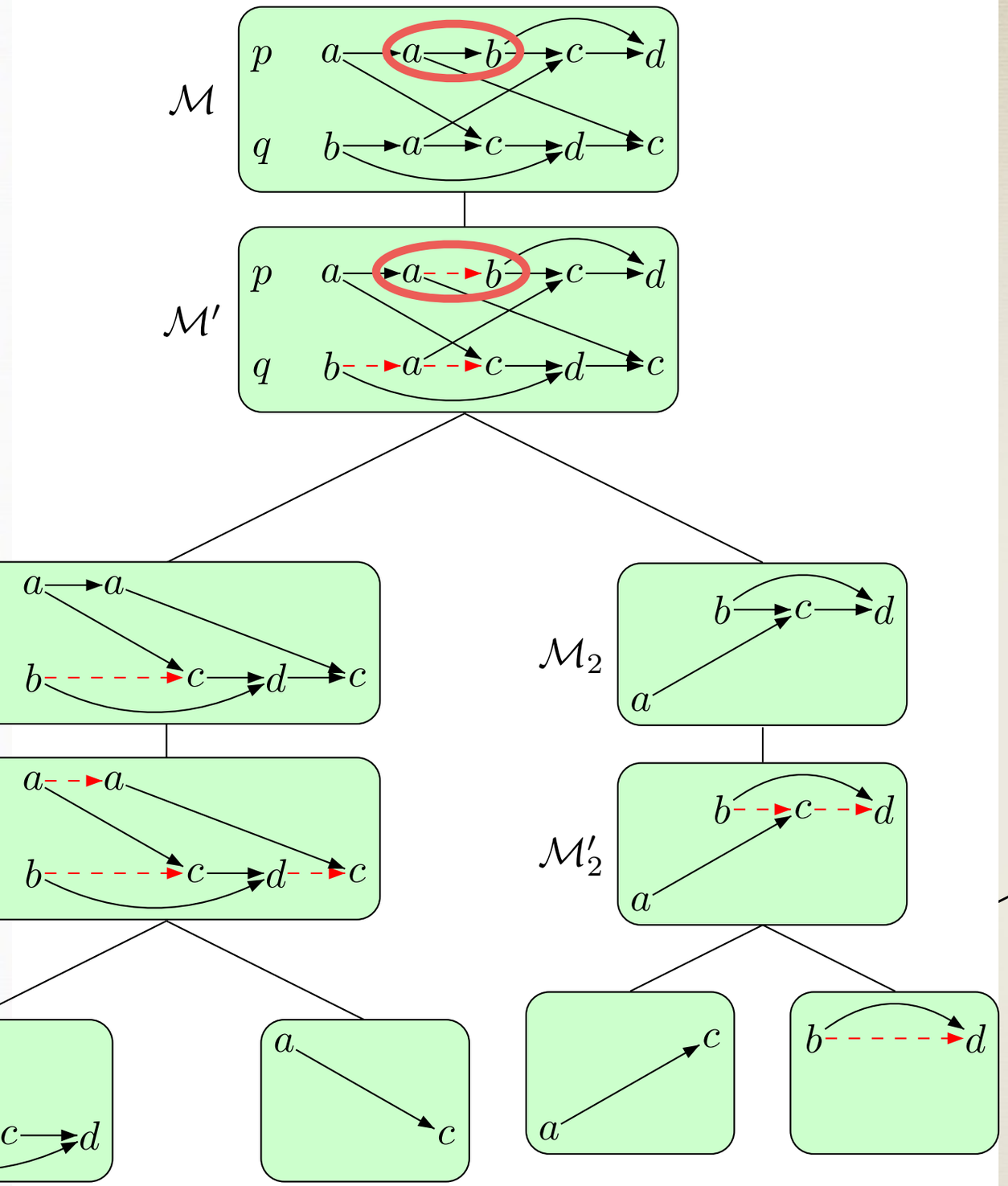
Vertices

Data edges

Process edges



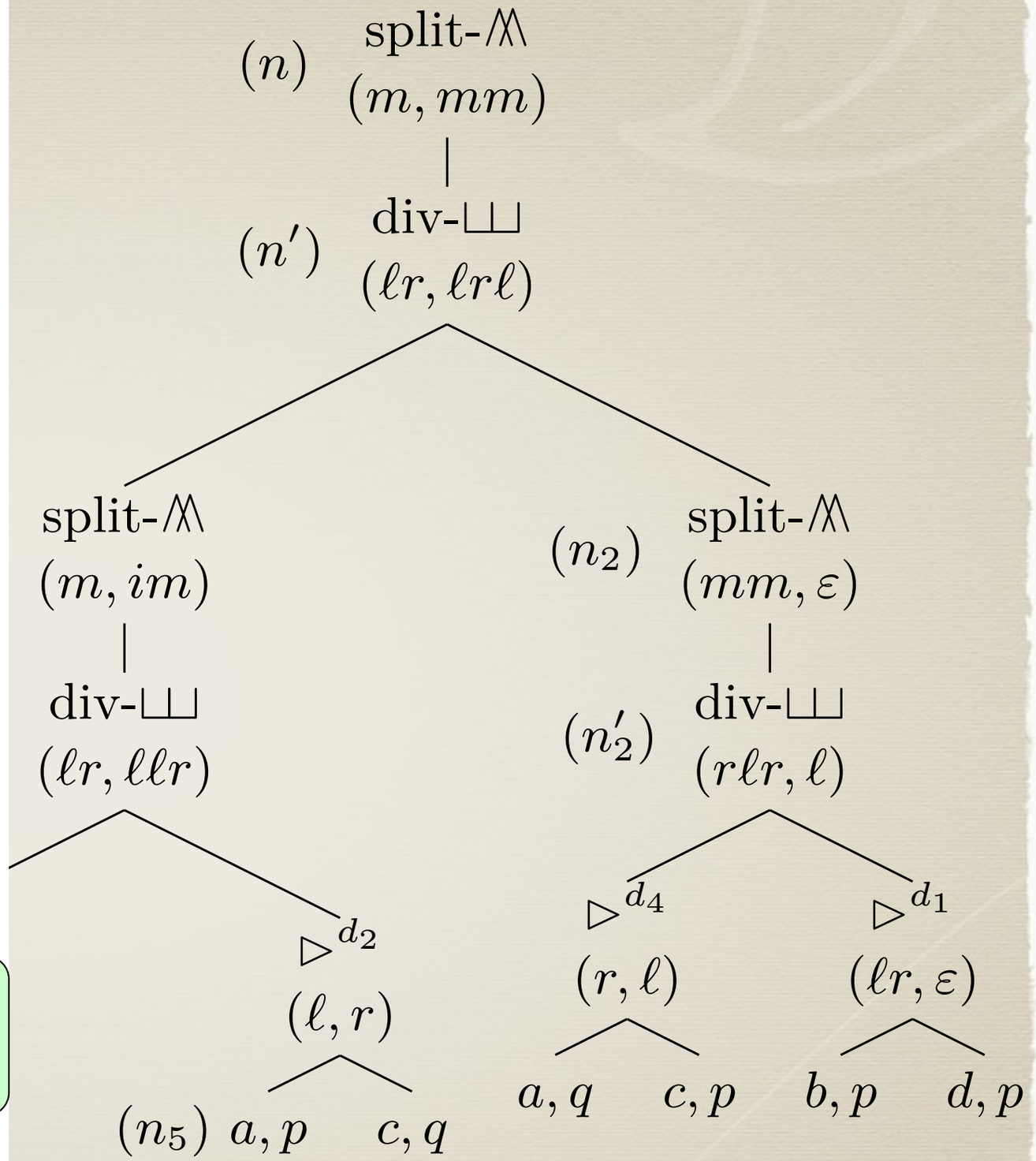
Tree interpretation in Disambiguated Split Terms



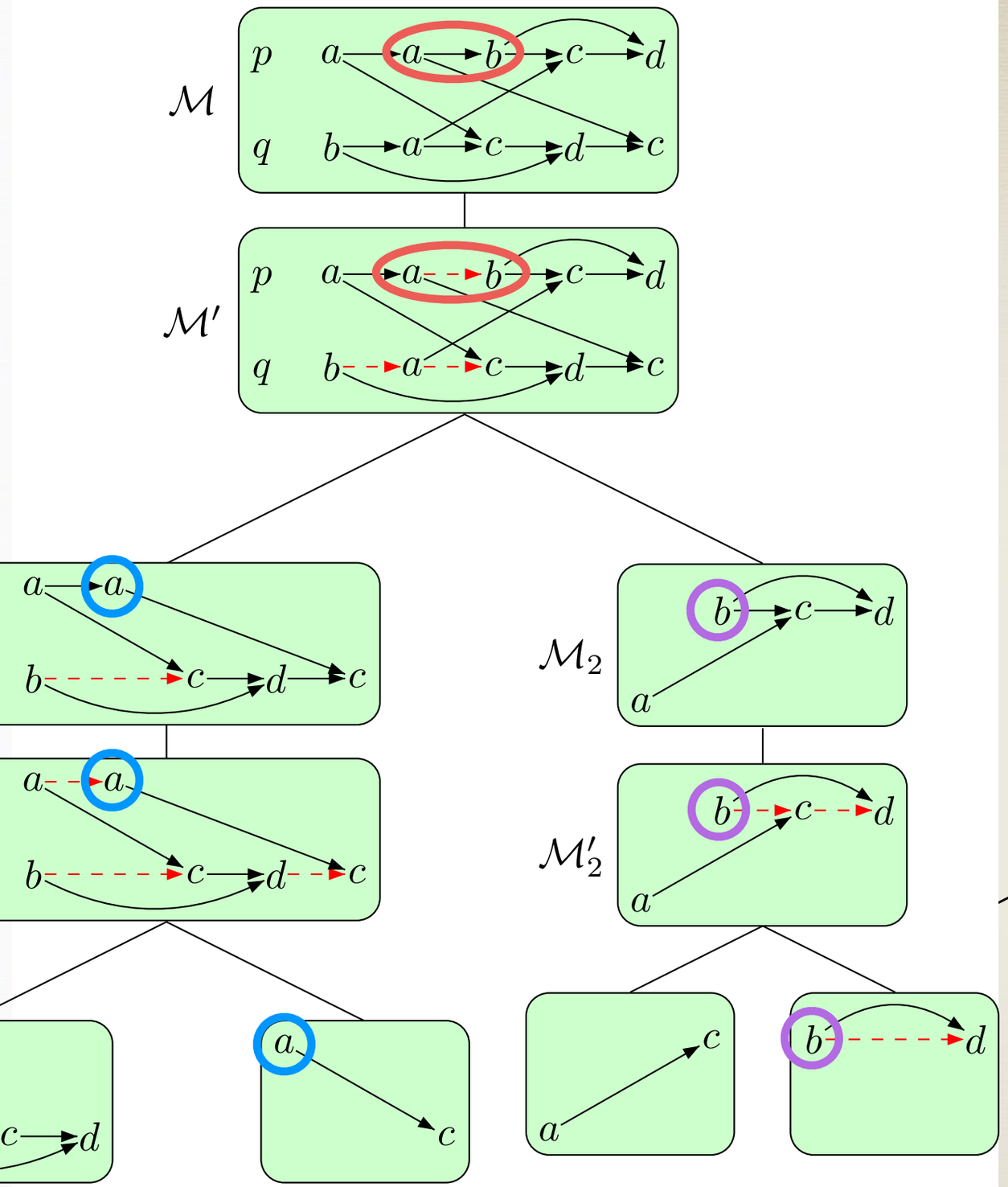
Vertices

Data edges

Process edges



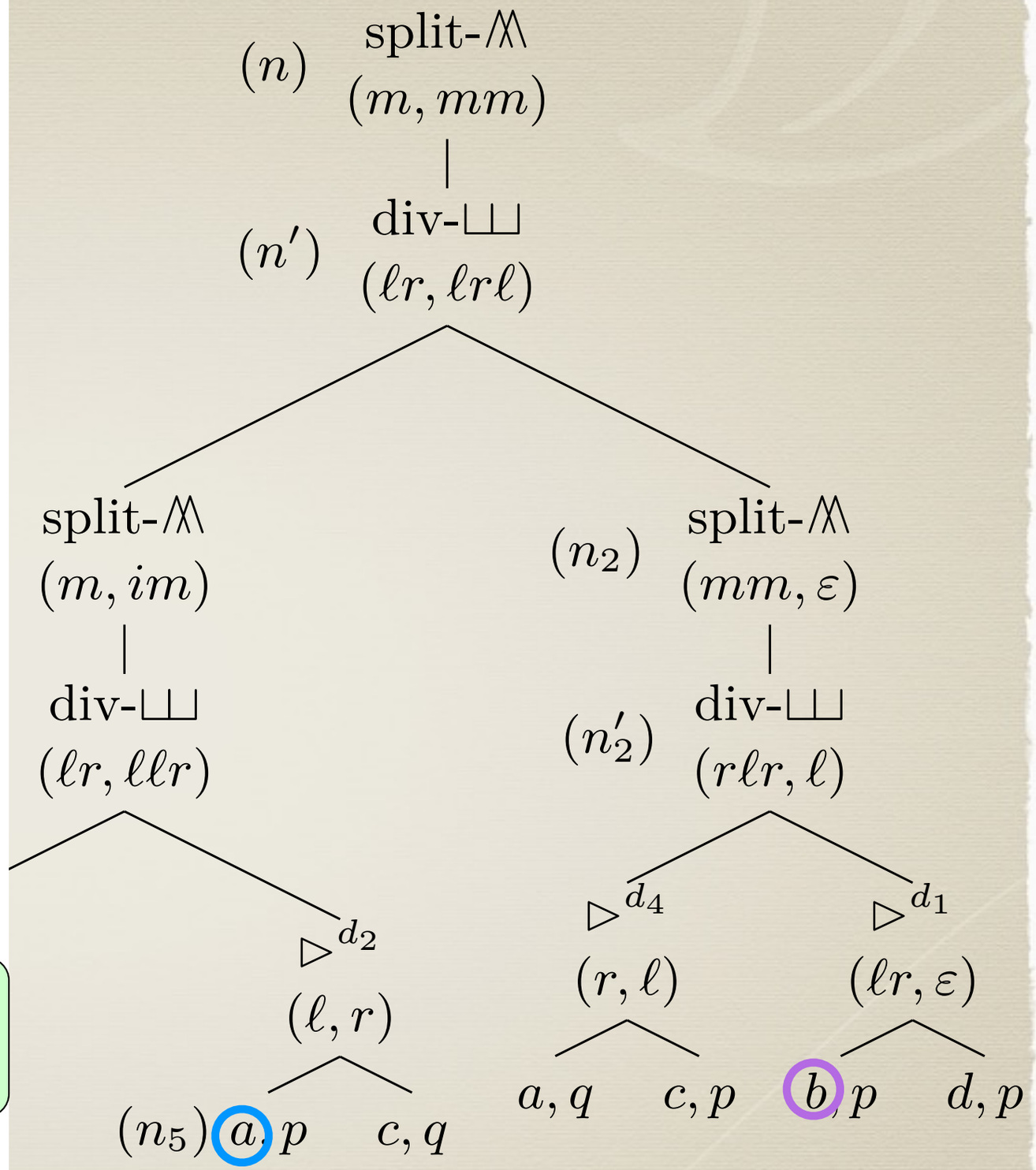
Tree interpretation in Disambiguated Split Terms



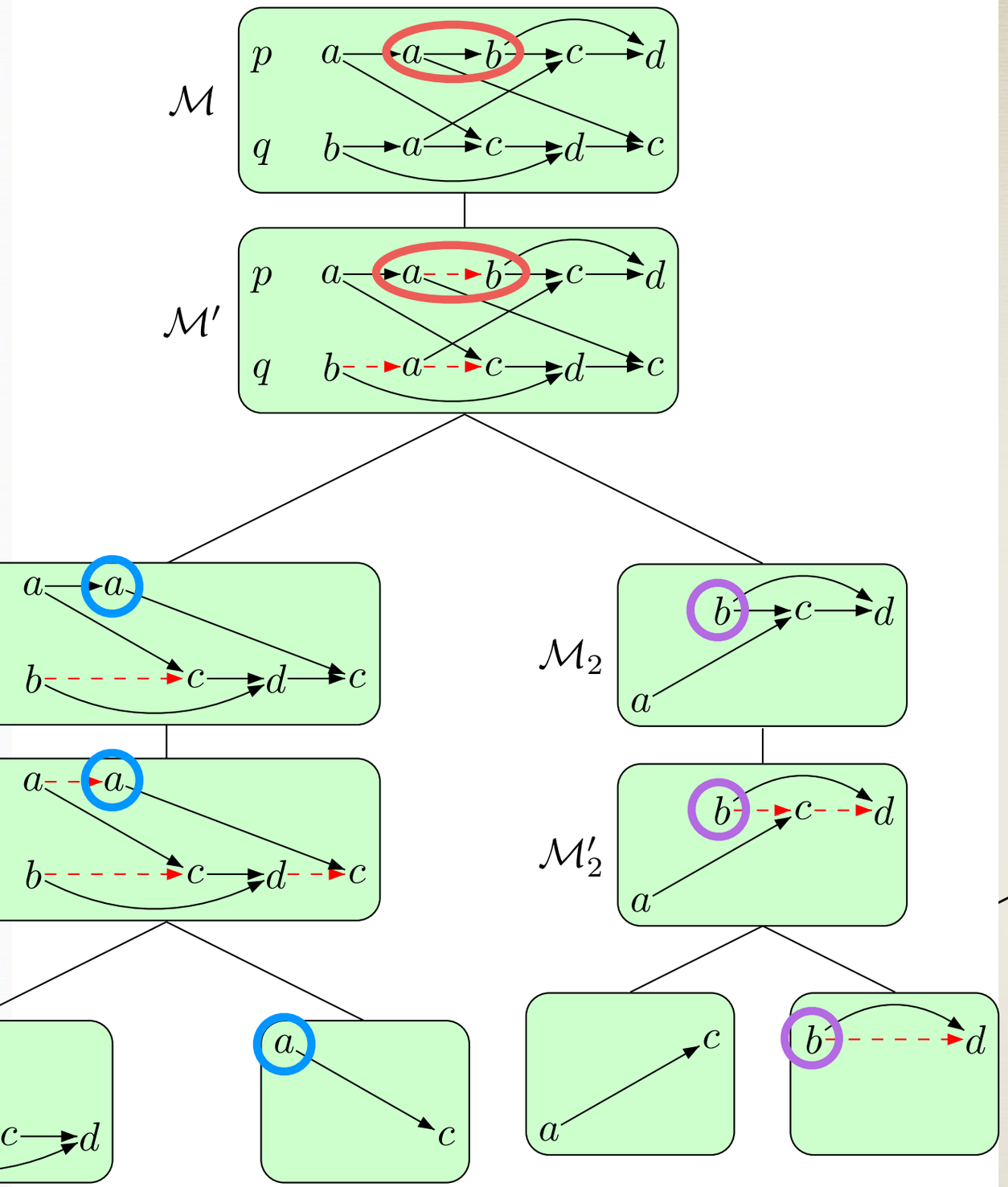
Vertices

Data edges

Process edges



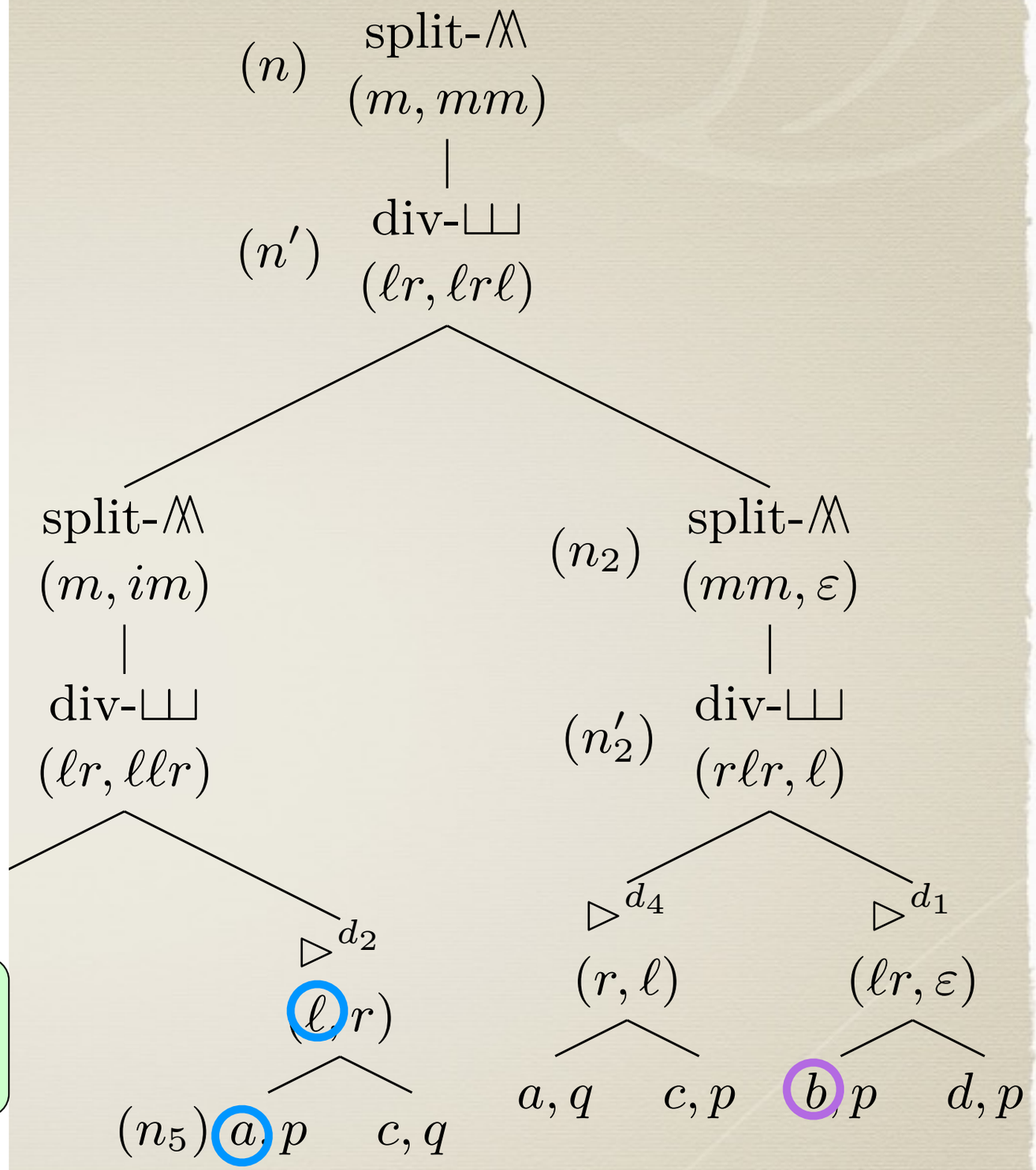
Tree interpretation in Disambiguated Split Terms



