Expressive power of pebble automata

or

Unary positive transitive closure logic on trees

Szeged

October 2006

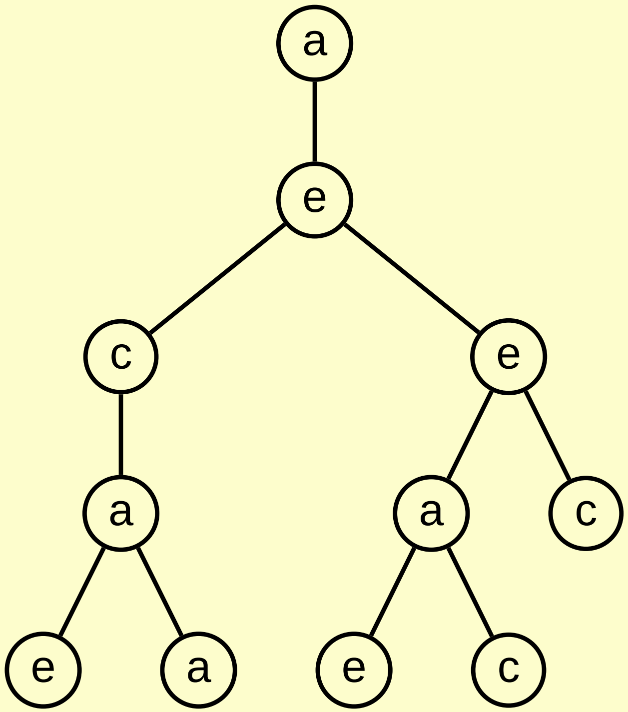
Thomas Schwentick

Joint work with

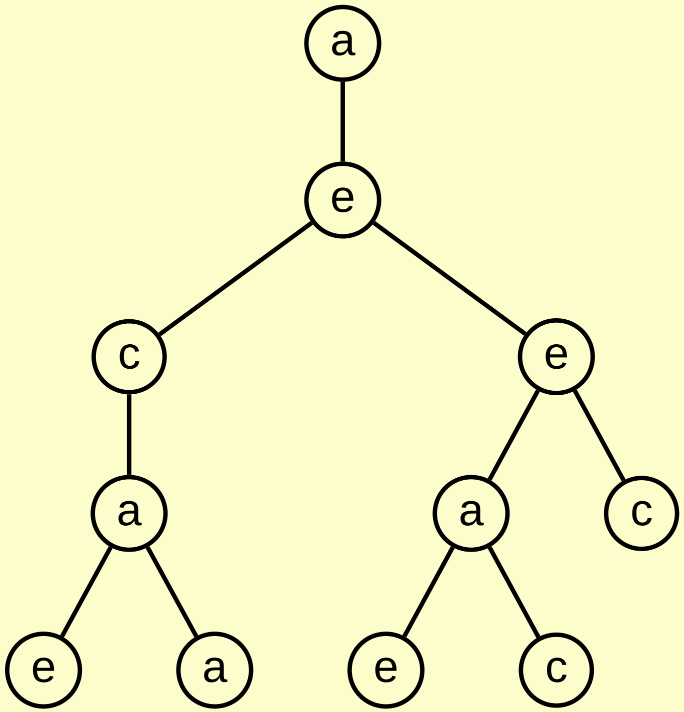
Mikołaj Bojańczyk, Mathias Samuelides, Luc Segoufin

How do string automata generalize to trees?

Parallel Tree Automata

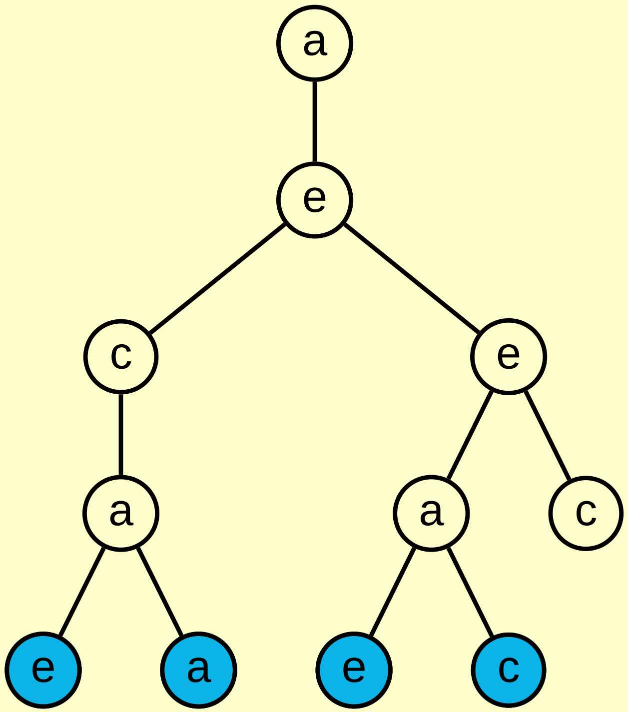


Sequential Tree Automata

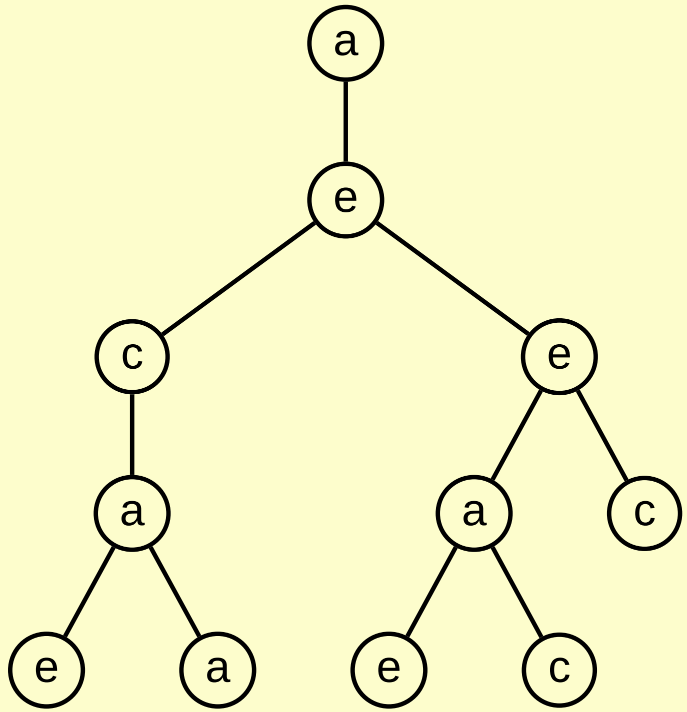


How do string automata generalize to trees?

Parallel Tree Automata

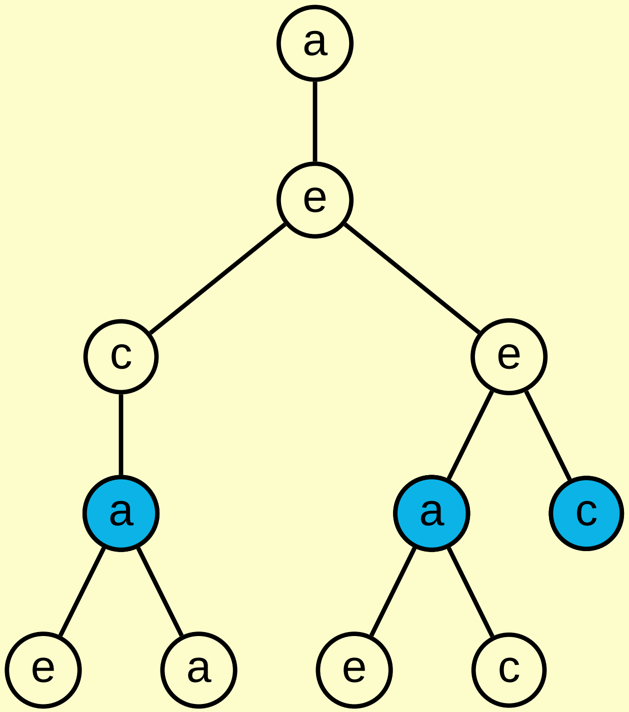


Sequential Tree Automata

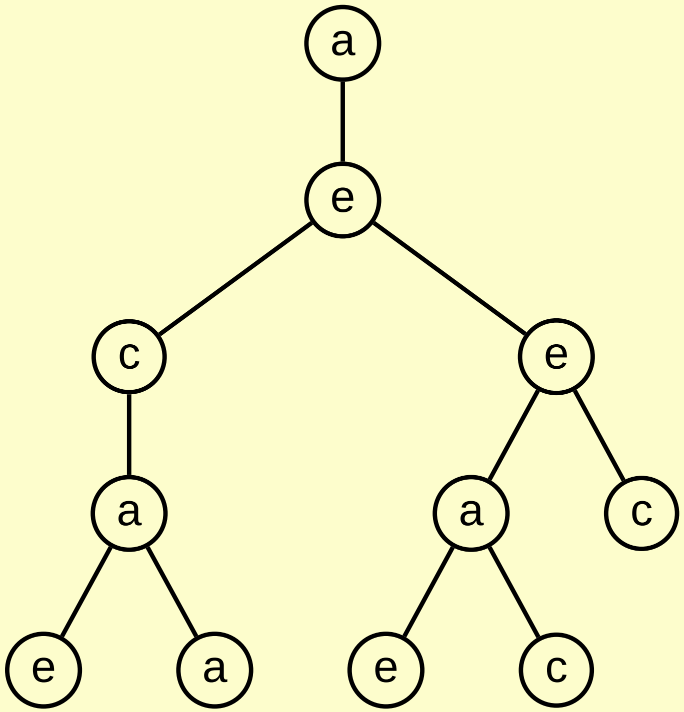


How do string automata generalize to trees?

Parallel Tree Automata

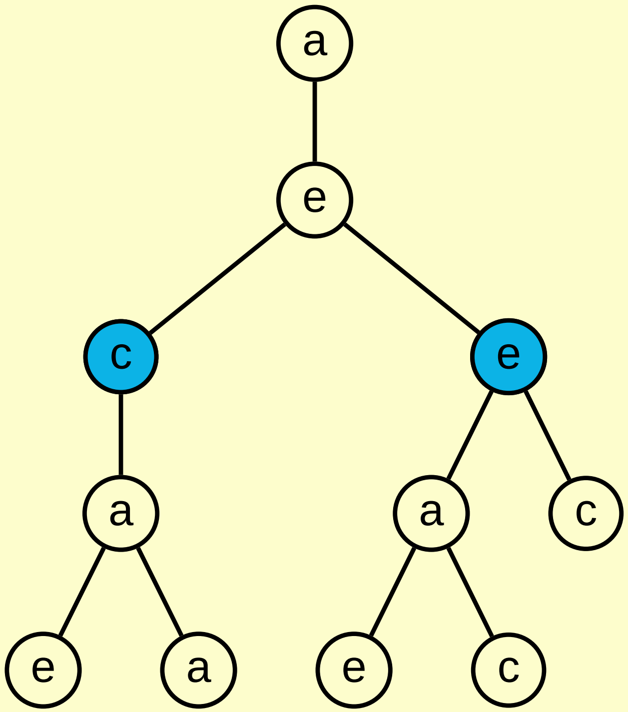


Sequential Tree Automata

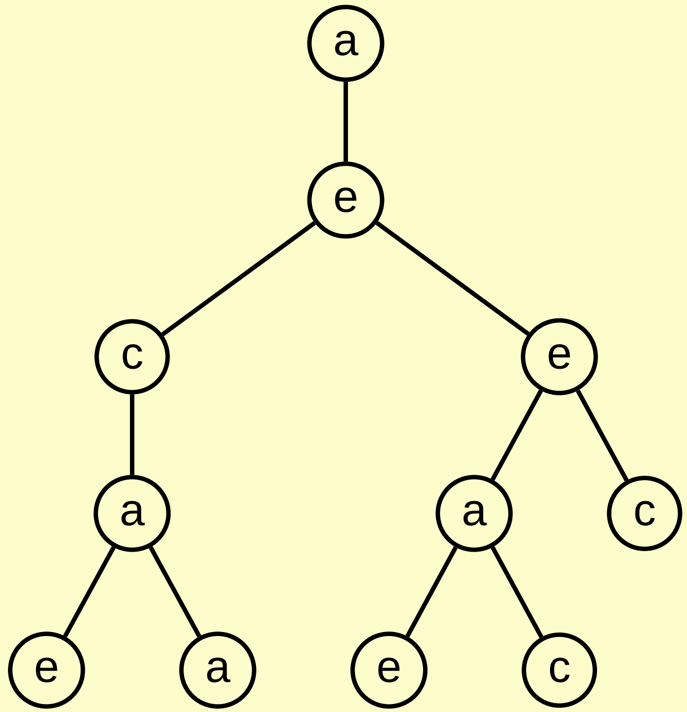


How do string automata generalize to trees?

Parallel Tree Automata

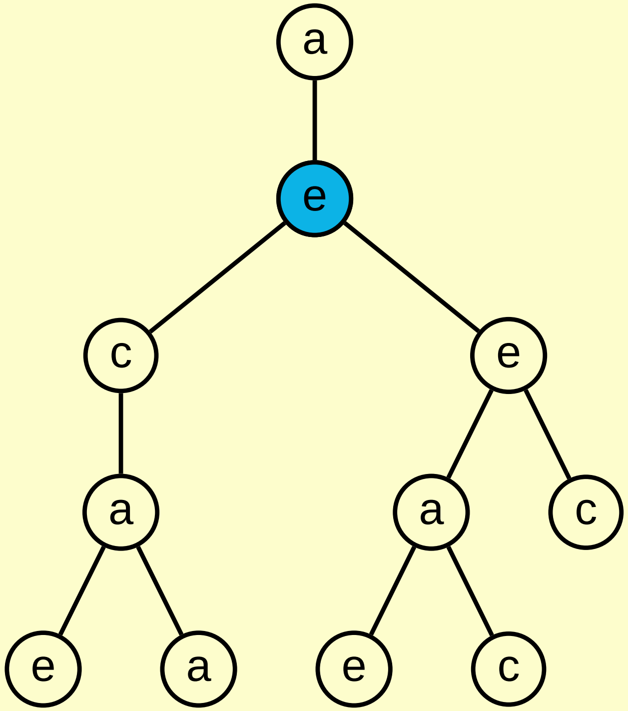


Sequential Tree Automata

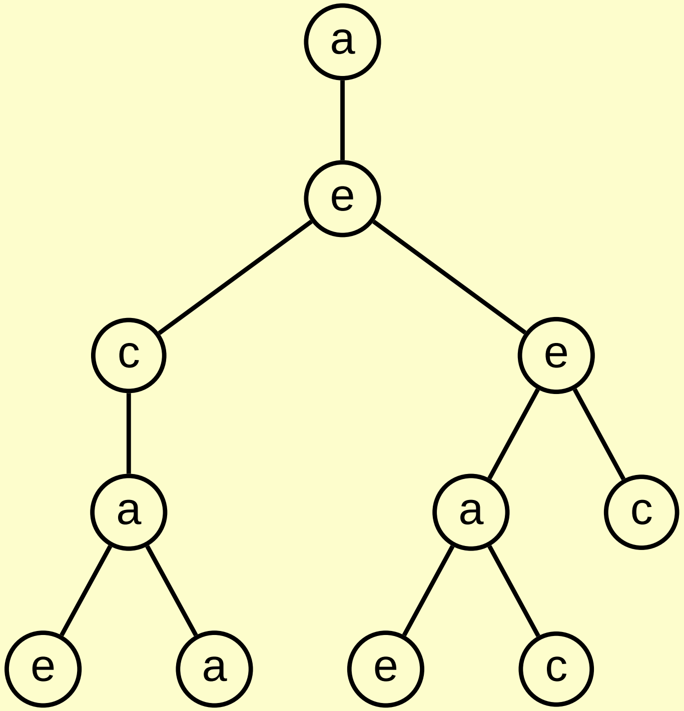


How do string automata generalize to trees?

Parallel Tree Automata

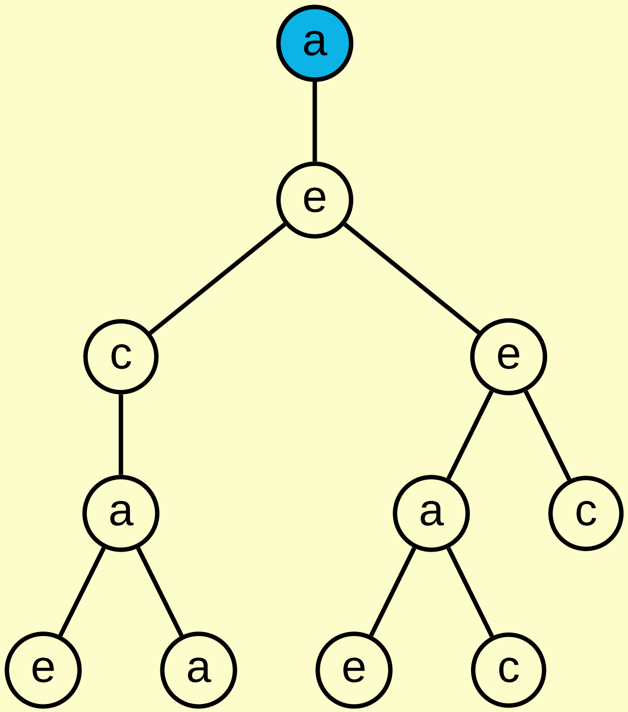


Sequential Tree Automata

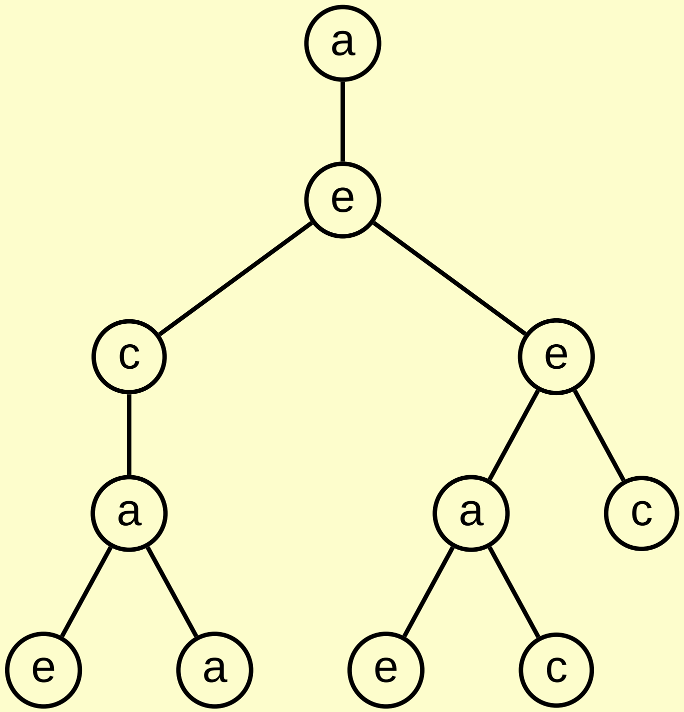


How do string automata generalize to trees?

Parallel Tree Automata

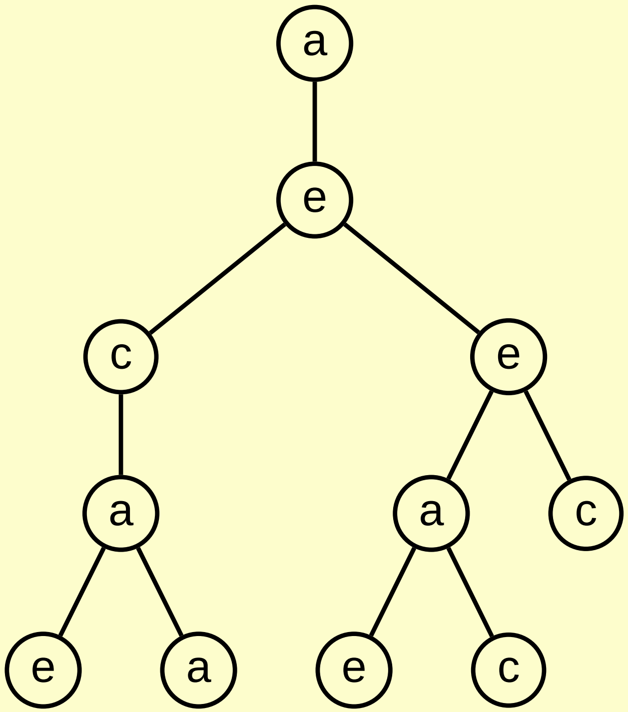


Sequential Tree Automata

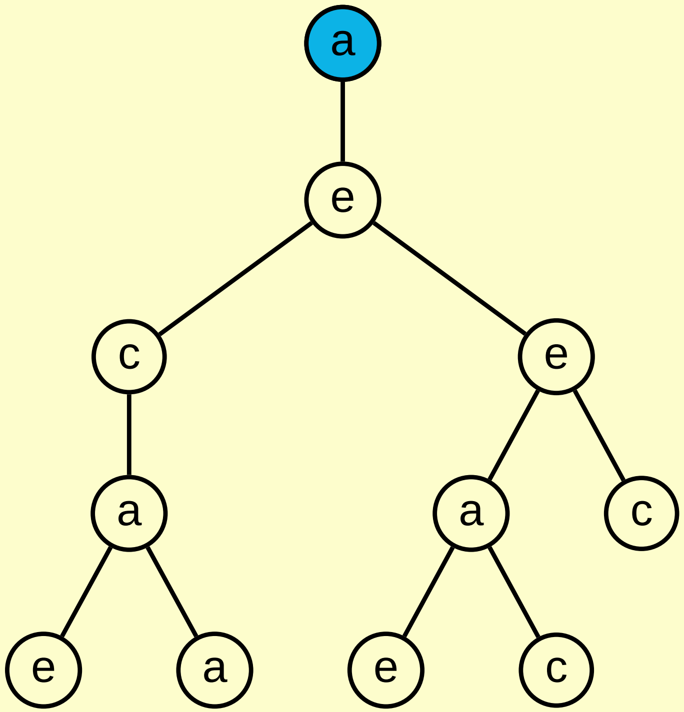


How do string automata generalize to trees?

Parallel Tree Automata

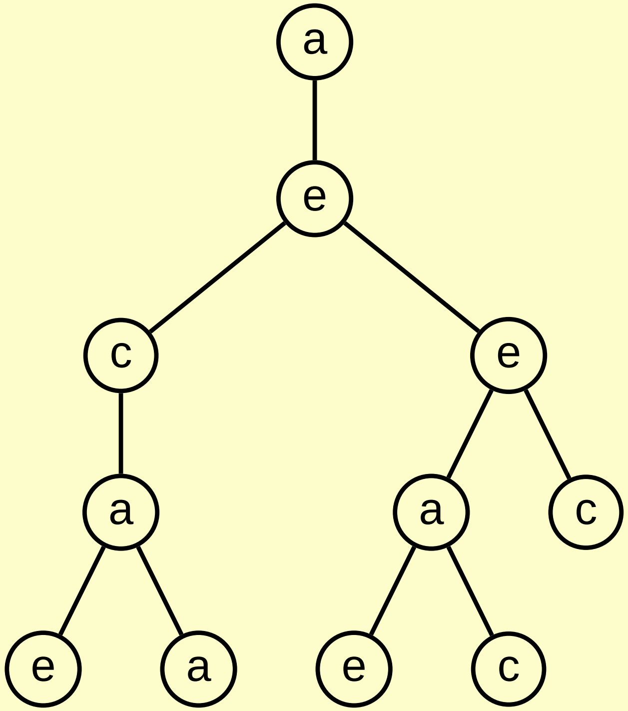


Sequential Tree Automata

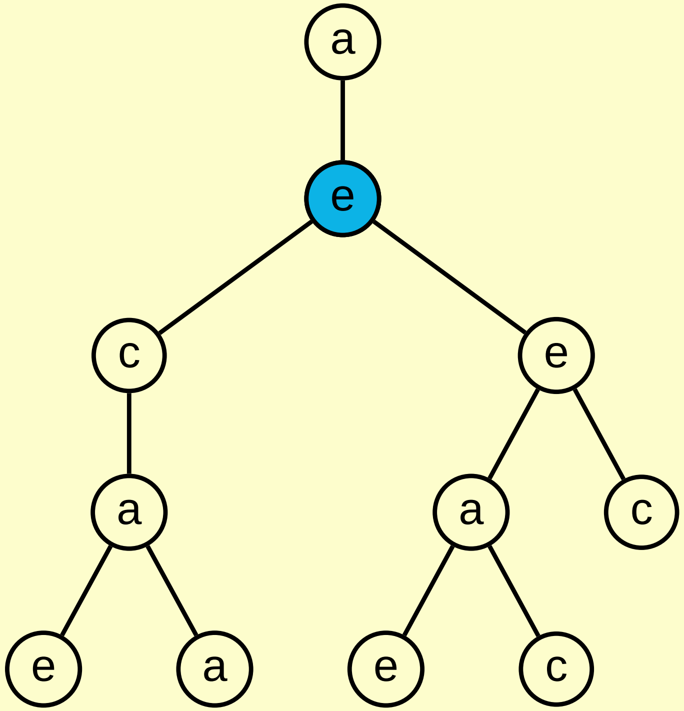


How do string automata generalize to trees?

Parallel Tree Automata

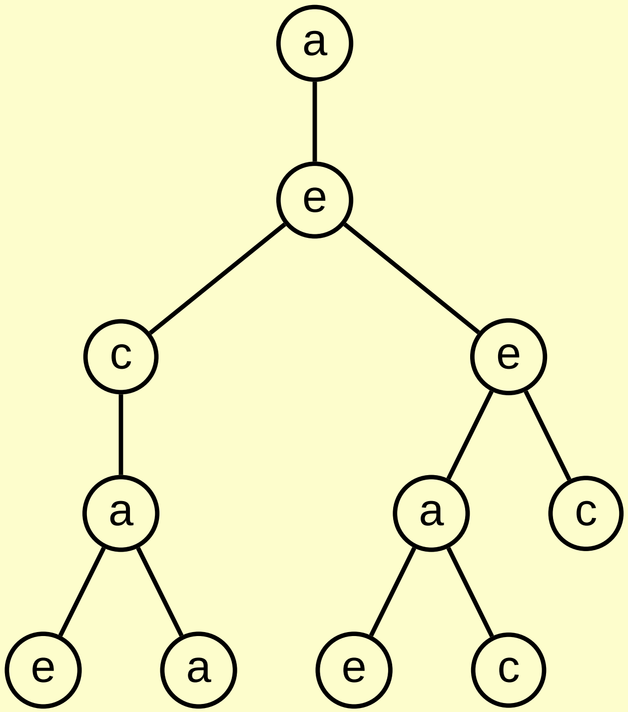


Sequential Tree Automata

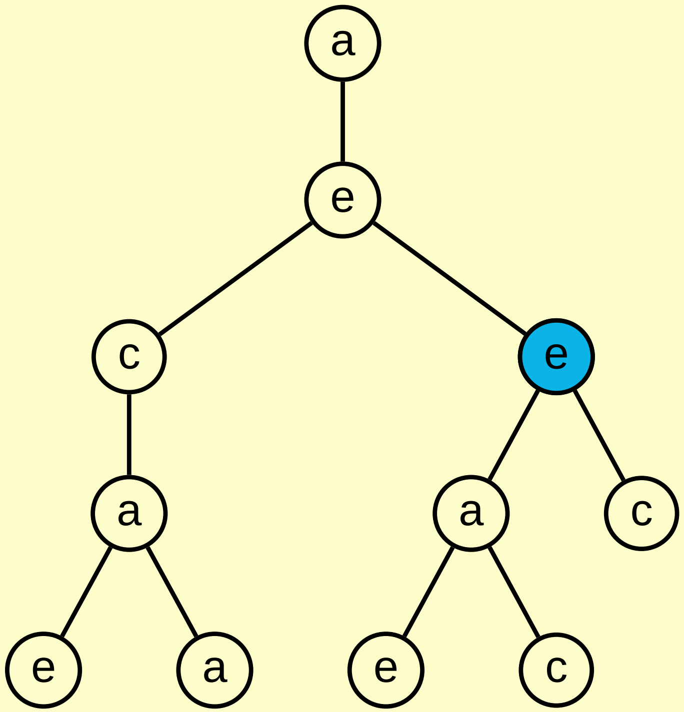


How do string automata generalize to trees?