Vertices

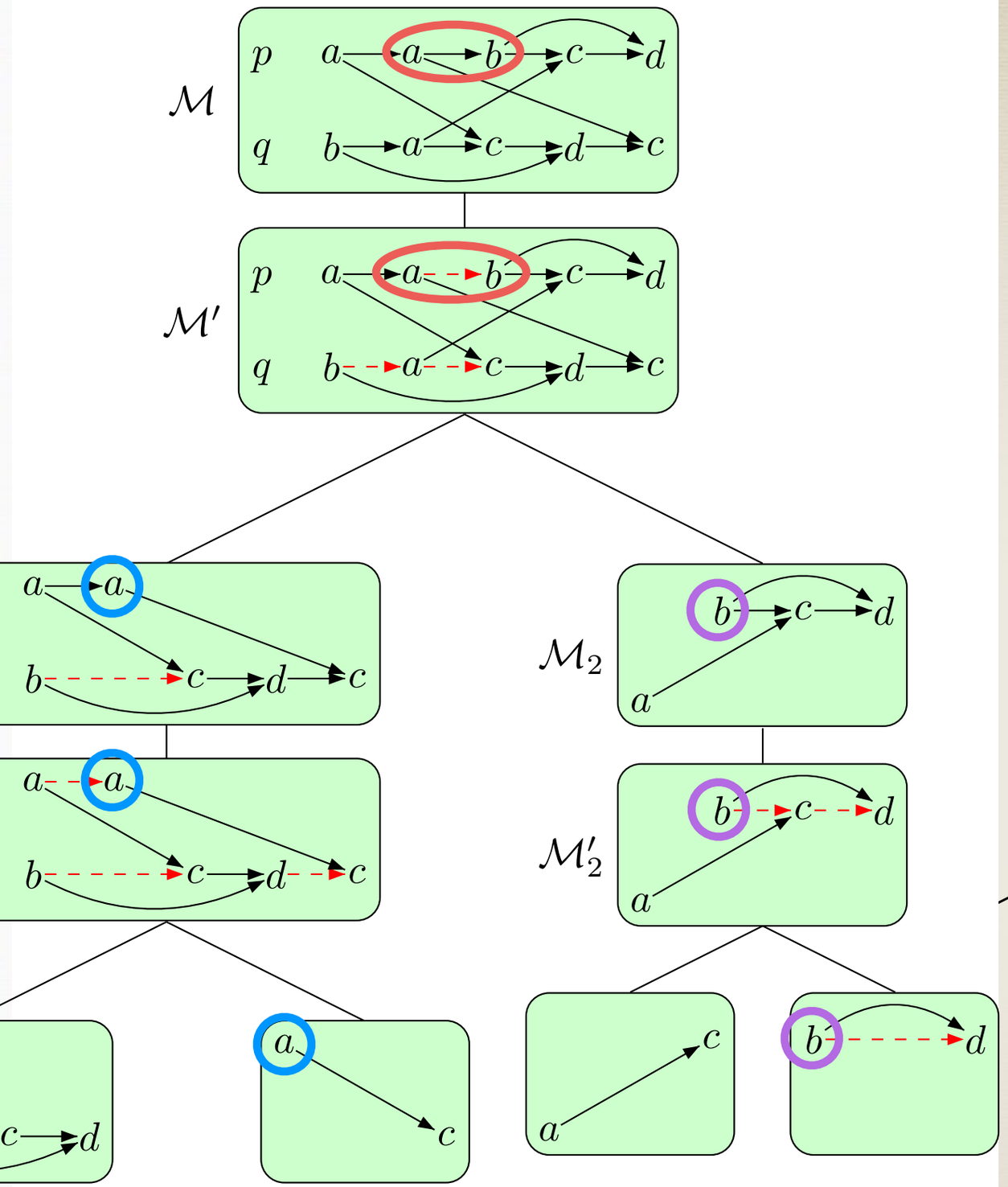
Data edges

Process edges



Tree interpretation in

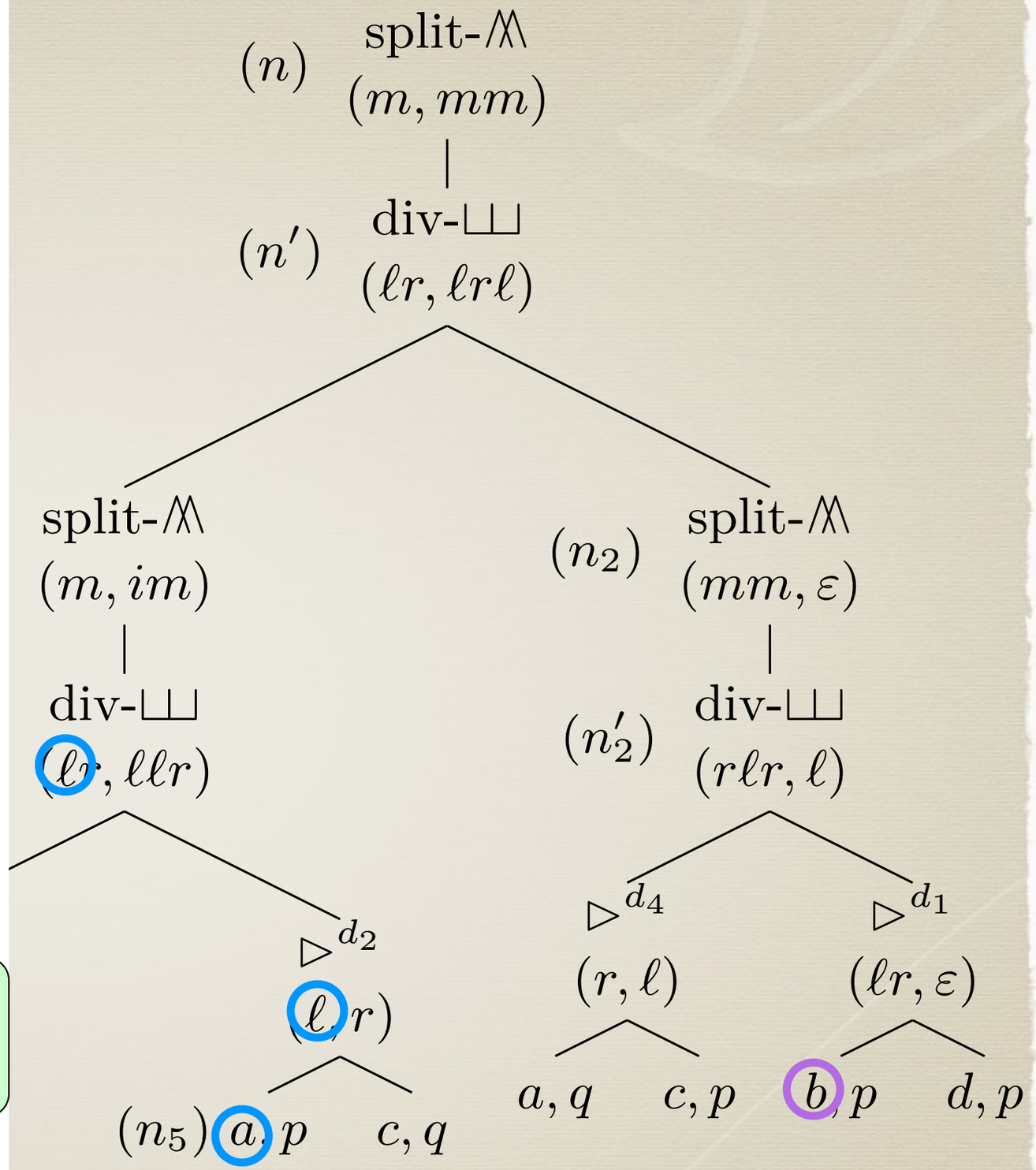
Disambiguated Split Terms



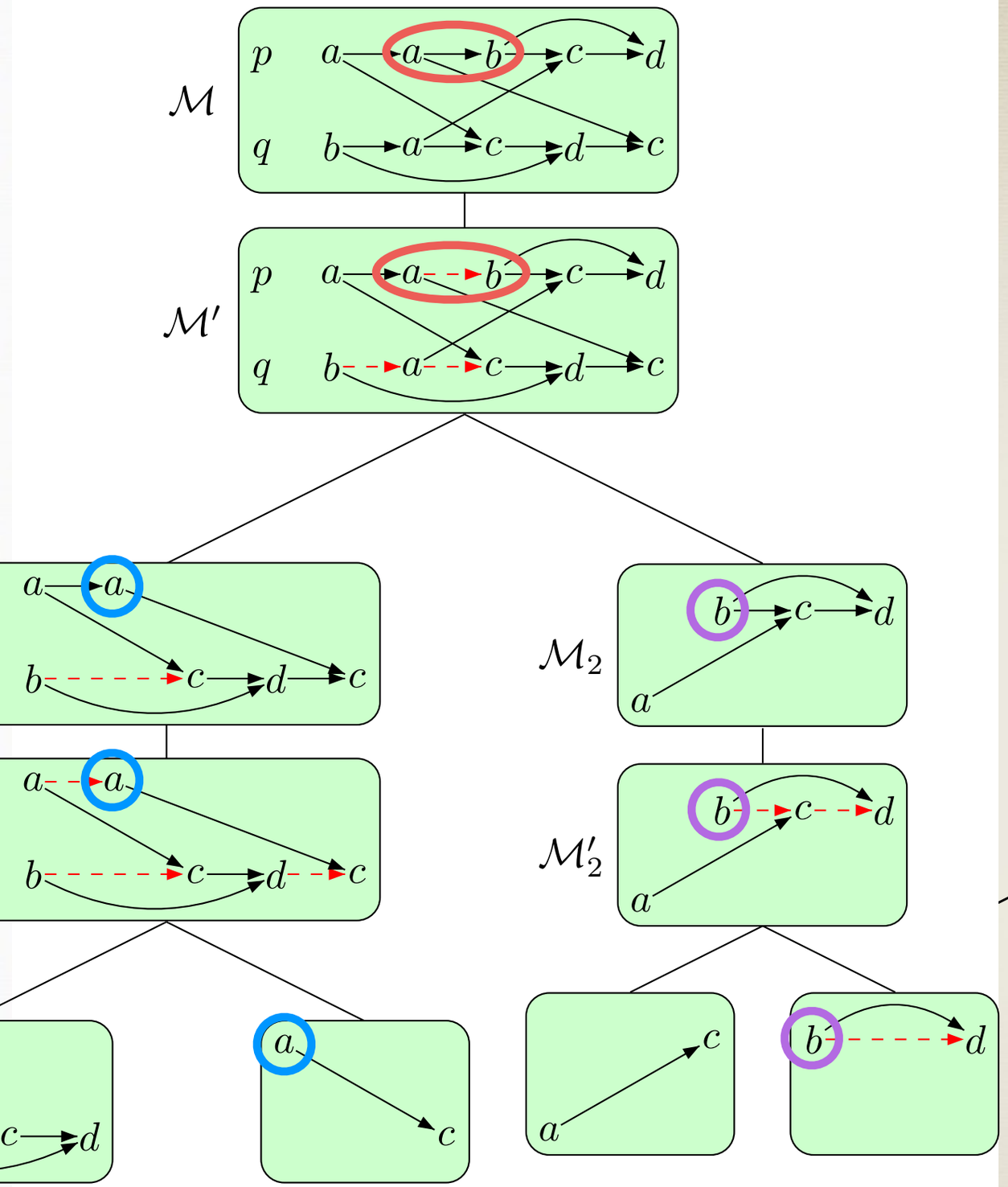
Vertices

Data edges

Process edges



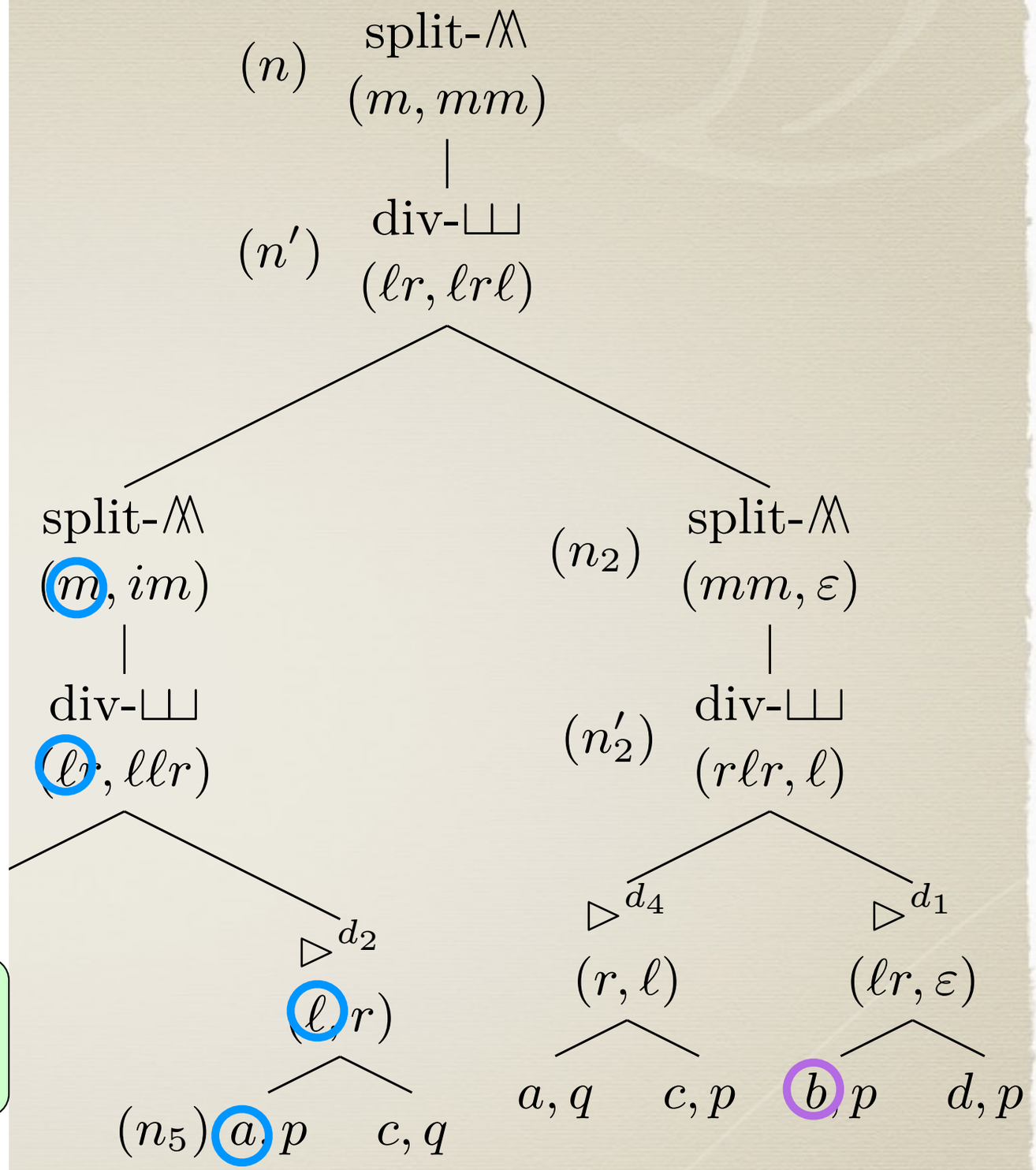
Tree interpretation in Disambiguated Split Terms



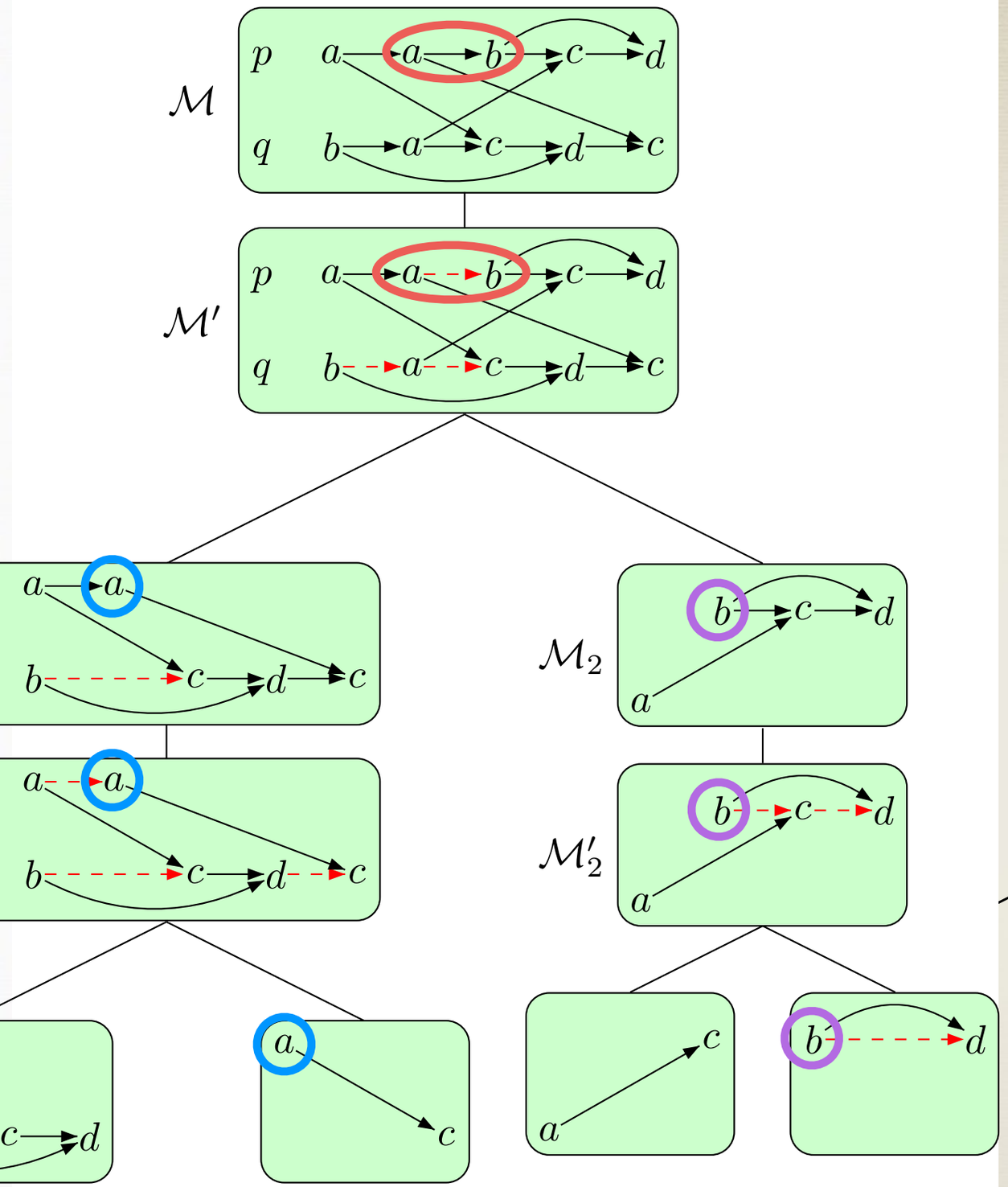
Vertices

Data edges

Process edges



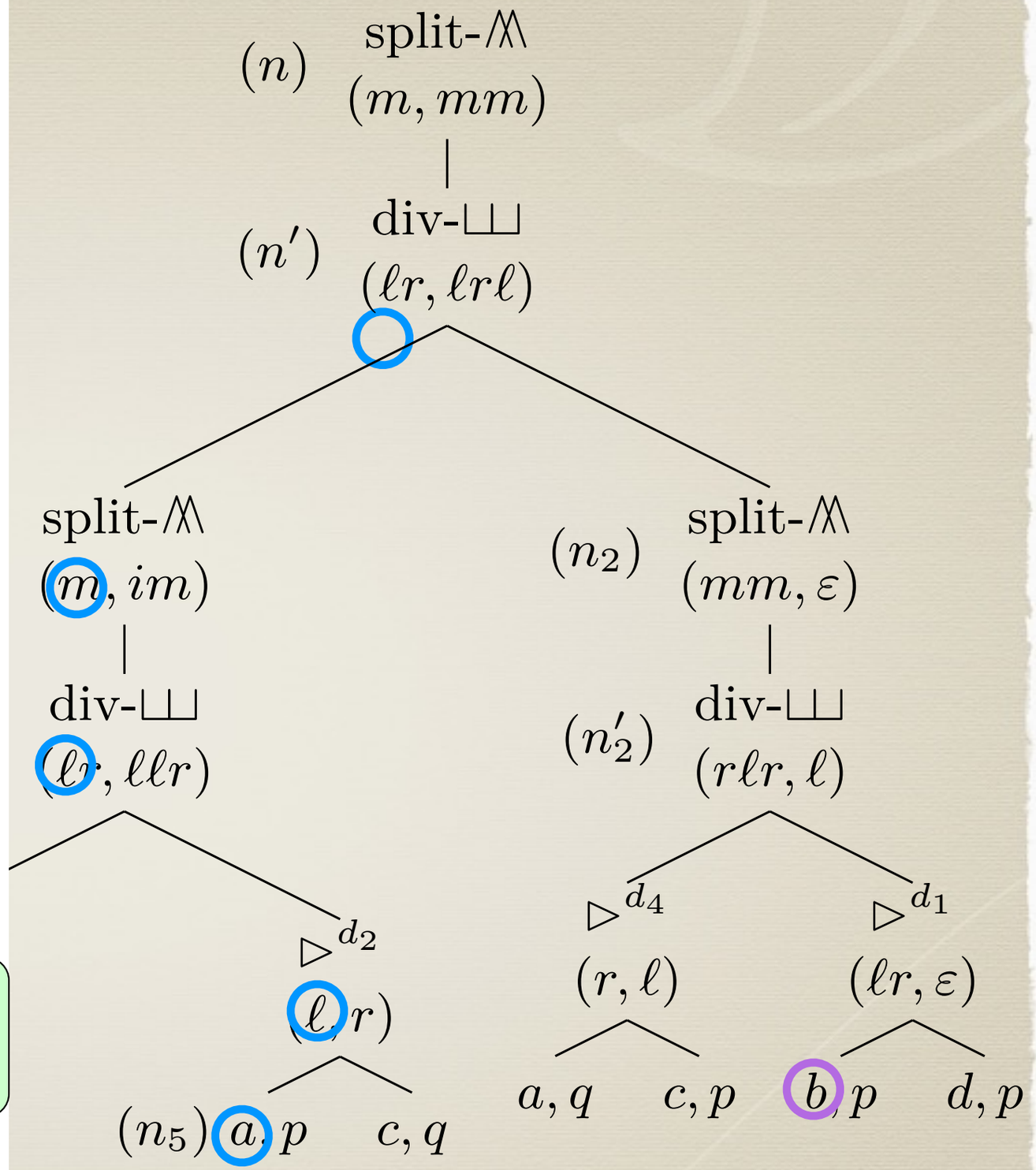
Tree interpretation in Disambiguated Split Terms