Parallel Tree Automata

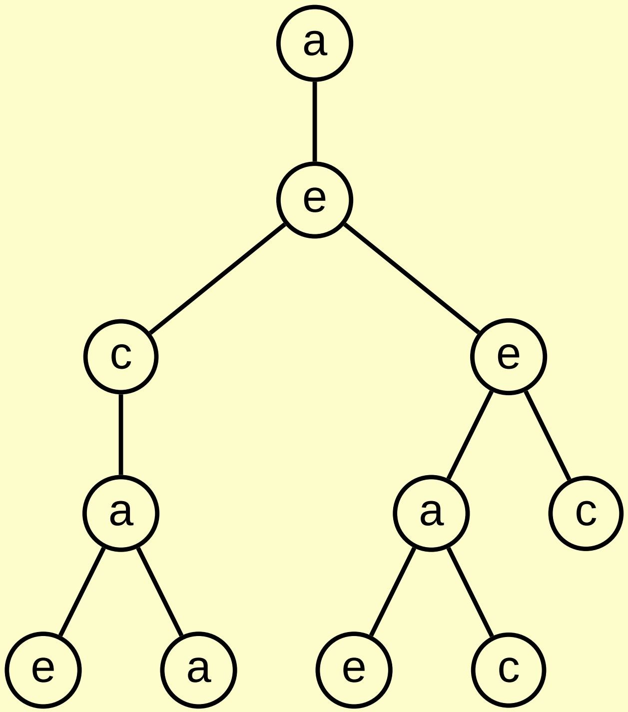


Sequential Tree Automata

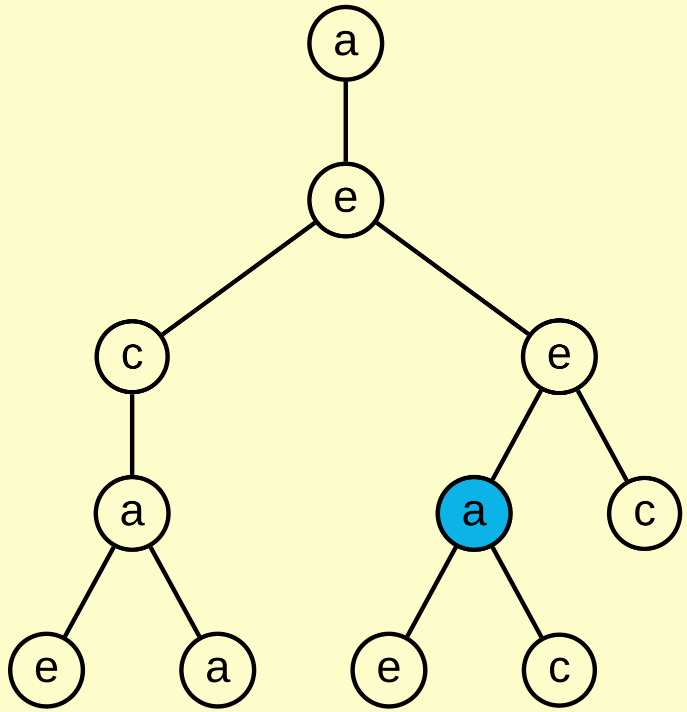


How do string automata generalize to trees?

Parallel Tree Automata

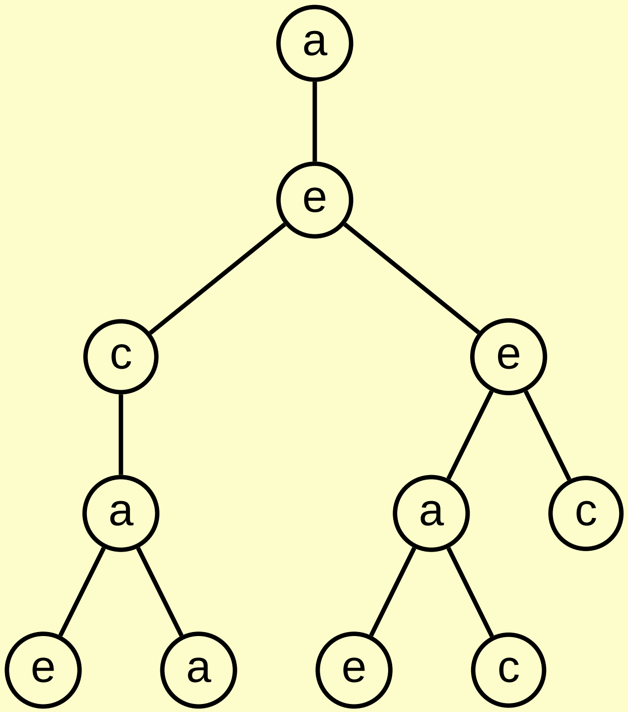


Sequential Tree Automata

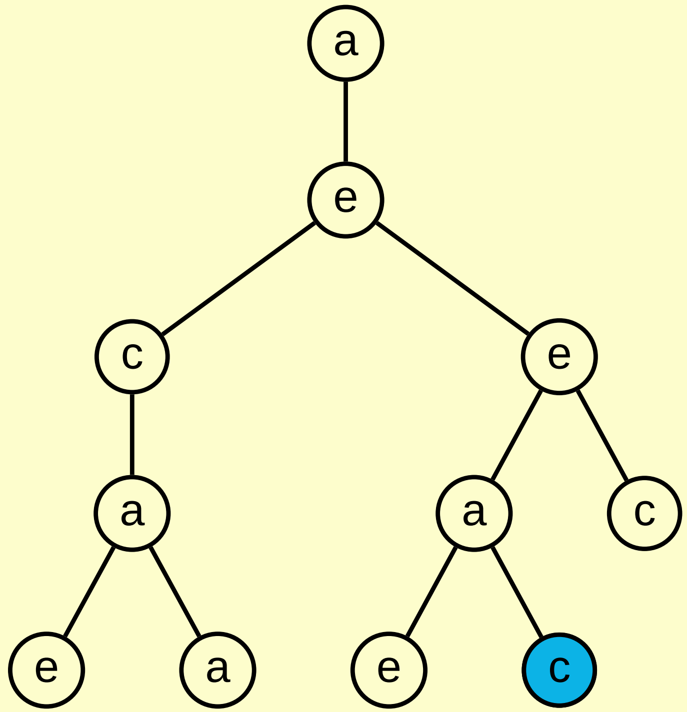


How do string automata generalize to trees?

Parallel Tree Automata

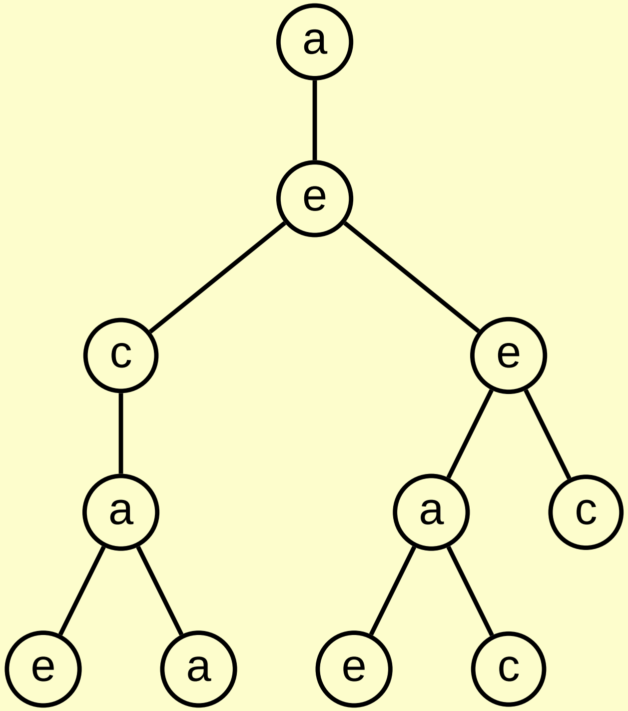


Sequential Tree Automata

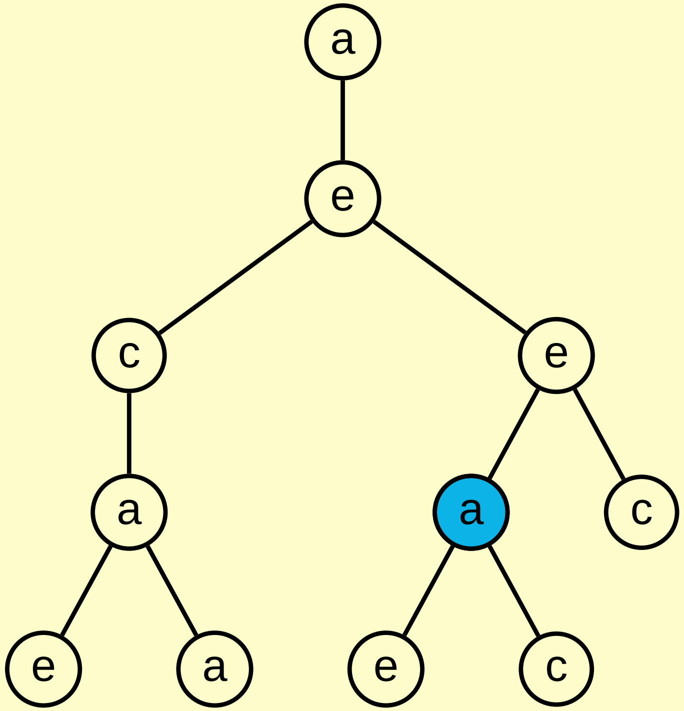


How do string automata generalize to trees?

Parallel Tree Automata

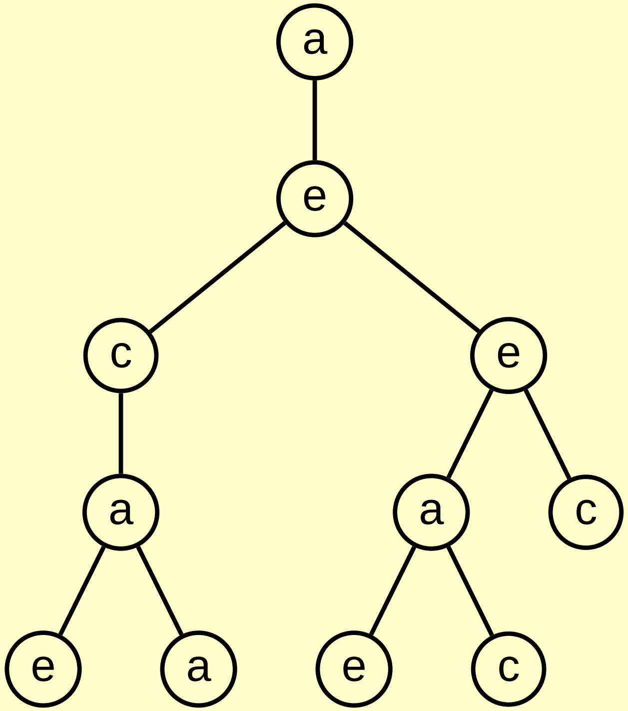


Sequential Tree Automata

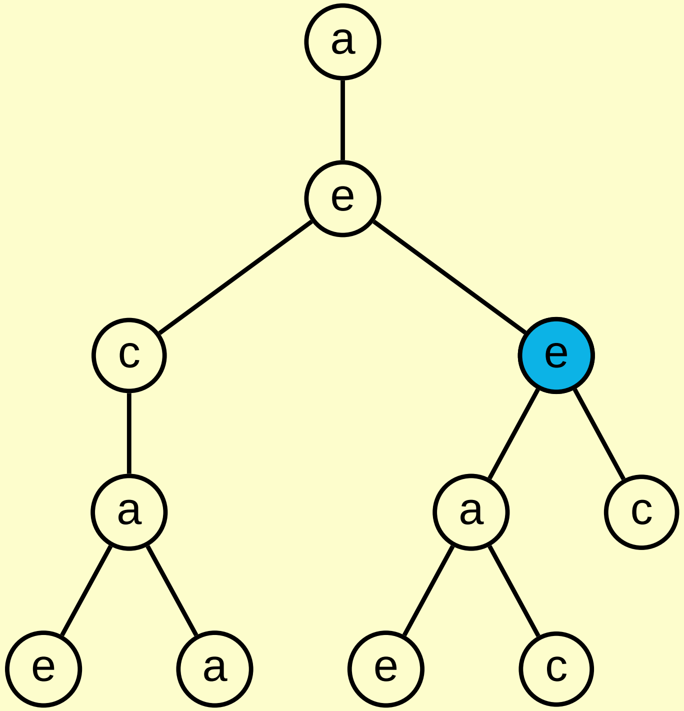


How do string automata generalize to trees?

Parallel Tree Automata

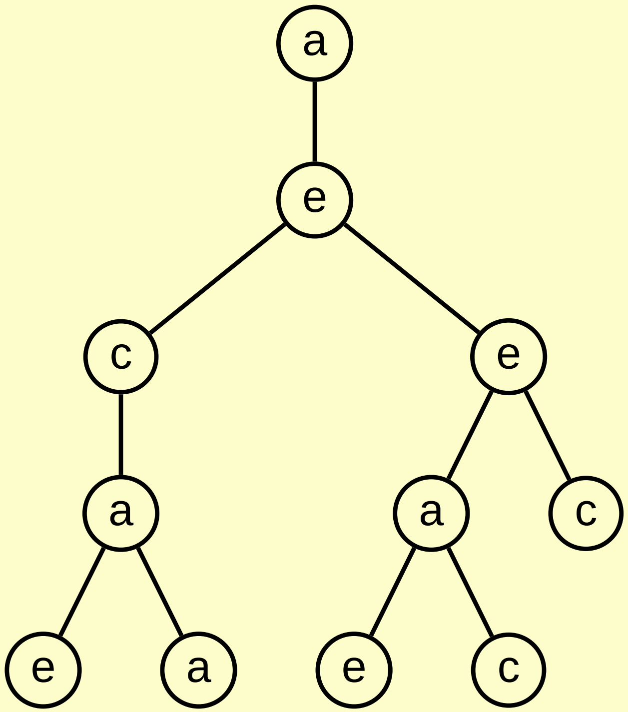


Sequential Tree Automata

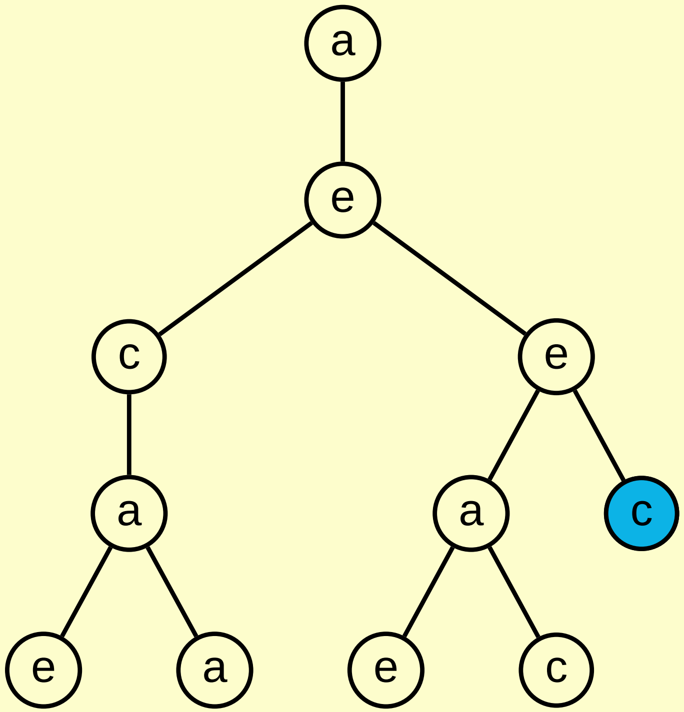


How do string automata generalize to trees?

Parallel Tree Automata

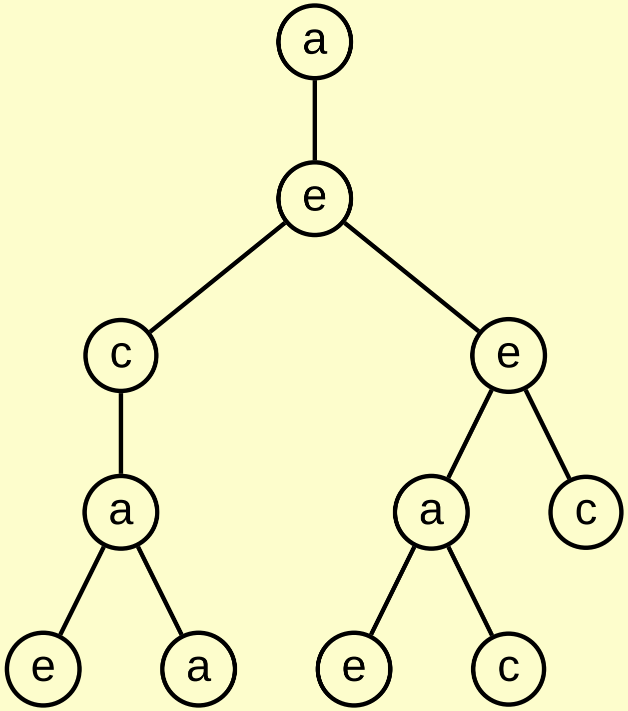


Sequential Tree Automata

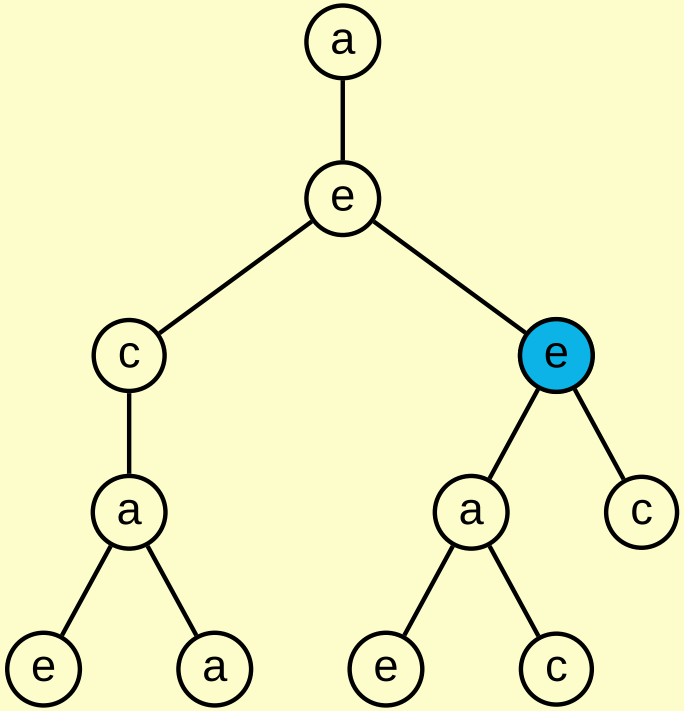


How do string automata generalize to trees?

Parallel Tree Automata

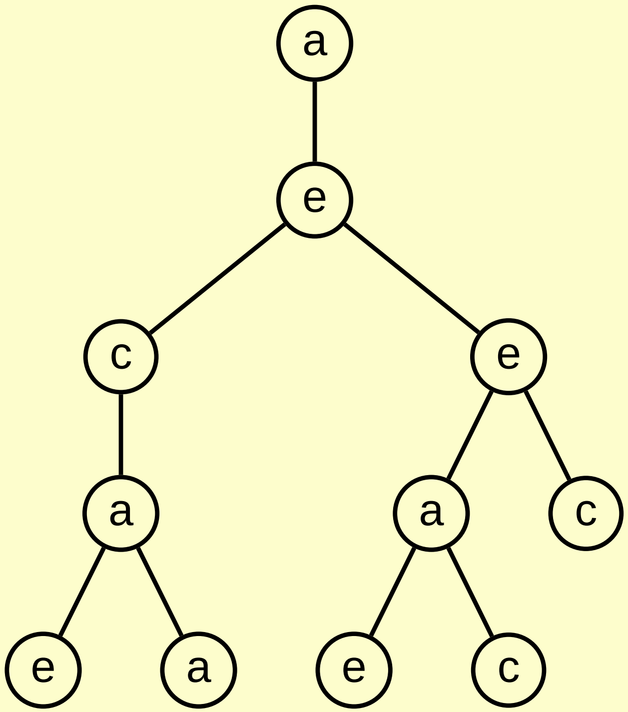


Sequential Tree Automata

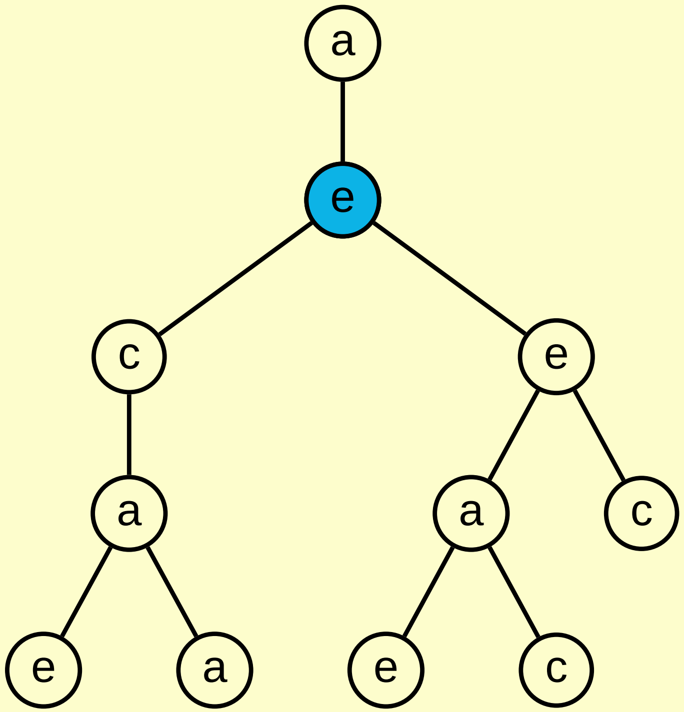


How do string automata generalize to trees?

Parallel Tree Automata

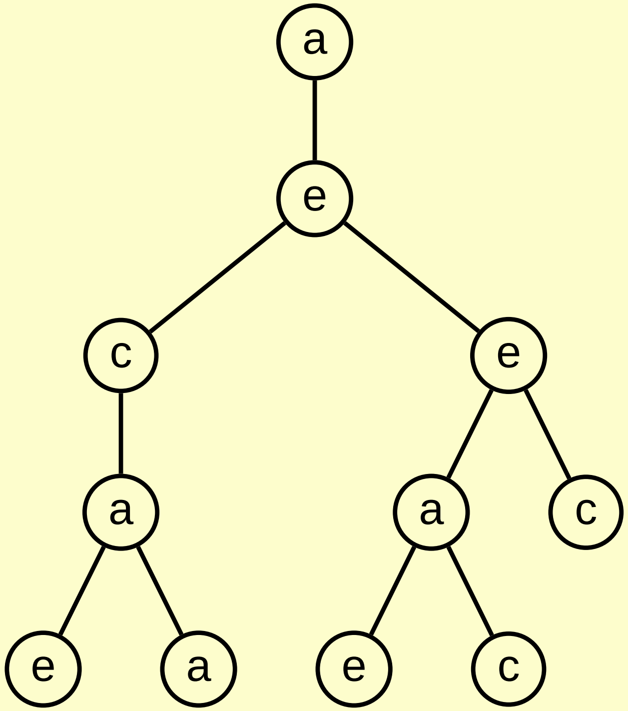


Sequential Tree Automata

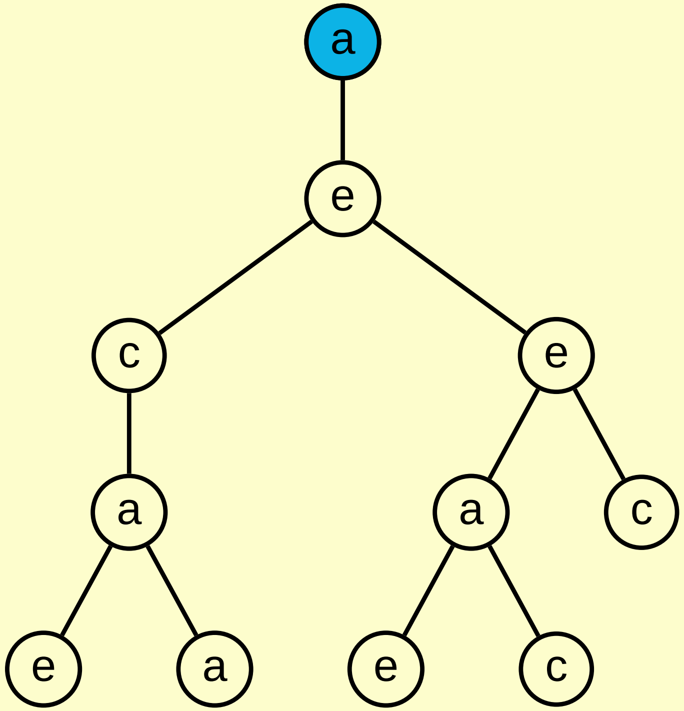


How do string automata generalize to trees?

Parallel Tree Automata



Sequential Tree Automata

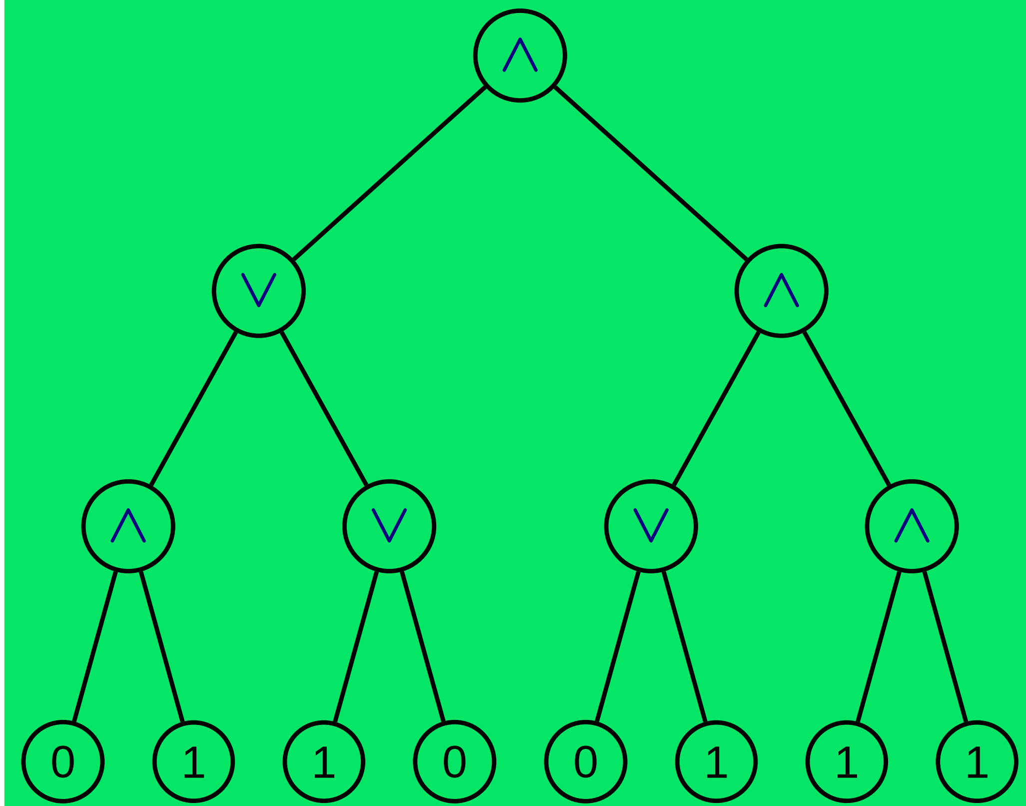


Can sequential automata do anything useful?

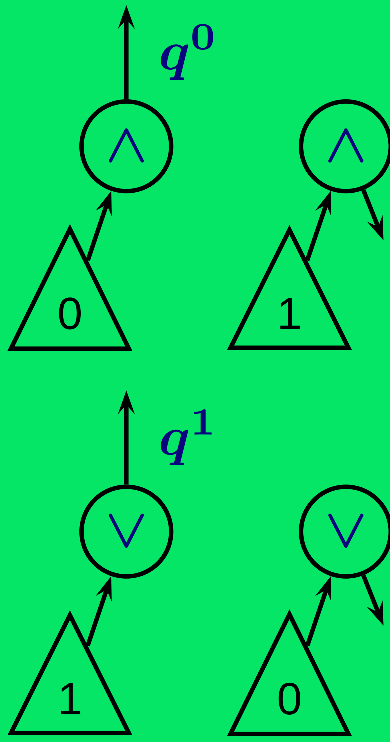
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

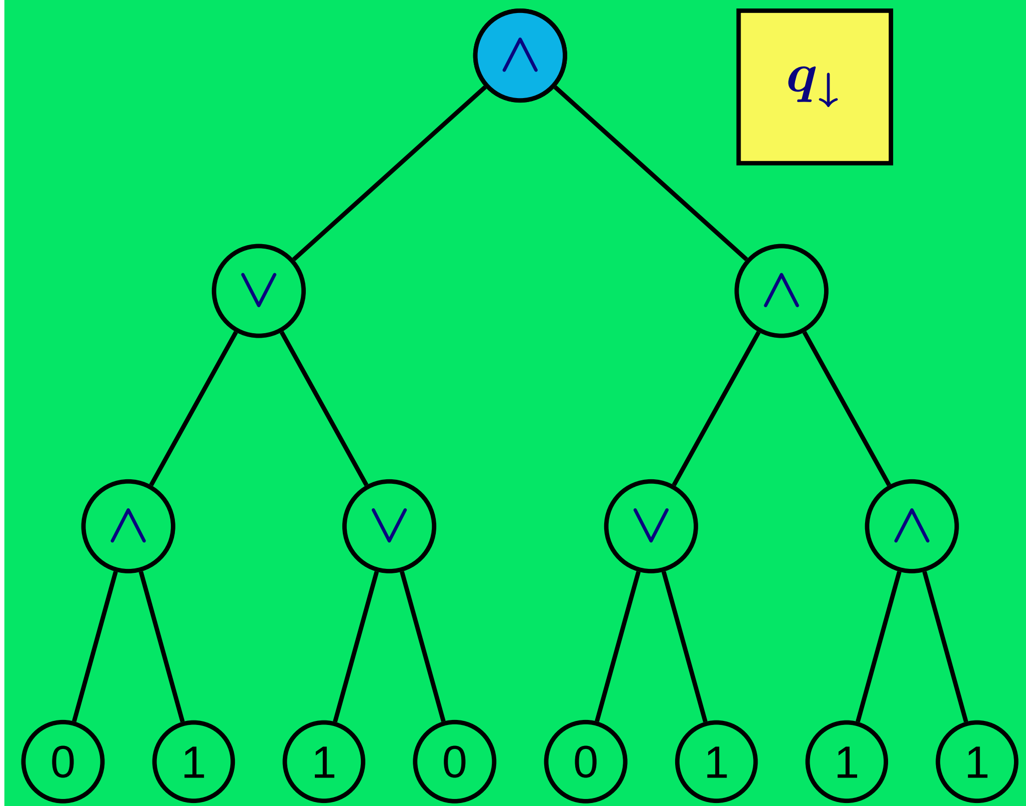


Can sequential automata do anything useful?