Vertices

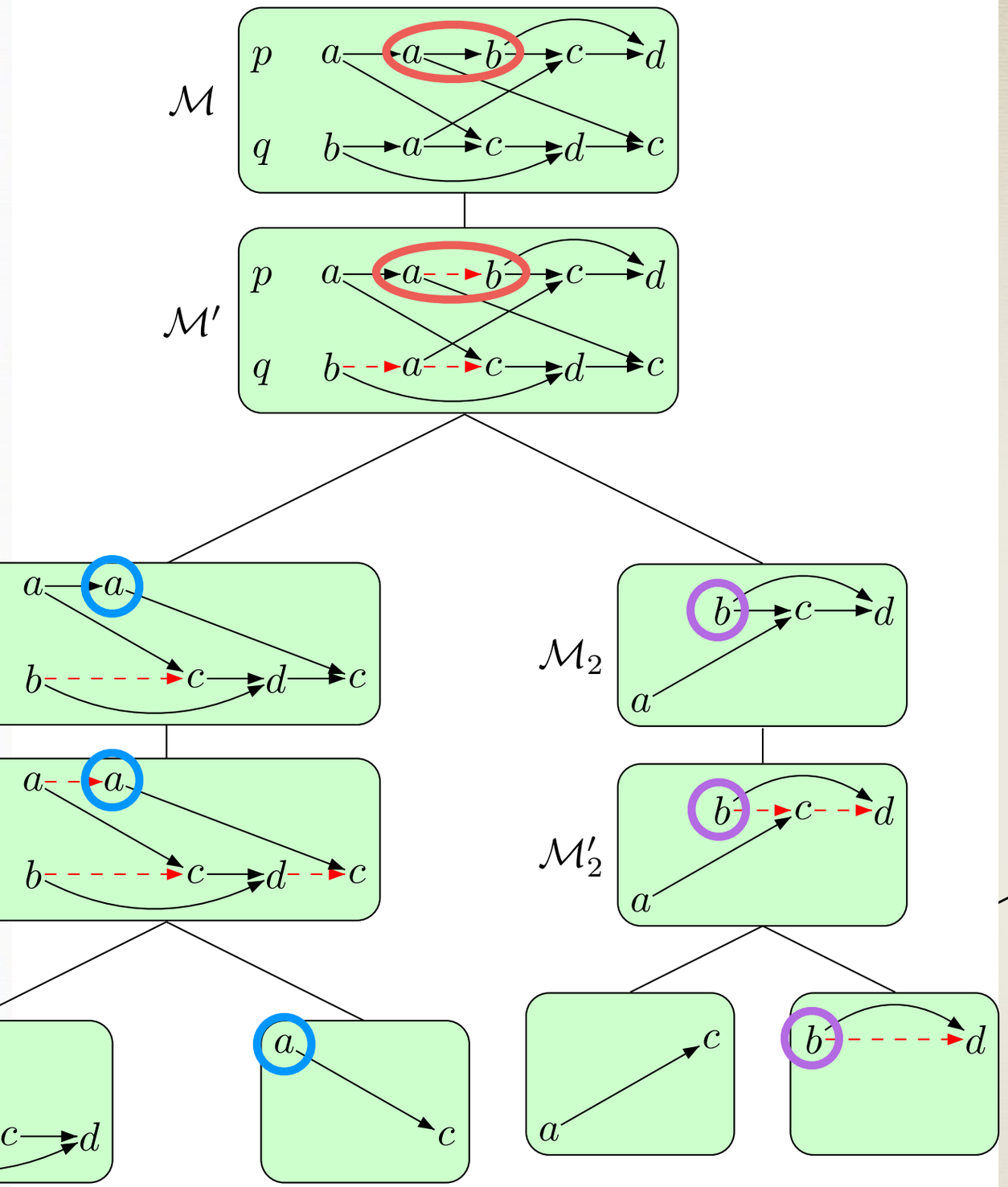
Data edges

Process edges



Tree interpretation in

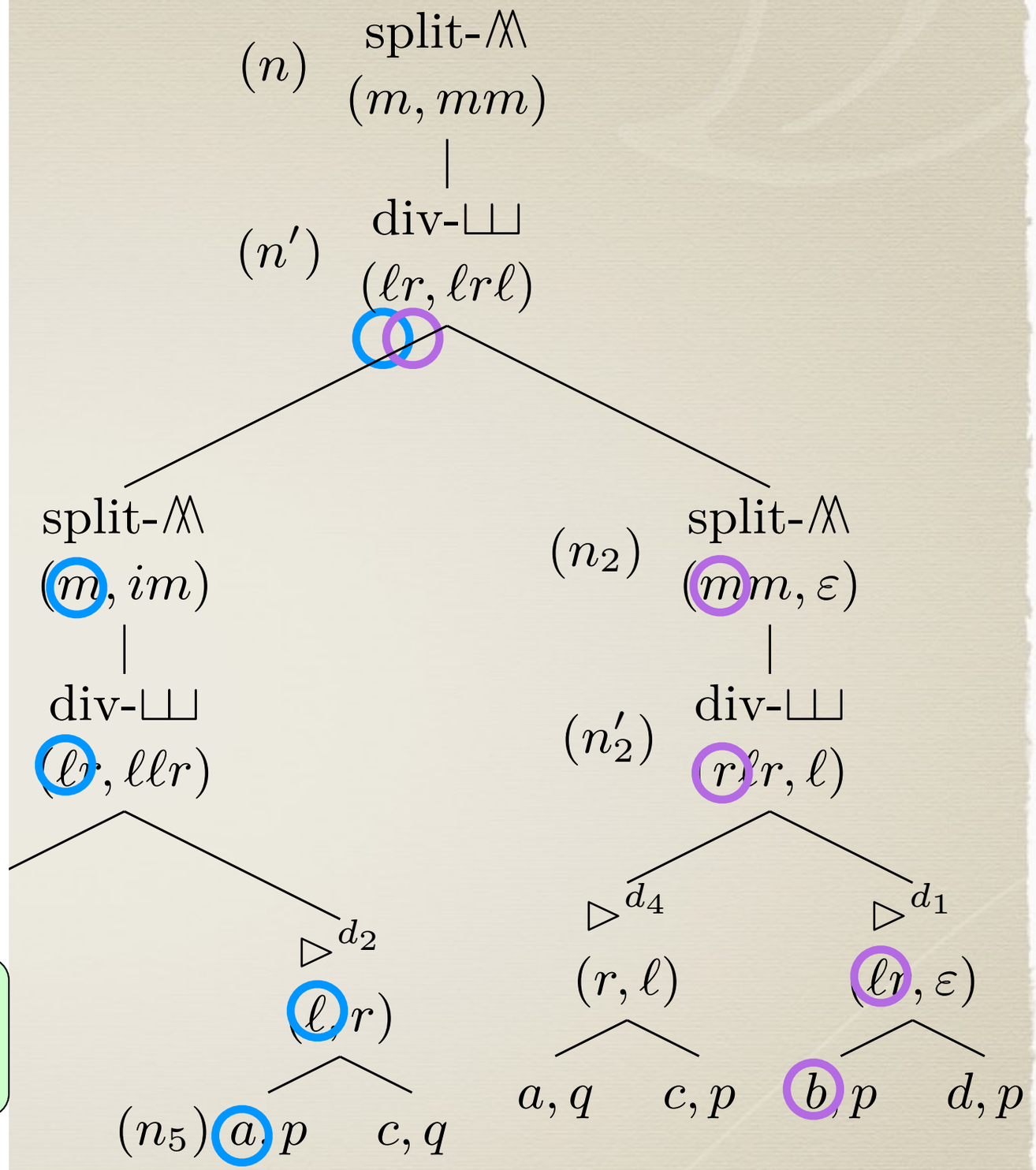
Disambiguated Split Terms



Vertices

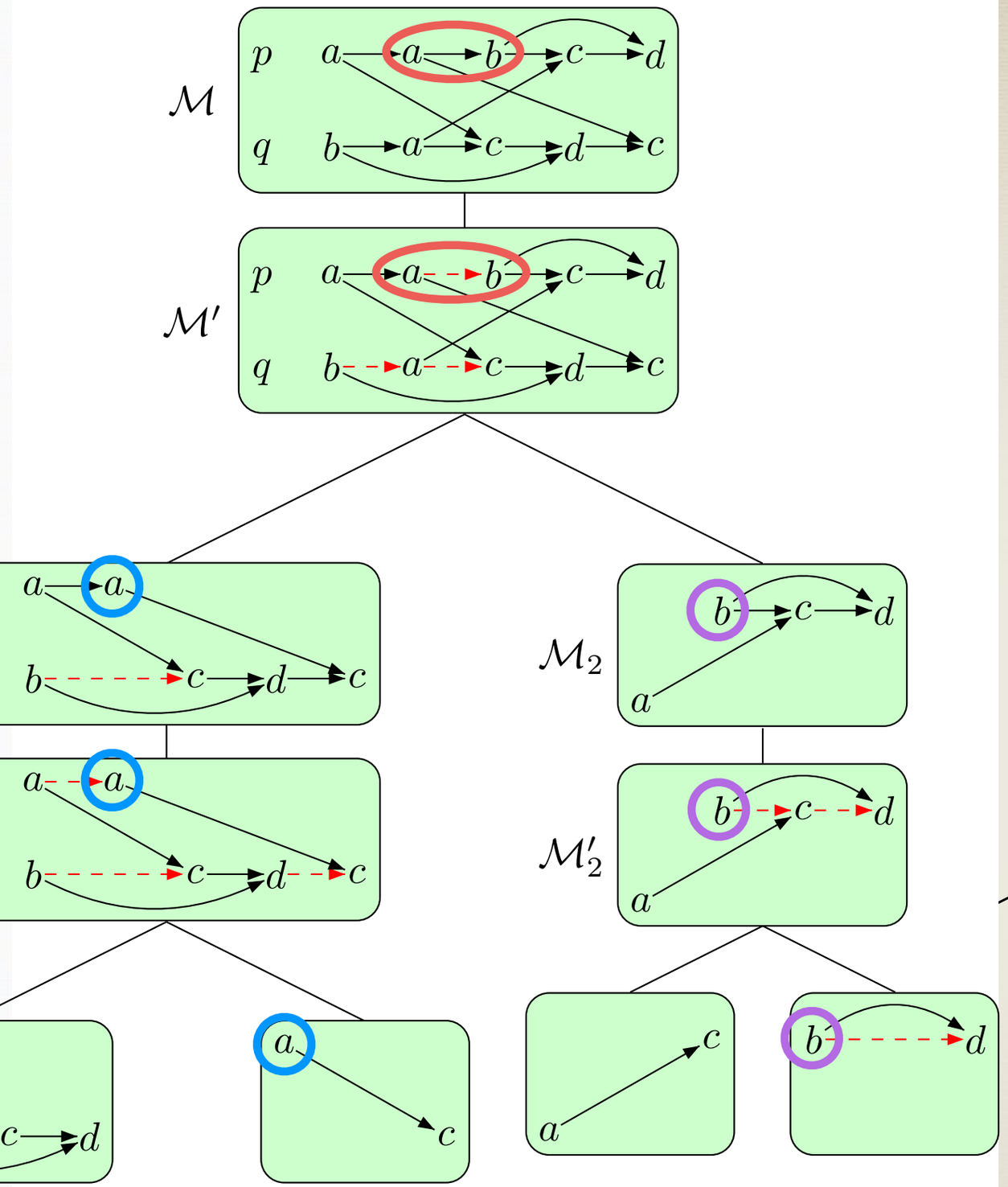
Data edges

Process edges



Tree interpretation in

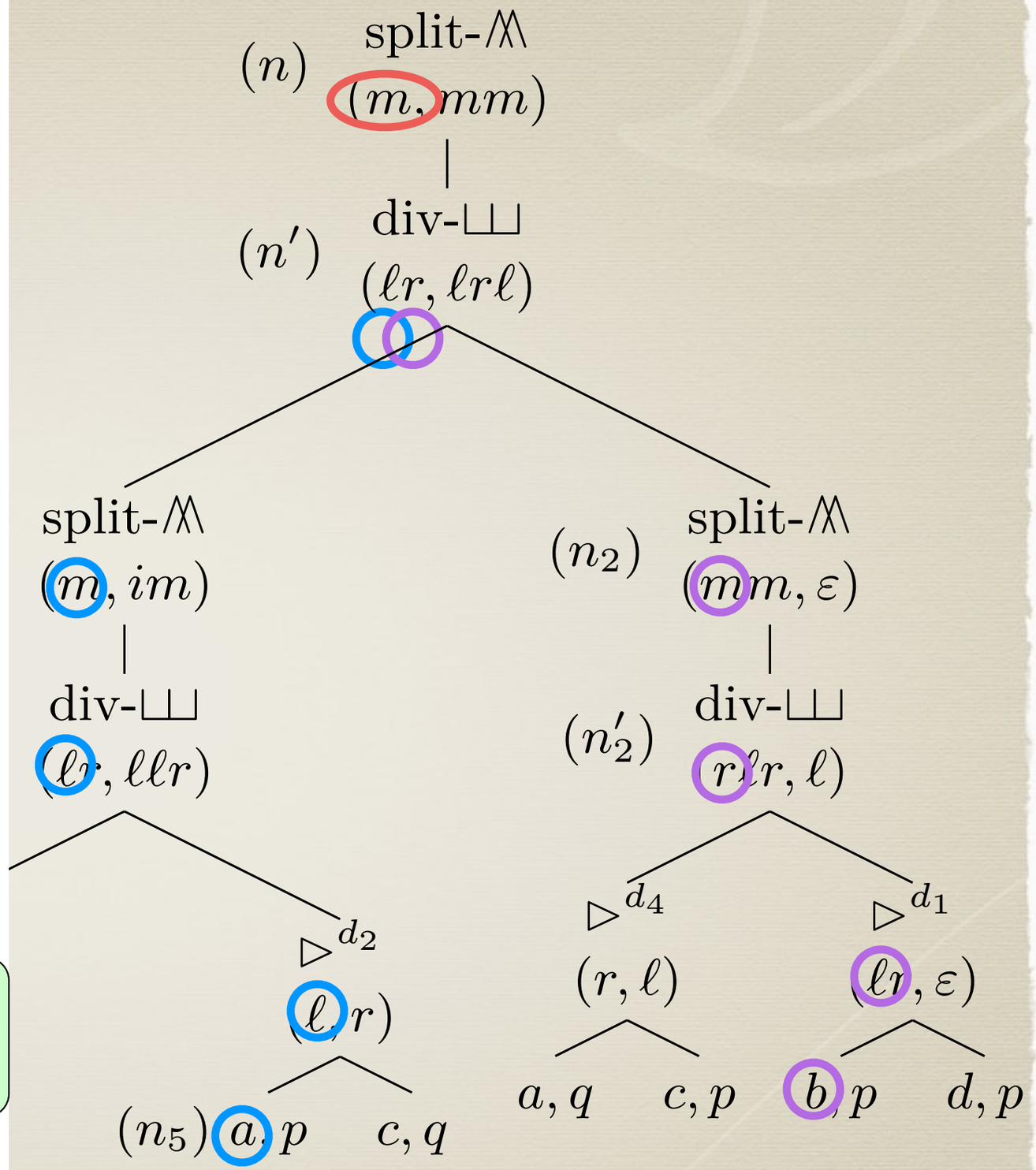
Disambiguated Split Terms



Vertices

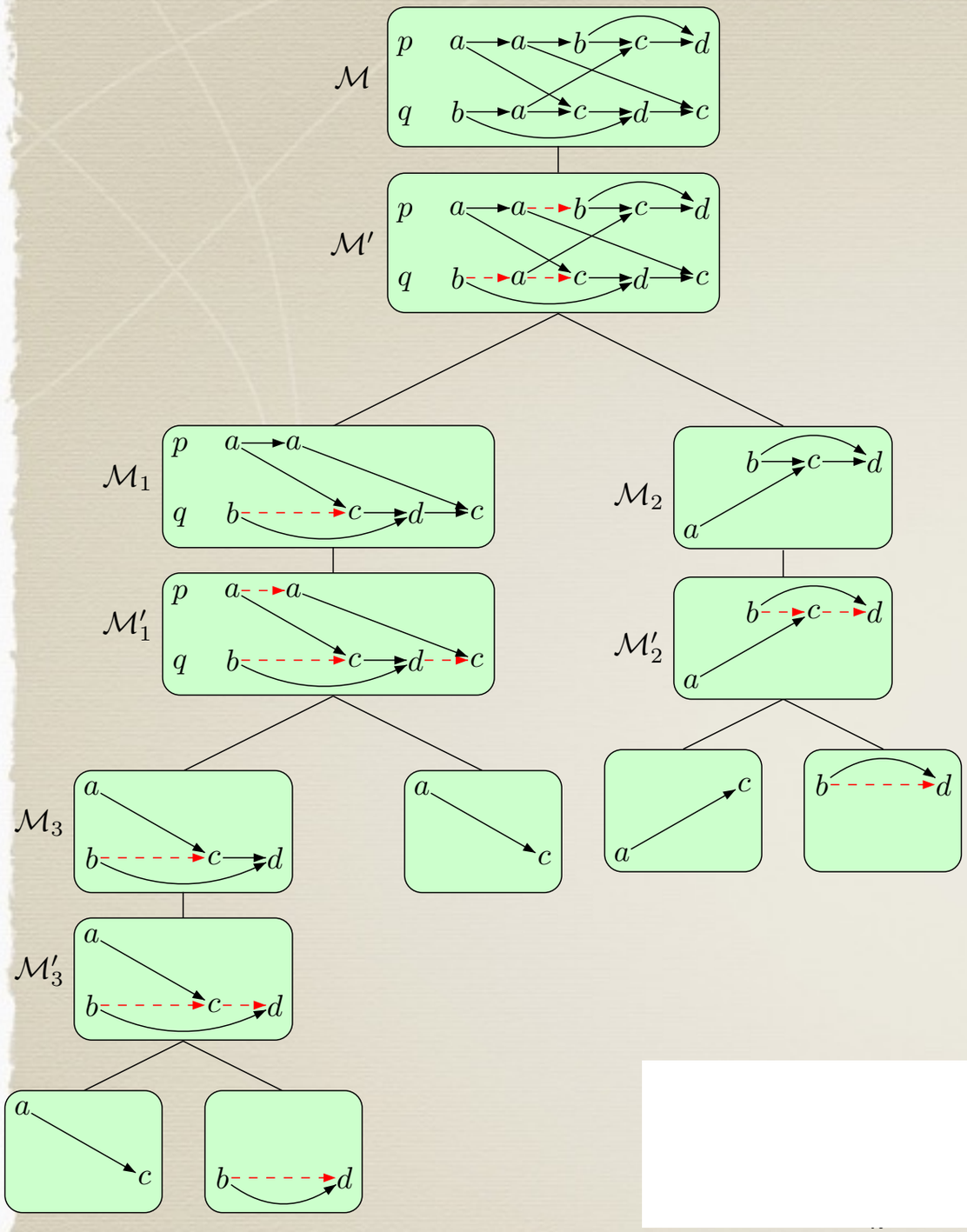
Data edges

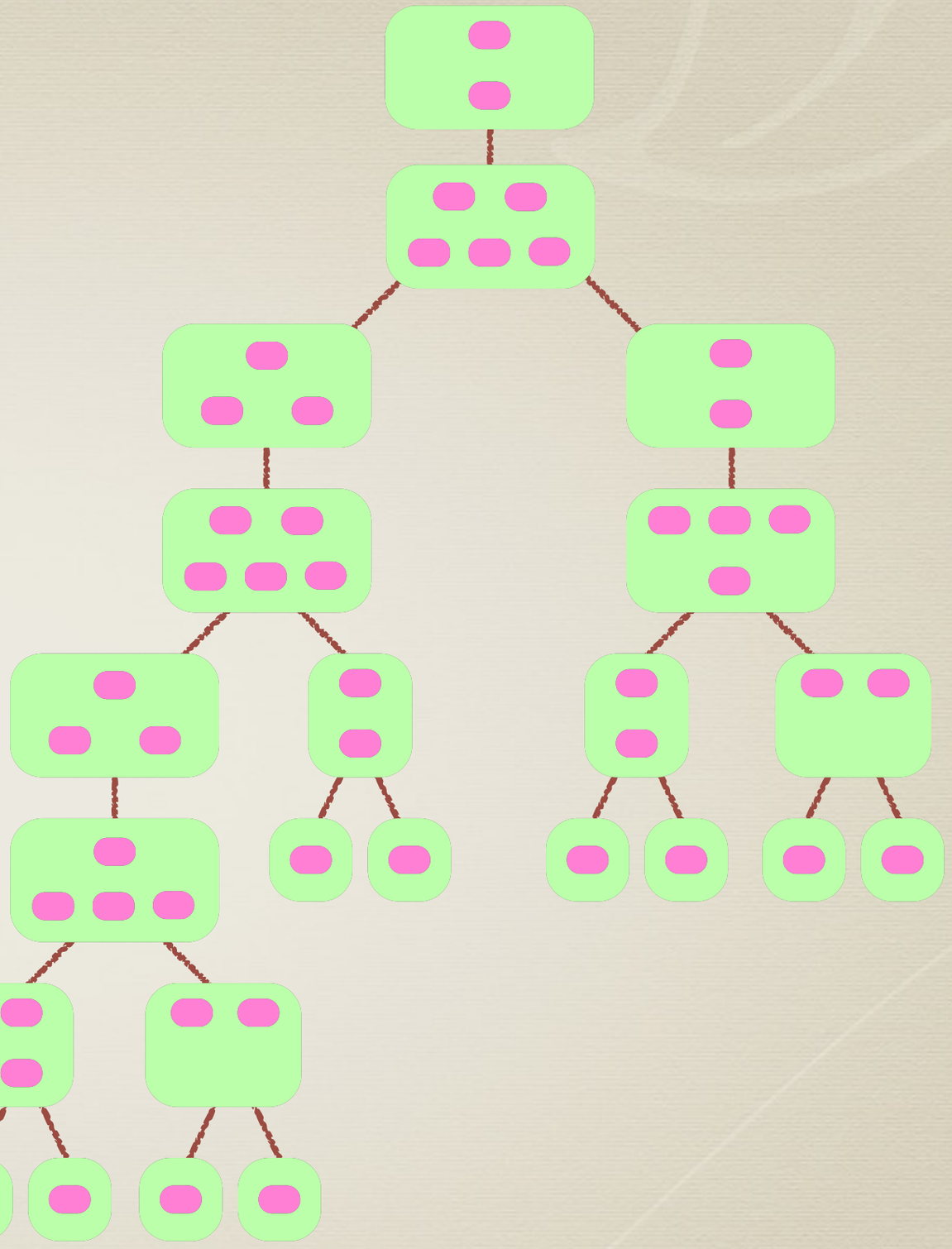
Process edges



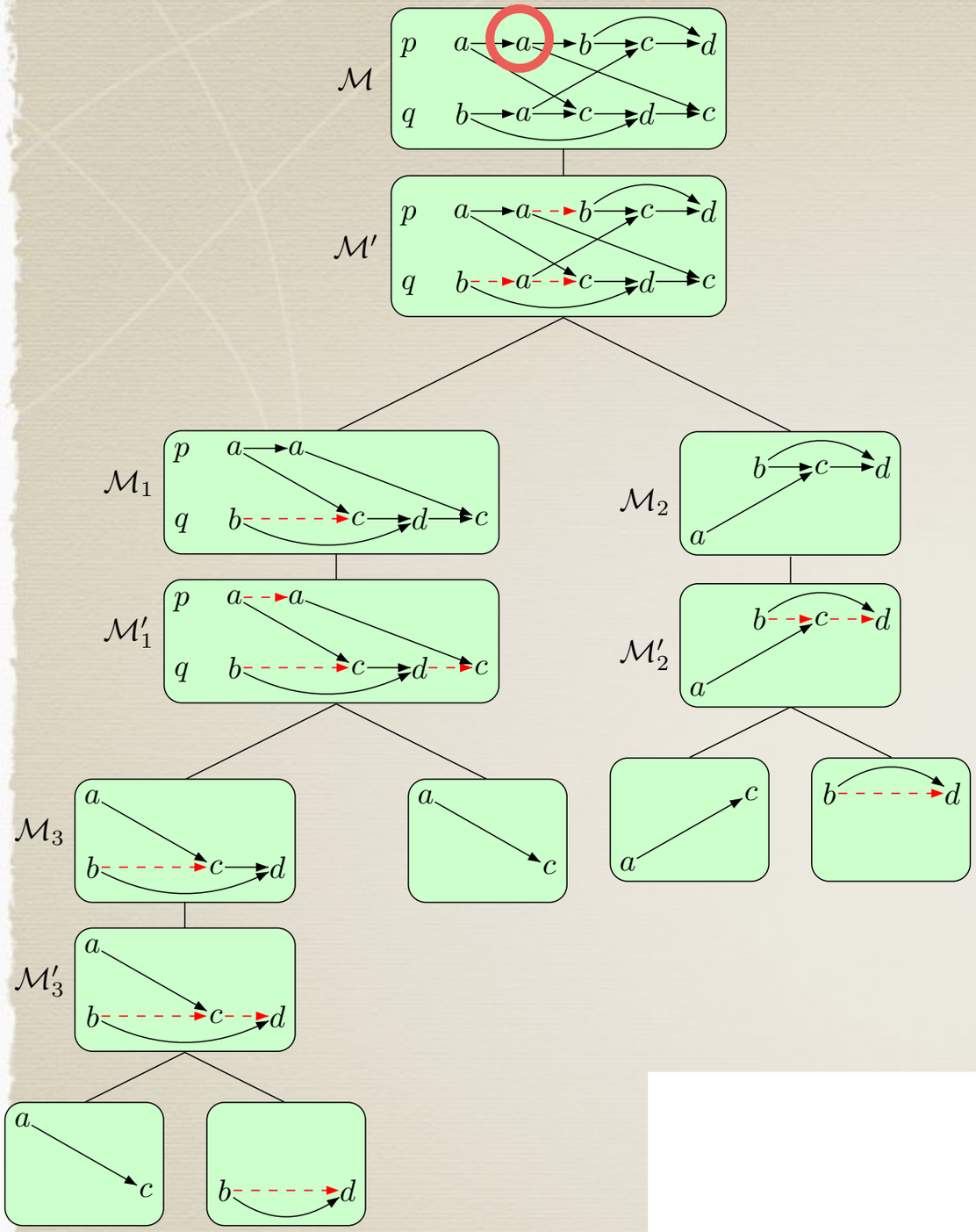
Tree interpretation in

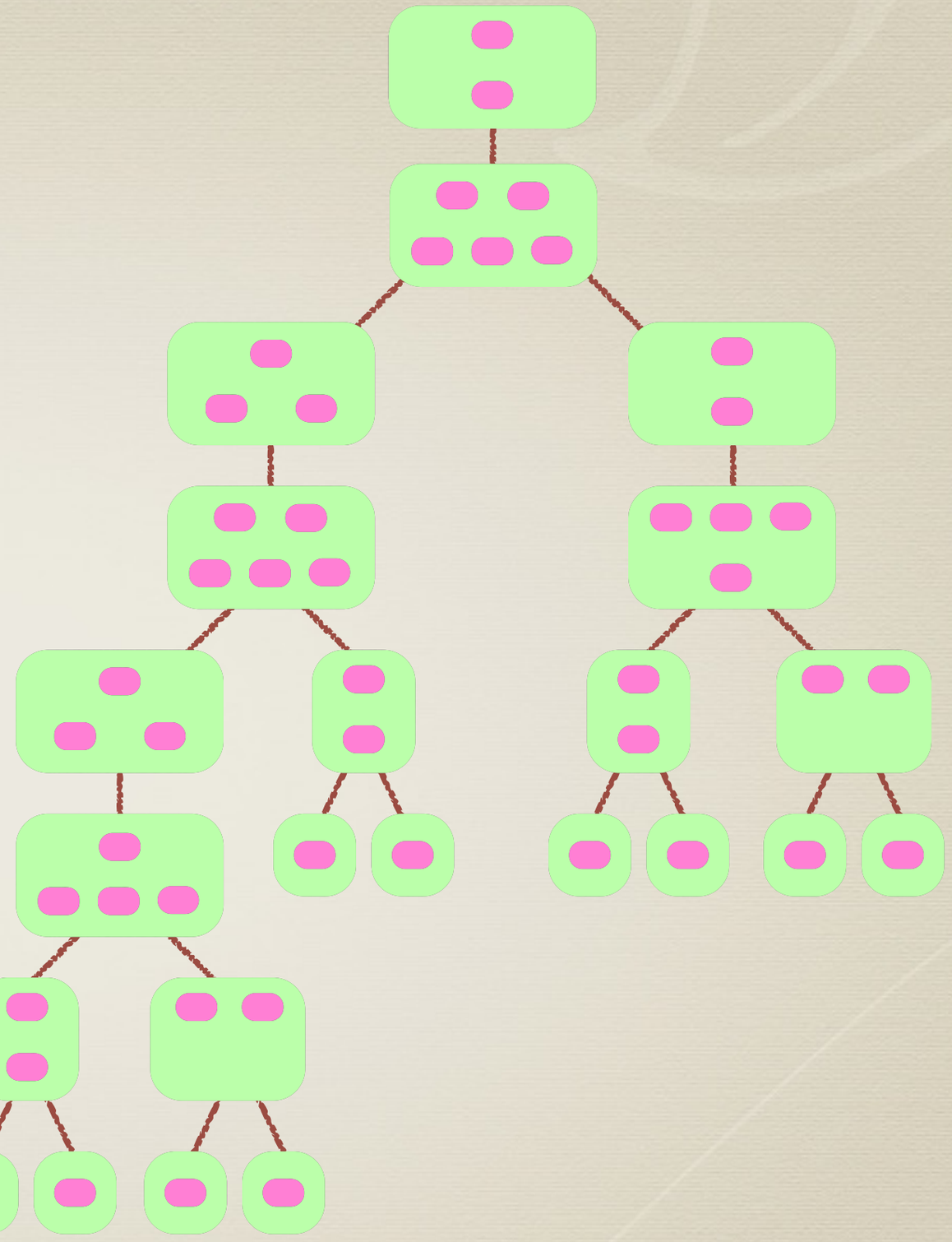
Disambiguated Split Terms





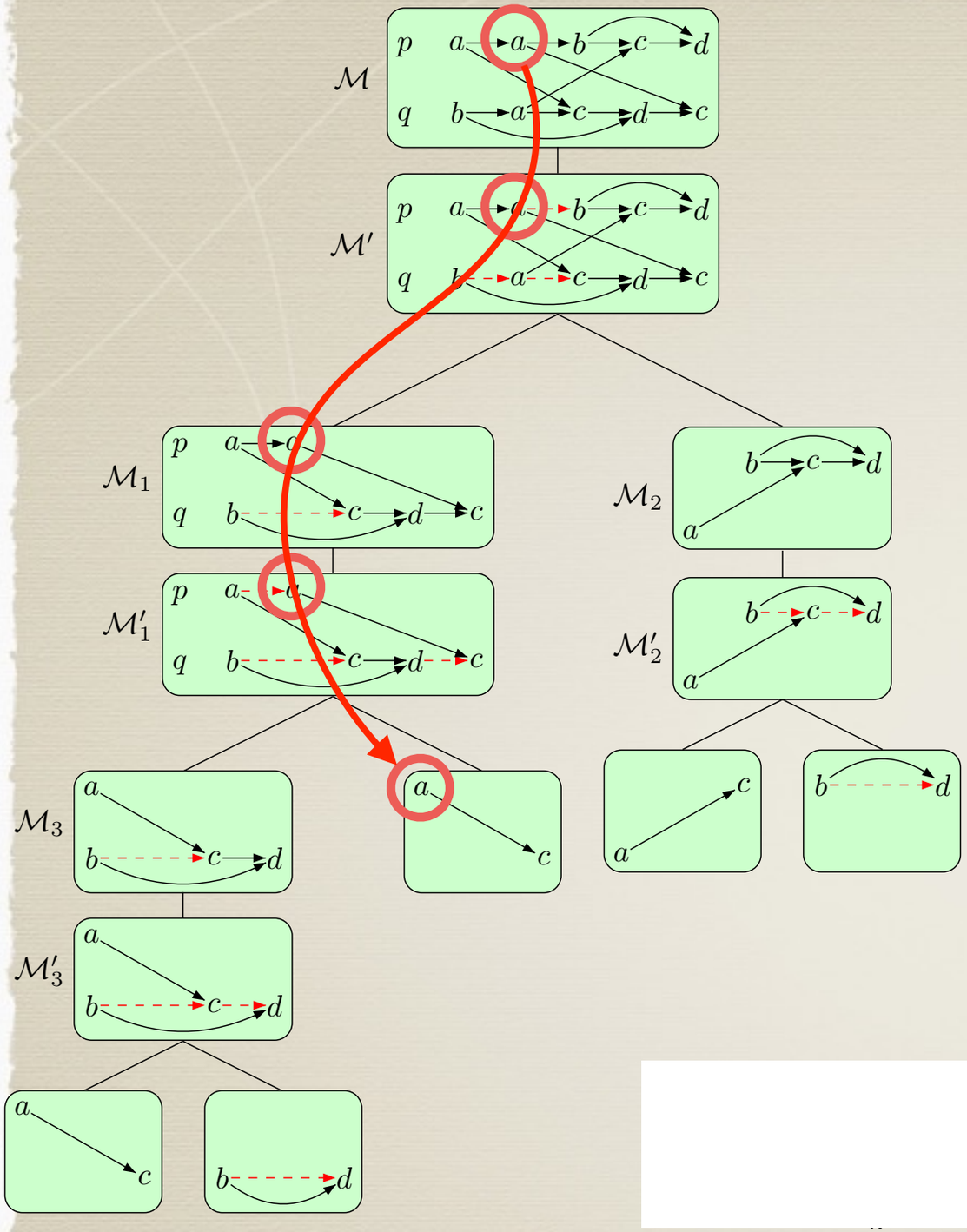
Tree interpretation in

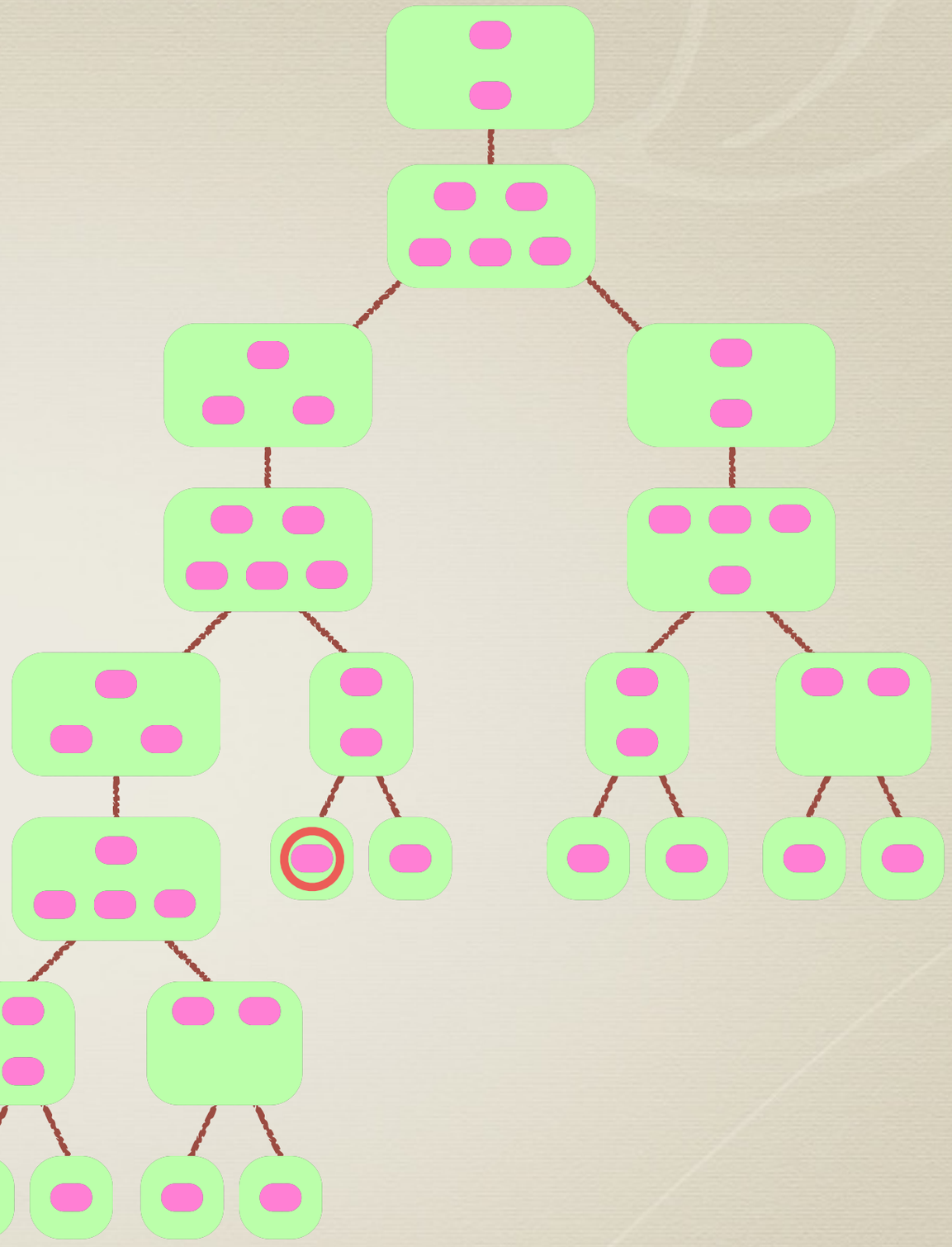




Vertices

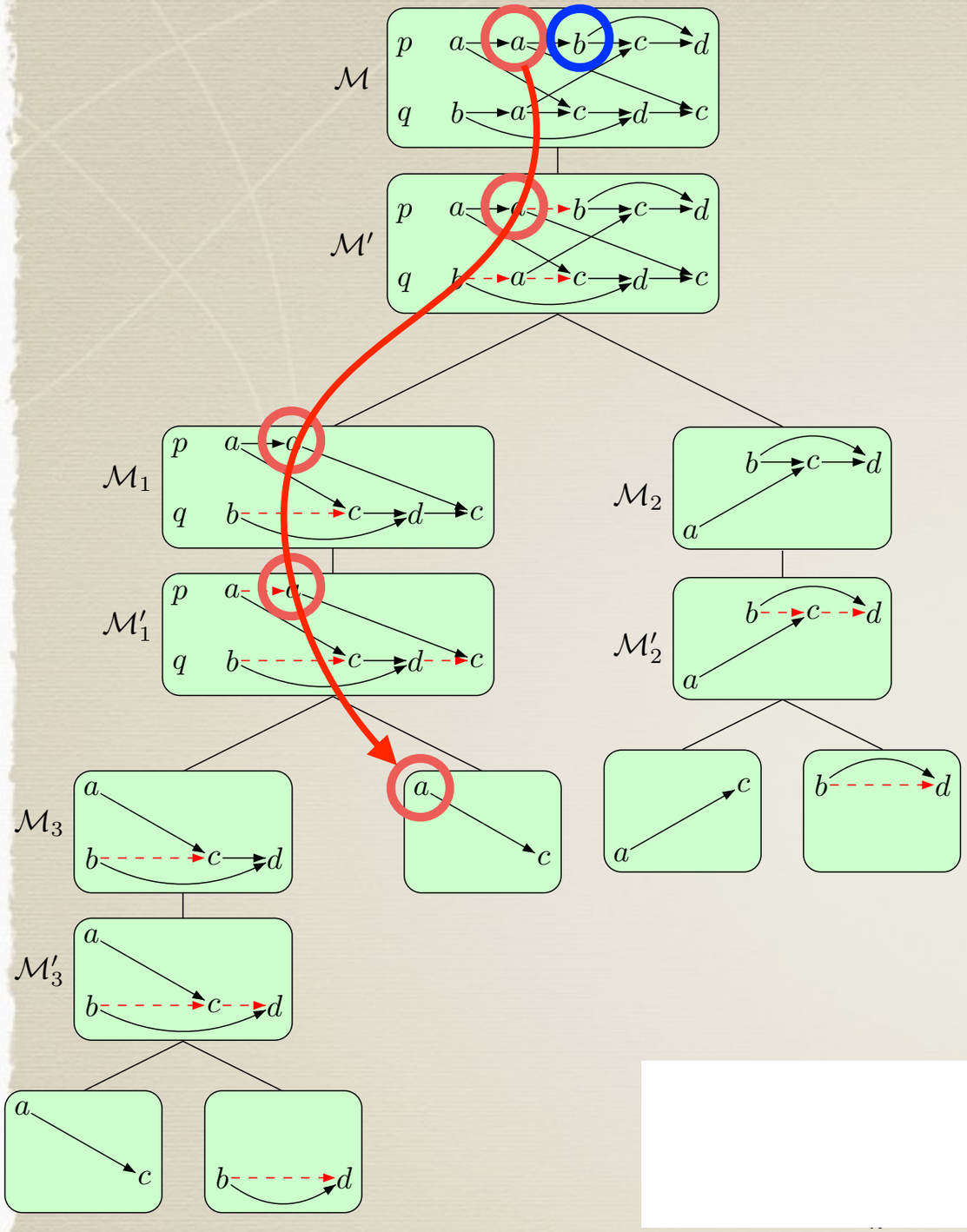
Tree interpretation in

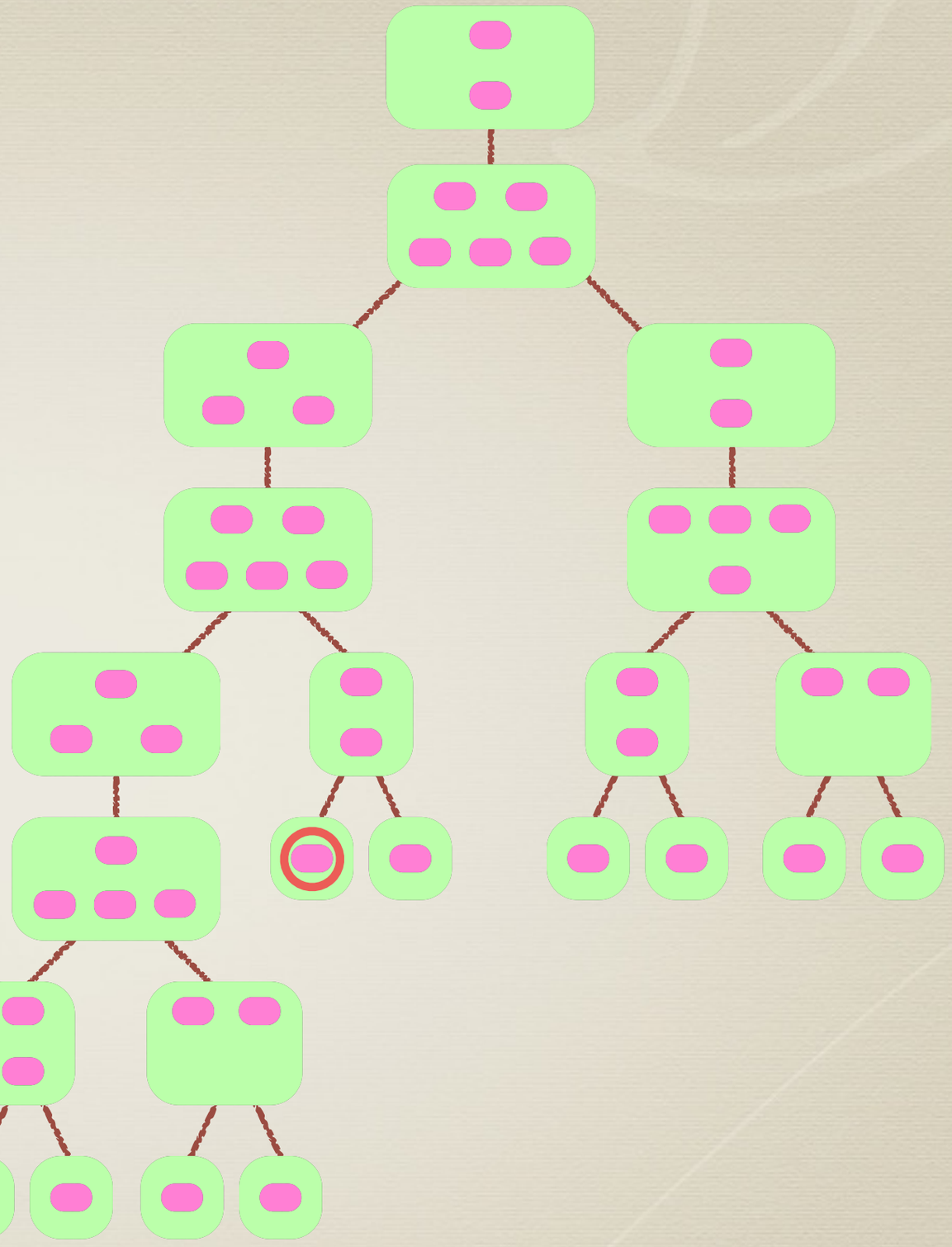




Vertices

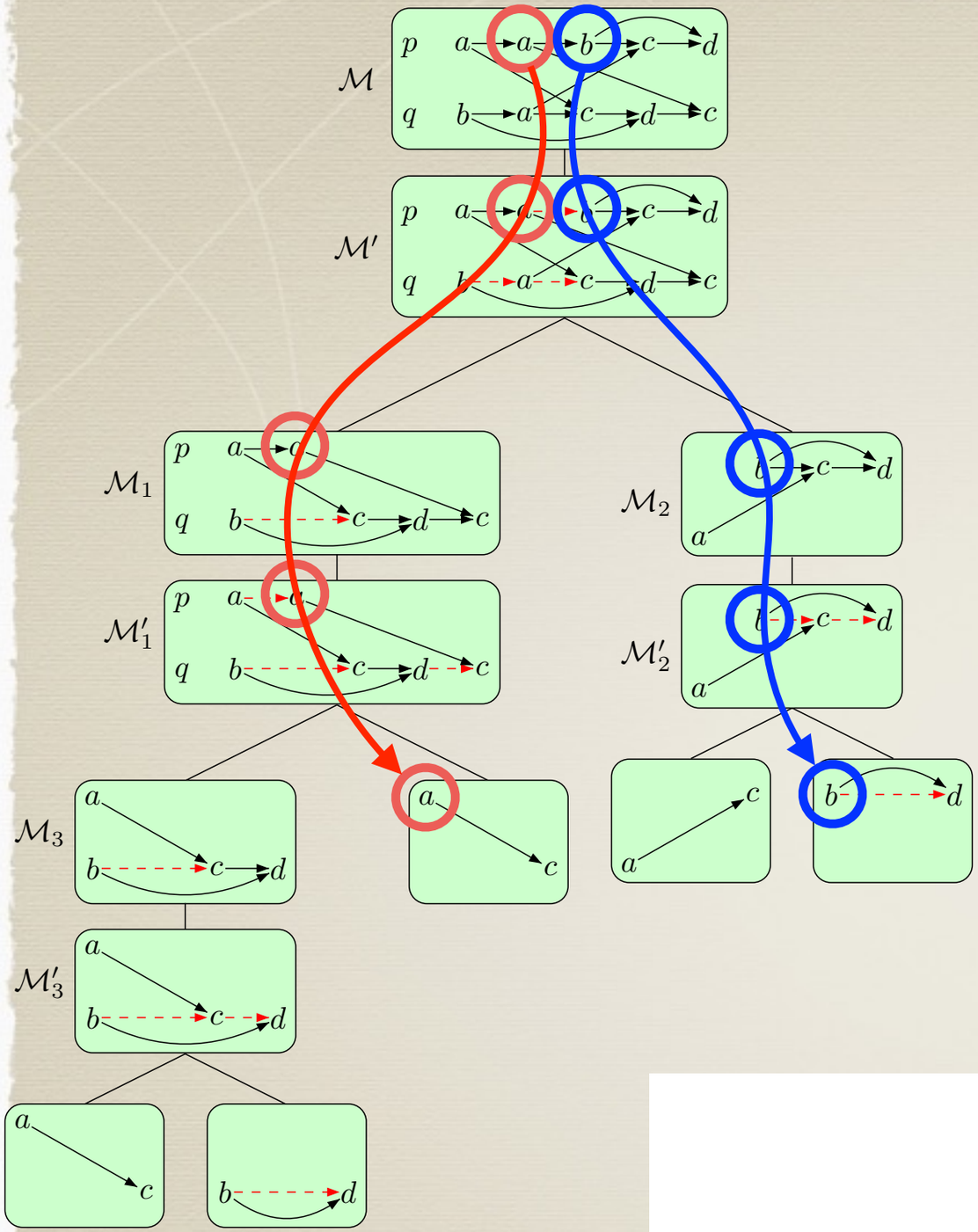
Tree interpretation in



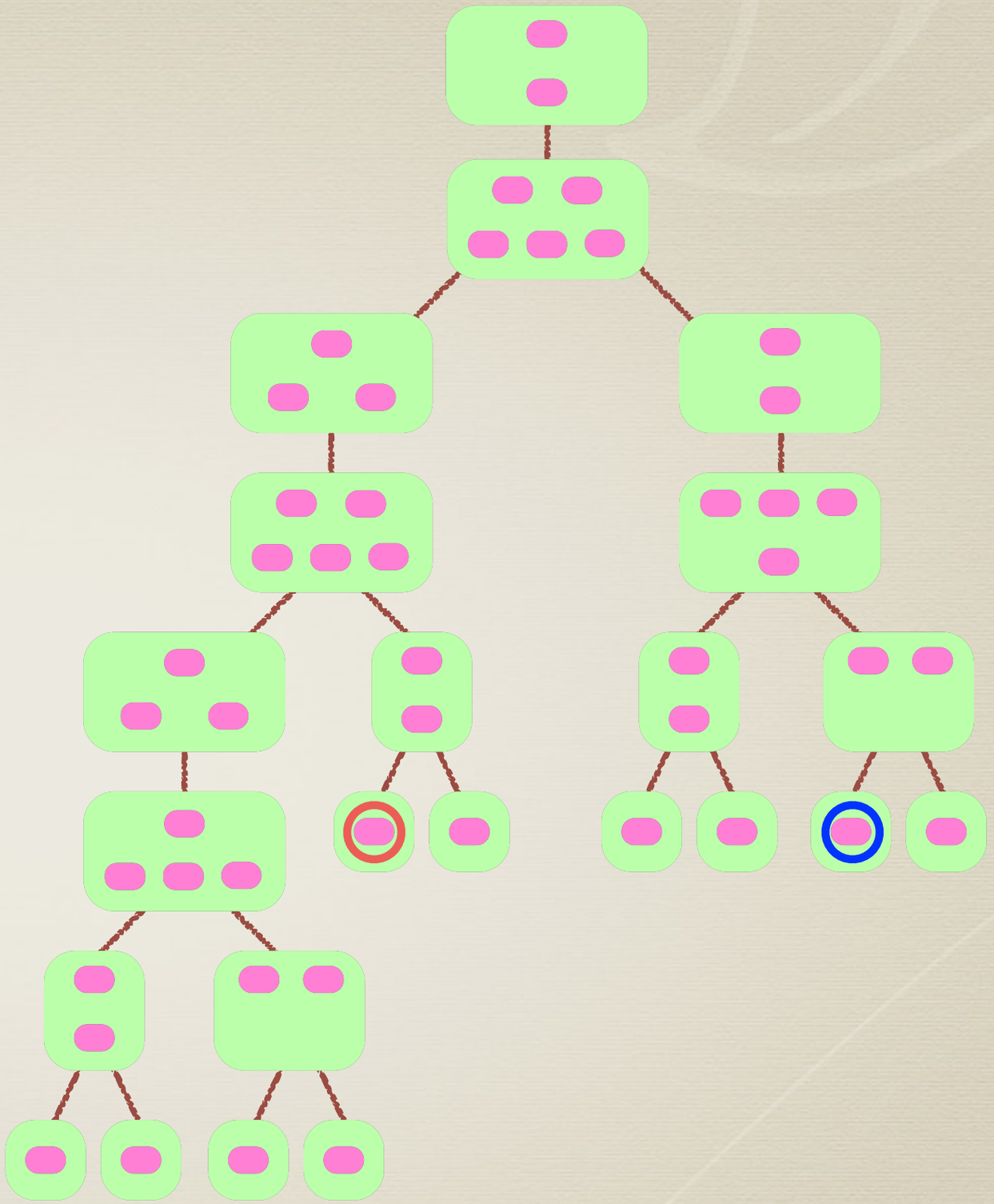