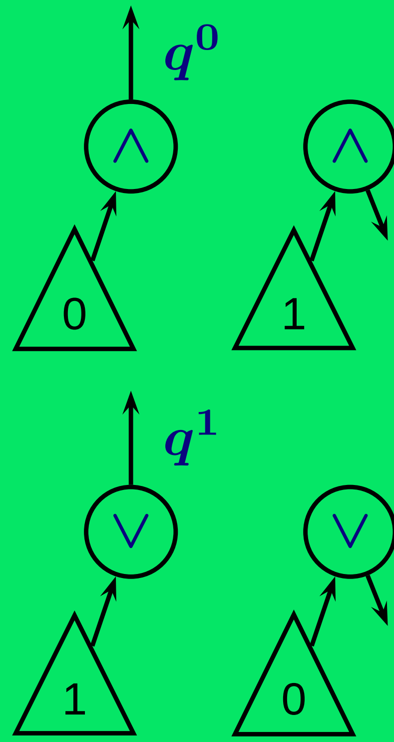
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

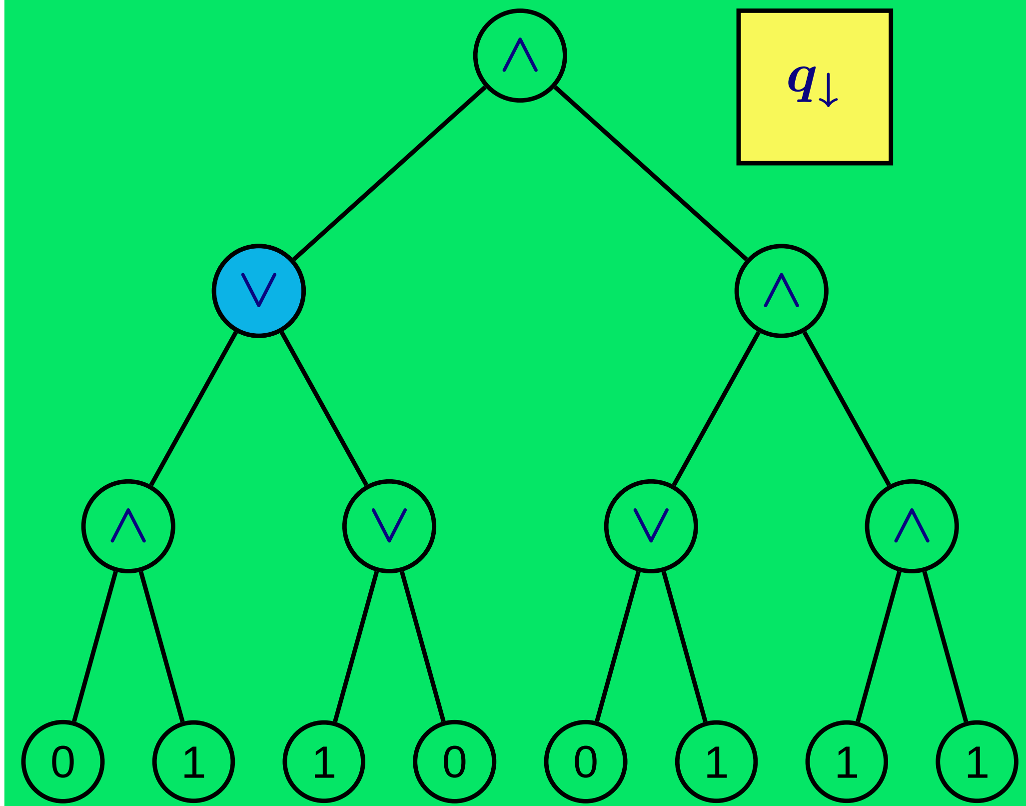


Can sequential automata do anything useful?

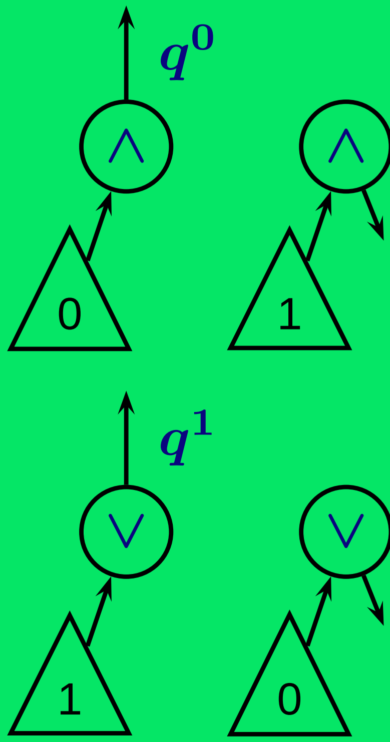
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

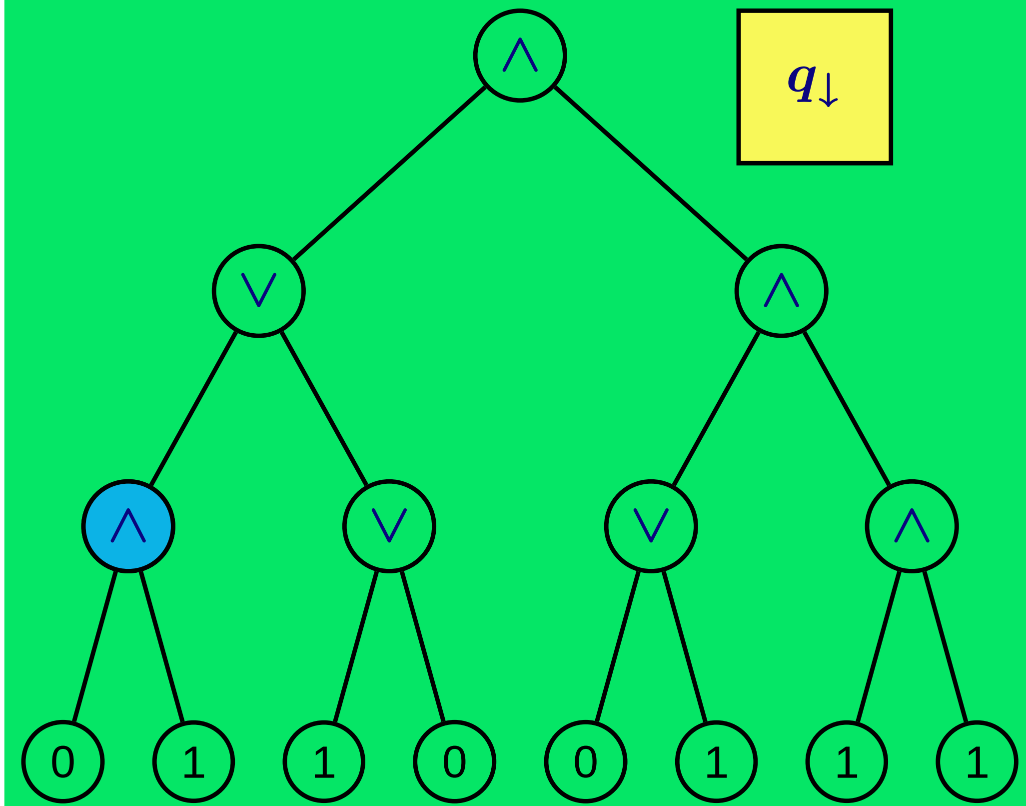


Can sequential automata do anything useful?

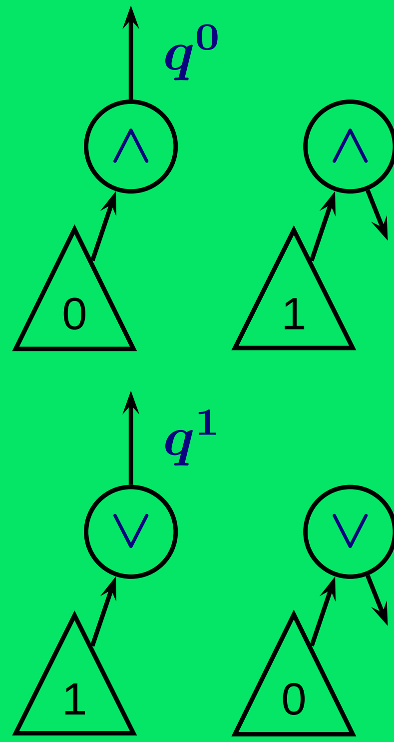
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

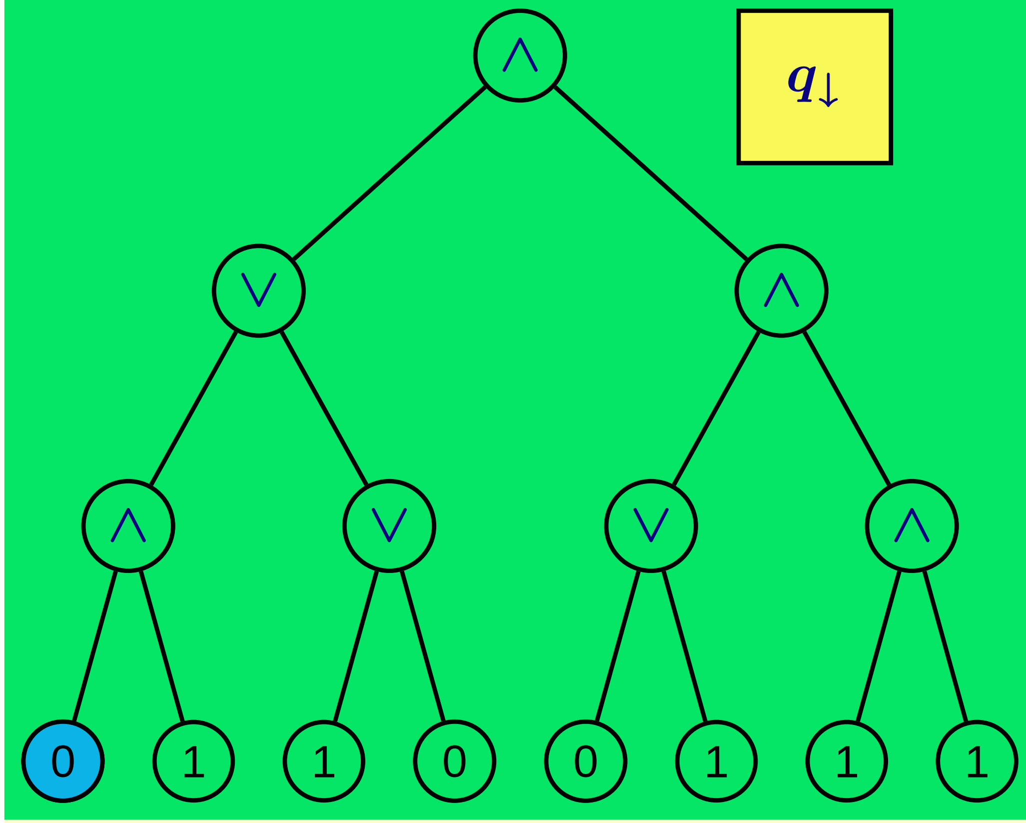


Can sequential automata do anything useful?

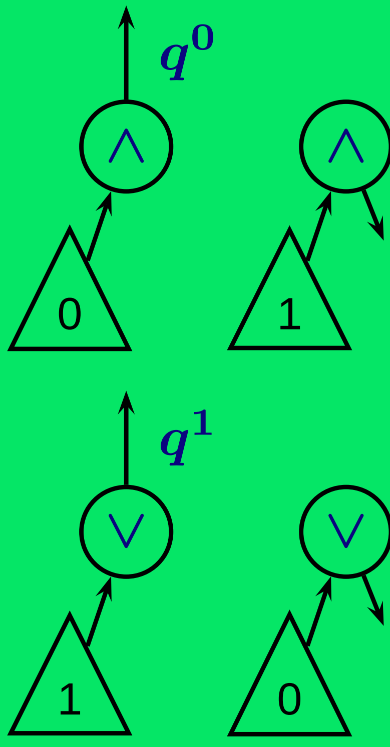
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

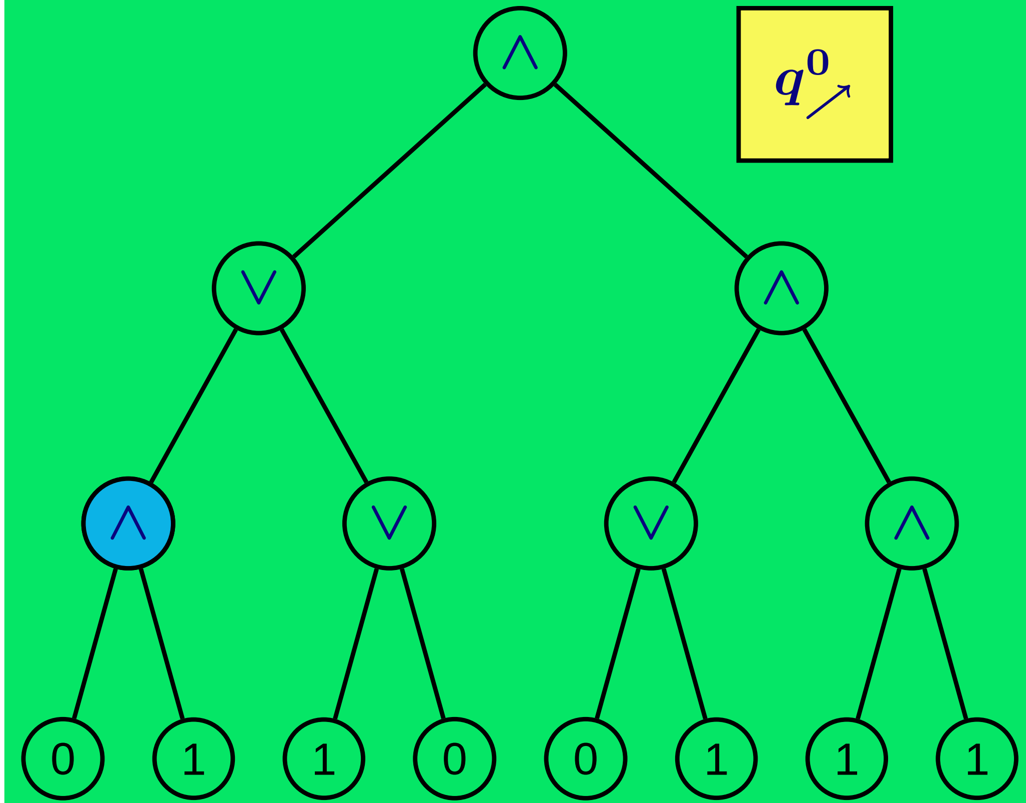


Can sequential automata do anything useful?

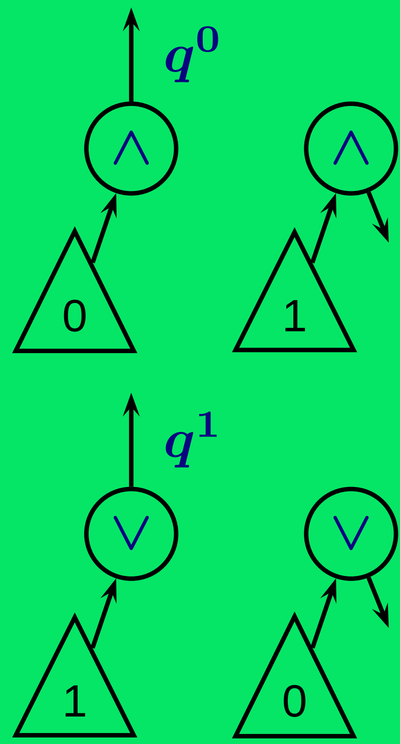
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

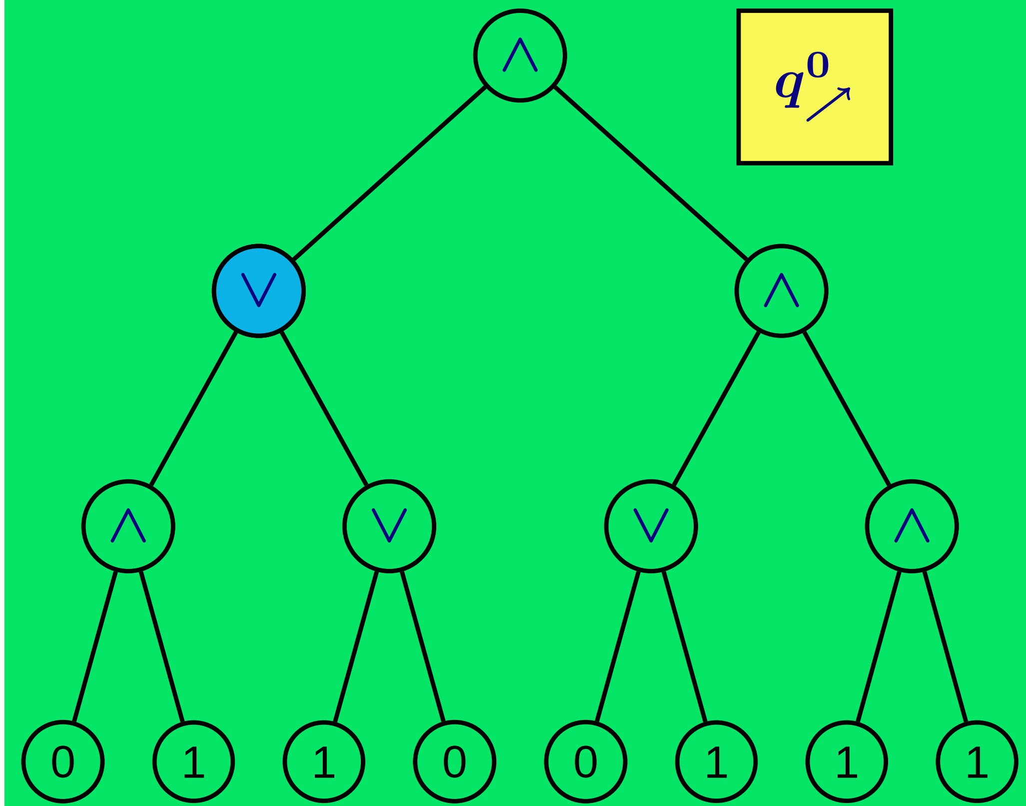


Can sequential automata do anything useful?

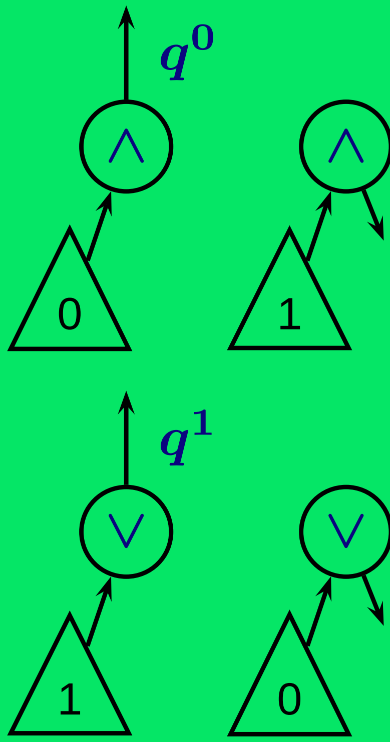
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

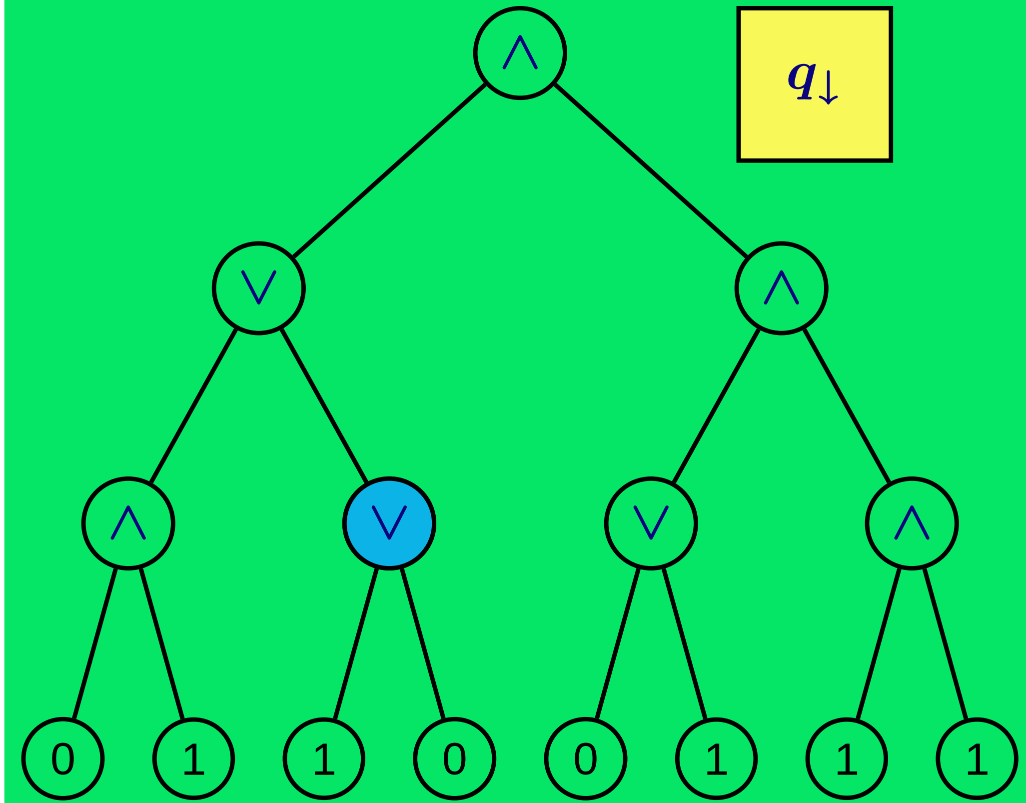


Can sequential automata do anything useful?

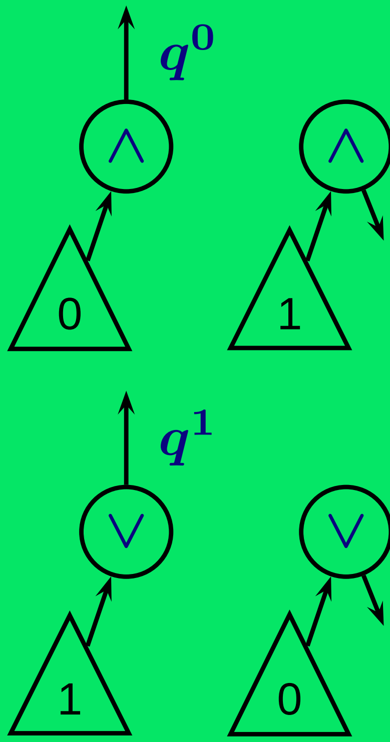
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

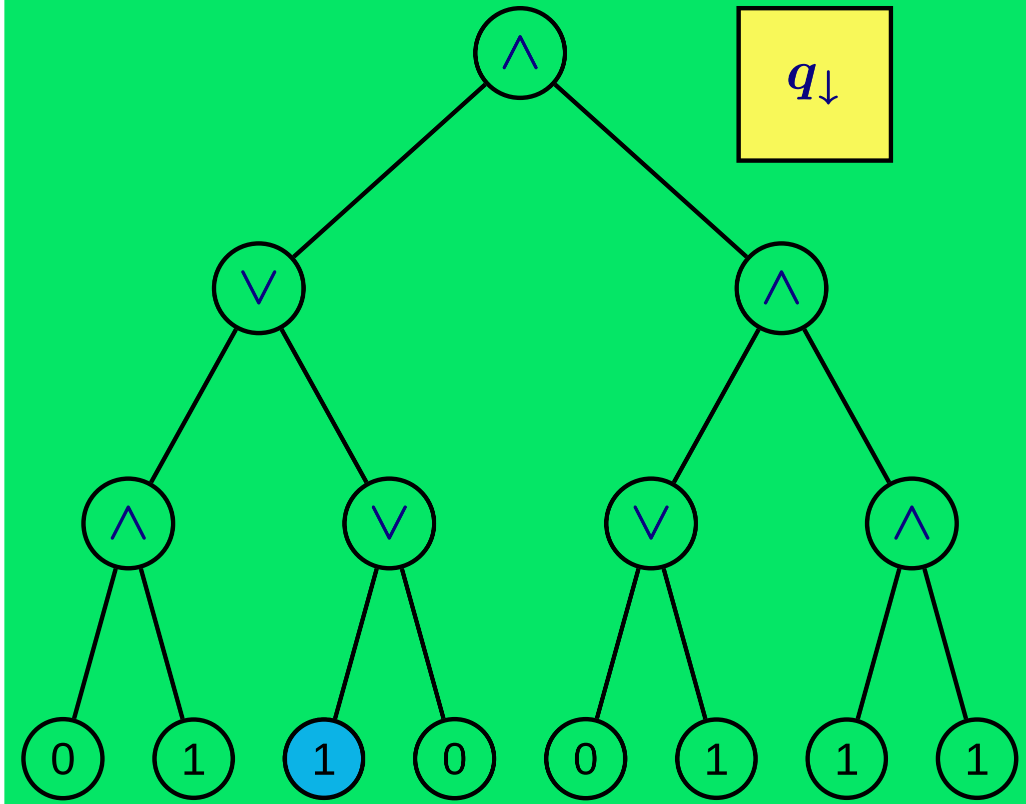


Can sequential automata do anything useful?

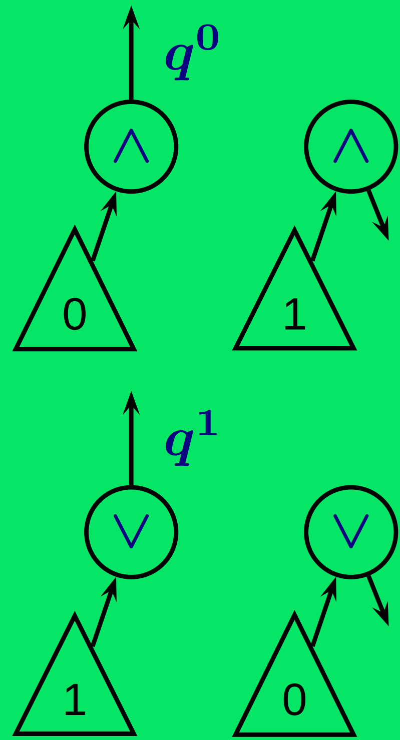
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

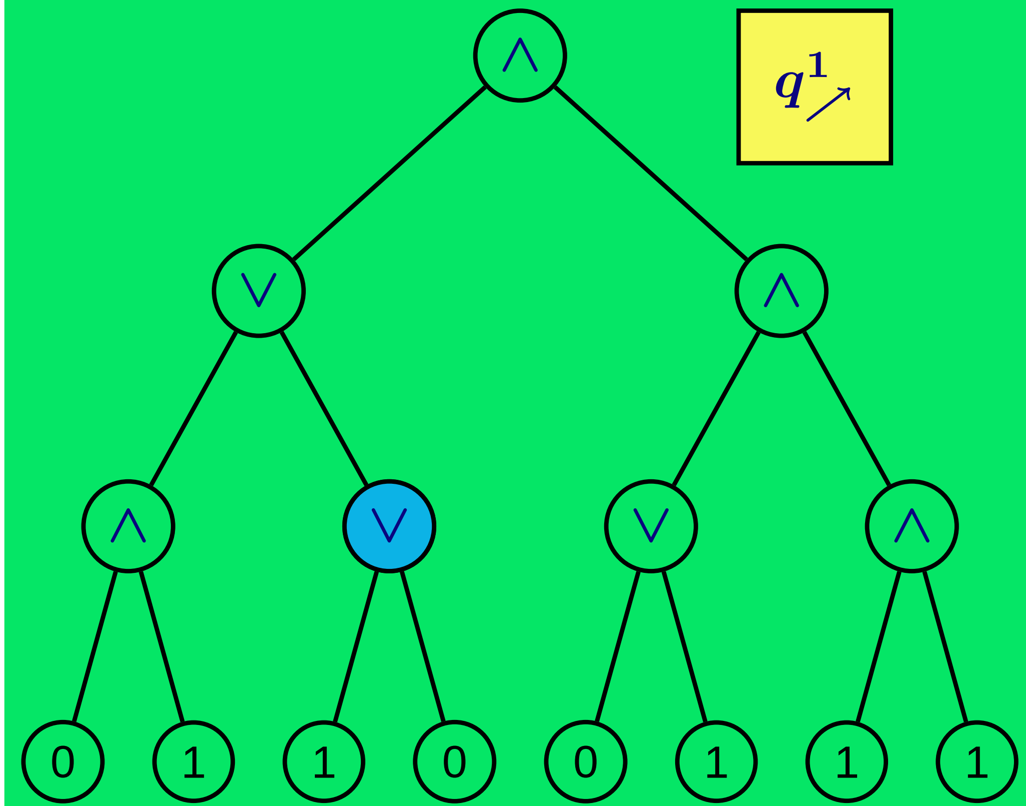


Can sequential automata do anything useful?