Vertices

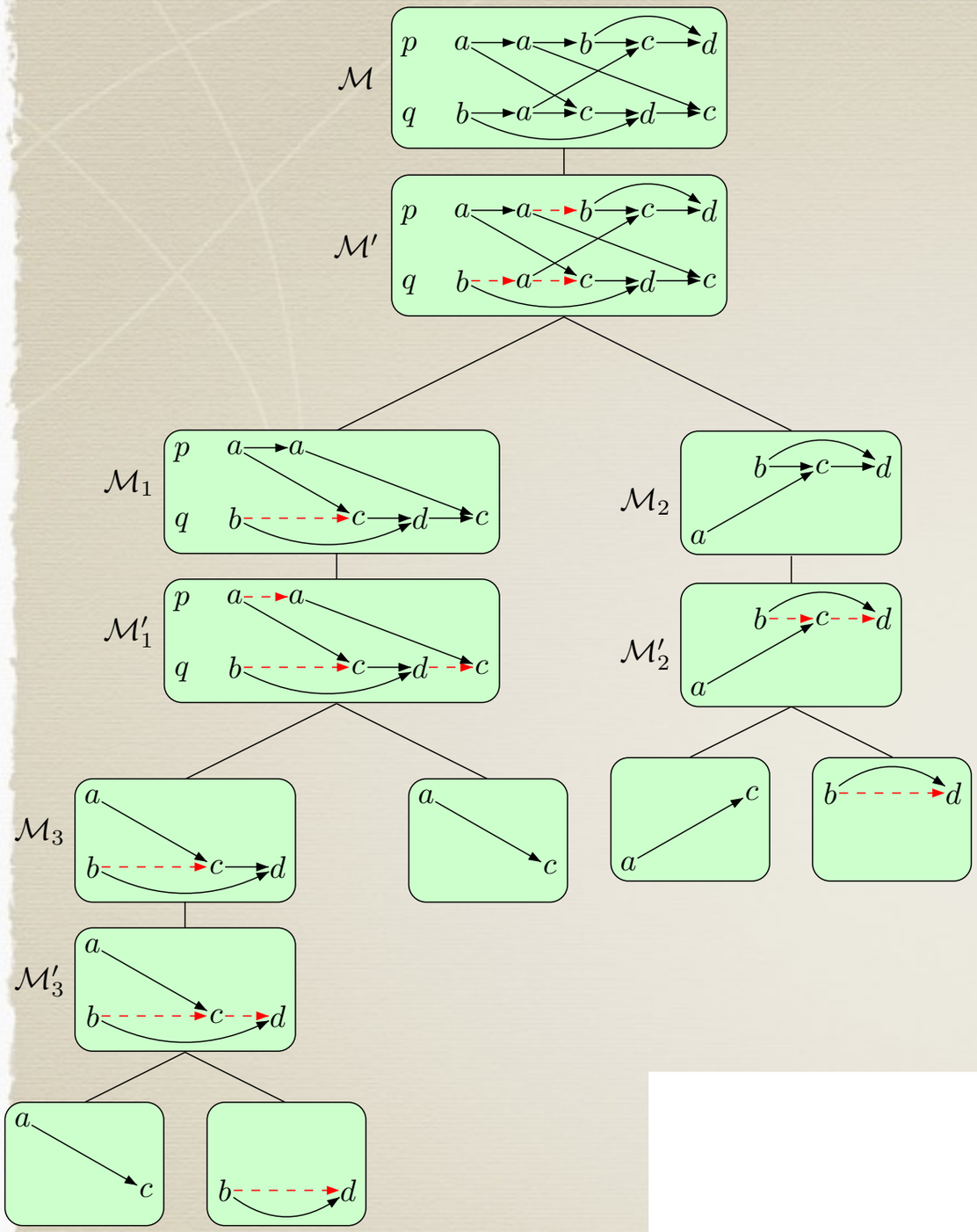
Tree interpretation in

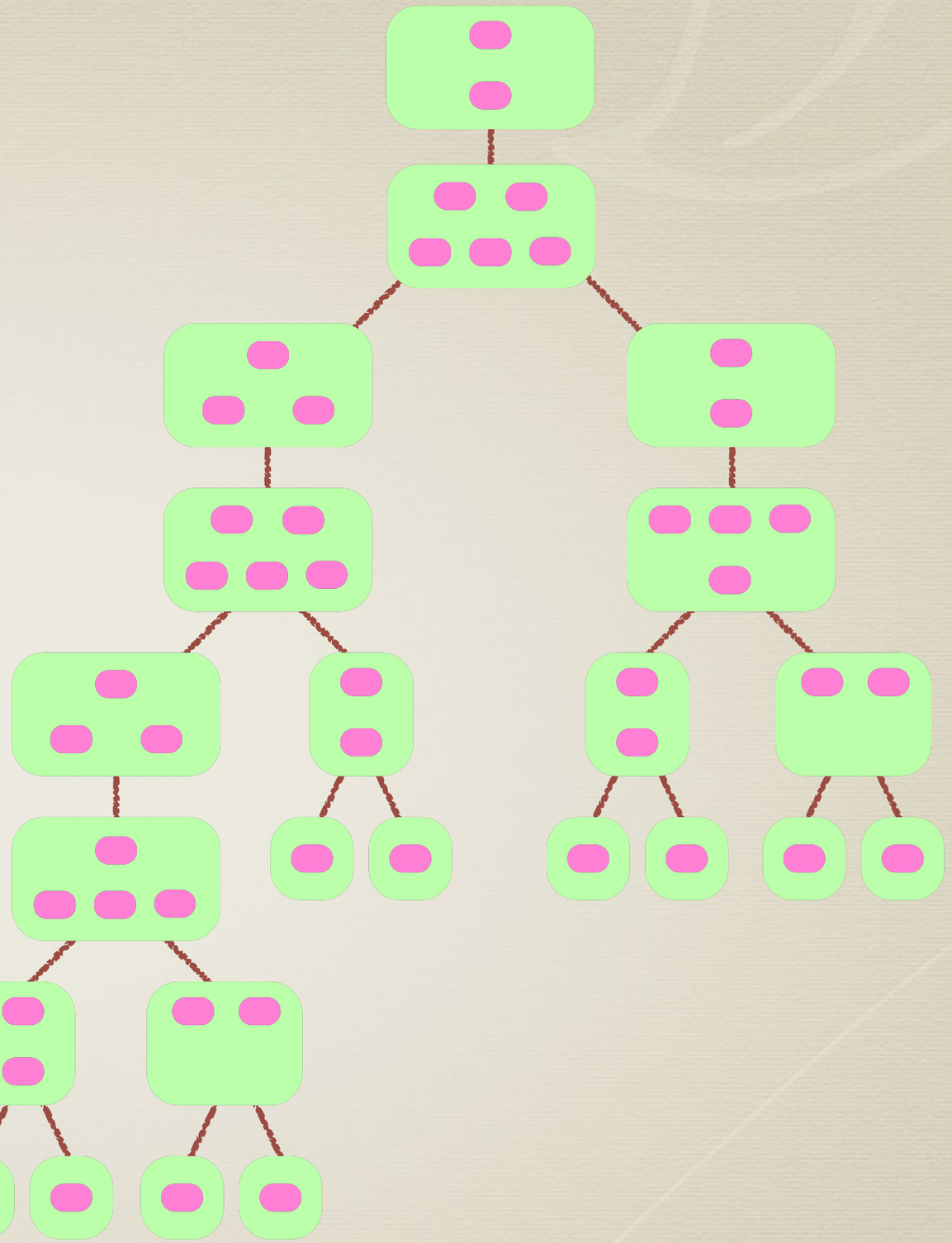


Vertices



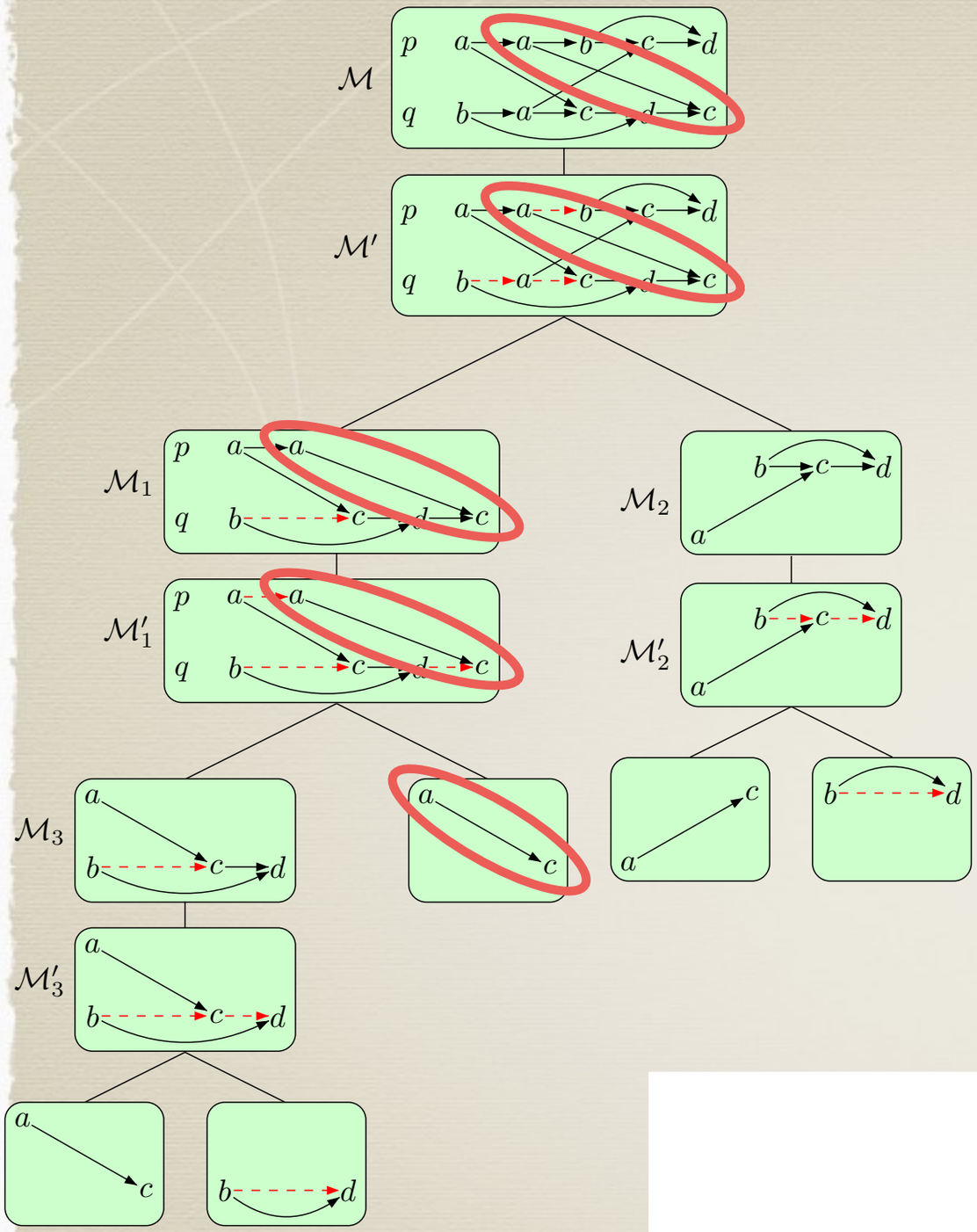
Tree interpretation in

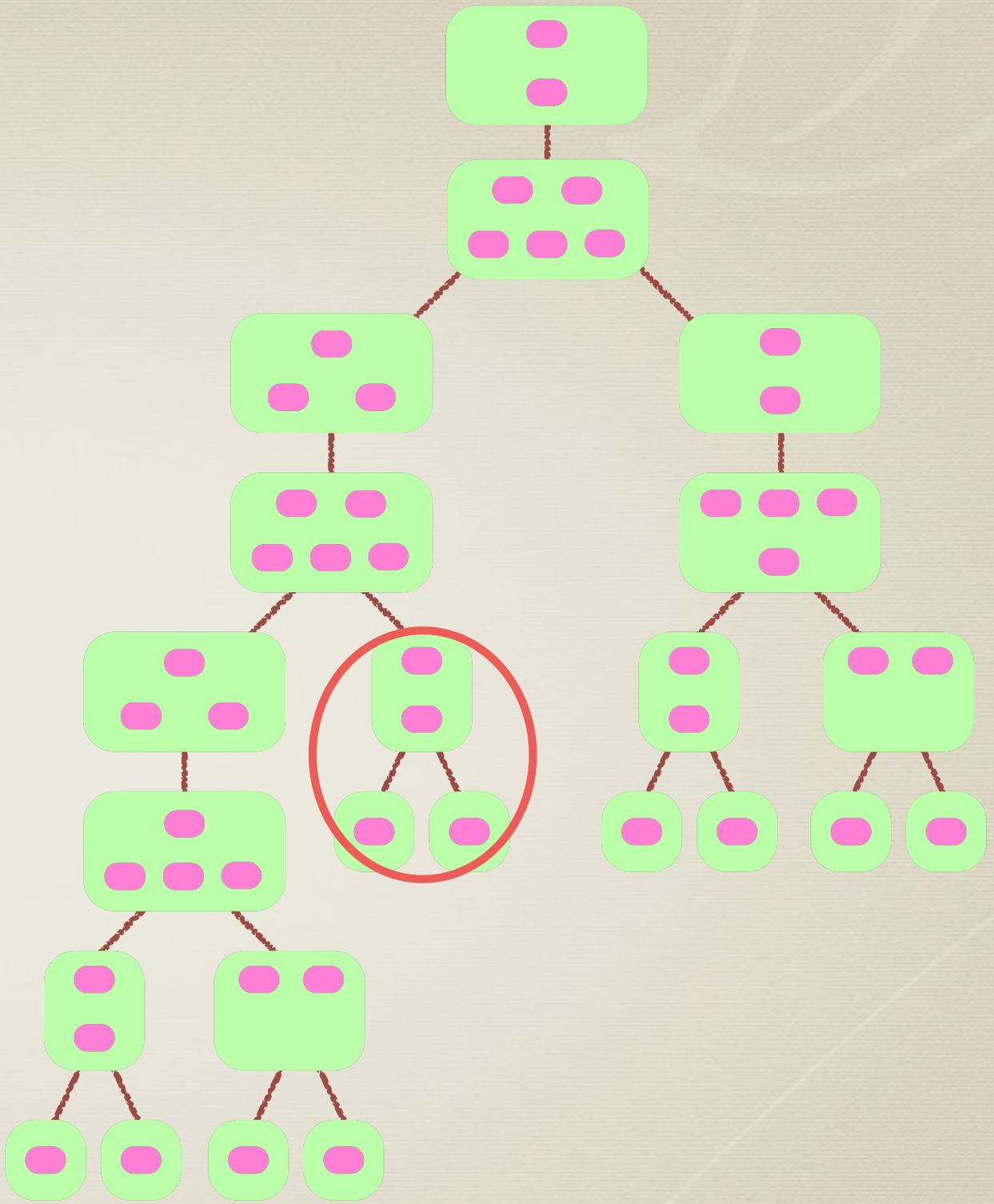




Vertices

Tree interpretation in

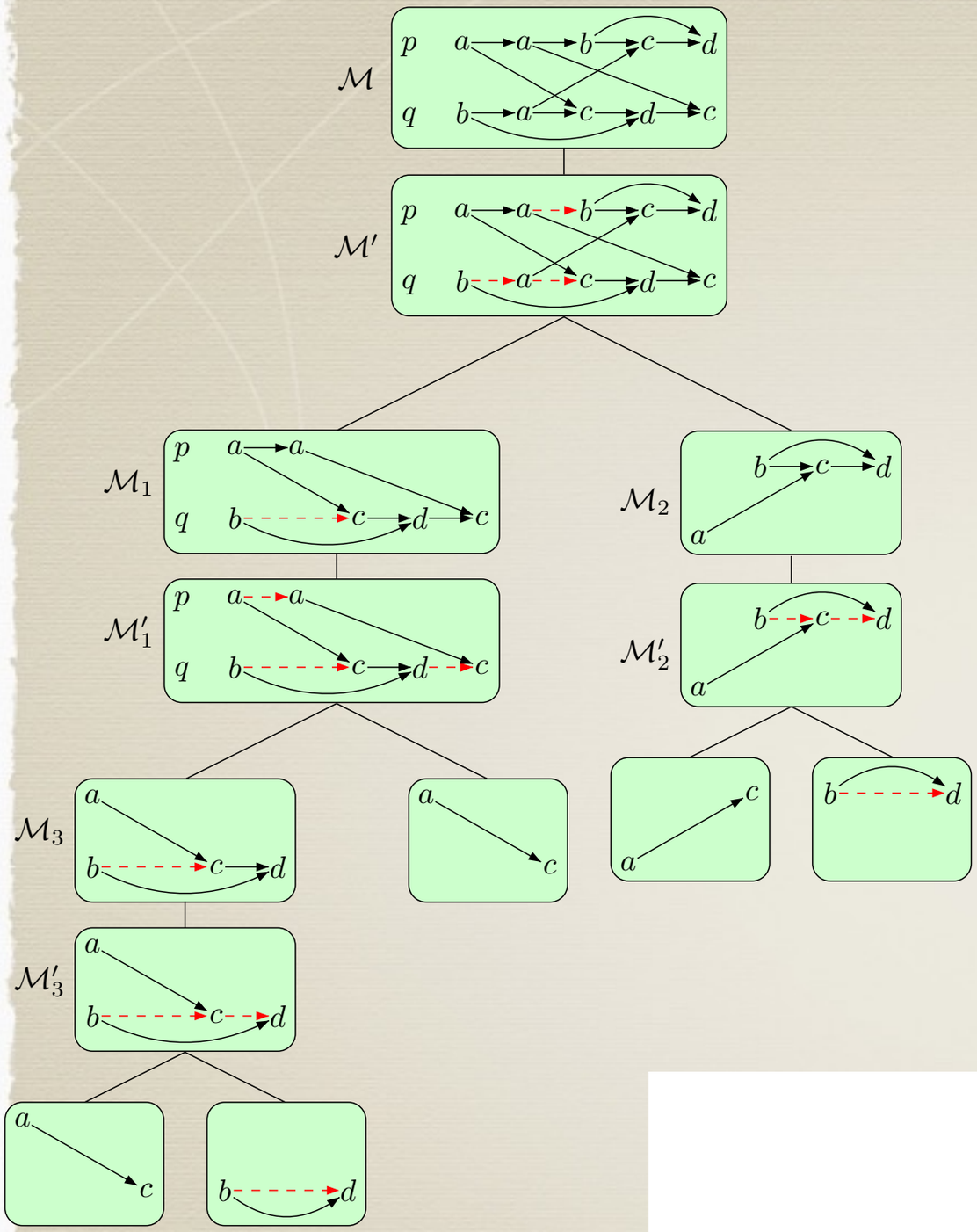


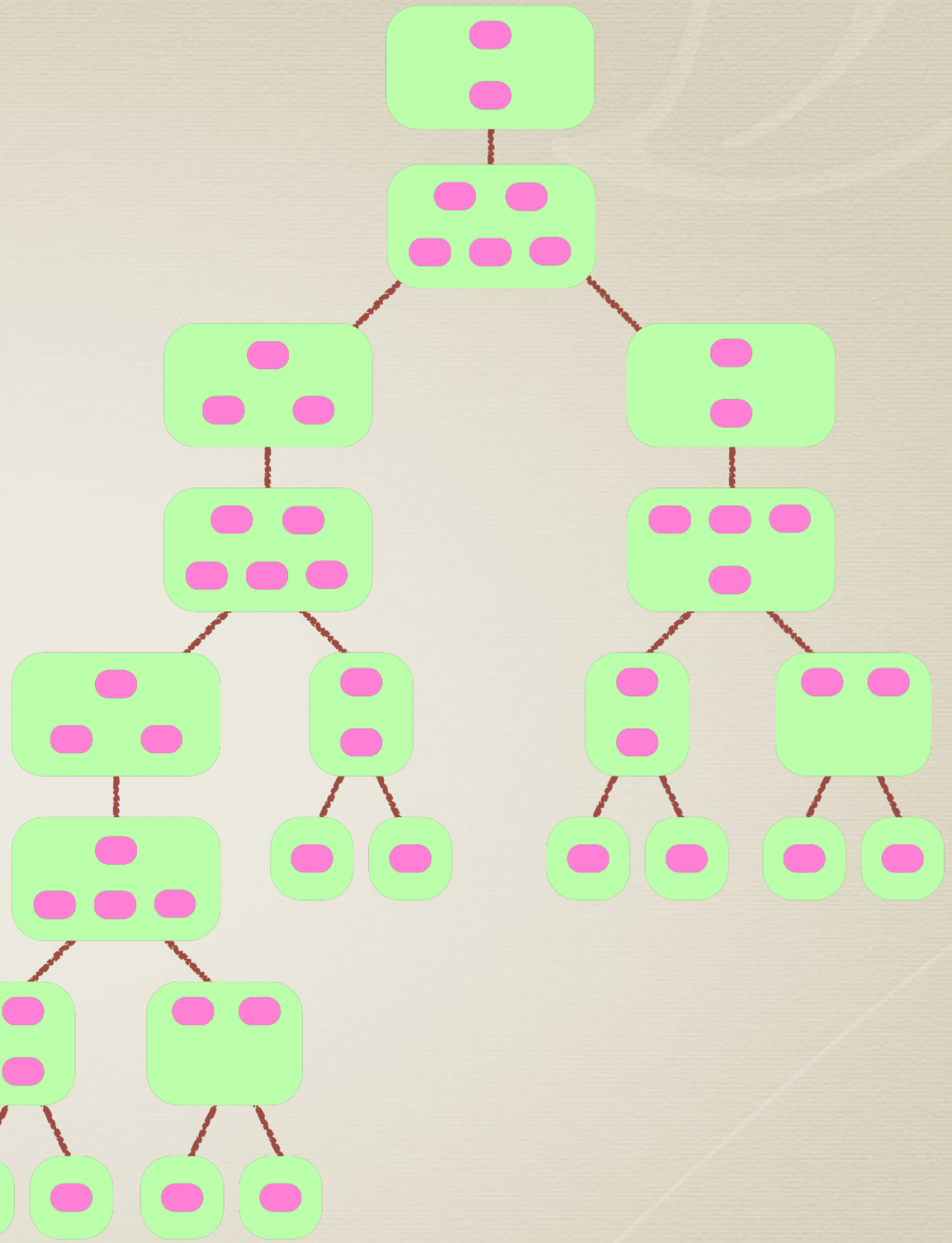


Vertices

Data edges

Tree interpretation in

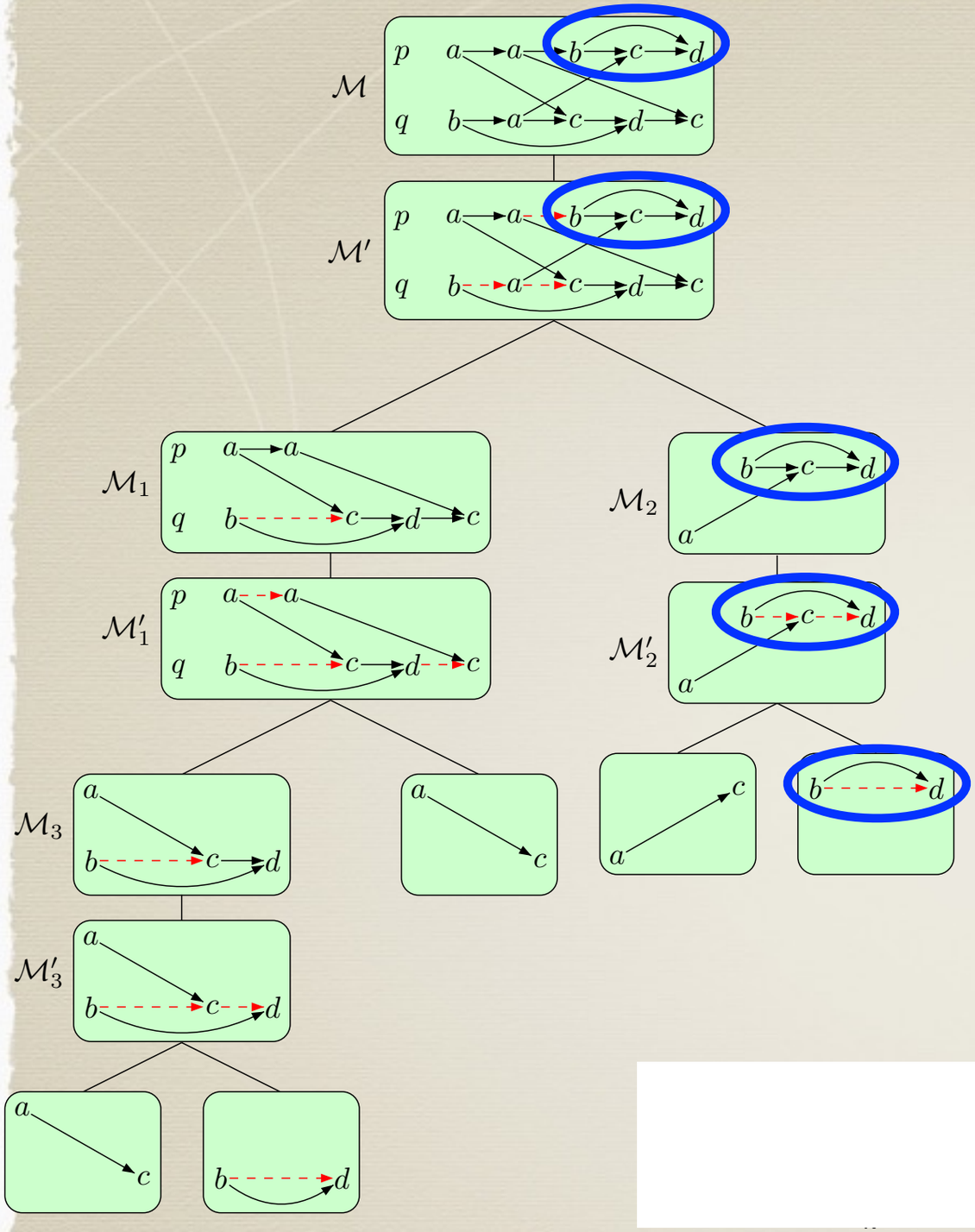


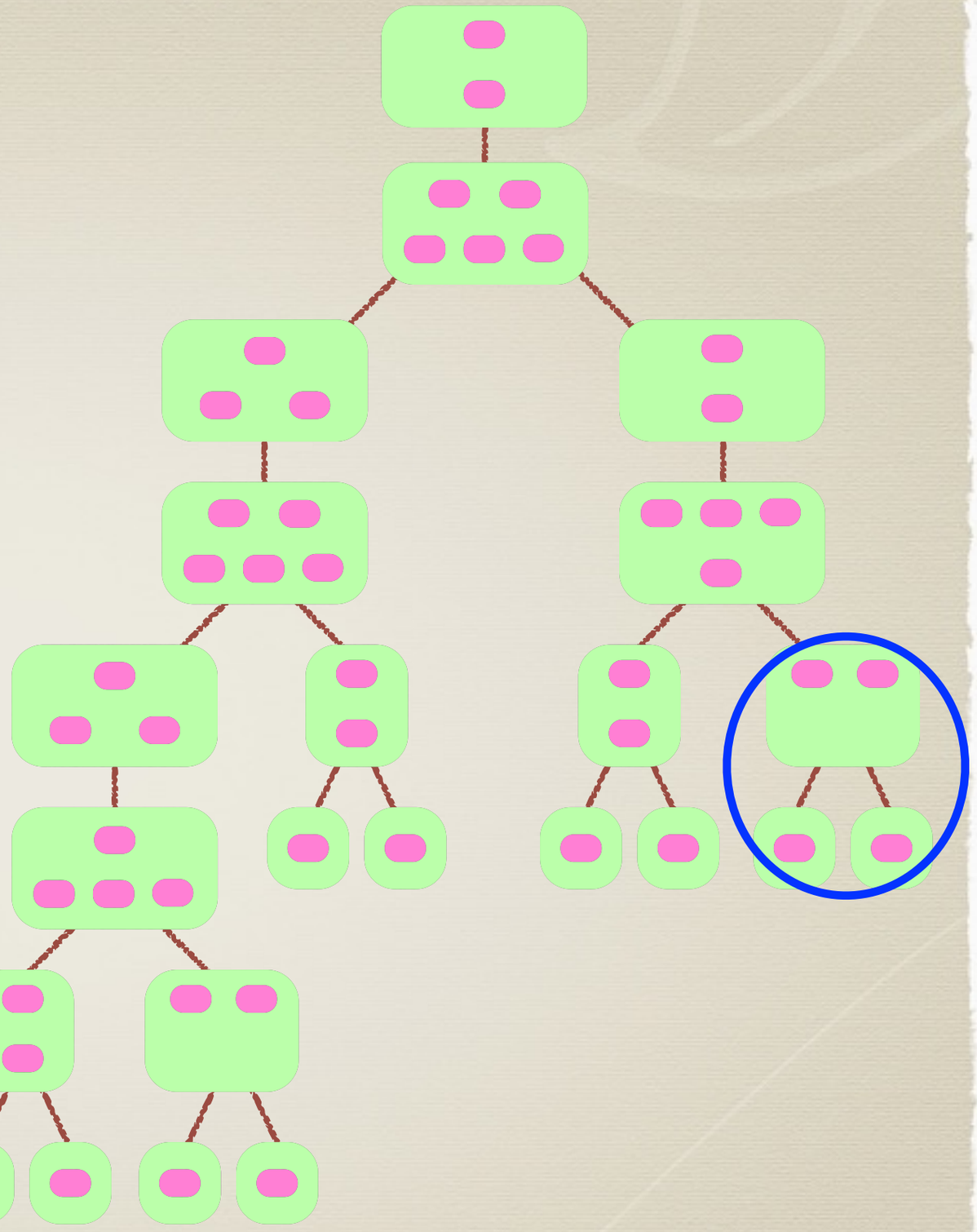


Vertices

Data edges

Tree interpretation in

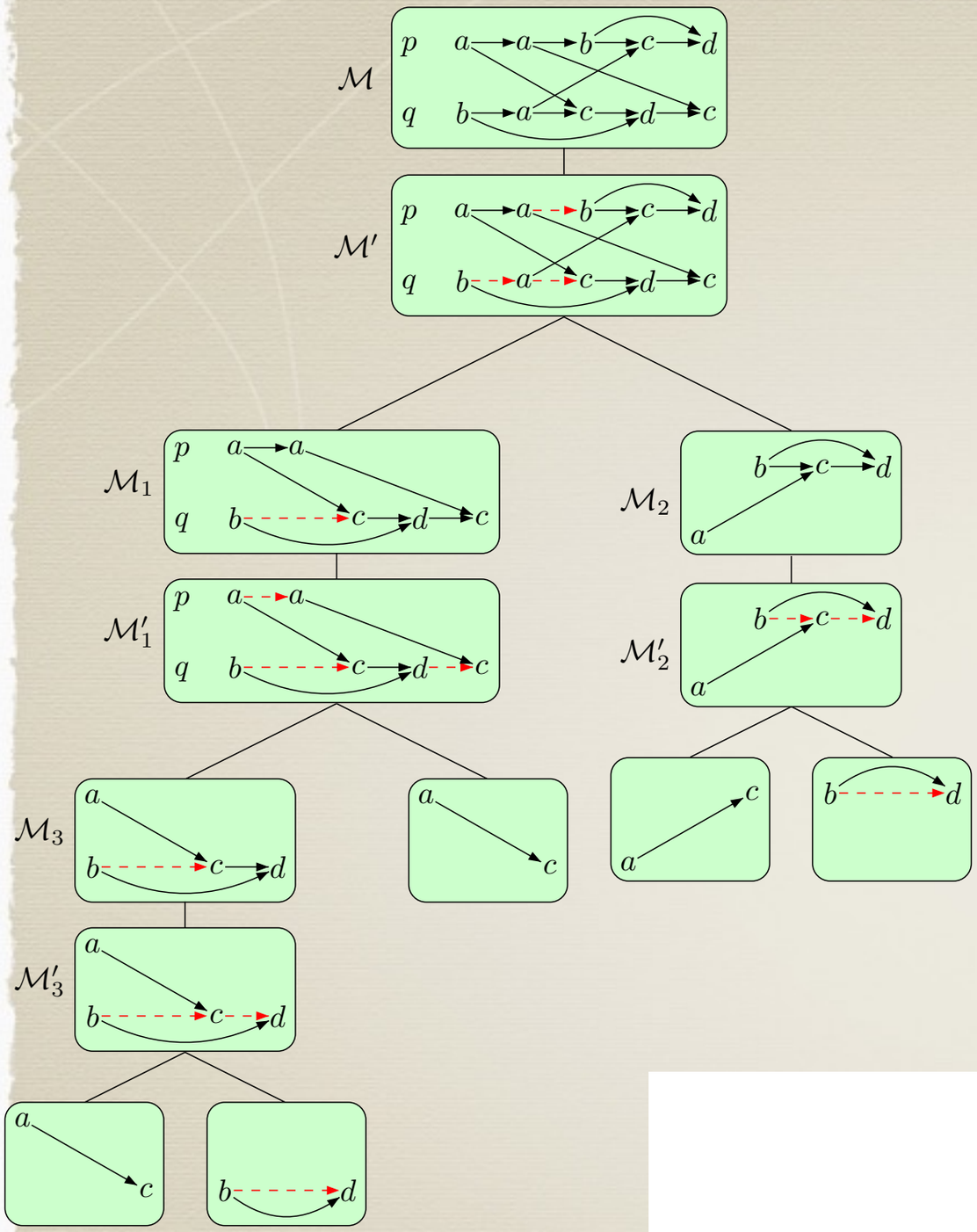


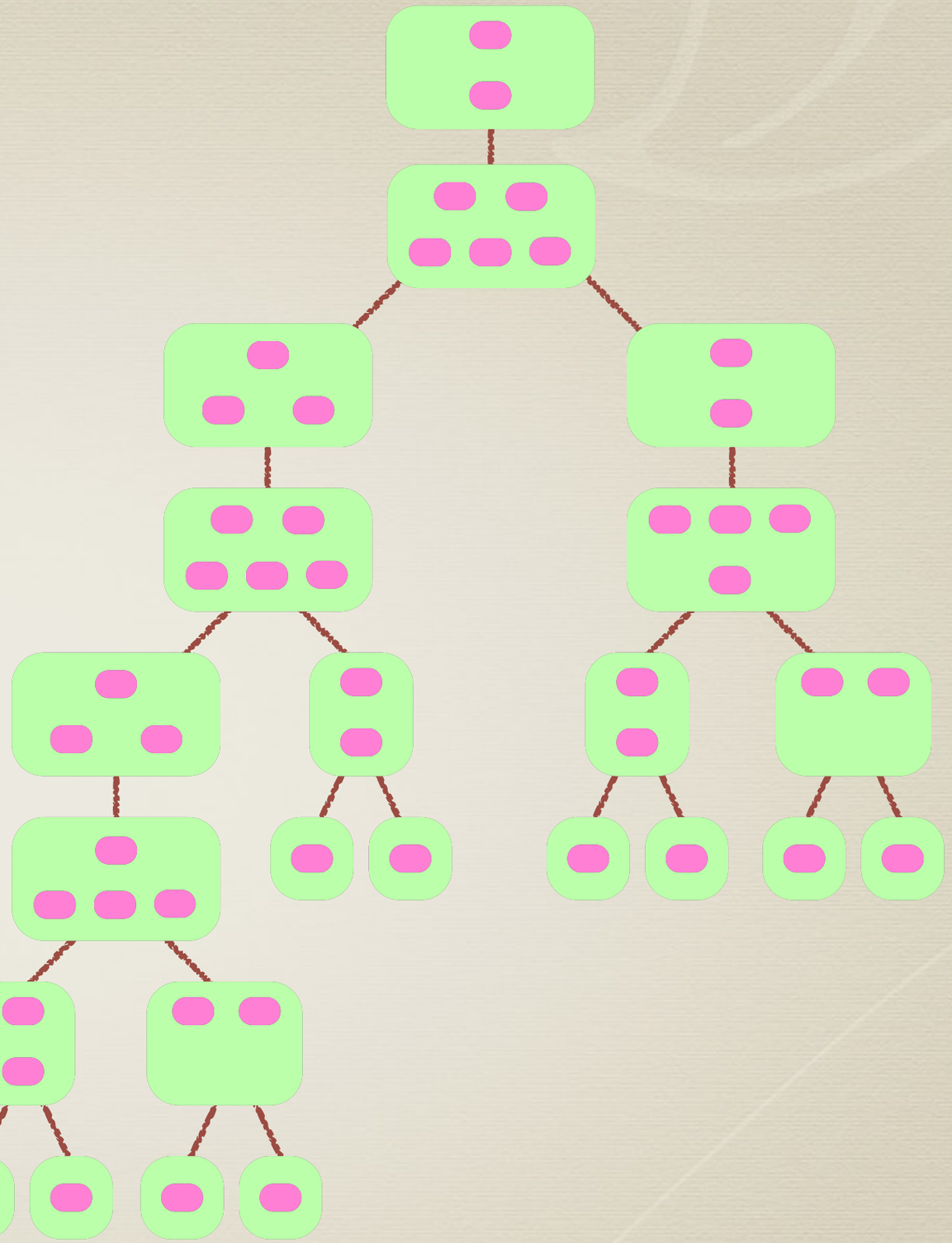


Vertices

Data edges

Tree interpretation in

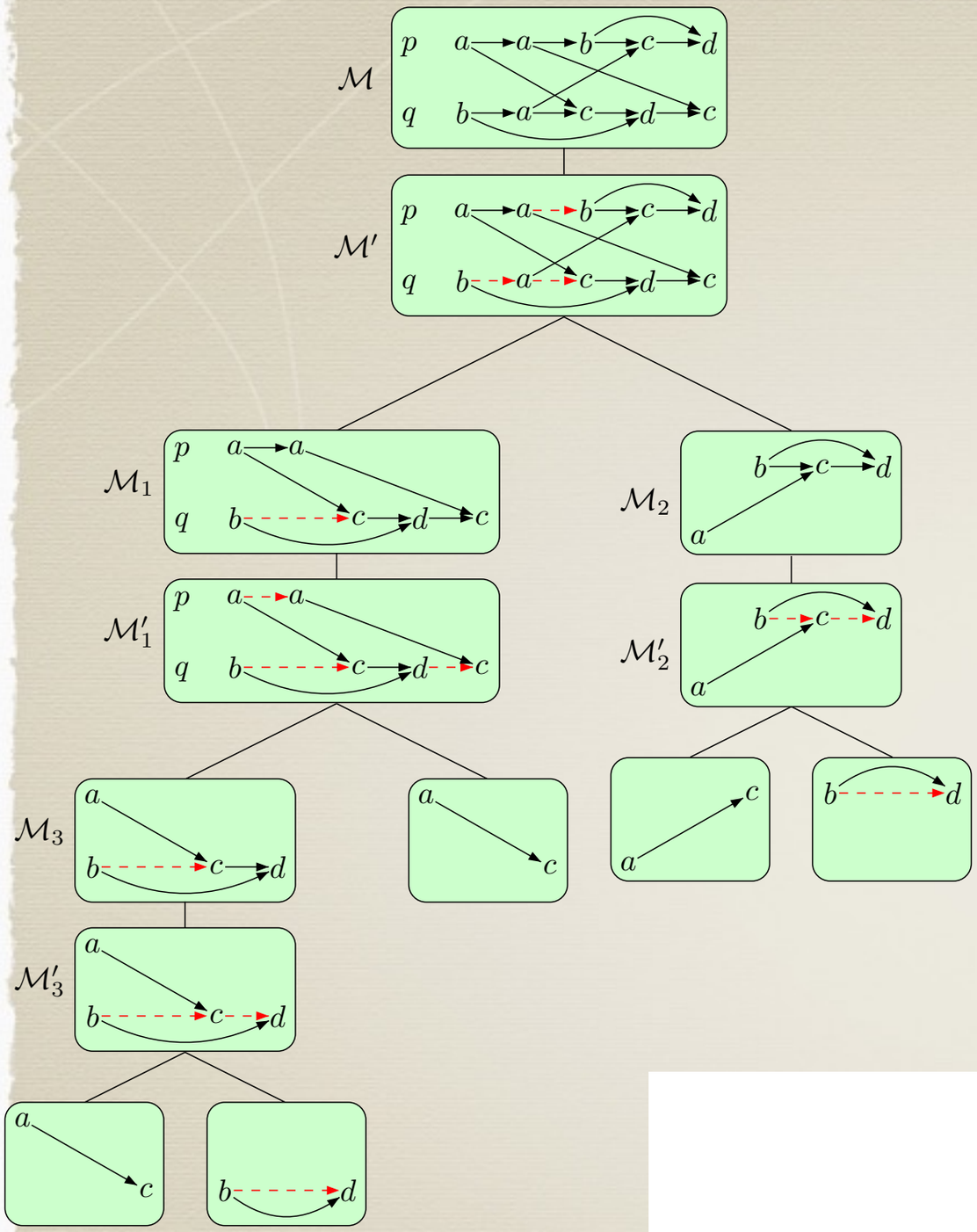


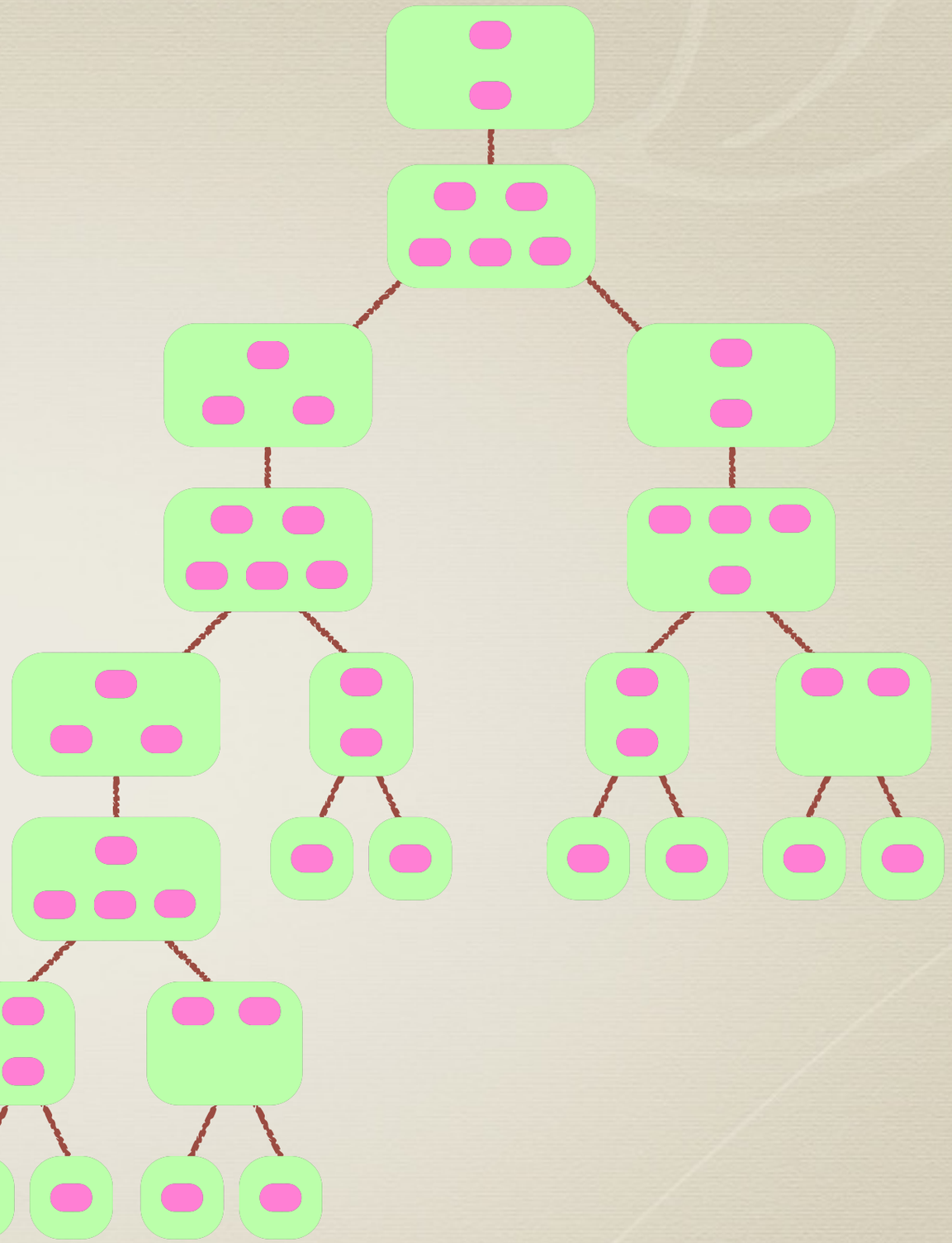


Vertices

Data edges

Tree interpretation in



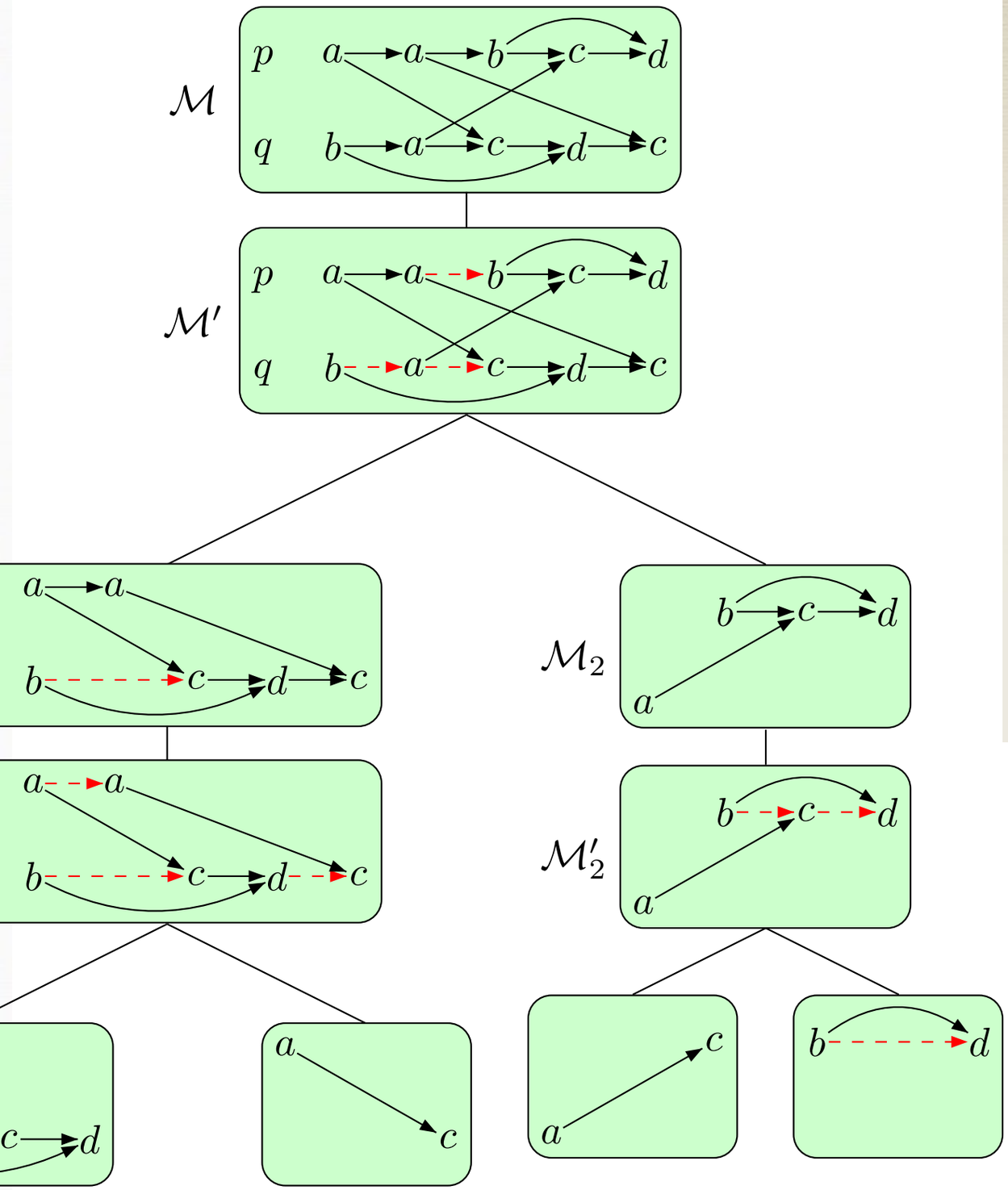


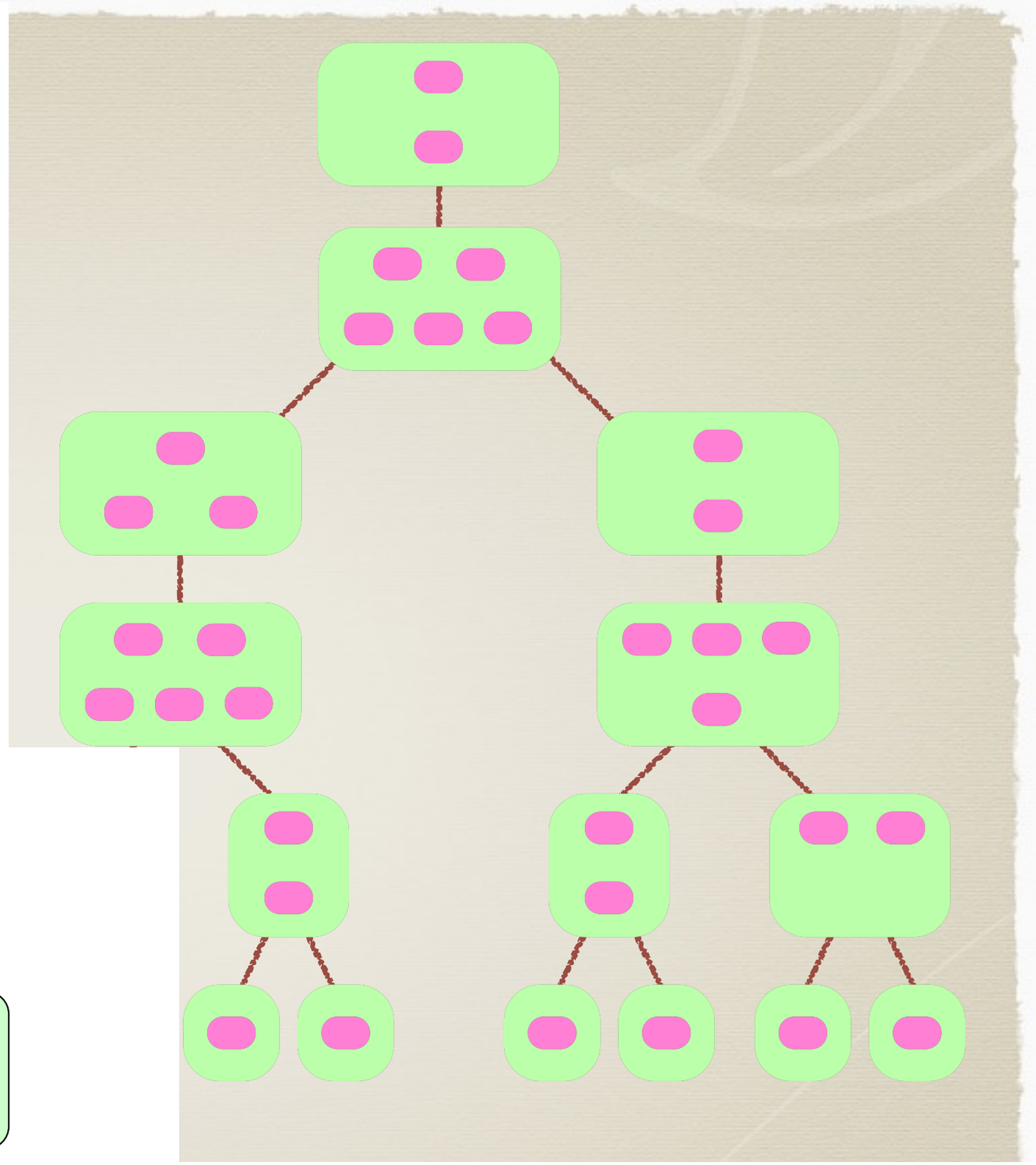
Vertices

Data edges

Process edges

Tree interpretation in