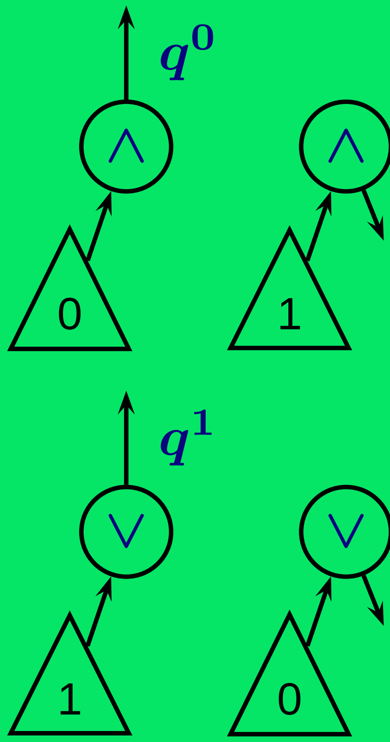
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

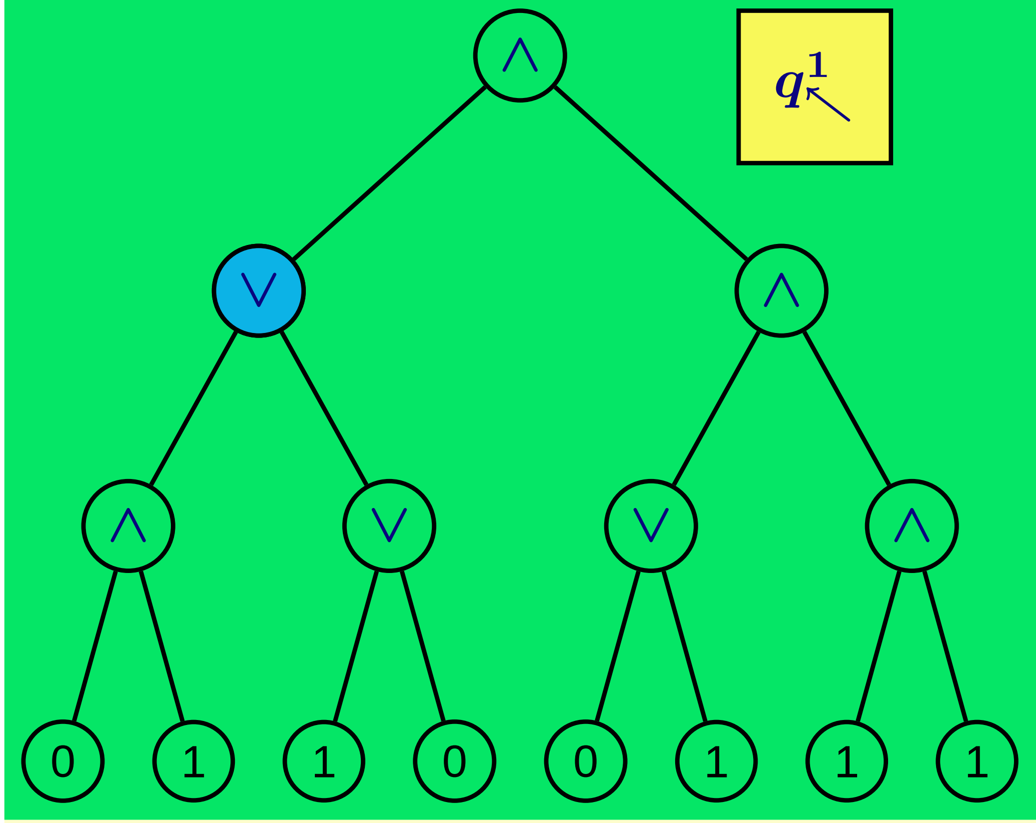


Can sequential automata do anything useful?

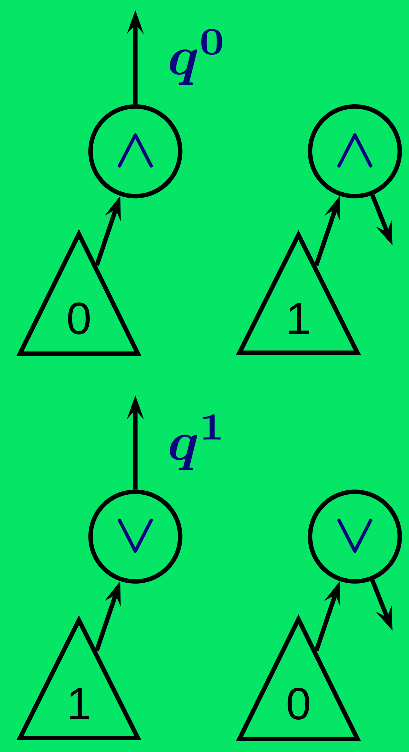
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

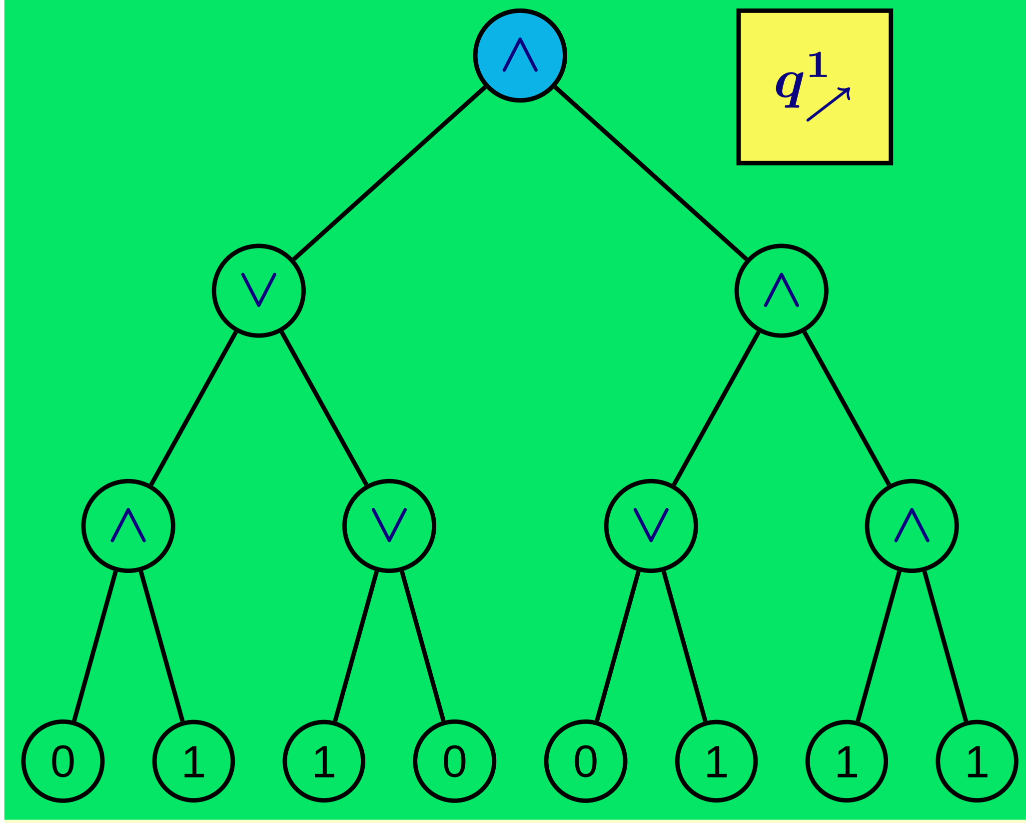


Can sequential automata do anything useful?

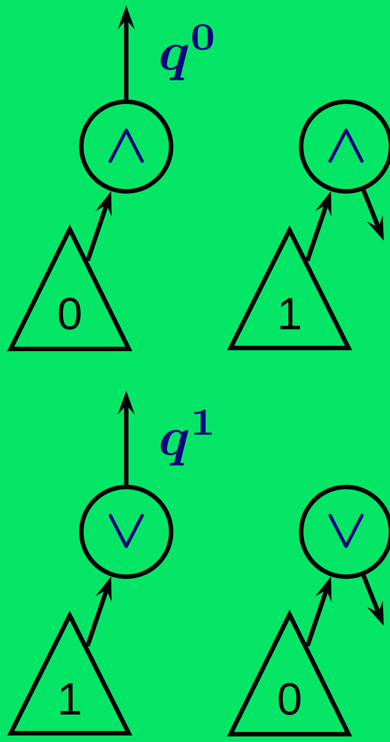
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

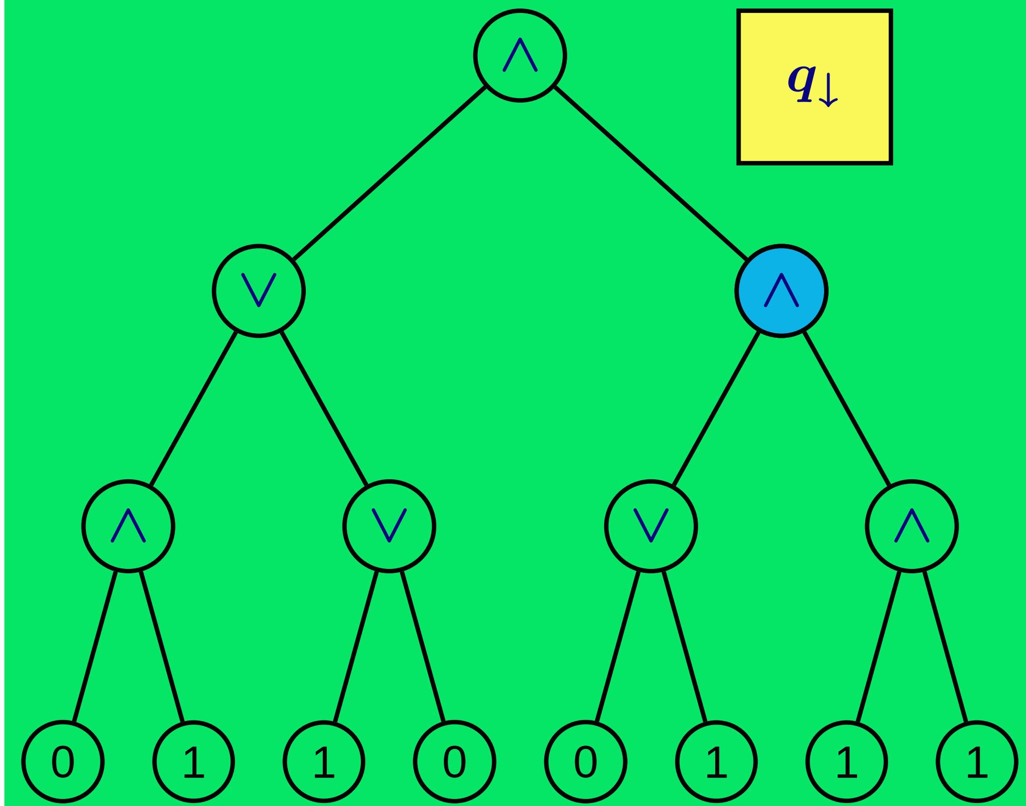


Can sequential automata do anything useful?

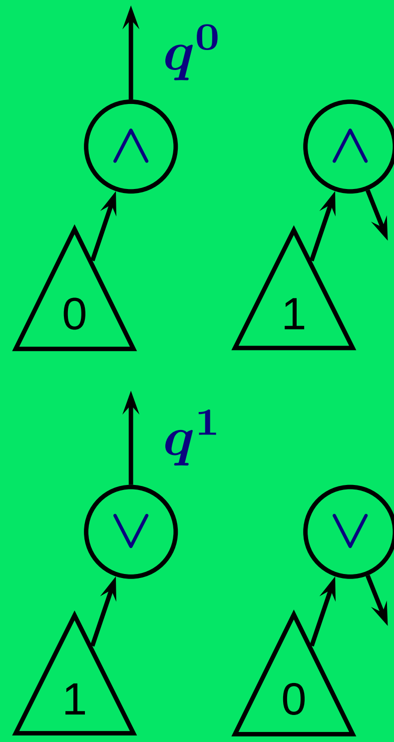
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

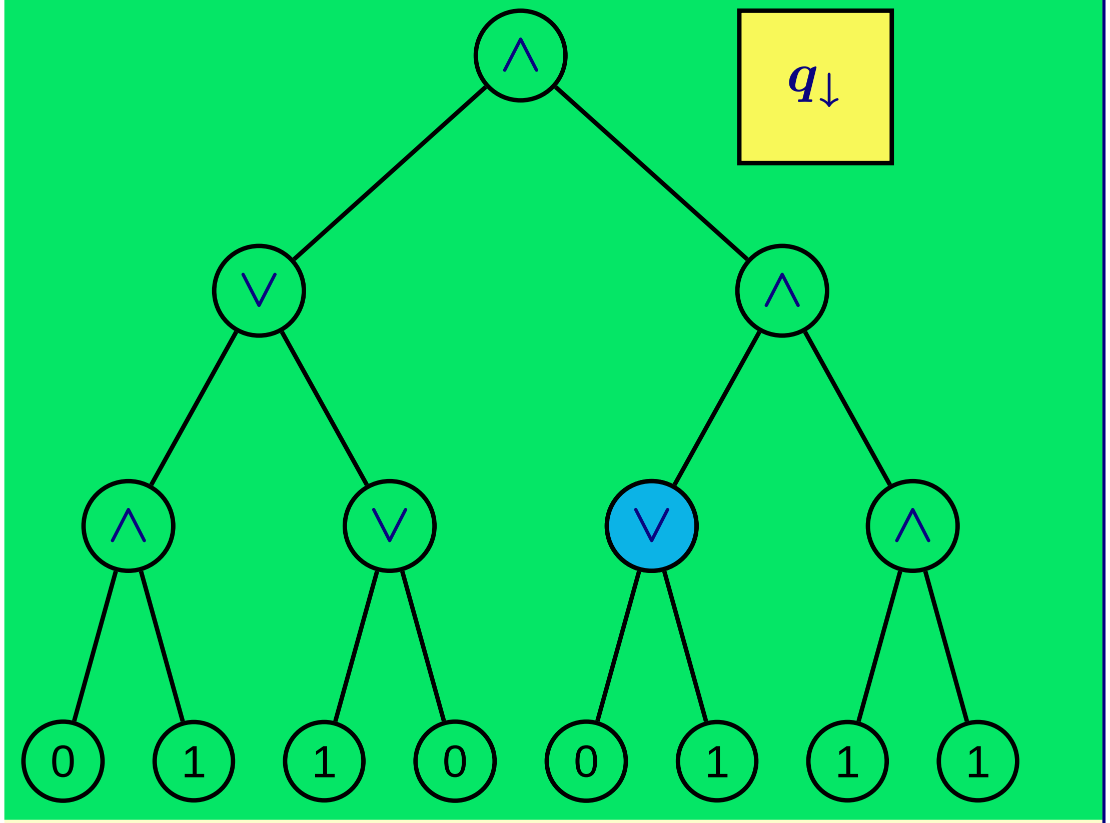


Can sequential automata do anything useful?

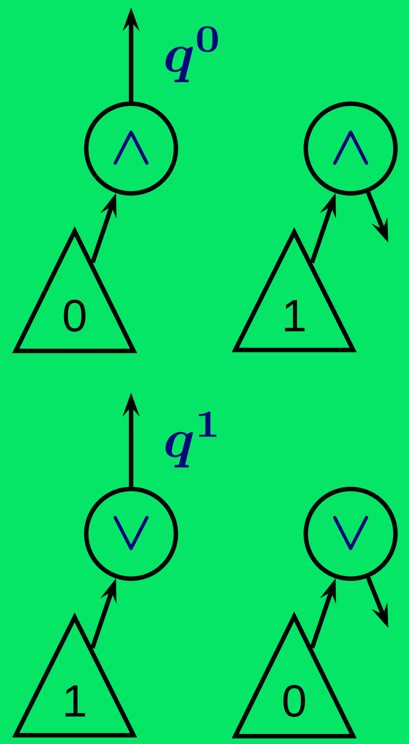
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

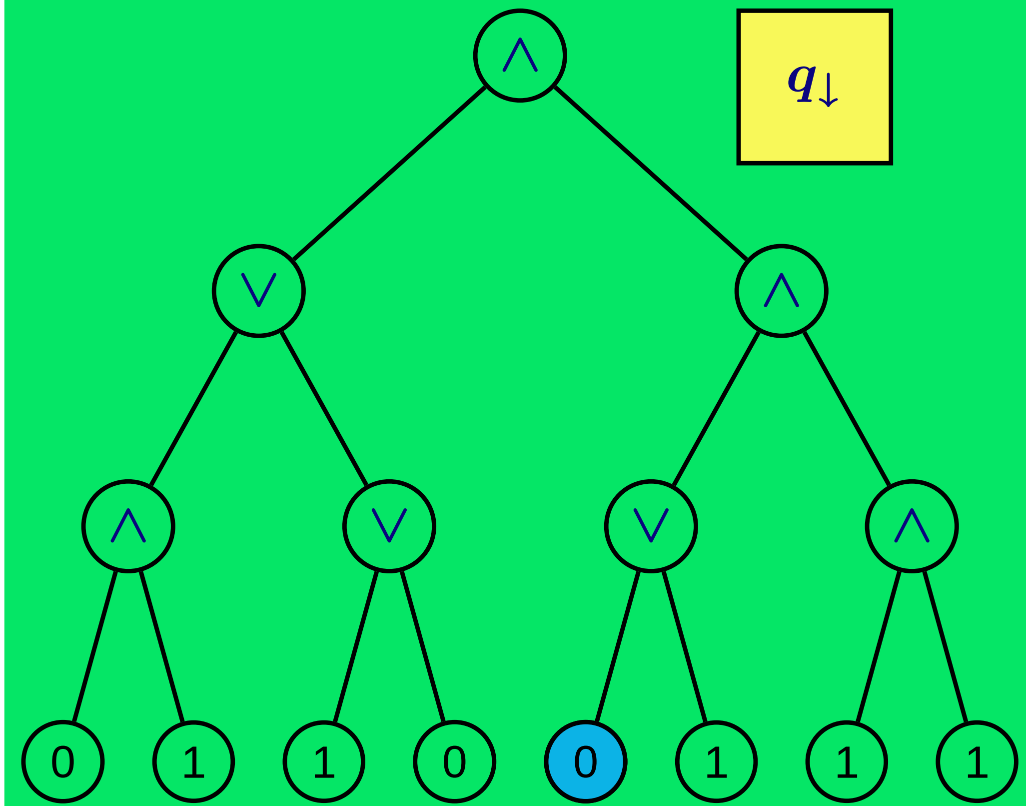


Can sequential automata do anything useful?

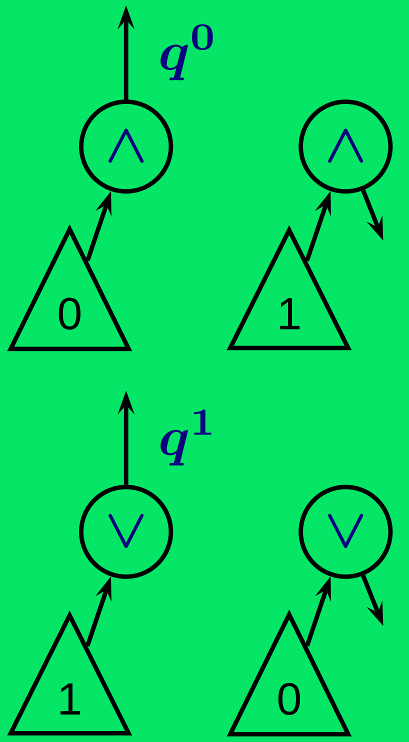
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

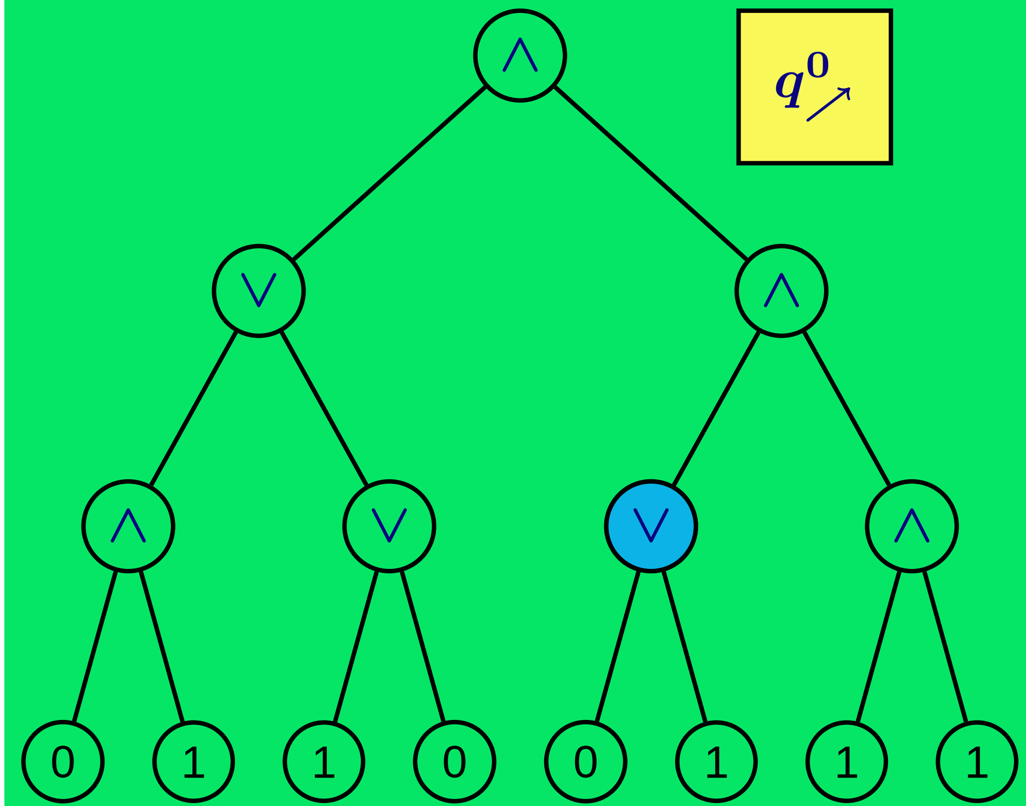


Can sequential automata do anything useful?

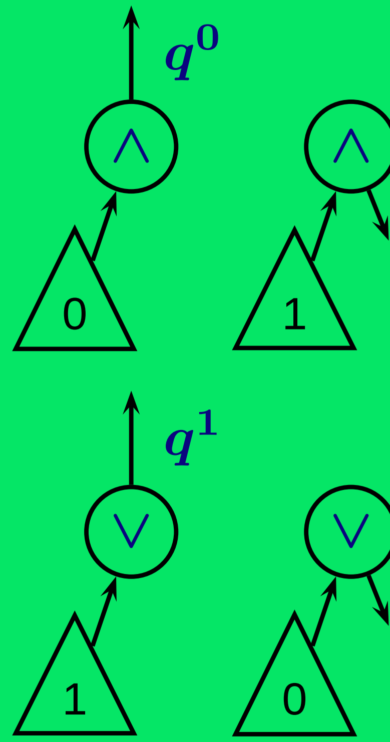
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

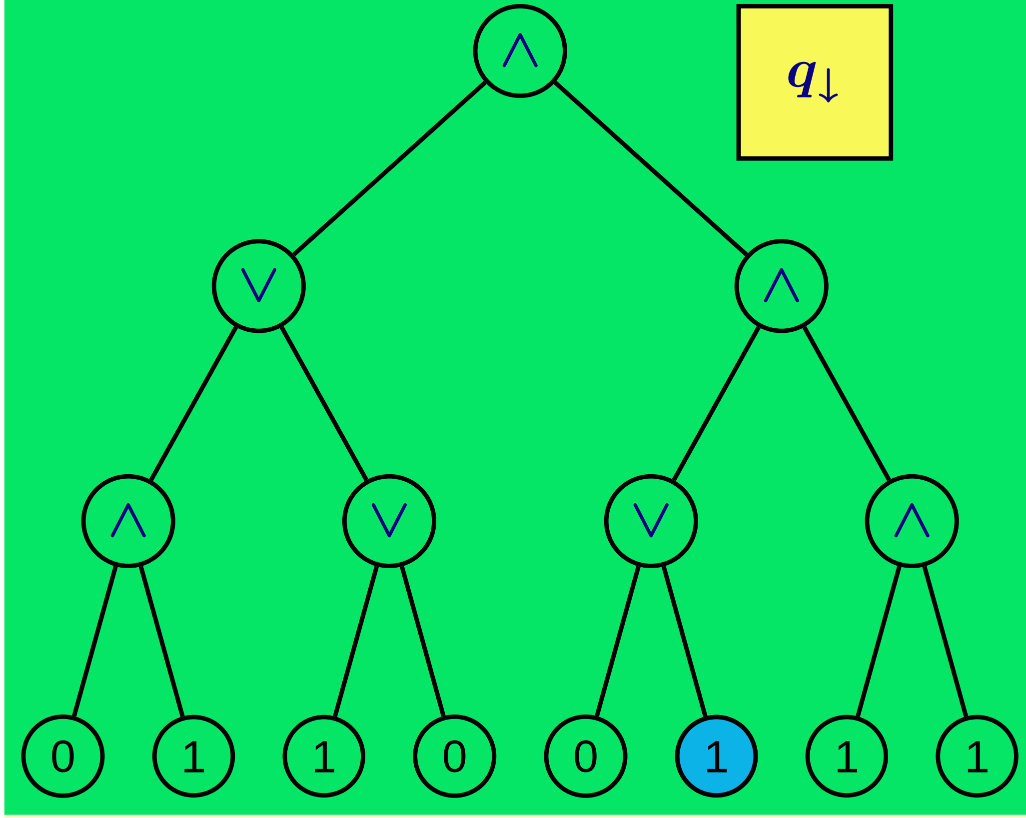


Can sequential automata do anything useful?

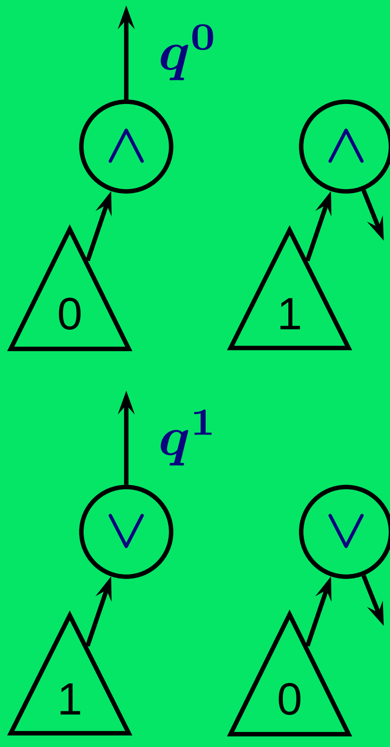
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

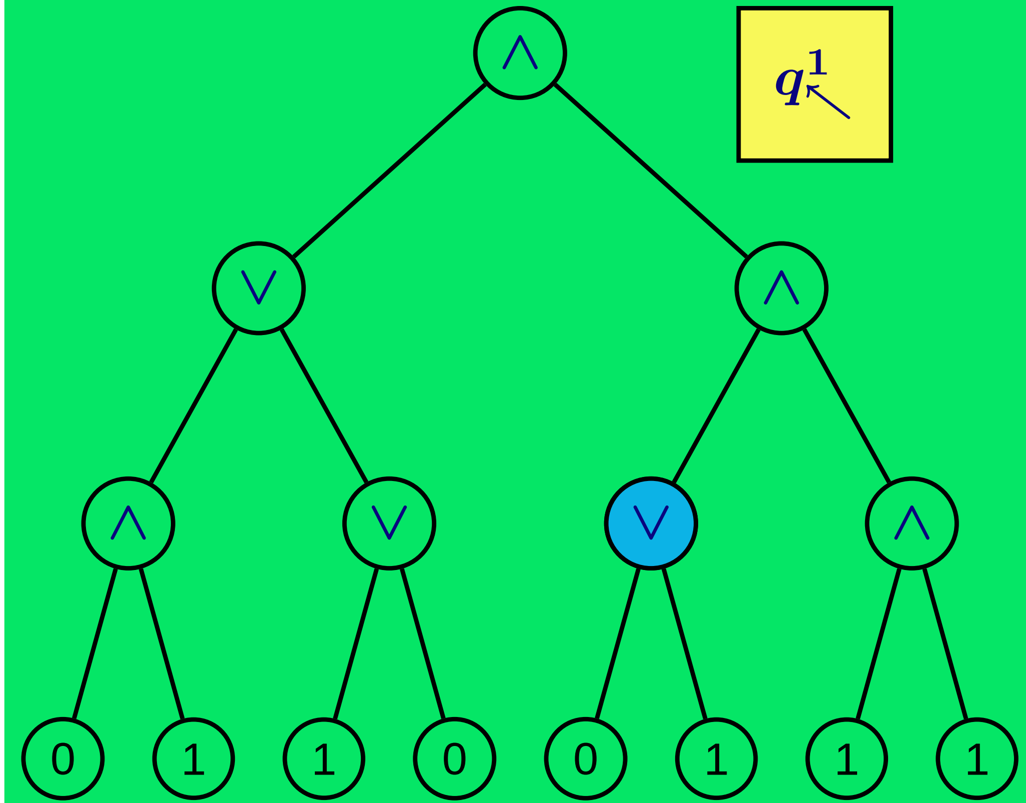


Can sequential automata do anything useful?