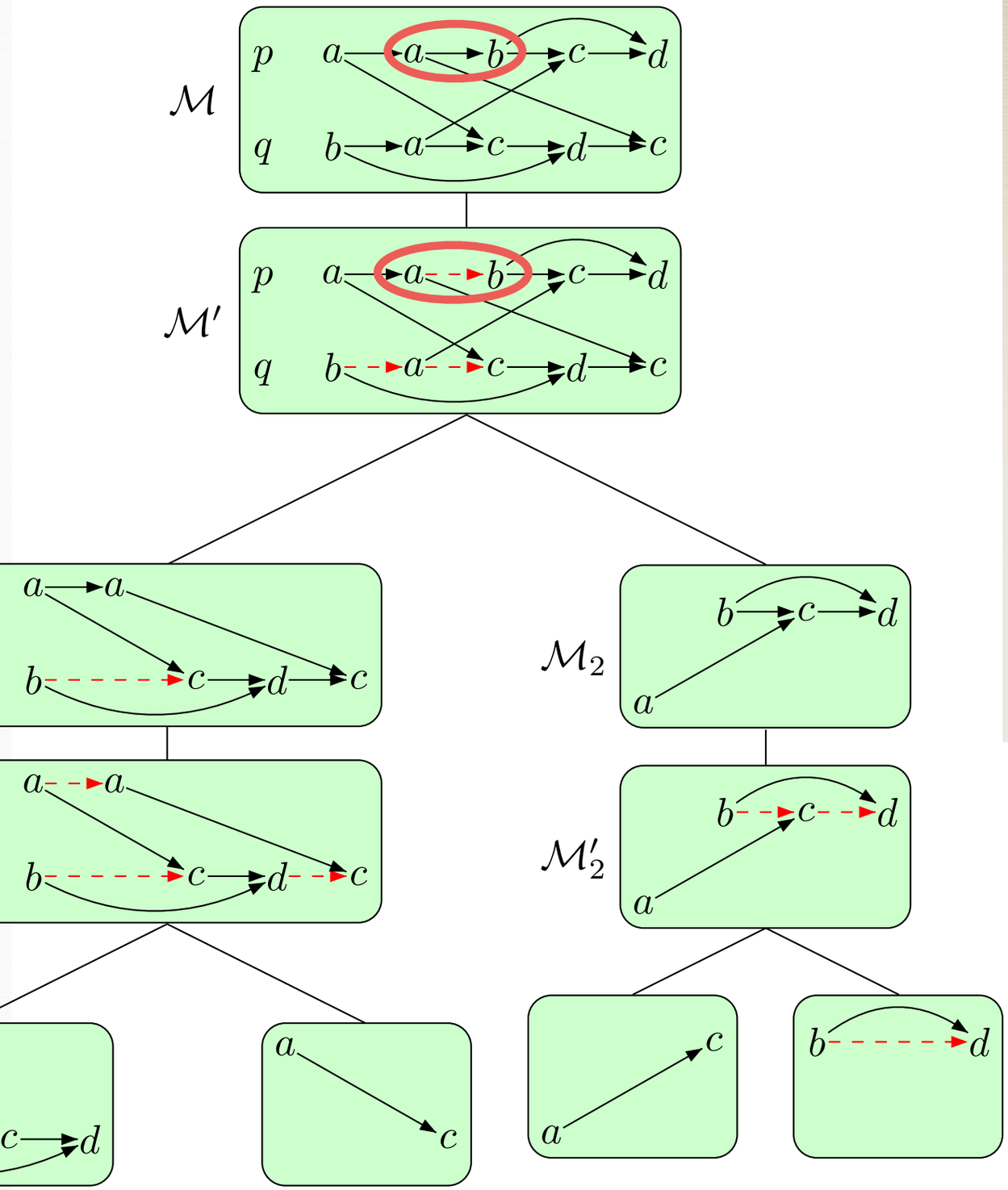
Vertices

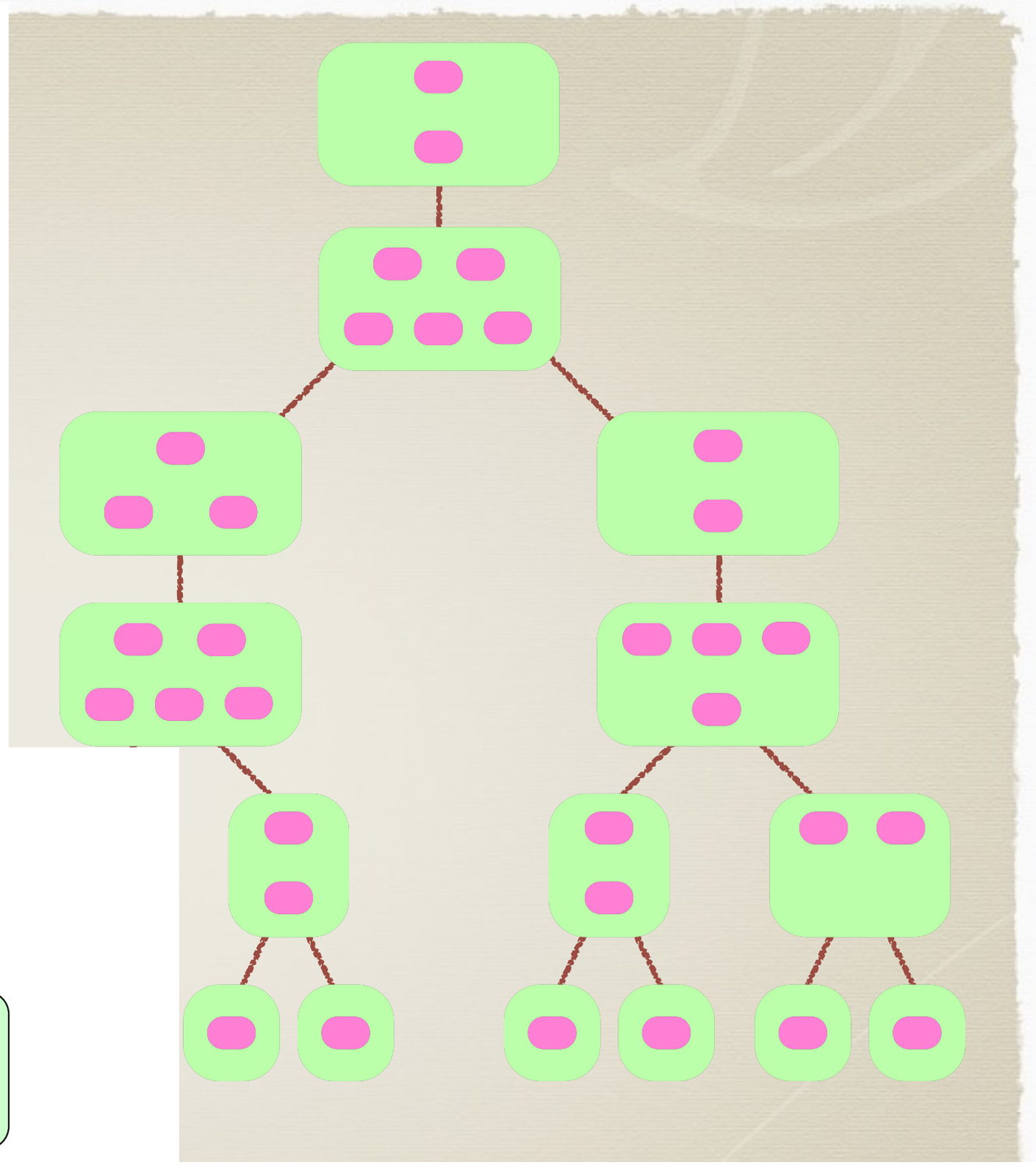
Data edges

Process edges

Tree interpretation in

Abstract Tree Decomposition





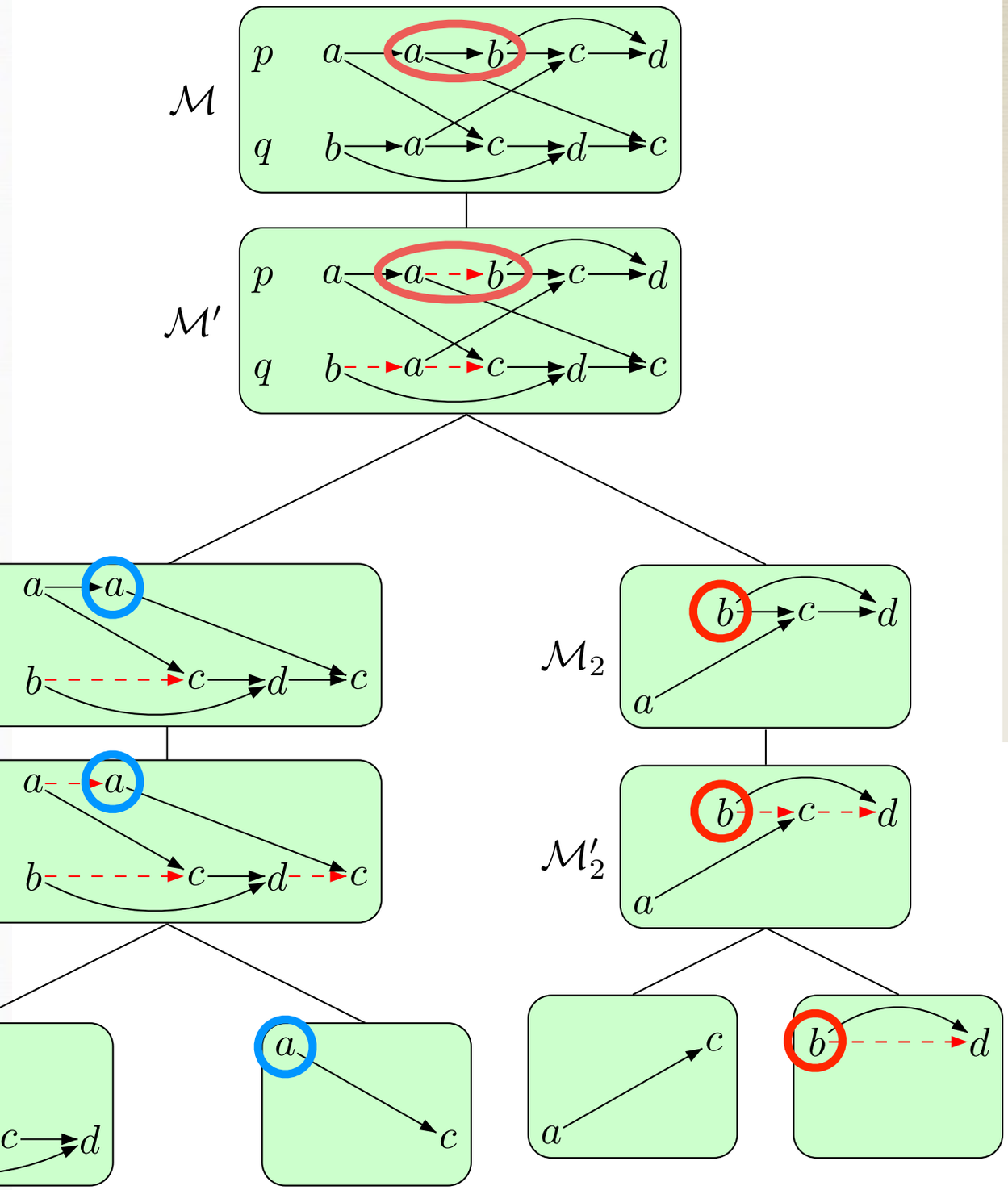
Vertices

Data edges

Process edges

Tree interpretation in

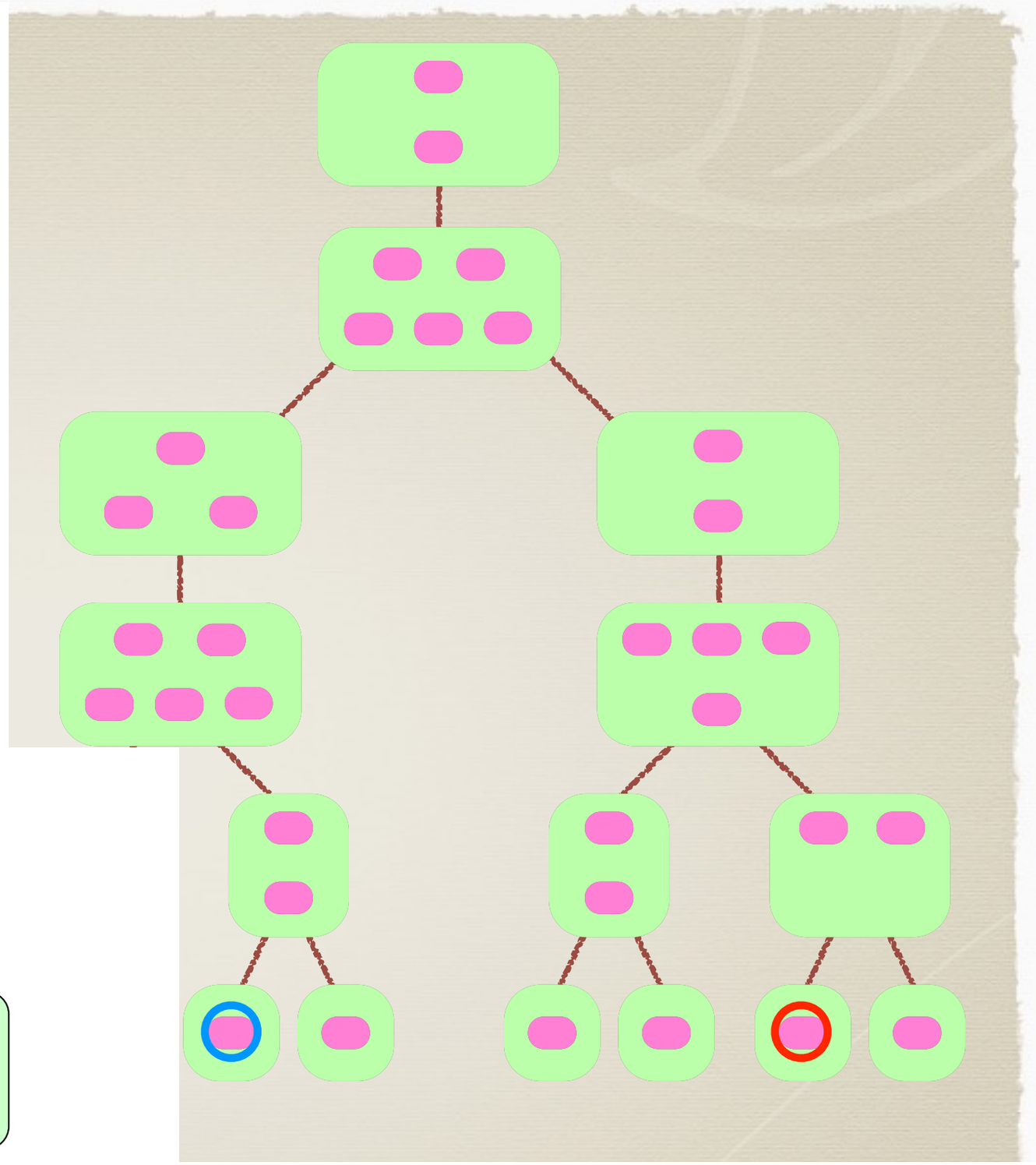
Abstract Tree Decomposition



Vertices

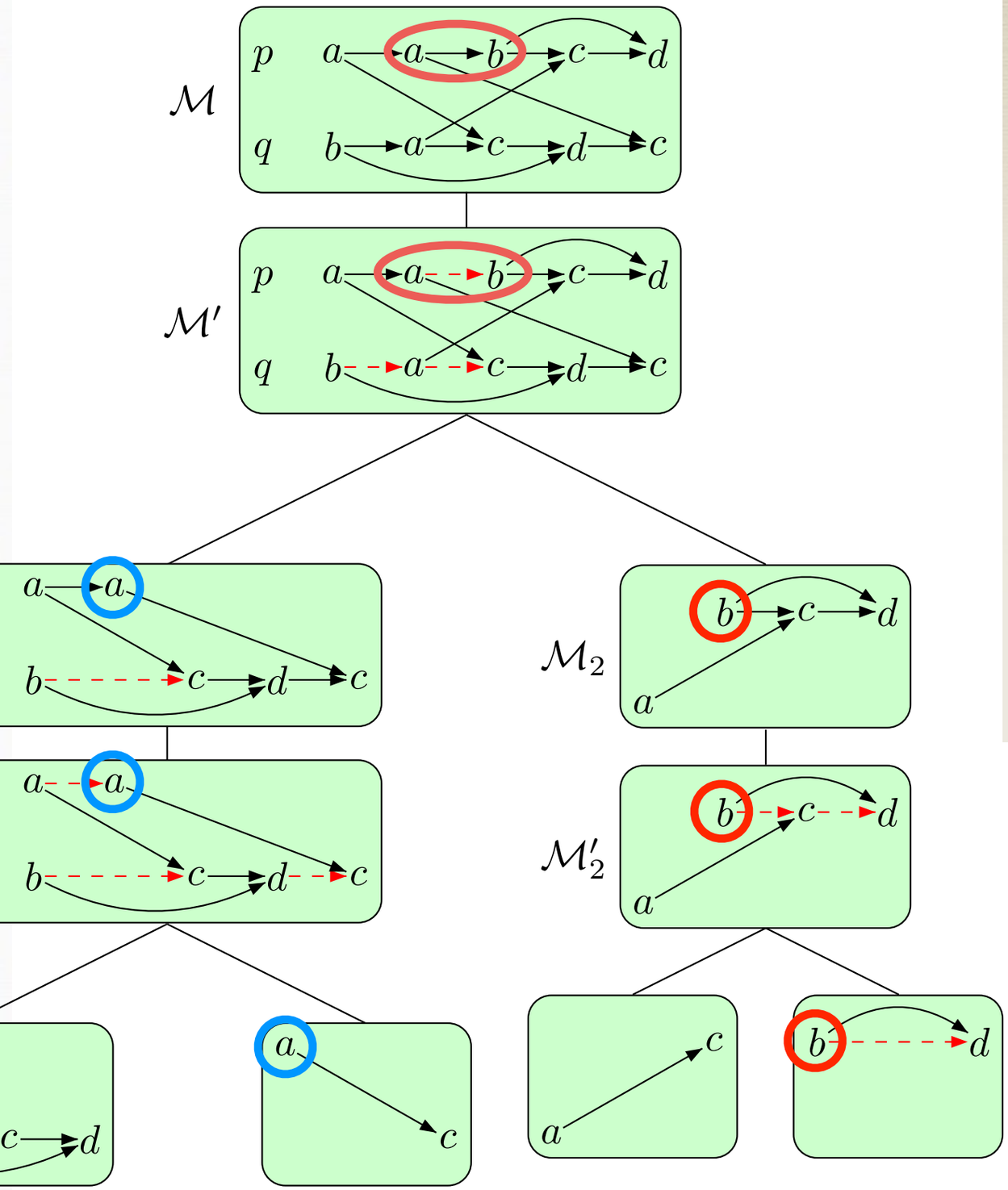
Data edges

Process edges



Tree interpretation in

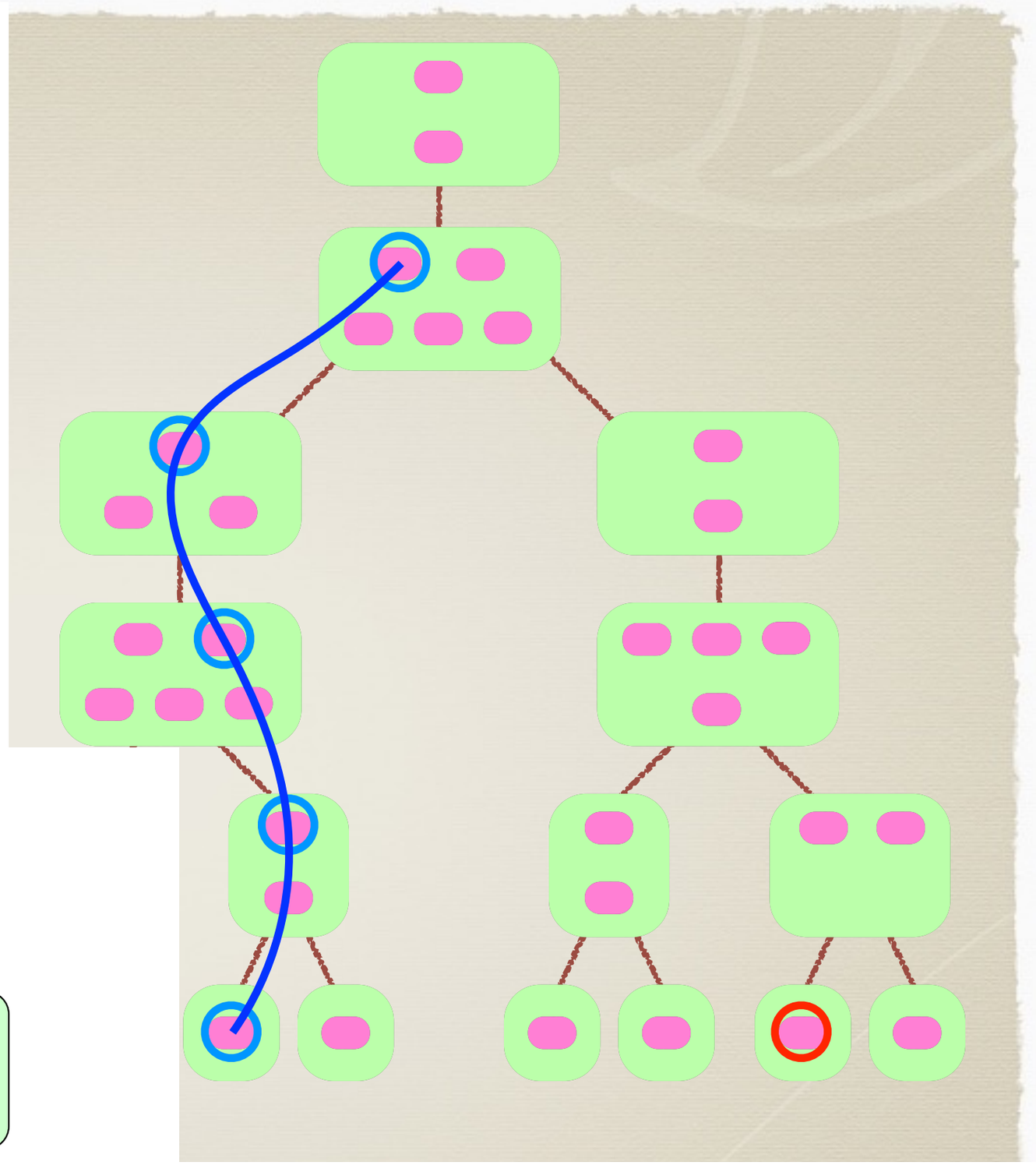