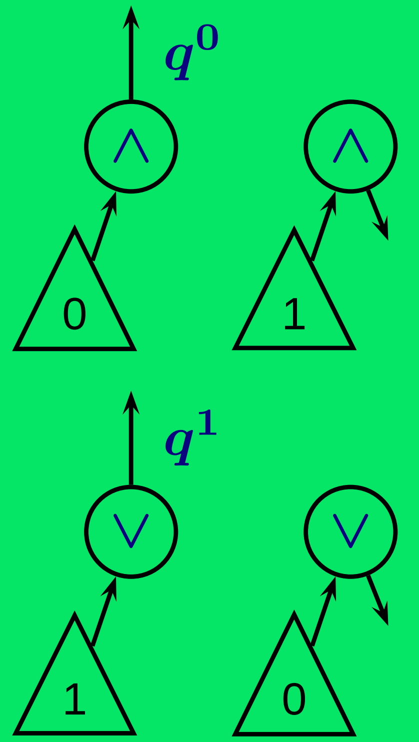
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

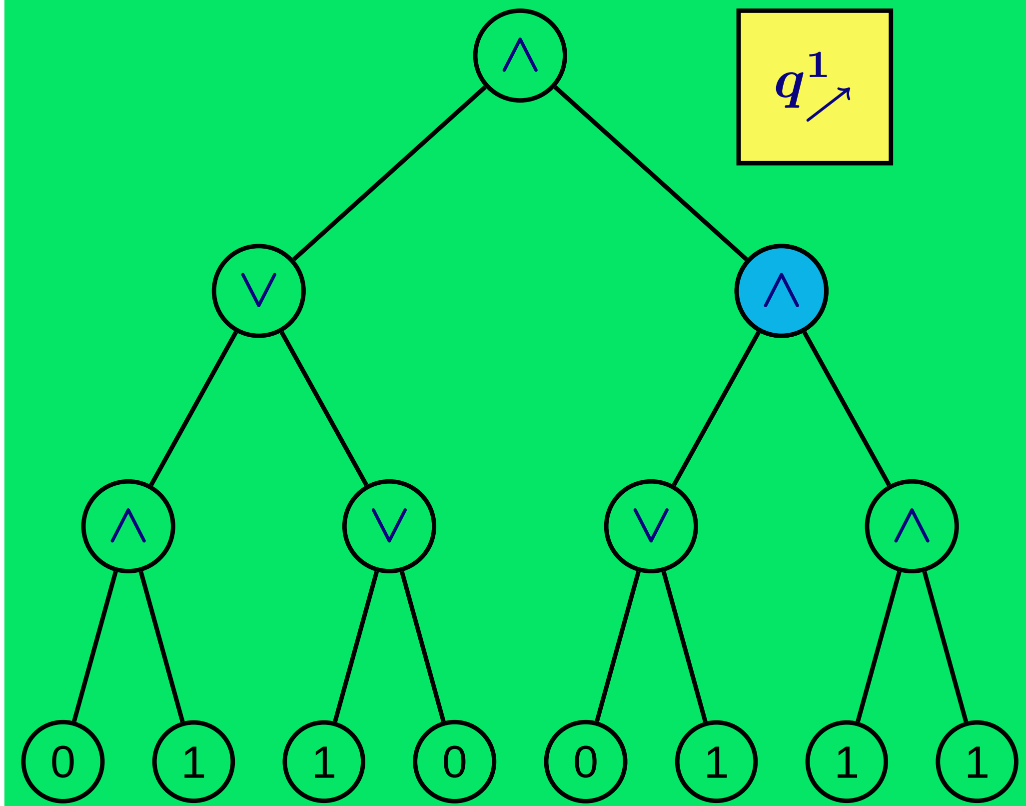


Can sequential automata do anything useful?

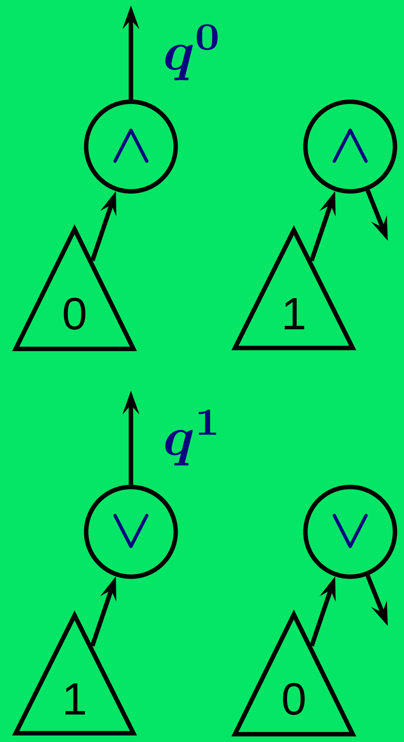
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

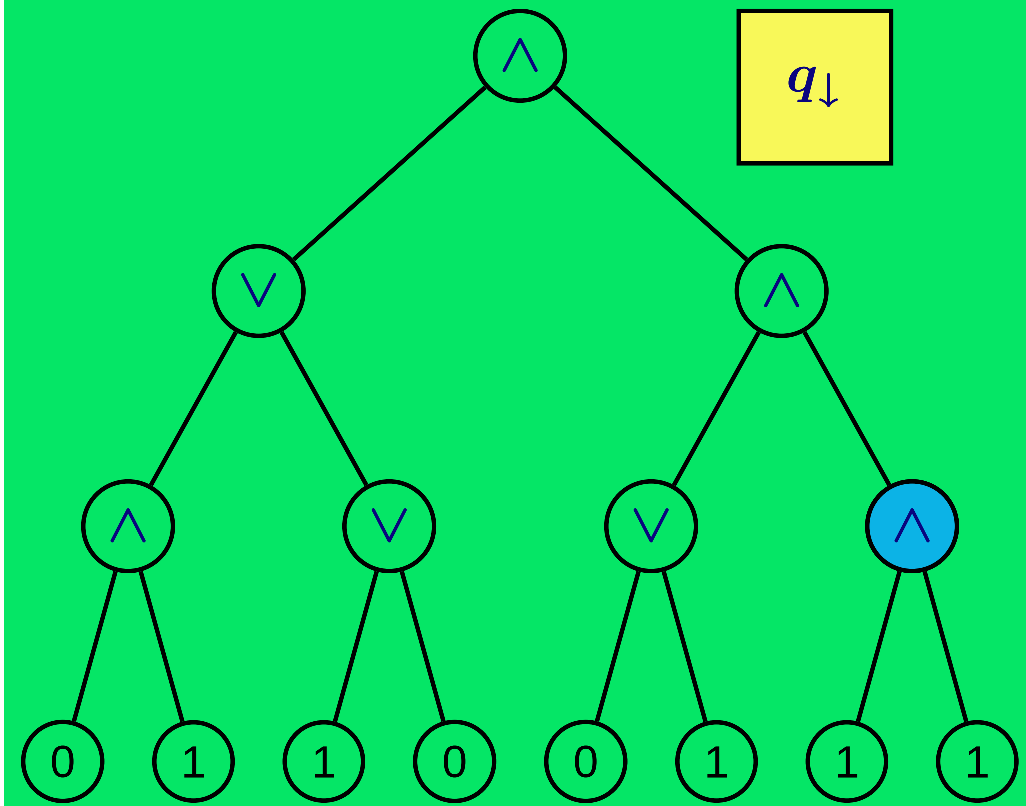


Can sequential automata do anything useful?

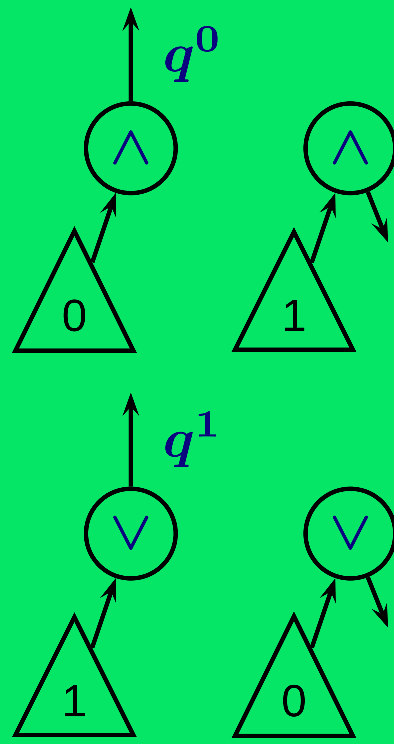
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

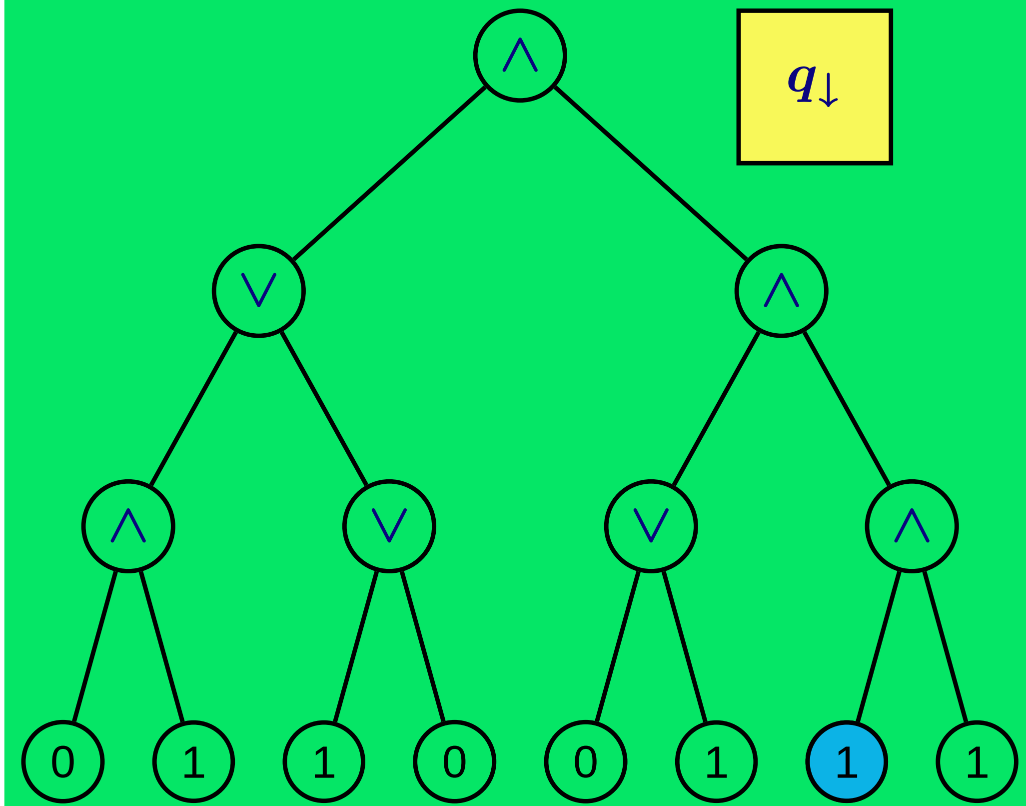


Can sequential automata do anything useful?

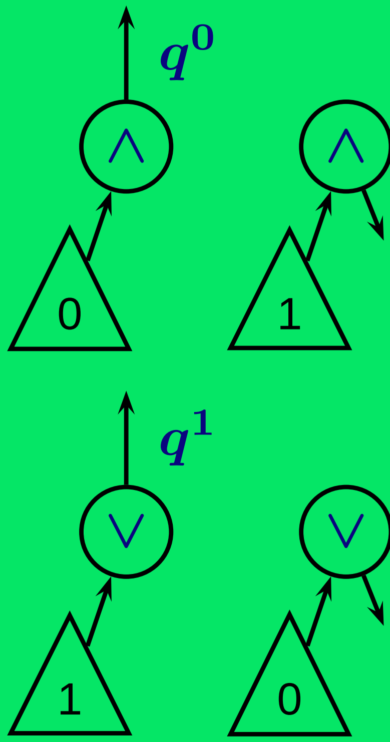
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

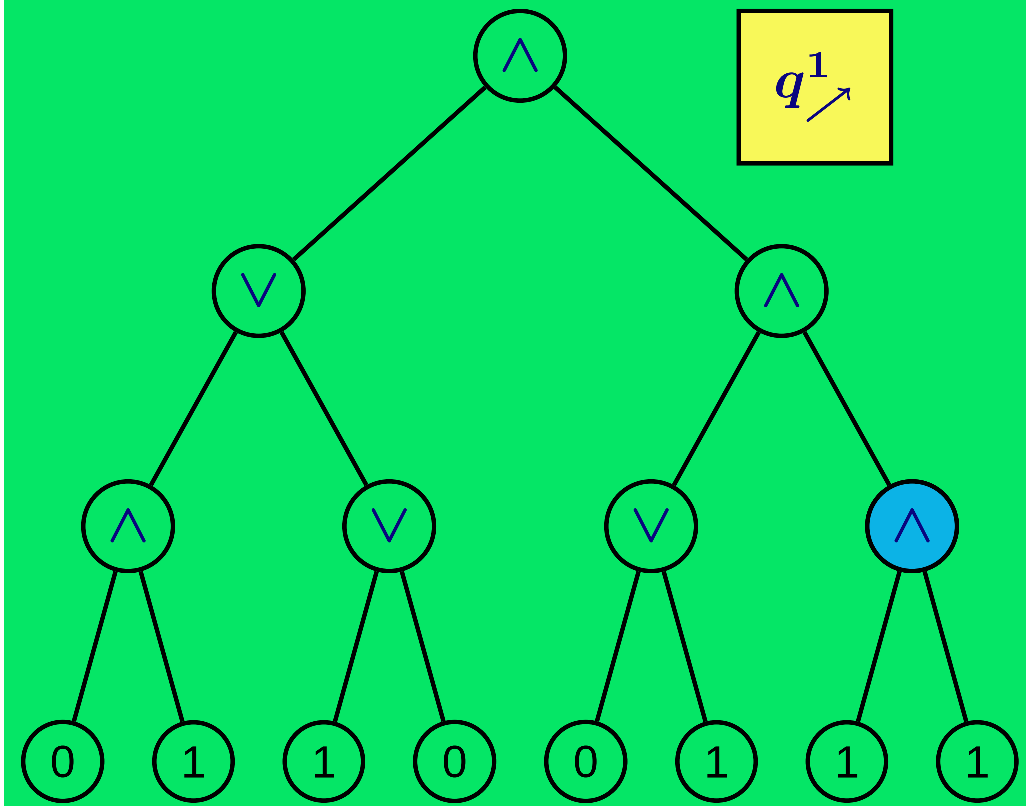


Can sequential automata do anything useful?

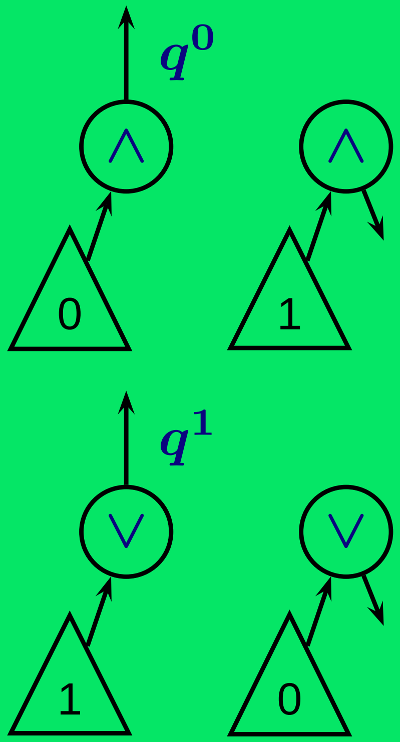
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

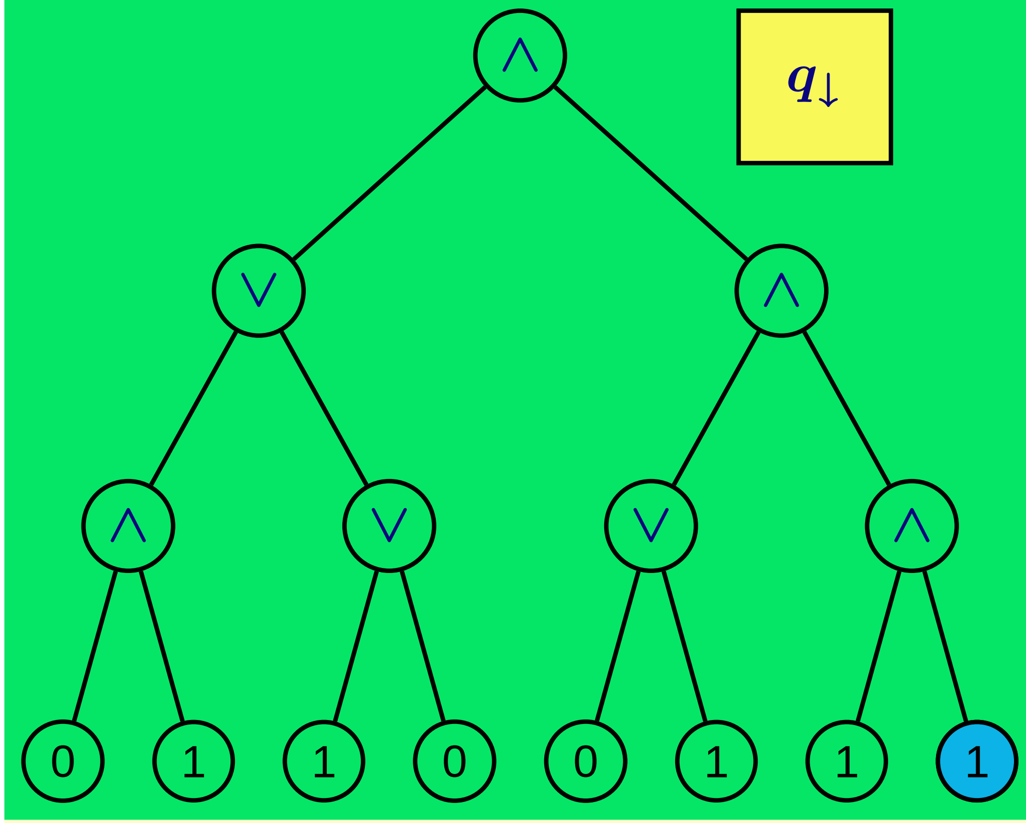


Can sequential automata do anything useful?

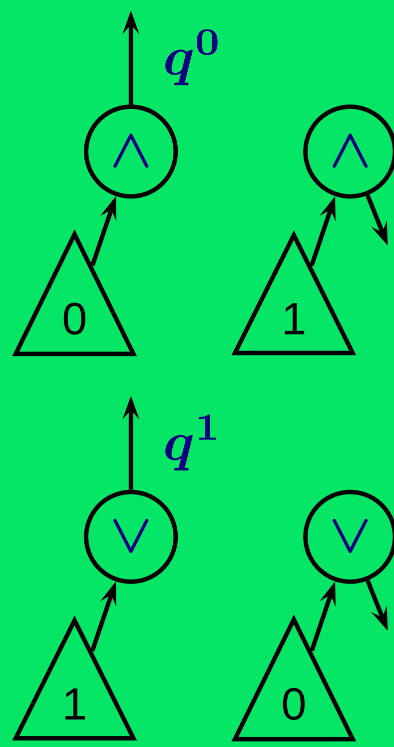
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

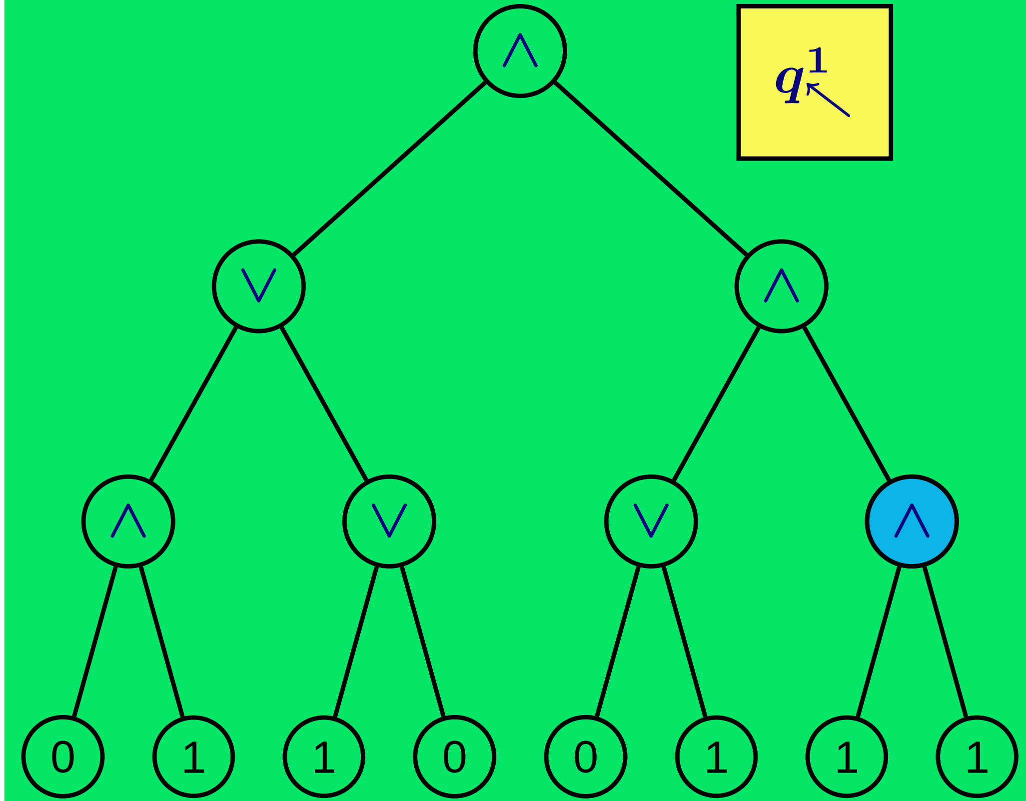


Can sequential automata do anything useful?

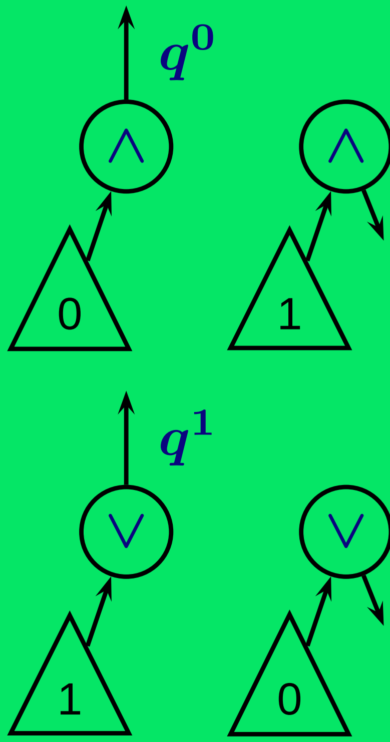
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

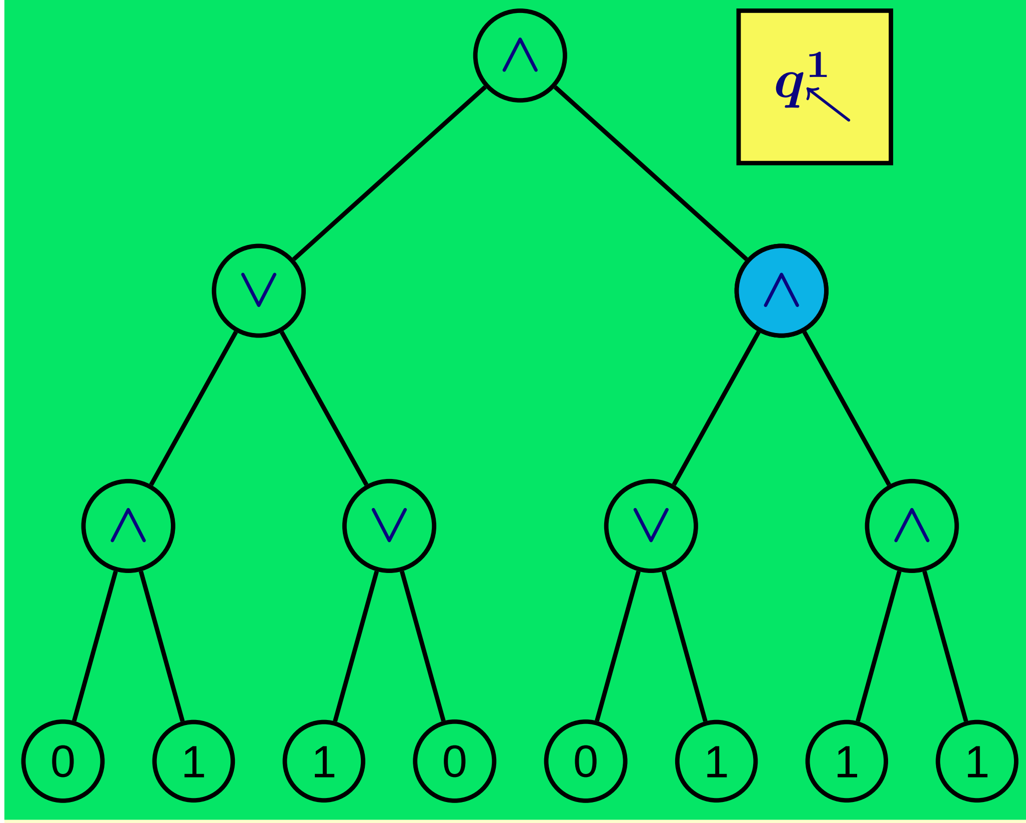


Can sequential automata do anything useful?

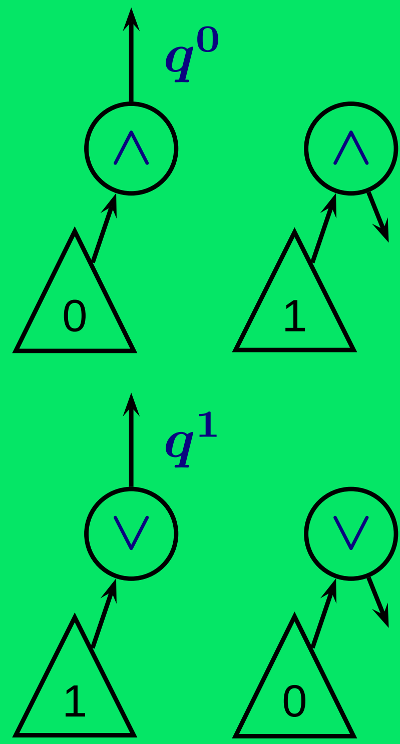
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea

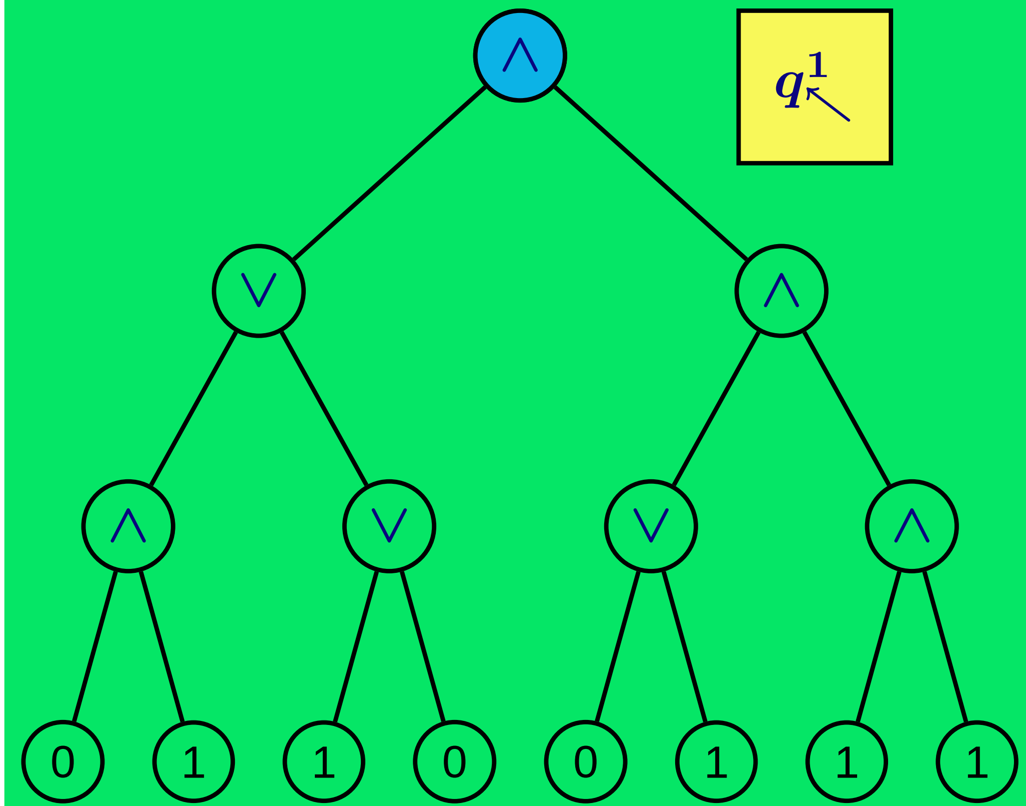


Can sequential automata do anything useful?

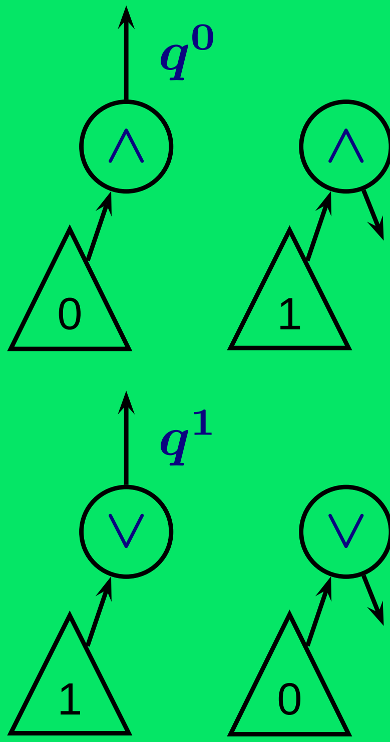
Fact

- Tree-walk automata can evaluate tree-shaped Boolean circuits
- 5 states

Example



Idea



What is the expressive power of sequential tree automata?