Abstract Tree Decomposition



Vertices

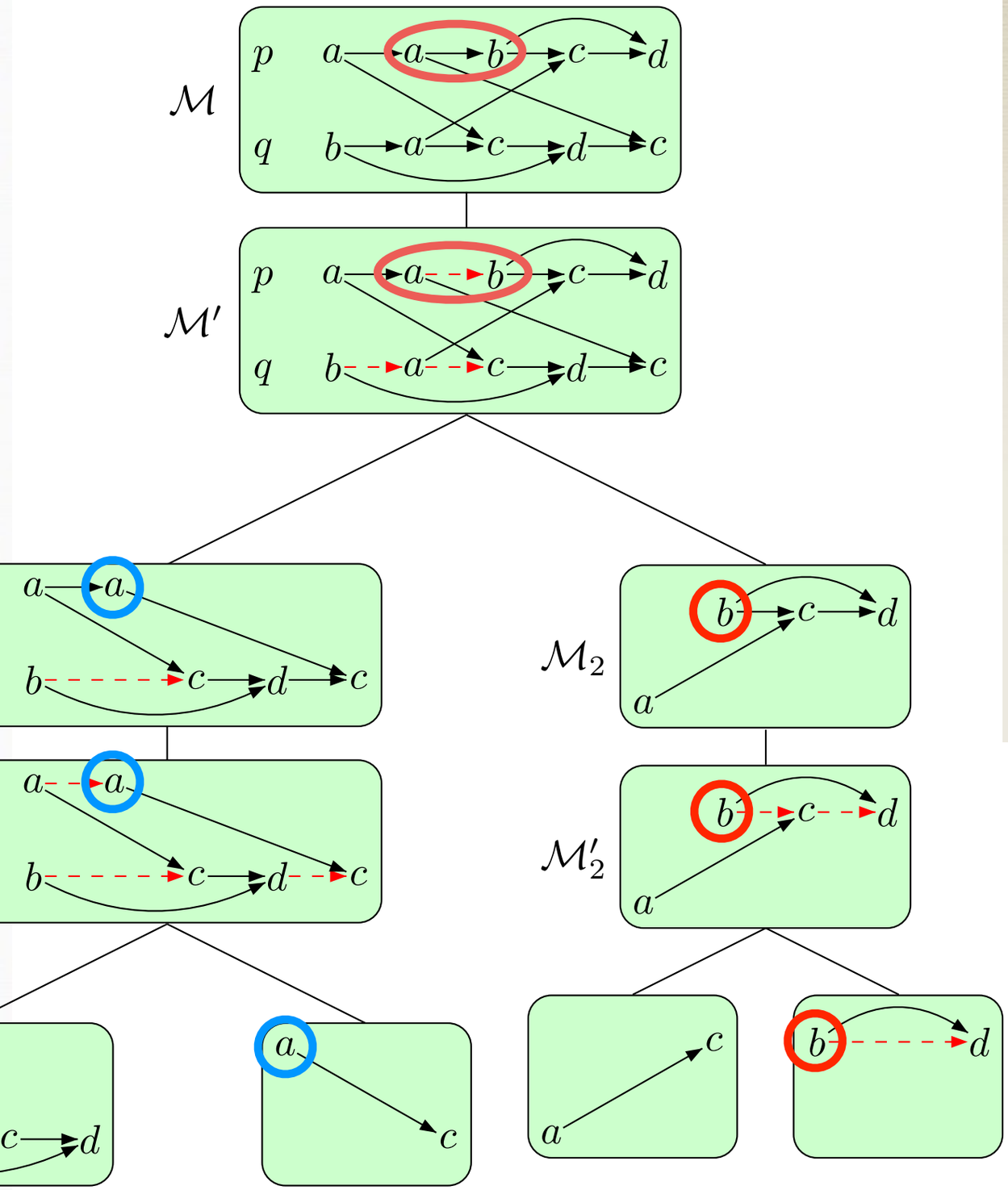
Data edges

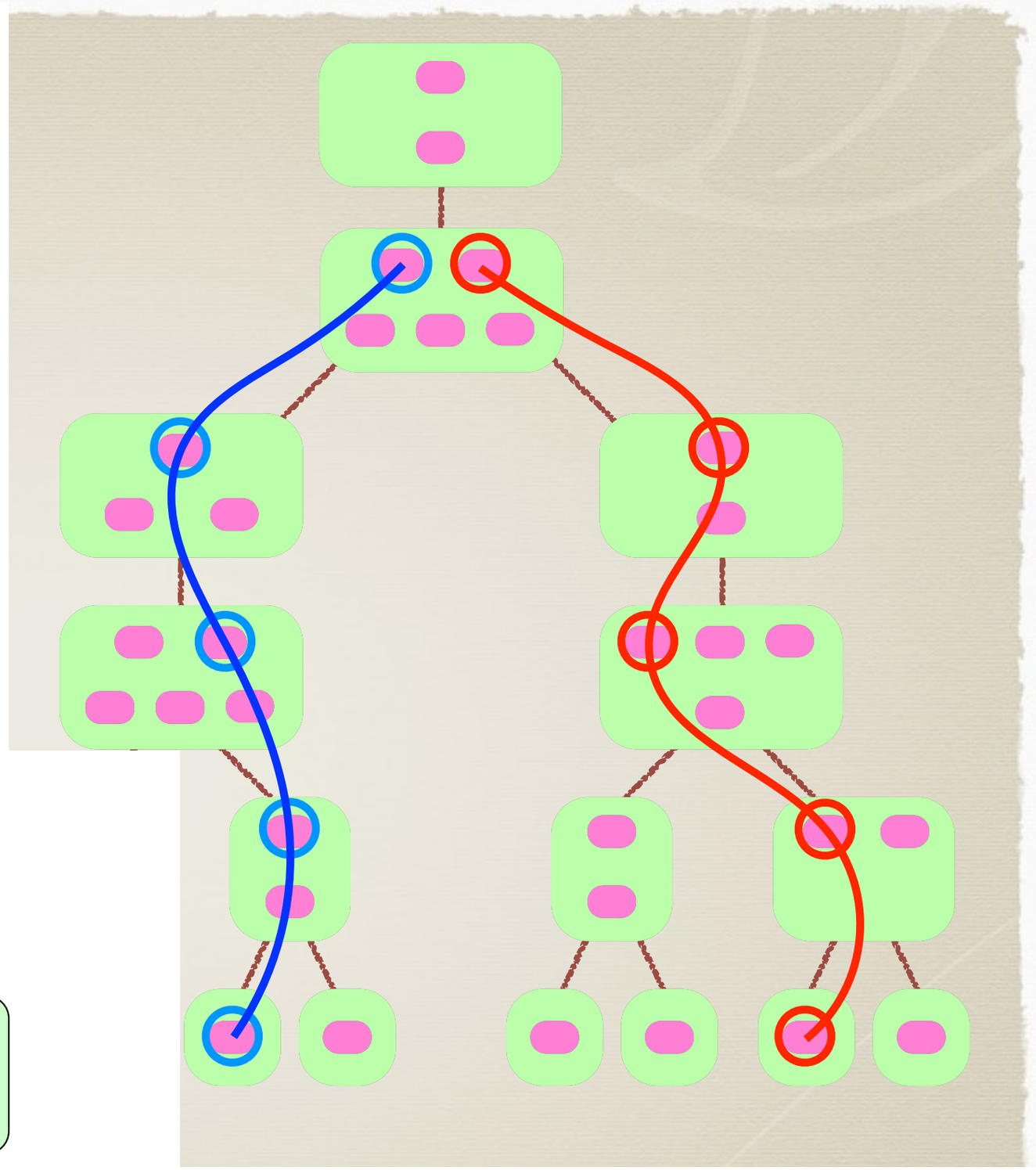
Process edges



Tree interpretation in

Abstract Tree Decomposition





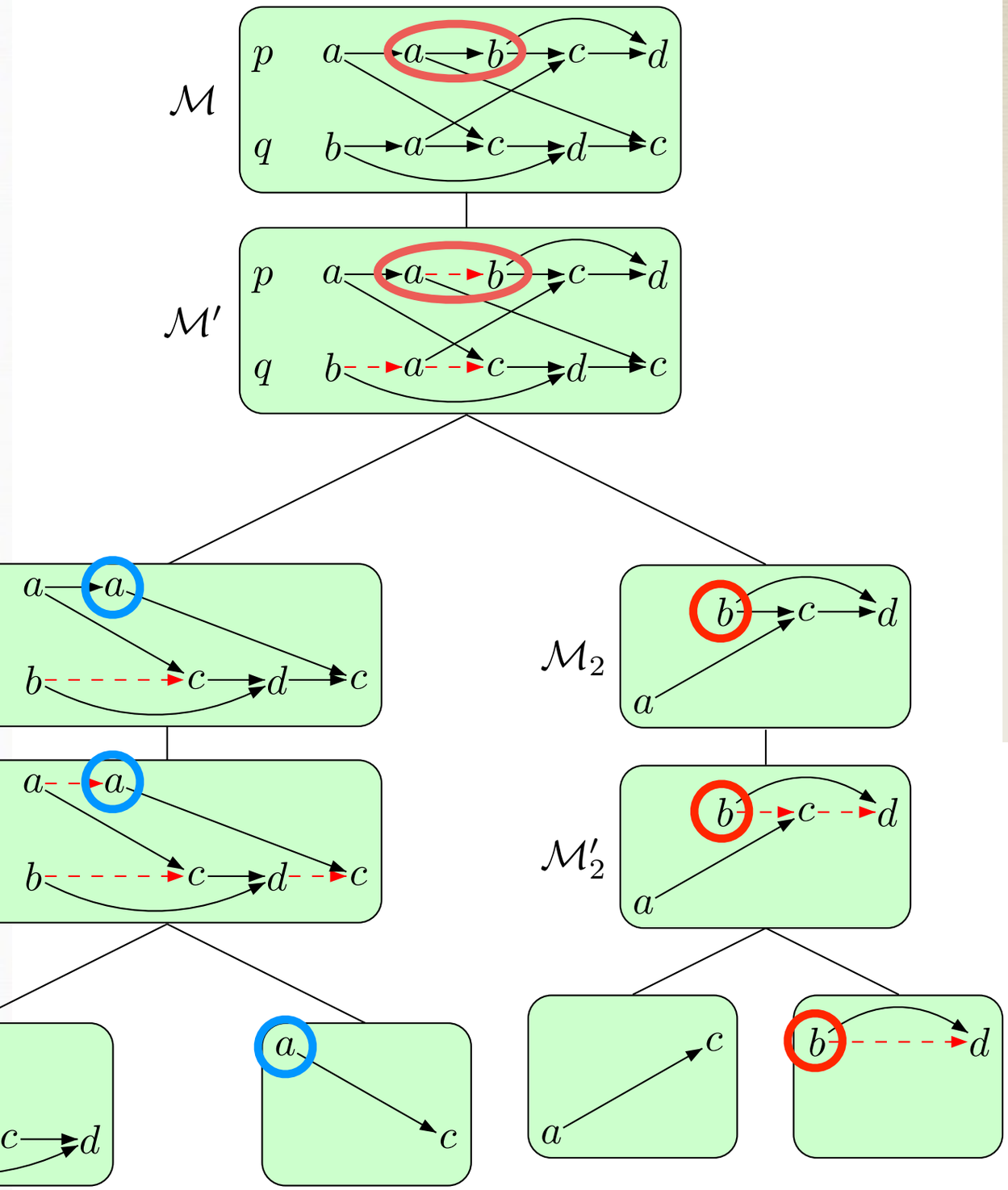
Vertices

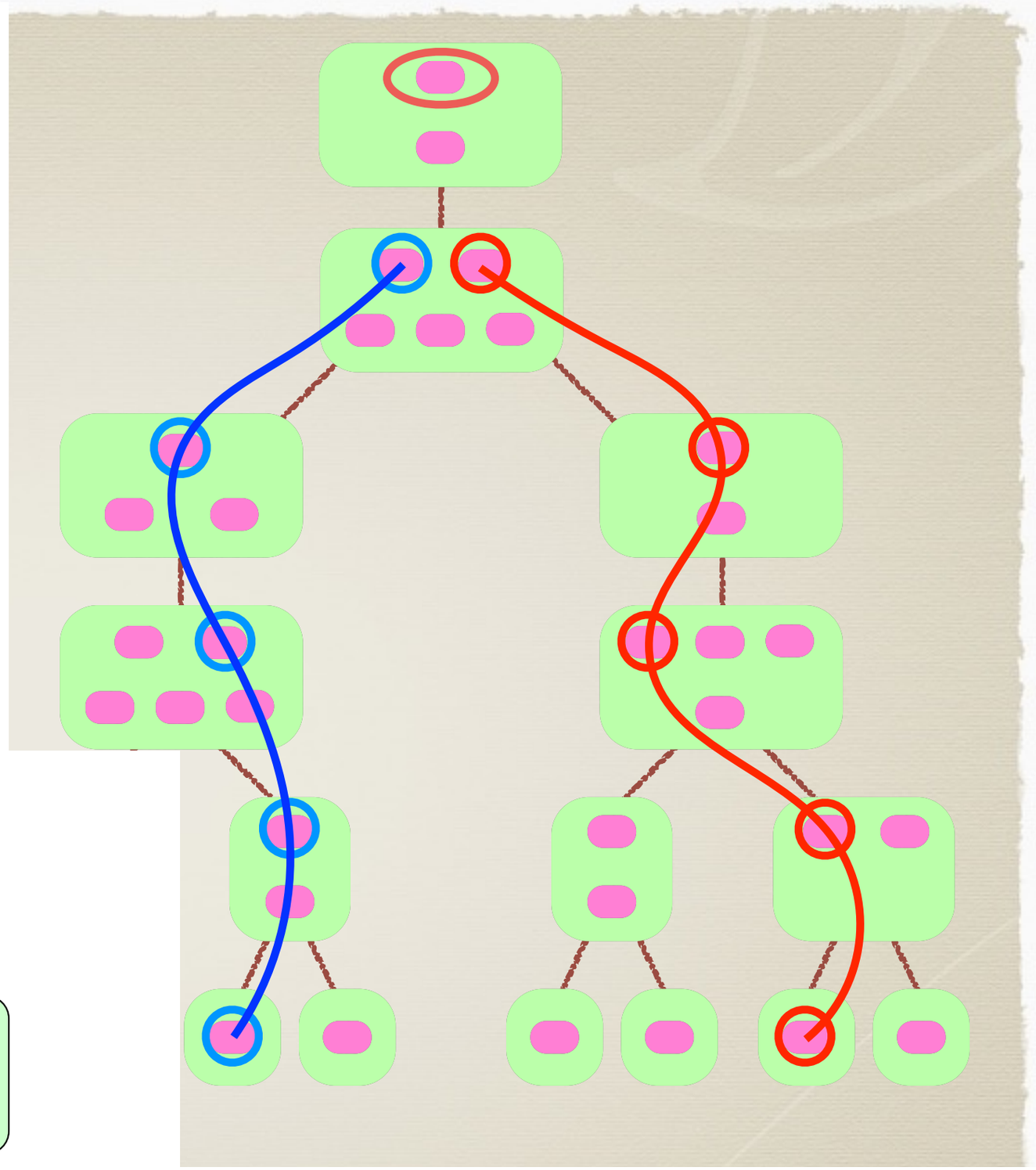
Data edges

Process edges

Tree interpretation in

Abstract Tree Decomposition





Vertices

Data edges

Process edges

Tree interpretation in

Abstract Tree Decomposition