- Parallel tree automata ≡ Regular Tree Languages
- Sequential automata are clearly weaker

What is the expressive power of sequential tree automata?

- Parallel tree automata $\equiv$
  Regular Tree Languages
- Sequential automata are clearly weaker
- Clearly?

What is the expressive power of sequential tree automata?

- Parallel tree automata $\equiv$ **Regular Tree Languages**
- Sequential automata are clearly weaker
- Clearly?
- Whether TWA can recognize all regular tree languages had been open for quite some years

What is the expressive power of sequential tree automata?

- Parallel tree automata $\equiv$
  Regular Tree Languages
- Sequential automata are clearly weaker
- Clearly?
- Whether TWA can recognize all regular tree languages had been open for quite some years

Theorem [Bojanczyk, Colcombet 05]

- Deterministic TWAs are weaker than nondeterministic TWAs
- Nondeterminstic TWAs do not capture all regular tree languages

What is the expressive power of sequential tree automata?

- Parallel tree automata $\equiv$
  Regular Tree Languages
- Sequential automata are clearly weaker
- Clearly?
- Whether TWA can recognize all regular tree languages had been open for quite some years

Theorem [Bojanczyk, Colcombet 05]

- Deterministic TWAs are weaker than nondeterministic TWAs
- Nondeterminstic TWAs do not capture all regular tree languages

- Engelfriet suggested an extension of TWAs:

What happens if we add...

What is the expressive power of sequential tree automata?

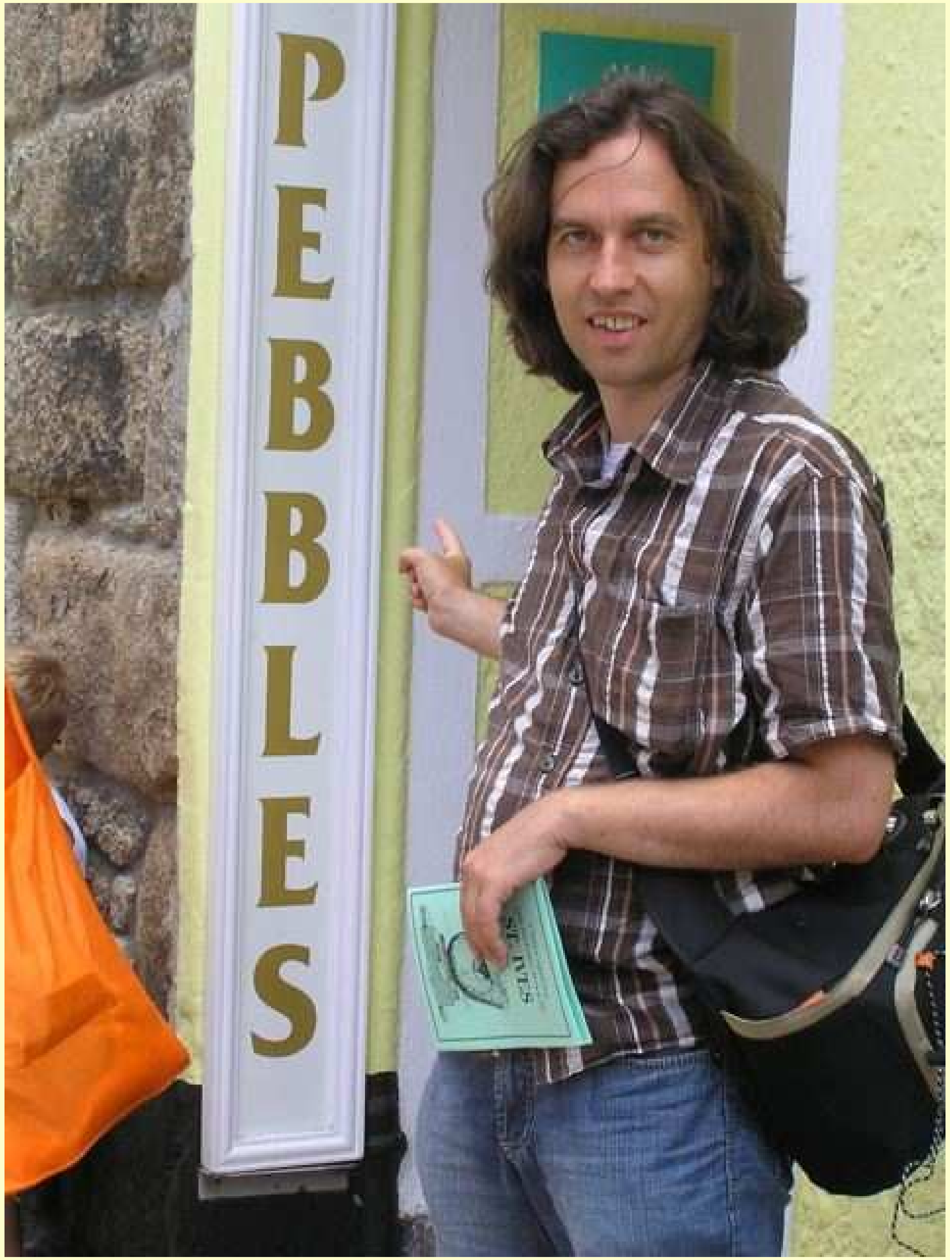
- Parallel tree automata $\equiv$ **Regular Tree Languages**
- Sequential automata are clearly weaker
- Clearly?
- Whether TWA can recognize all regular tree languages had been open for quite some years

Theorem [Bojanczyk, Colcombet 05]

- Deterministic TWAs are weaker than nondeterministic TWAs
- Nondeterminstic TWAs do not capture all regular tree languages

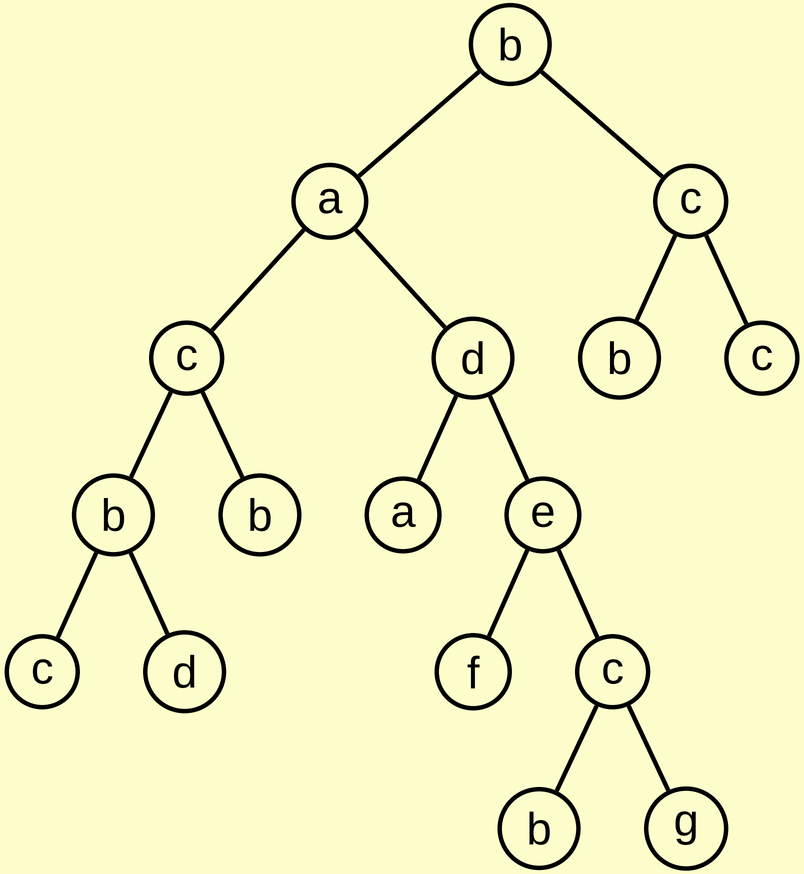
- Engelfriet suggested an extension of TWAs:

What happens if we add...



Contents

Pebble automata



Definition

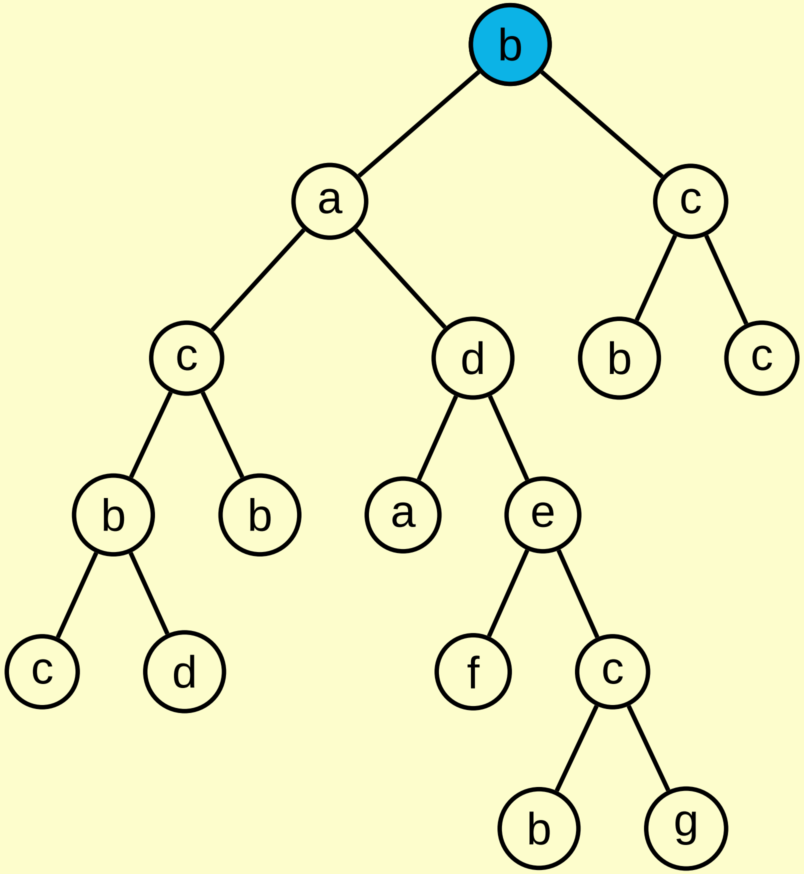
Pebble automata:

- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:
    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

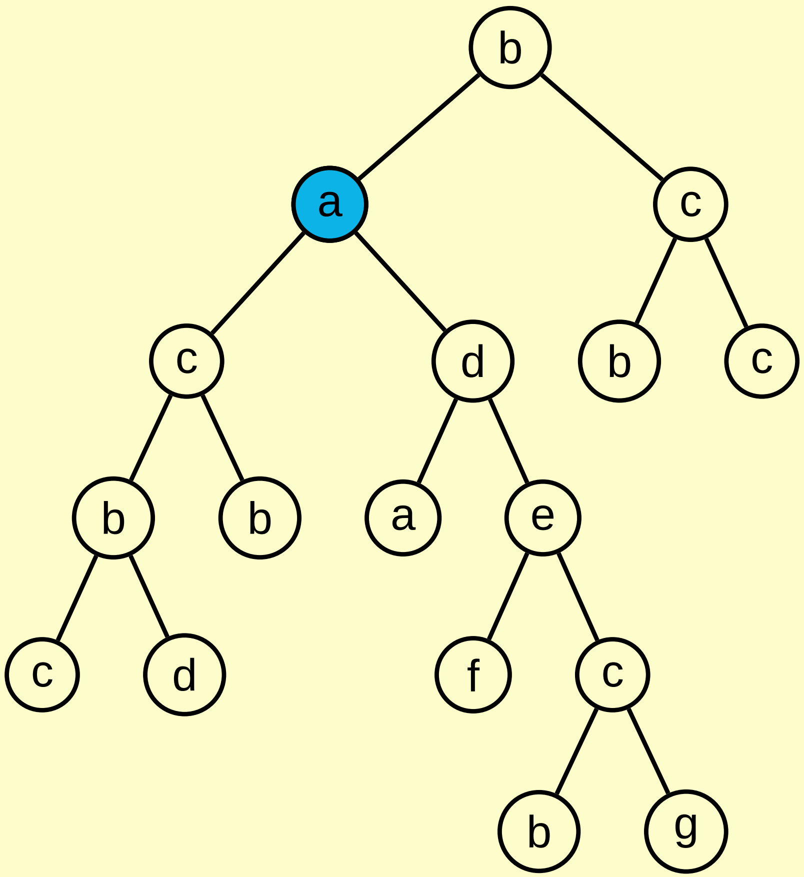
Pebble automata:

- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:
    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

Pebble automata:

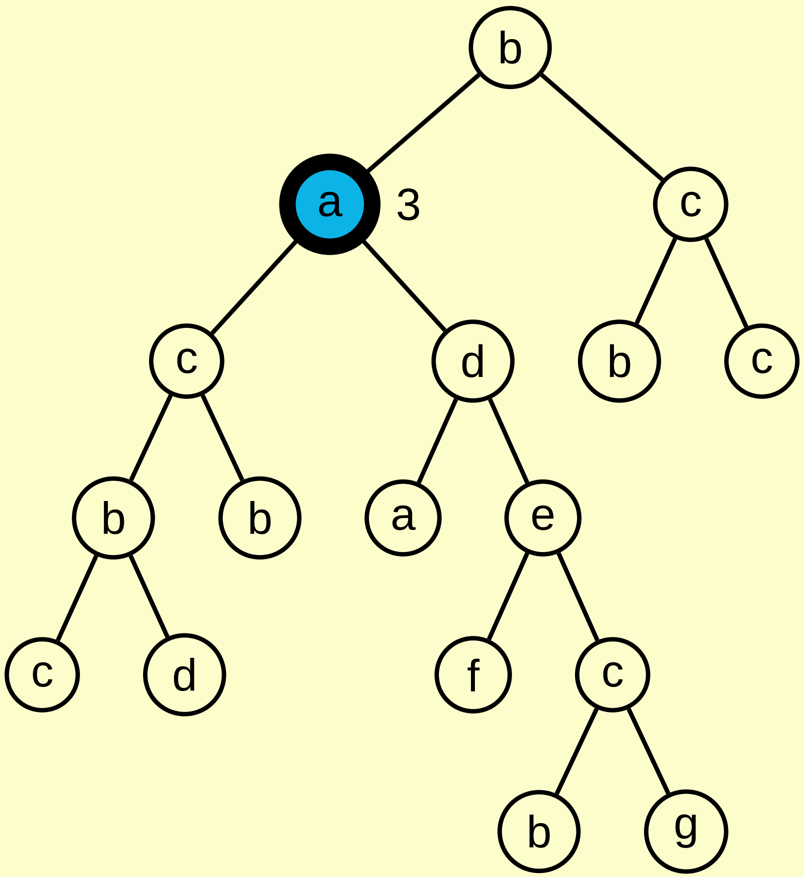
- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

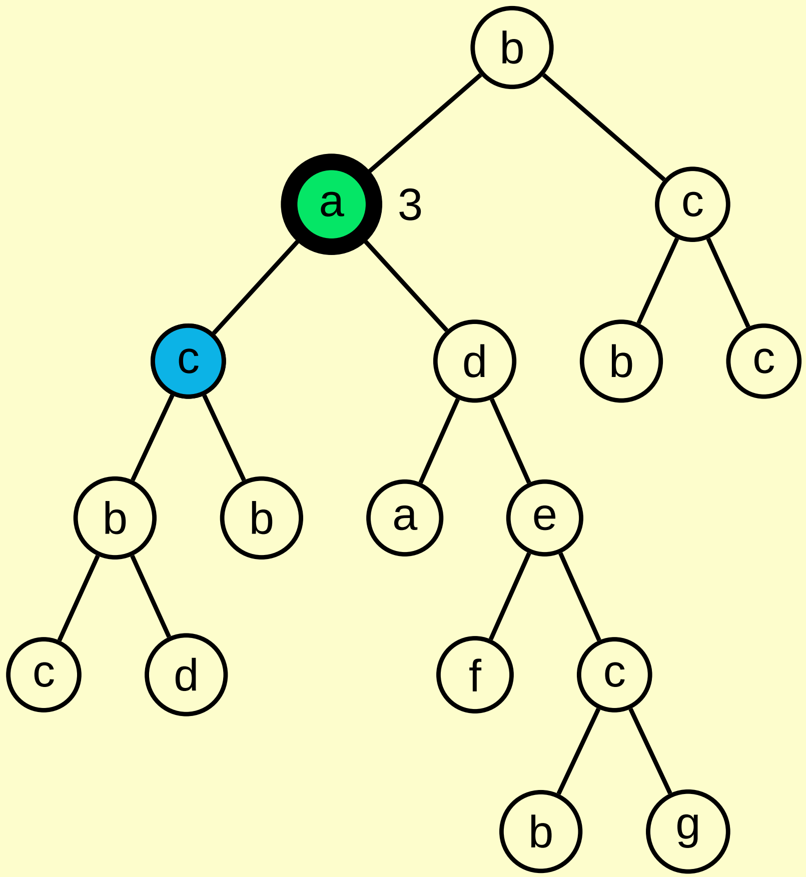
Pebble automata:

- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:
    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

Pebble automata:

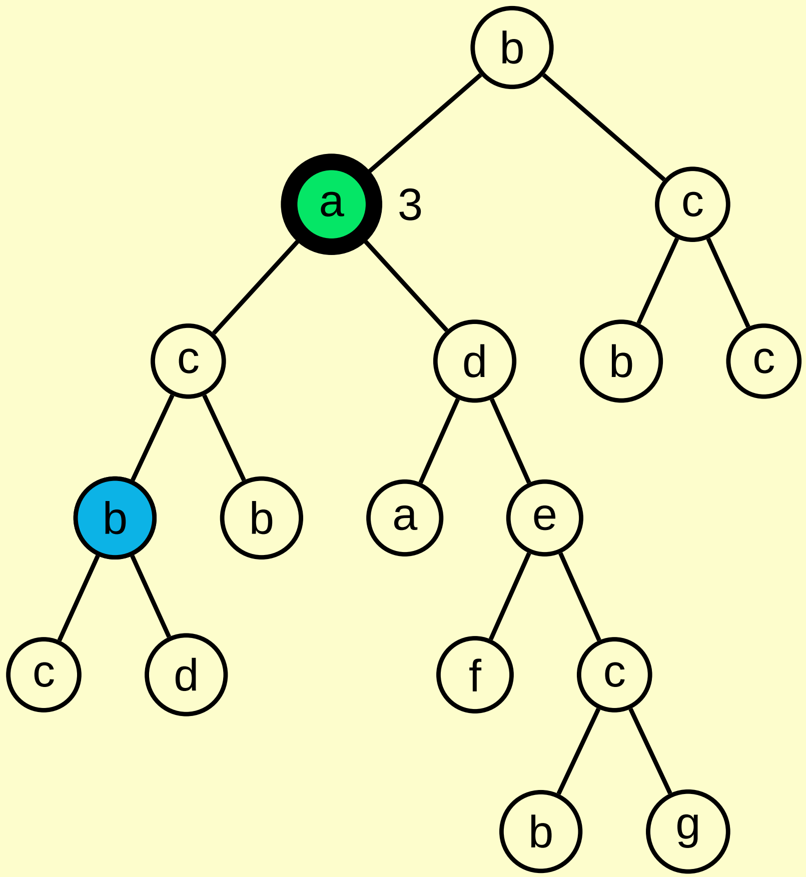
- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

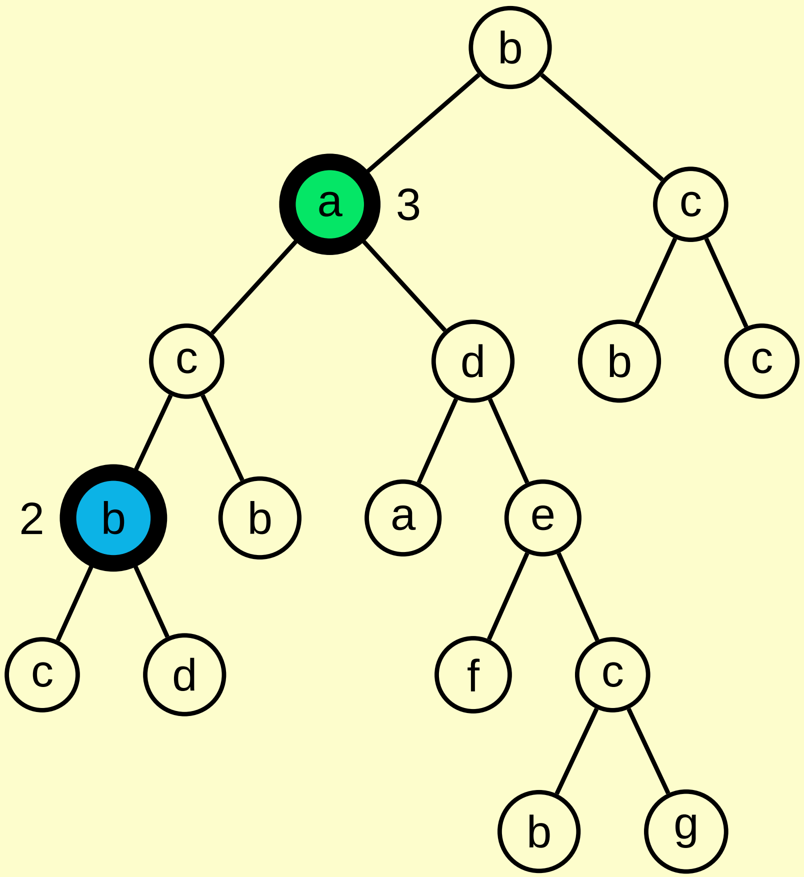
Pebble automata:

- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:
    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

Pebble automata:

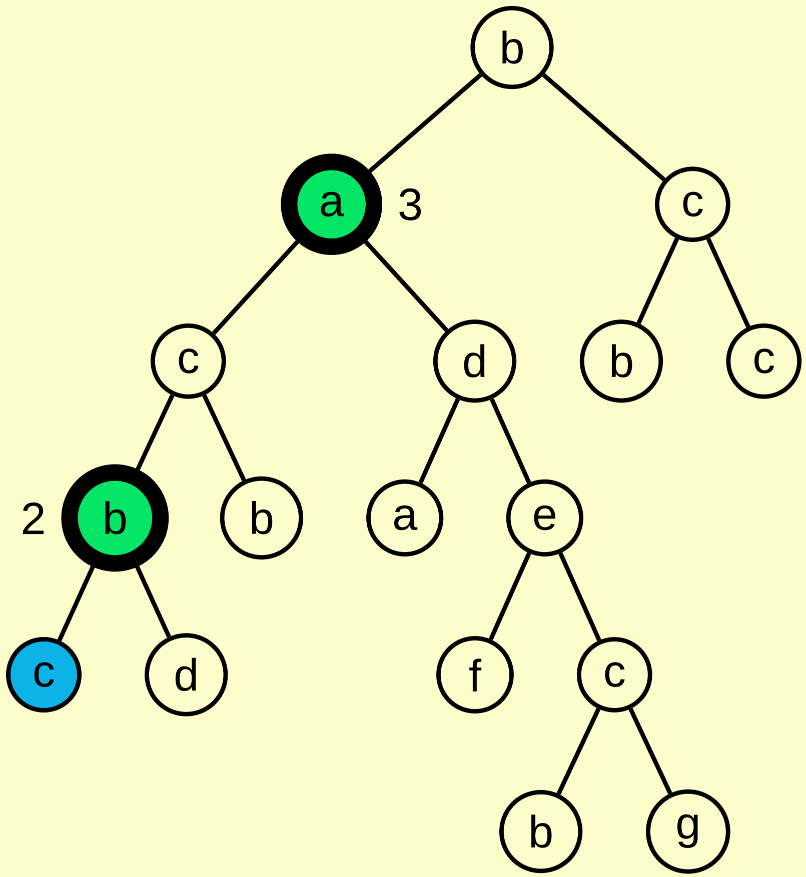
- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

Pebble automata:

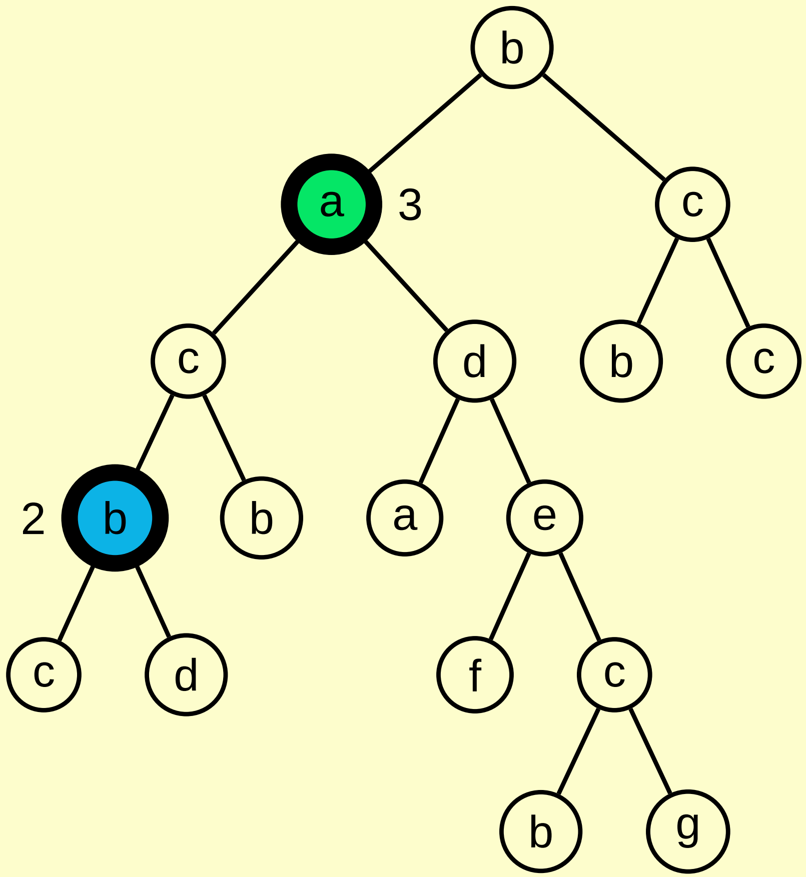
- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

Pebble automata:

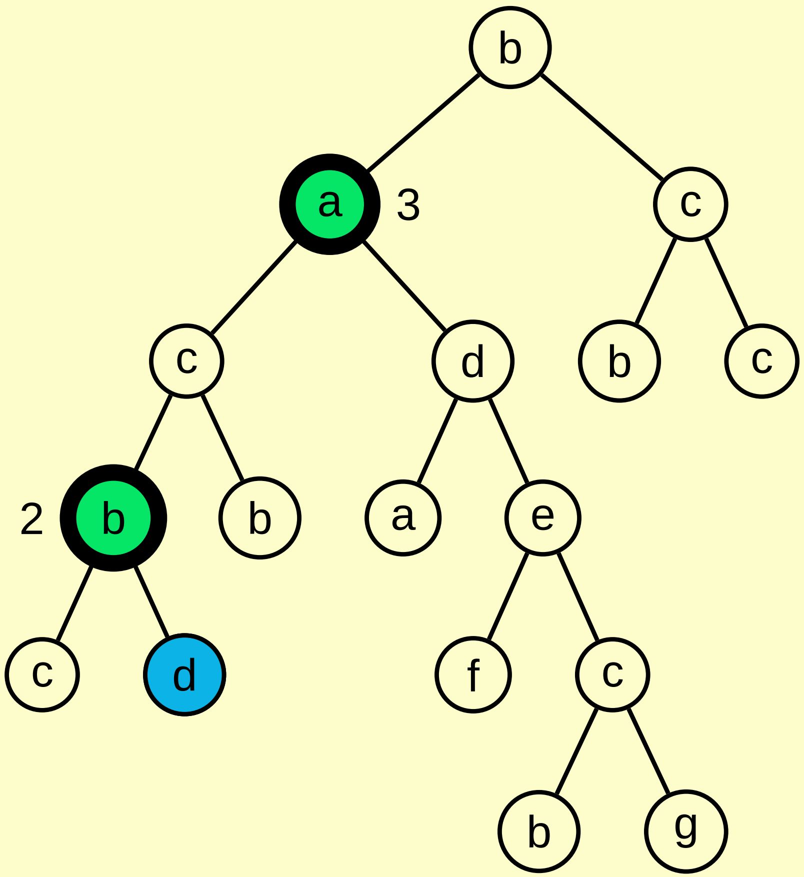
- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

Pebble automata:

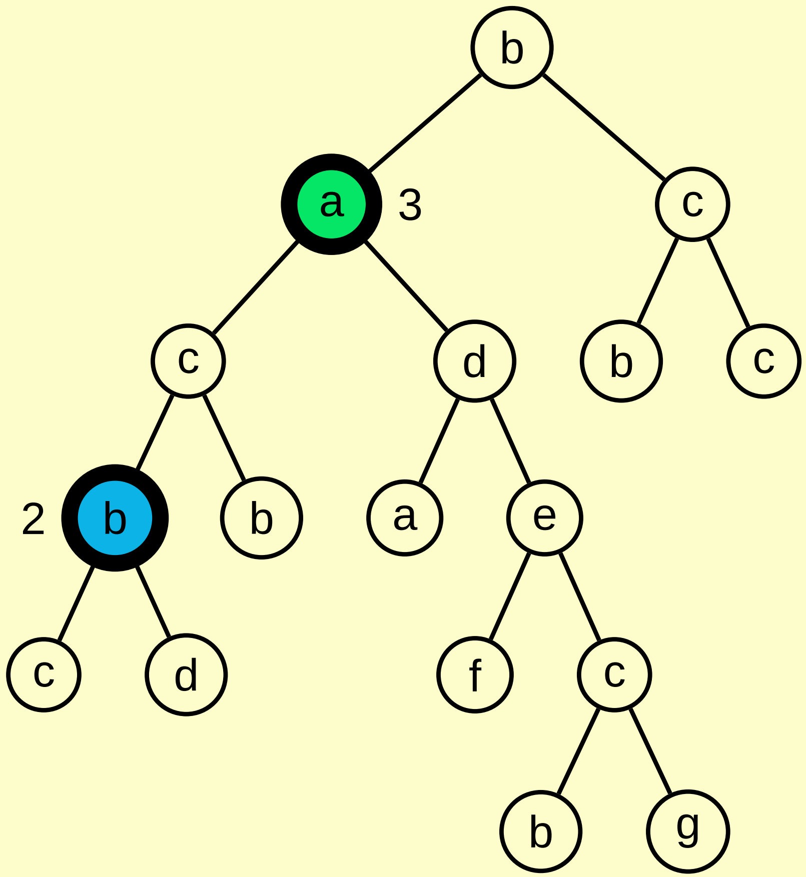
- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

Pebble automata:

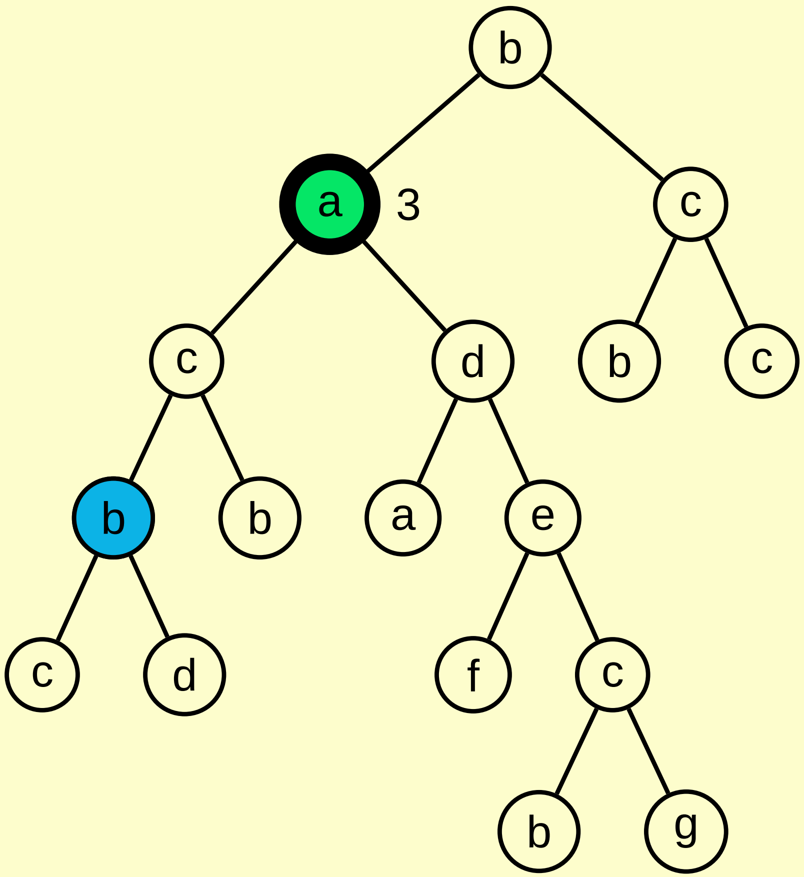
- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

Pebble automata:

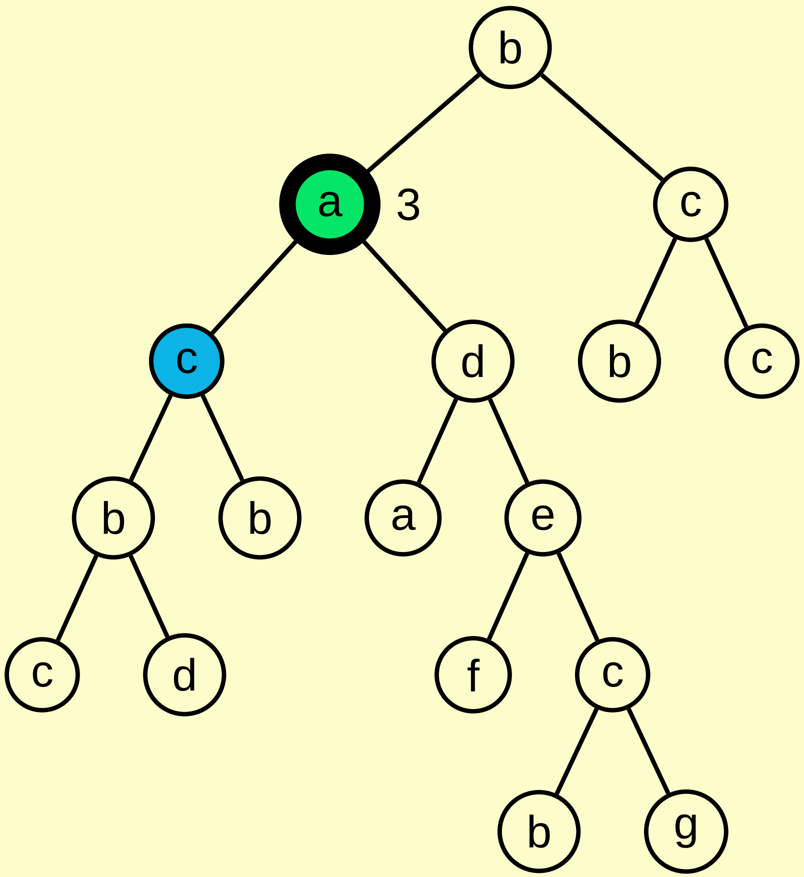
- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

Pebble automata:

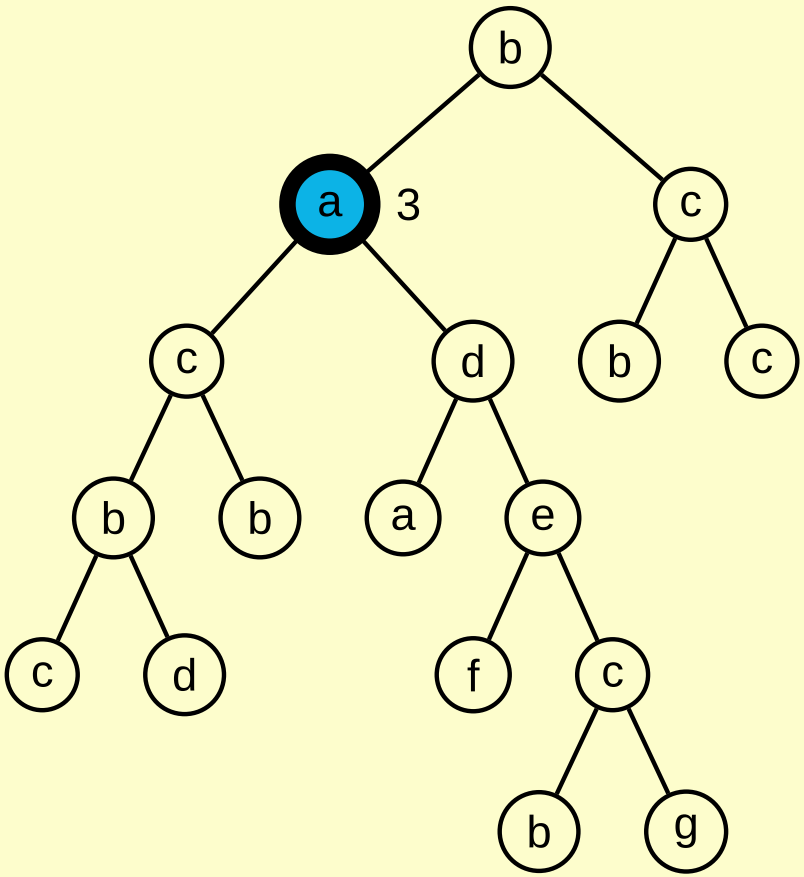
- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

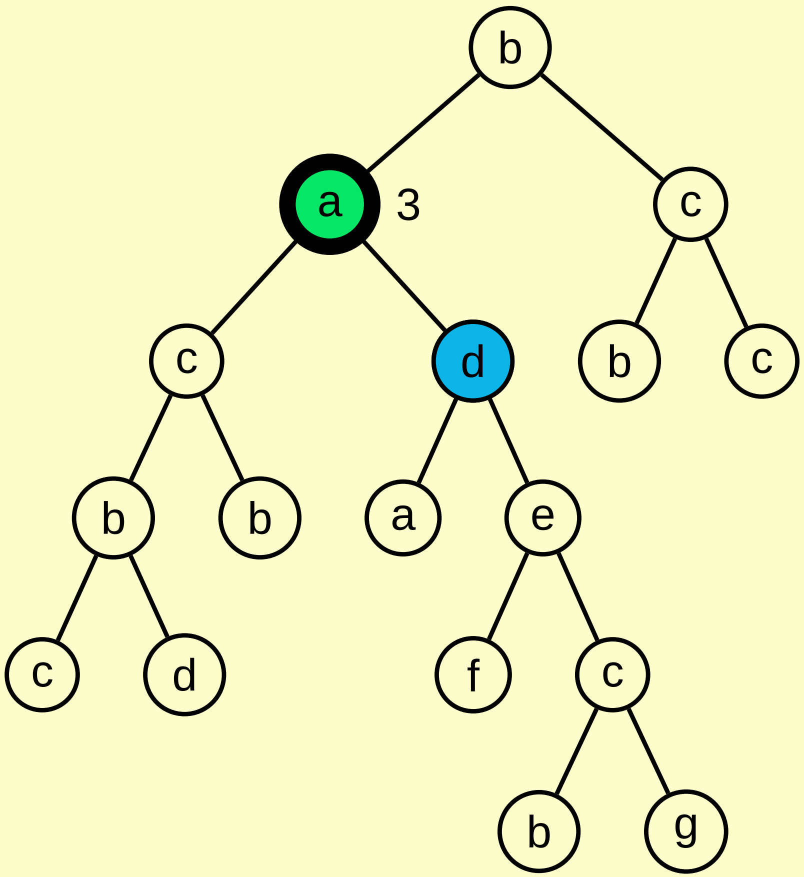
Pebble automata:

- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow$,stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:
    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

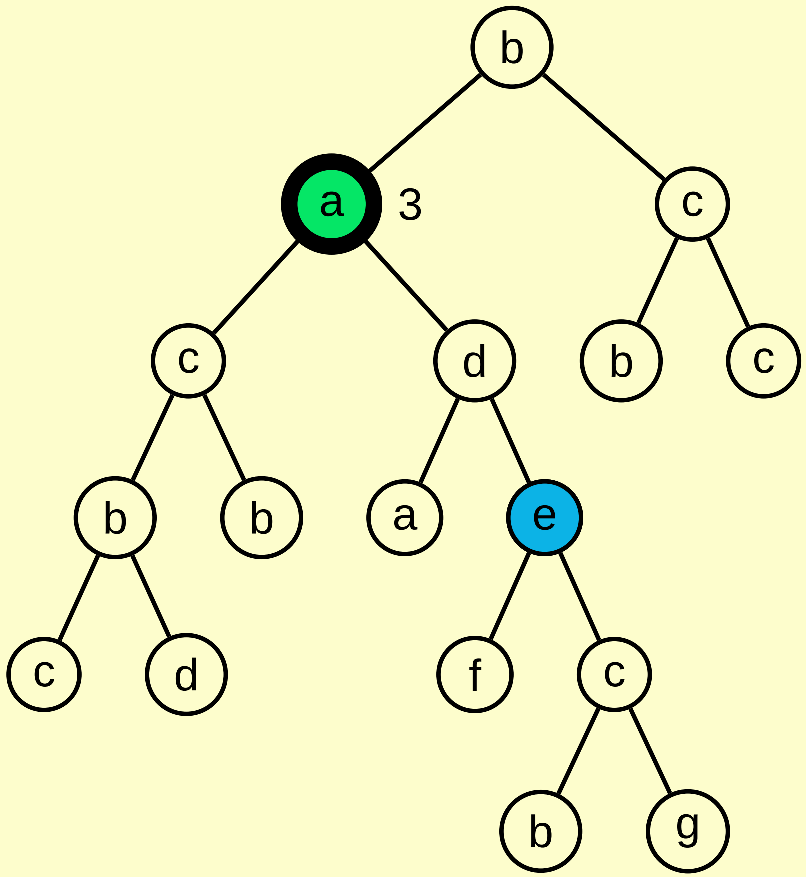
Pebble automata:

- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:
    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

Pebble automata:

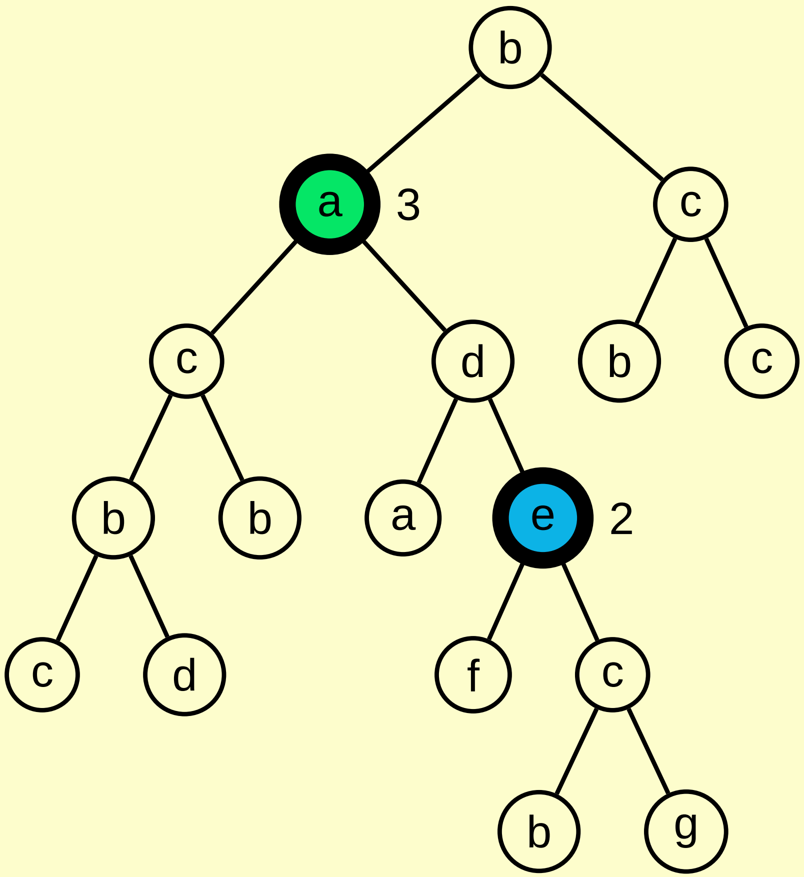
- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow, \text{stay}, \text{drop}, \text{lift}\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{\text{root}, \text{leftchild}, \text{rightchild}\} \times \{\text{leaf}, \text{innernode}\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

Pebble automata:

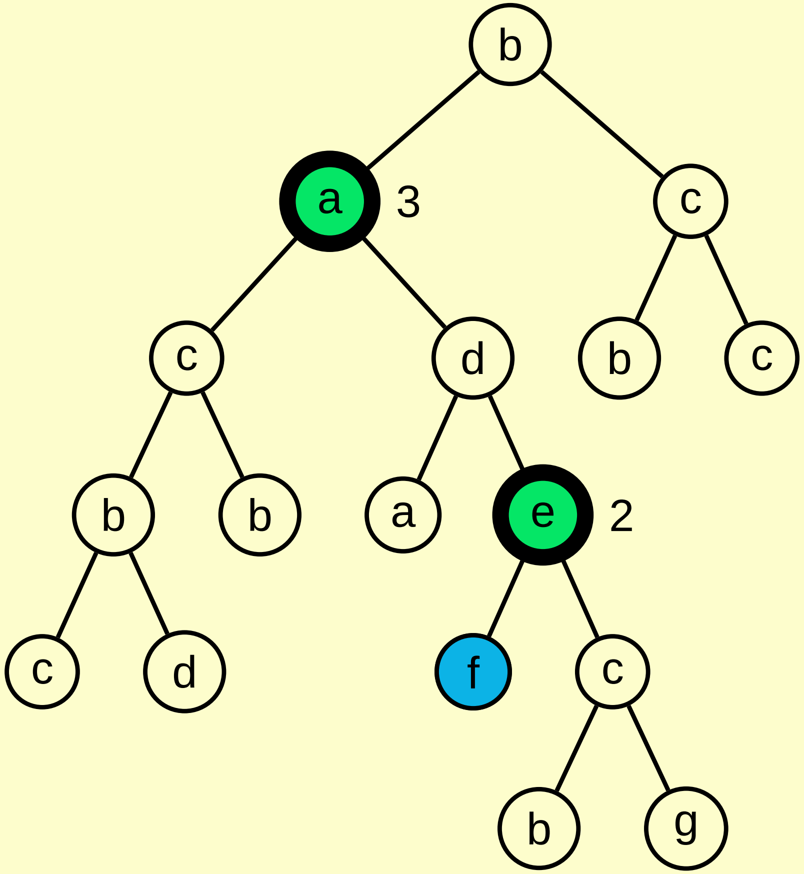
- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

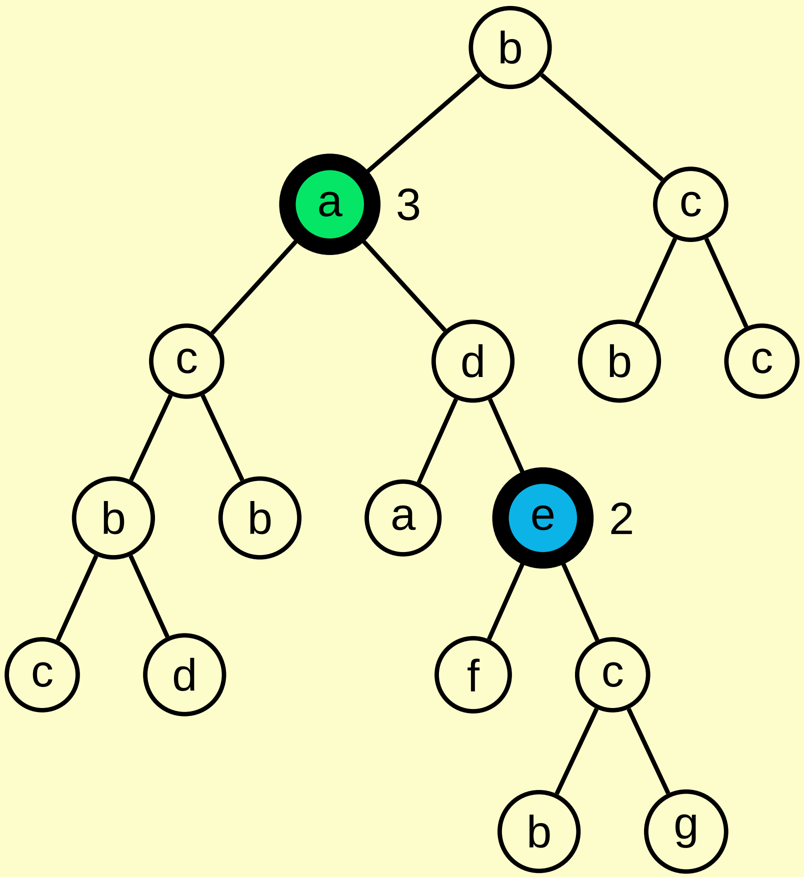
Pebble automata:

- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow$,stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:
    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

Pebble automata:

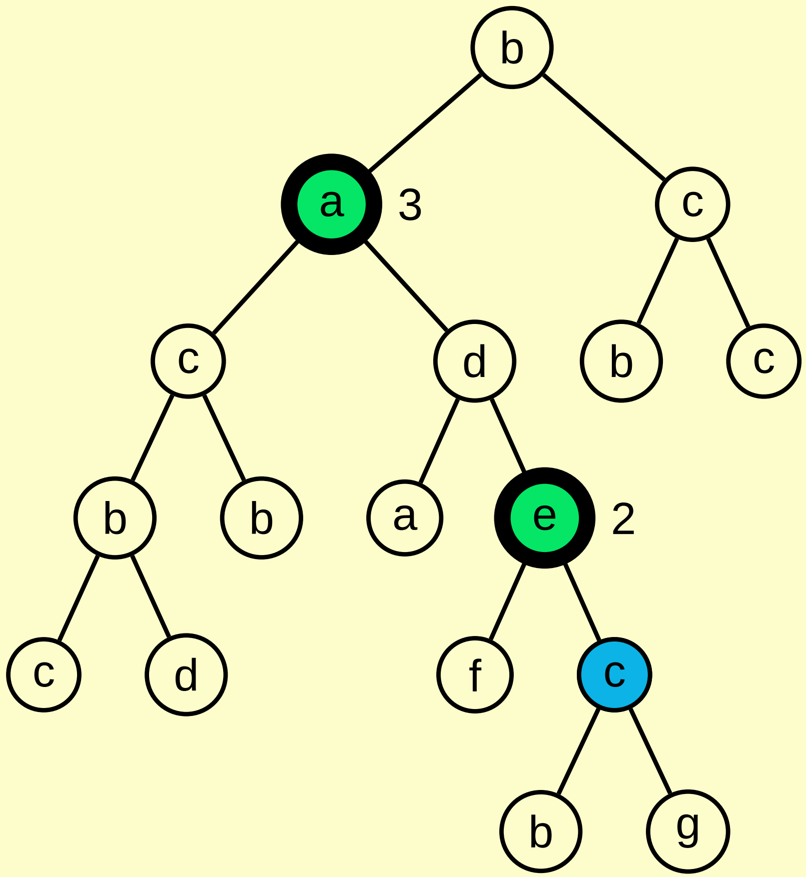
- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

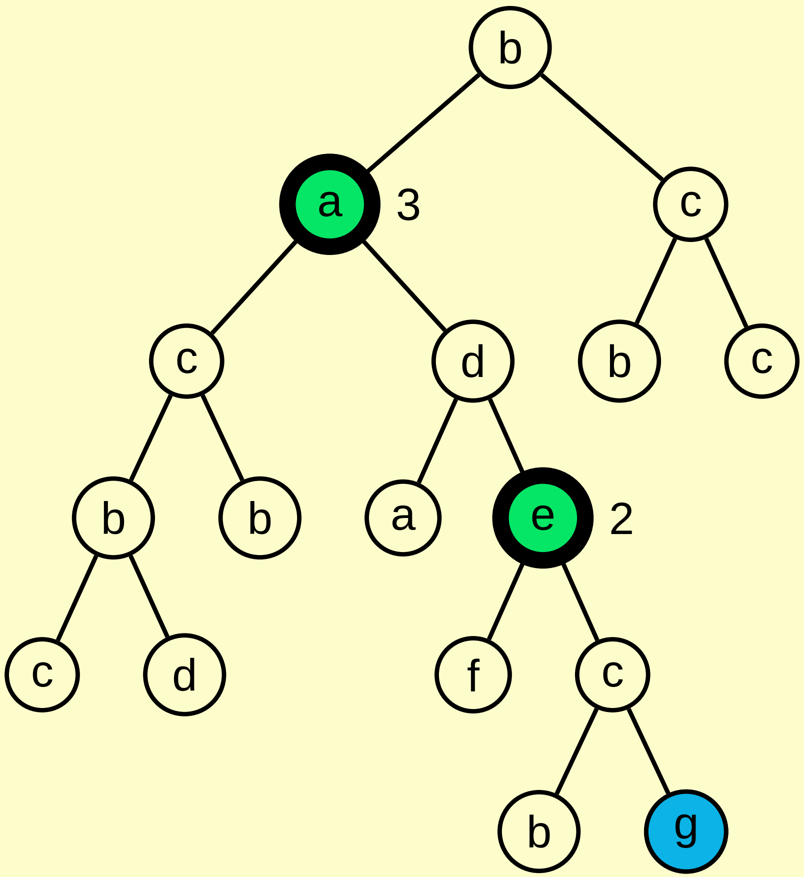
Pebble automata:

- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:
    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

Pebble automata:

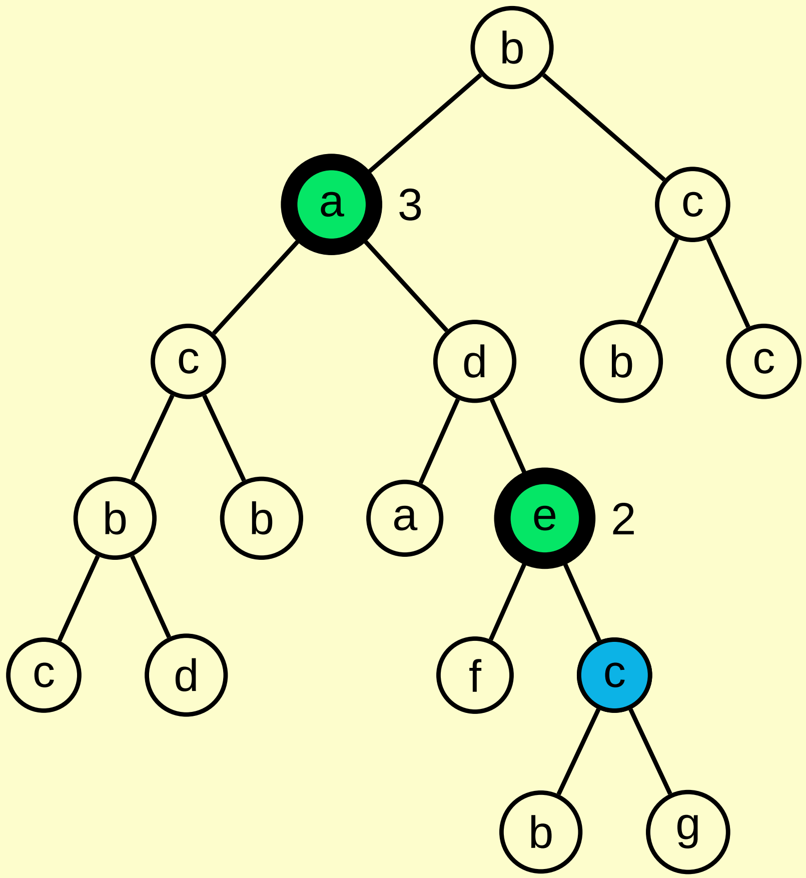
- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\}$ $\times$ $\{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

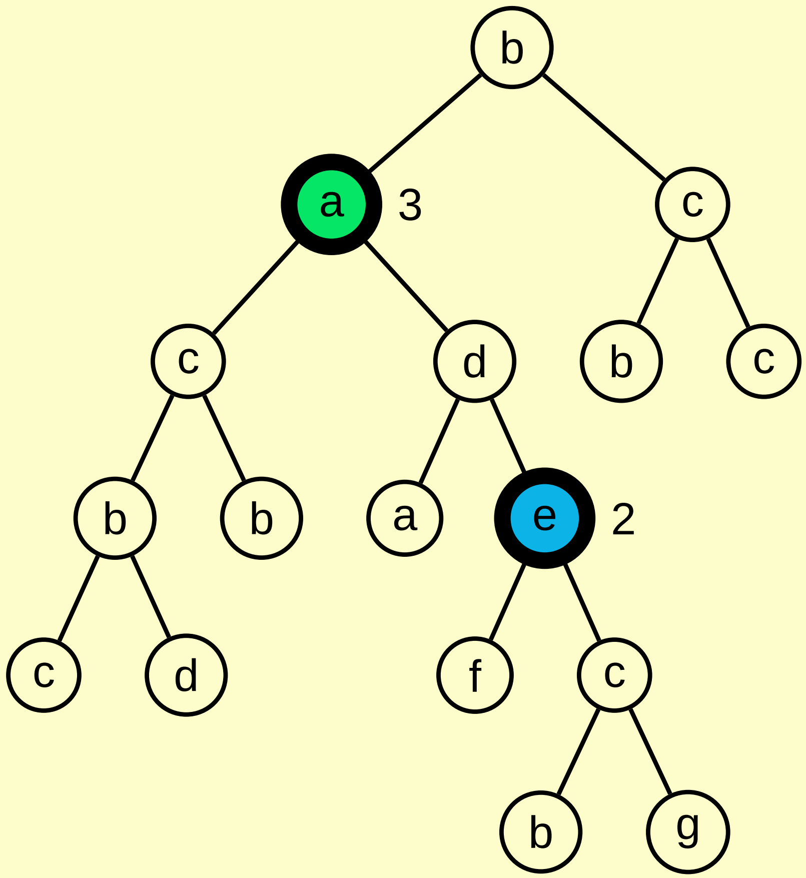
Pebble automata:

- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow, \text{stay,drop,lift}\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node: $\{\text{root,leftchild,rightchild}\} \times \{\text{leaf,innernode}\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong **D**eterministic pebble automata with n pebbles

Pebble automata



Definition

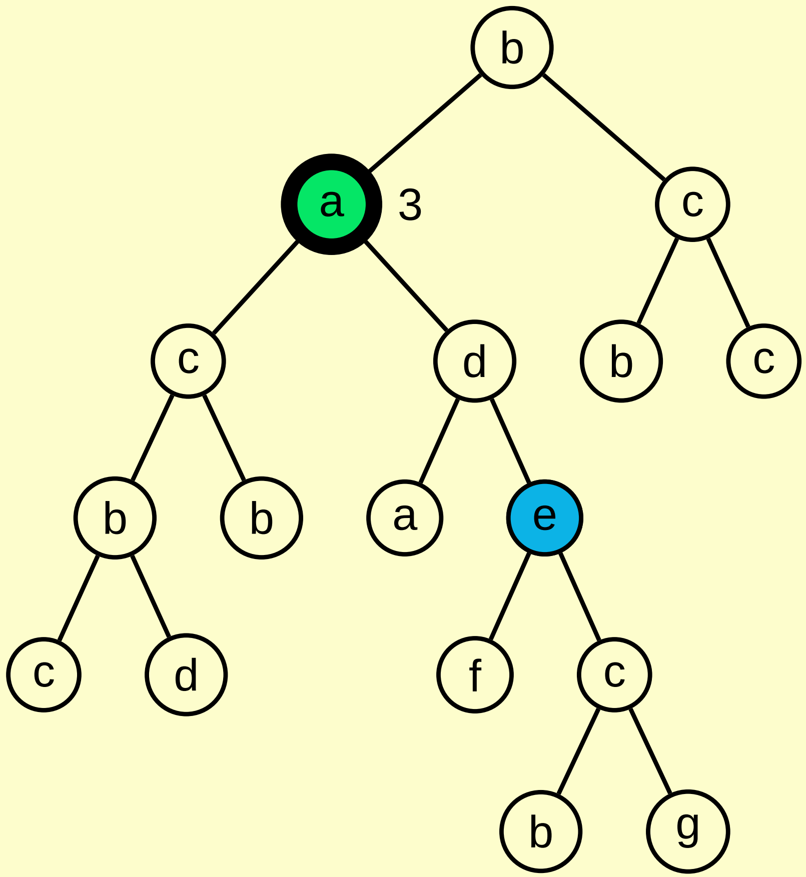
Pebble automata:

- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:
    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

Pebble automata:

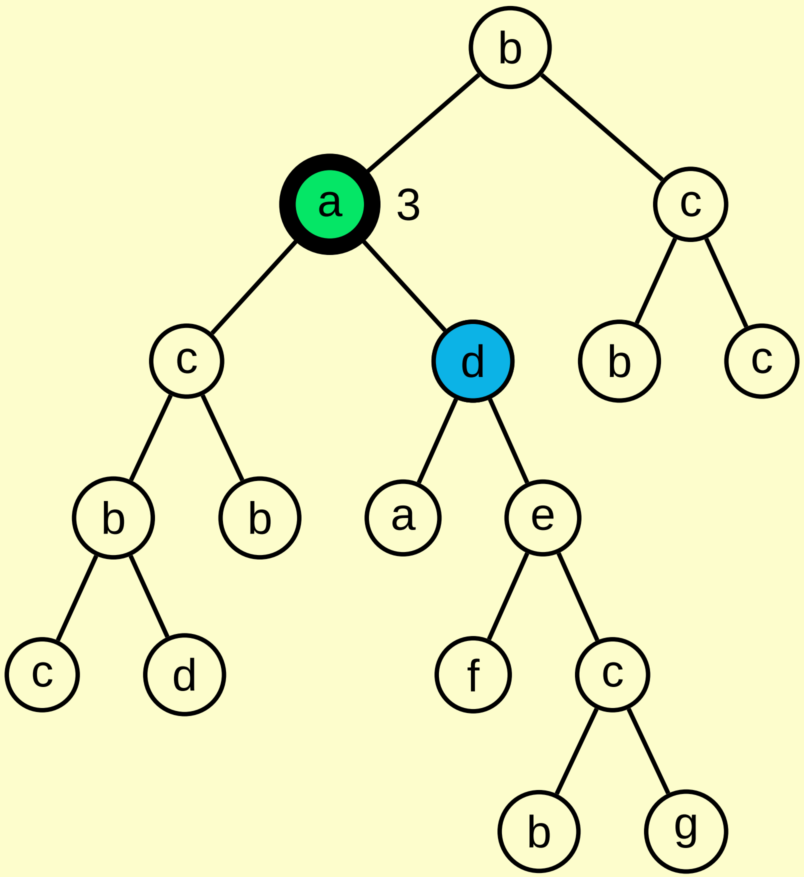
- states $\boldsymbol{Q}$
- pebbles $\boldsymbol{n, \ldots, 1}$
- Possible moves: $\boldsymbol{D = \{\uparrow, \searrow, \swarrow,}$stay,drop,lift$\boldsymbol{\}}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with $\boldsymbol{n}$ pebbles

Pebble automata



Definition

Pebble automata:

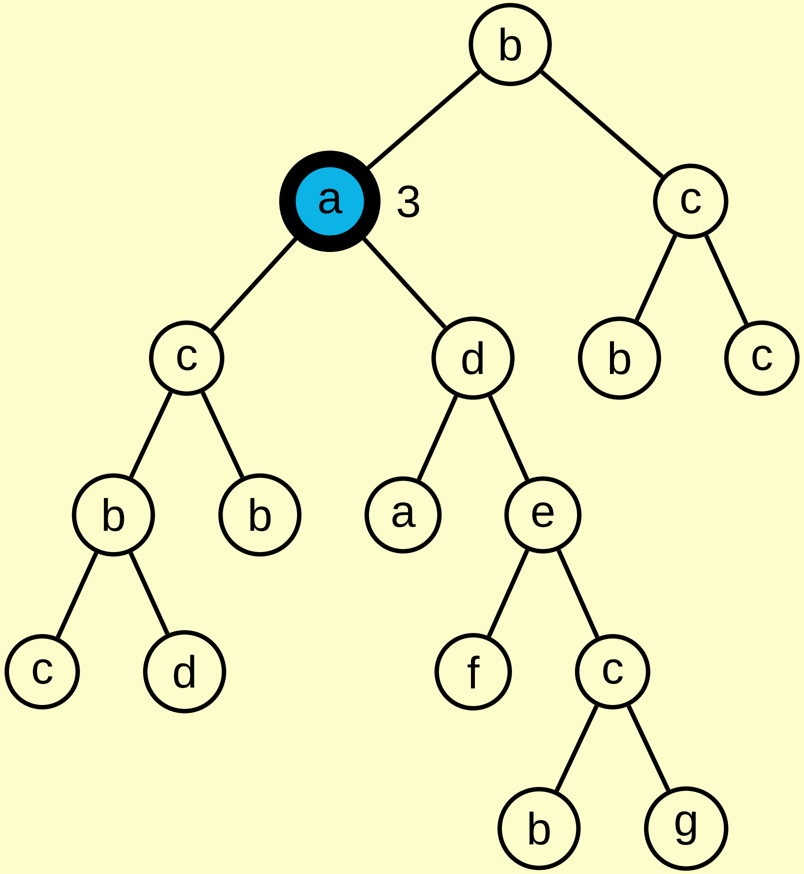
- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow$,stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



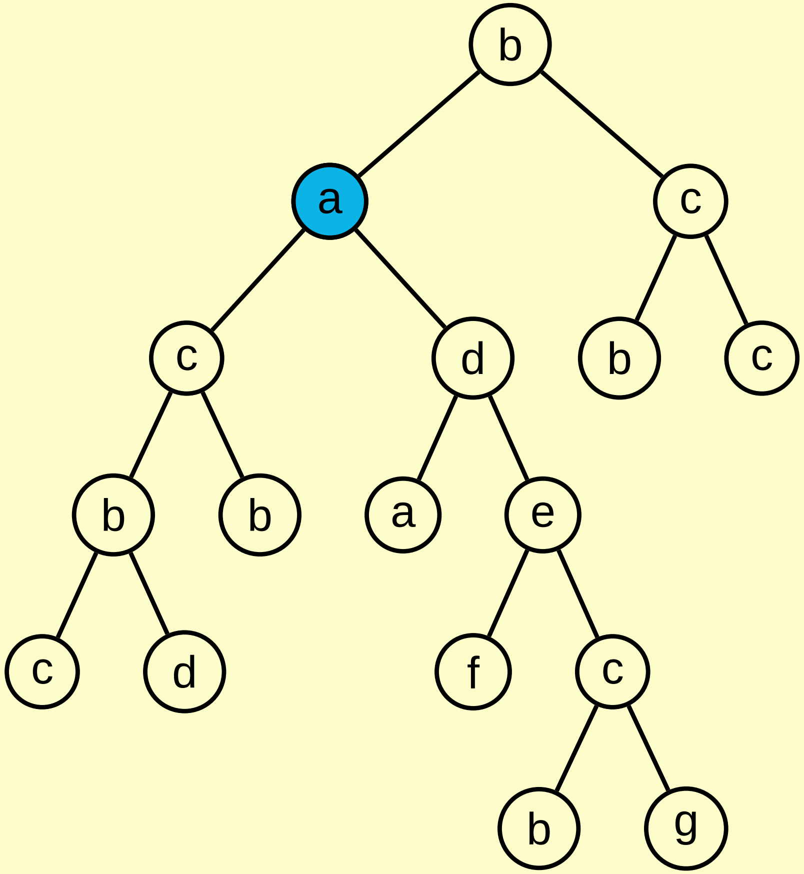
Definition

Pebble automata:

- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node: $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles



Definition

Pebble automata:

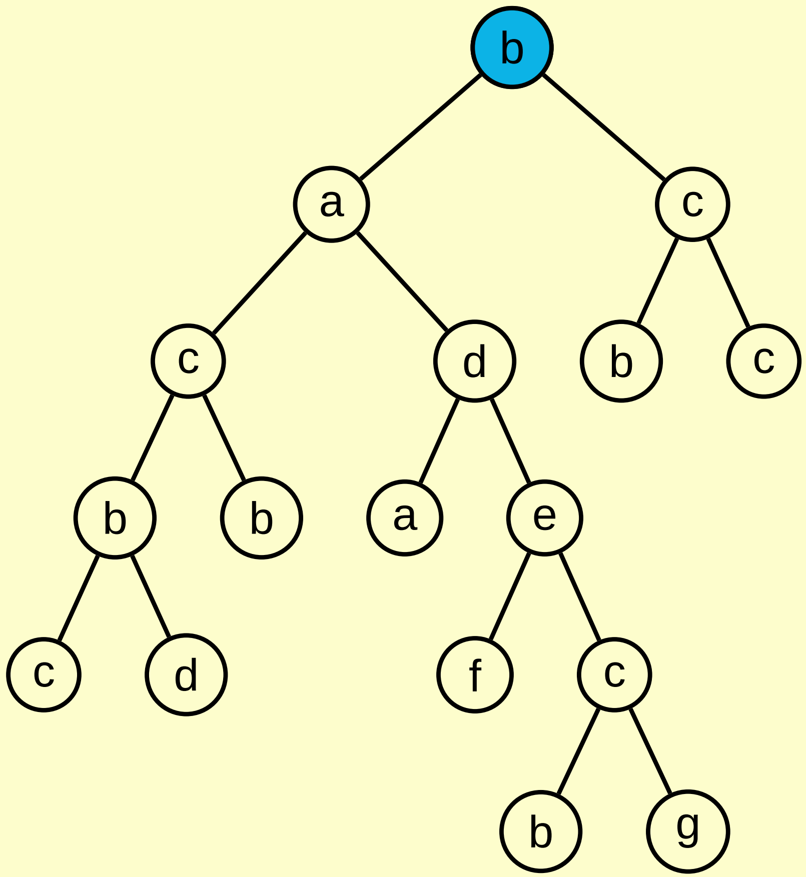
- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

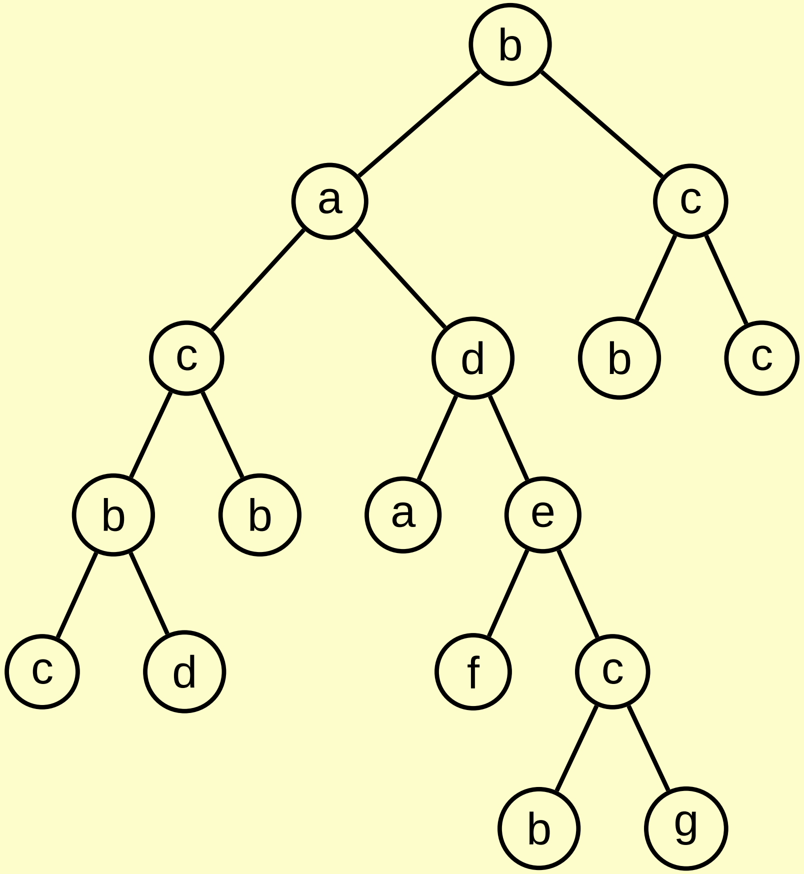
Pebble automata:

- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow$,stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:
    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong Deterministic pebble automata with n pebbles

Pebble automata



Definition

Pebble automata:

- states Q
- pebbles $n, \ldots, 1$
- Possible moves: $D = \{\uparrow, \searrow, \swarrow,$stay,drop,lift$\}$
- Depending on
  - state,
  - minimum pebble number,
  - set of pebbles at current node,
  - label,
  - type of node:

    $\{$root,leftchild,rightchild$\} \times \{$leaf,innernode$\}$

Notation: $\mathbf{sDPA}_n$: languages accepted by:

Strong **D**eterministic pebble automata with n pebbles

Some results...

... on pebble automata

- $\mathbf{PA} \subsetneq \mathbf{REG}$
- For each $n \geq \mathbf{0}$,
  - $\mathbf{PA}_n \subsetneq \mathbf{PA}_{n+1}$ and $\mathbf{DPA}_n \subsetneq \mathbf{DPA}_{n+1}$
  - $\mathbf{TWA} \not\subseteq \mathbf{DPA}_n$
  - $\mathbf{sPA}_n = \mathbf{PA}_n$ and $\mathbf{sDPA}_n = \mathbf{DPA}_n$
  - $\mathbf{sDPA}_n$ is closed under complement

... and a corollary on logics

FO+posTC$^{\mathbf{1}} \subsetneq$ MSO on binary trees

Proof summary

Theorem

FO+posTC^1 $\subsetneq$ MSO on binary trees

Outline

1. FO $+$ posTC^1 $= \mathbf{sPA}$ [Engelfriet, Hoogeboom 06]
2. Formalize behavior of a pebble automaton on a tree ($\longrightarrow$ type)
3. Strong pebbles do not give extra power: $\mathbf{sPA}_n = \mathbf{PA}_n$
4. For each n, define L_n and show that $L_n \in \mathbf{DPA}_n - \mathbf{PA}_{n-1}$
   (a) Towards a contradiction, fix A for L_n
   (b) Show that oracle automata do not accept L_1
   Fix $s \in L_1, t \notin L_1$ that can not be distinguished by oracle automaton of size $\leq m$
   (c) Define $s_1, \ldots, s_n \in L_n$, $t_1, \ldots, t_n \notin L_n$
   (d) Show: i-type of s_i and t_i are identical
   (otherwise: there is oracle automaton of size $\leq m$ distinguishing s from t)
5. Construct $L \in \mathbf{REG} - \mathbf{PA}$ from $L_1, L_2, L_3, \ldots$

Contents

Pebble automata and logic

Theorem [Engelfriet, Hoogeboom 06]

- **FO** $+$ DTC^{1} $= \mathbf{sDPA}$
- **FO** $+$ posTC^{1} $= \mathbf{sPA}$

(They prove a stronger result for multihead-automata)

Proof idea

"Logic $\Rightarrow$ automaton" requires a liberal pebble lifting policy:

- To check TC$_{x,y}[\varphi(x, y, \vec{z})](u, v)$:
  - Pebbles $n, \ldots, k$ on $\vec{z}$ do not move
  - Pebble $k-1$ on v, pebble $k-2$ on u
  - Guess u_1 by placing pebble $k-3$ on it
  - Check $\varphi(u, u_1, \vec{z})$ recursively
  - Go back to u_1, lift pebbles $k-2, k-3$
  - Put pebble $k-2$ on u_1
  - Continue with u_2...

$\longrightarrow$ **strong pebbles**: the minimum pebble can be lifted even if the head is somewhere else

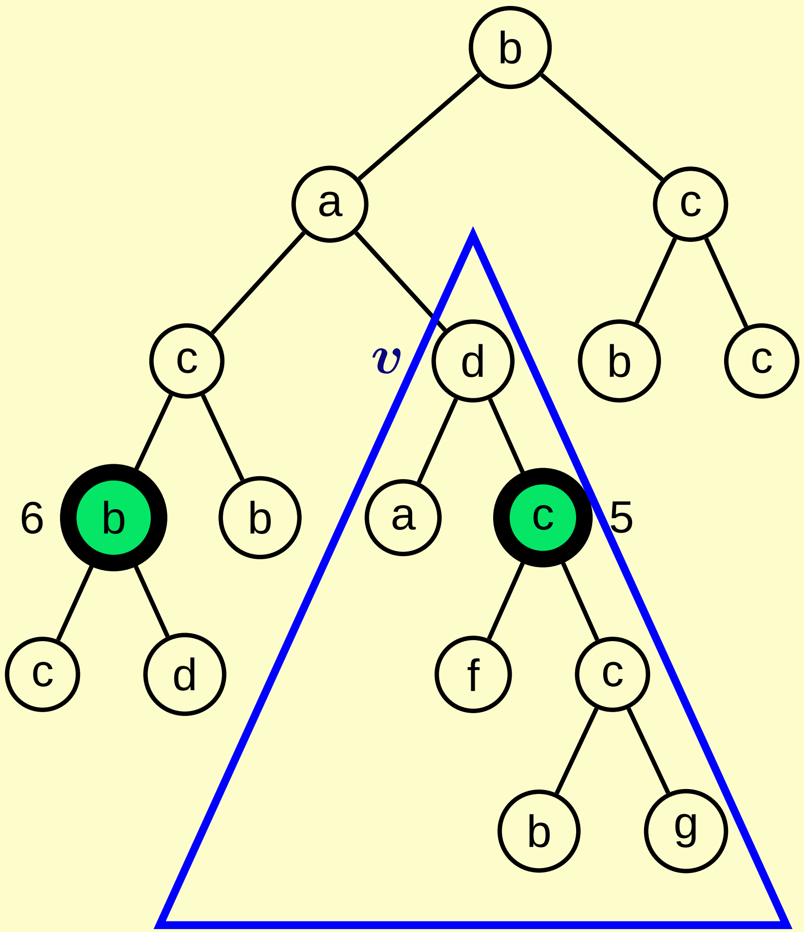
Contents

Tree loops...

We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?



Tree loops...

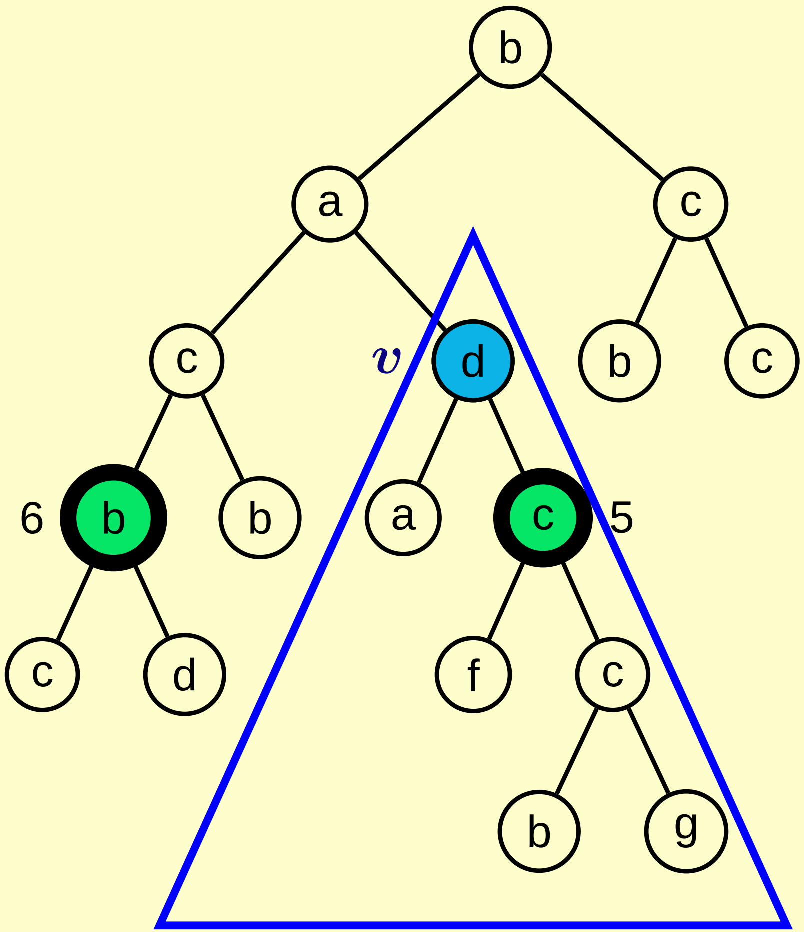
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \dots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \dots, v_n)$ to $(v, q, v_{i+1}, \dots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \;\; \Rightarrow$ head below v



Tree loops...

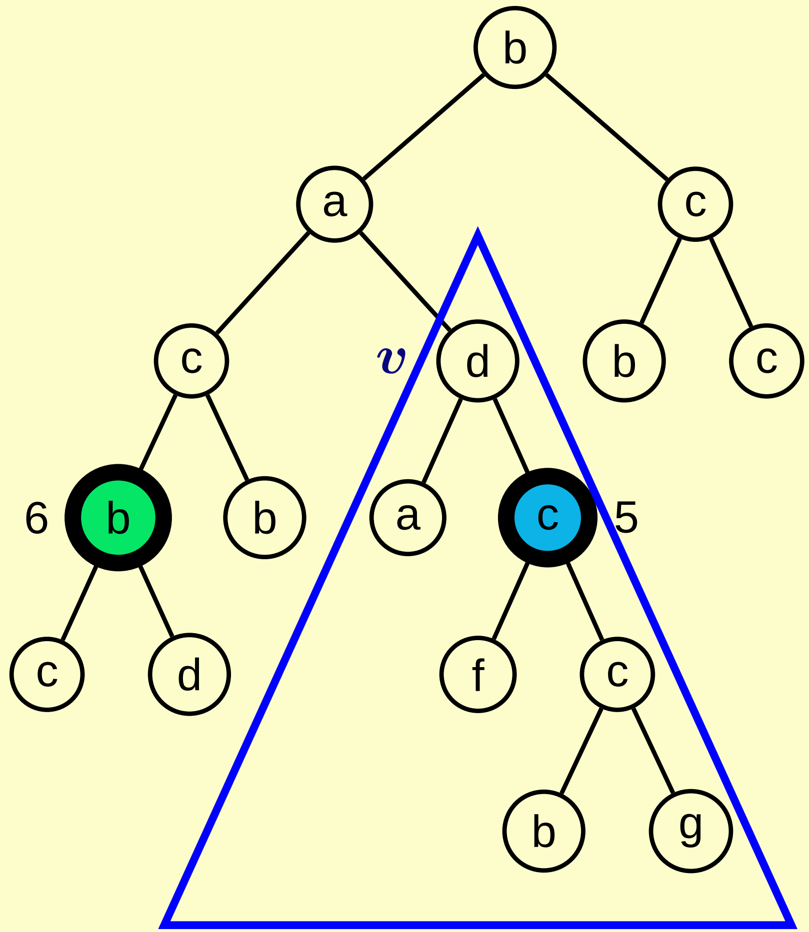
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \ldots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \quad \Rightarrow \quad$ head below v



Tree loops...

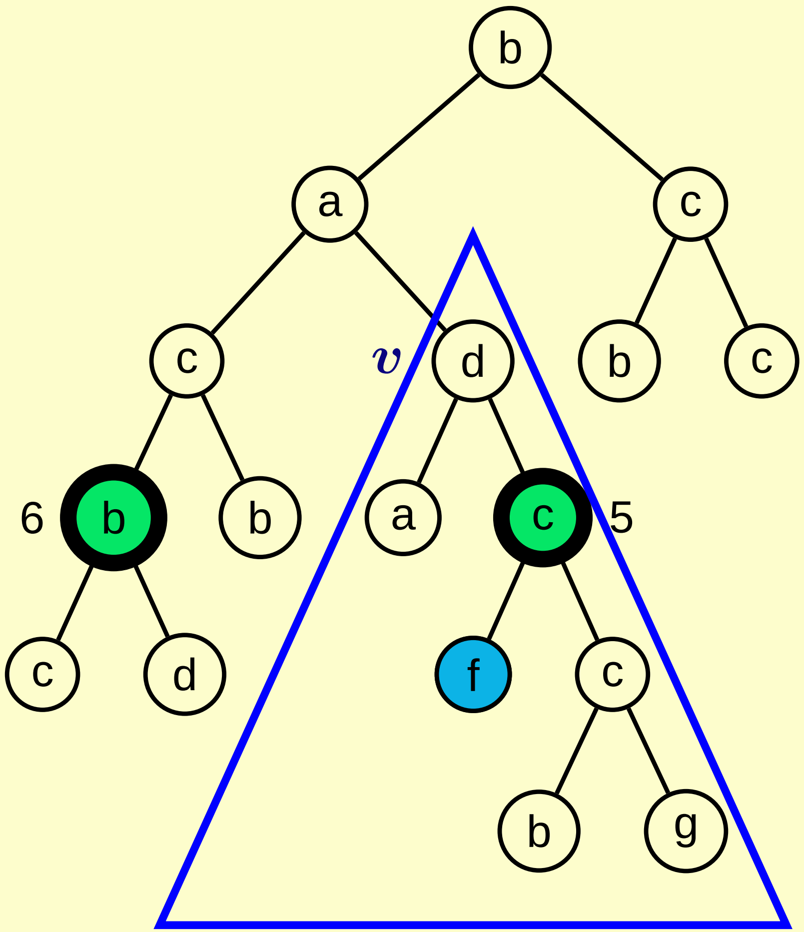
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \ldots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

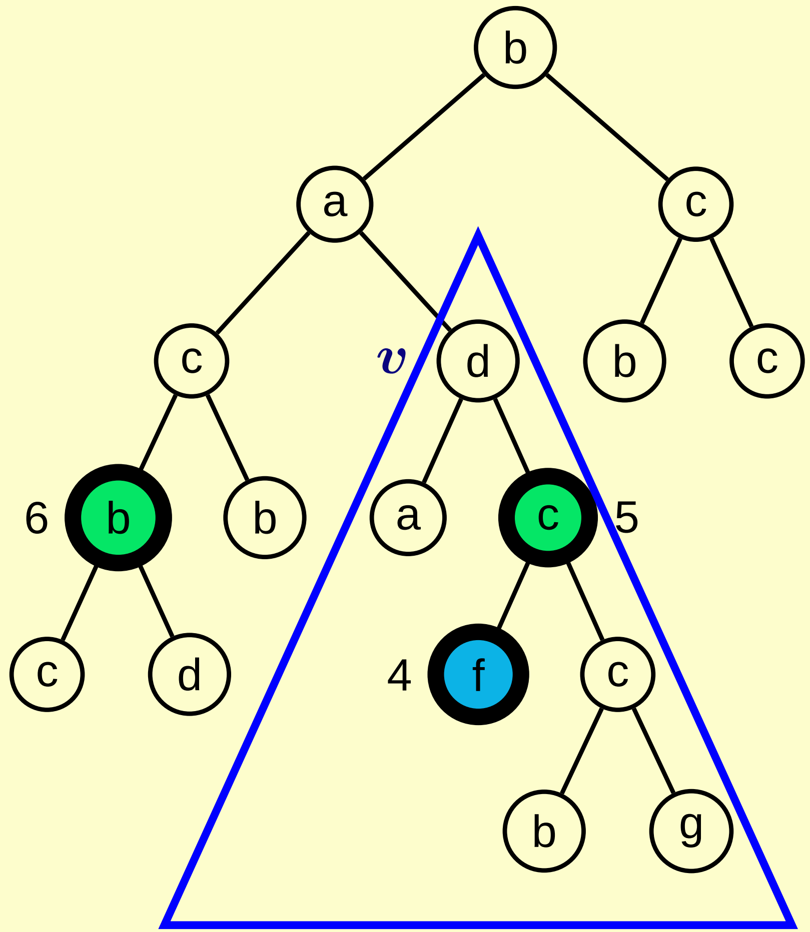
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \ldots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

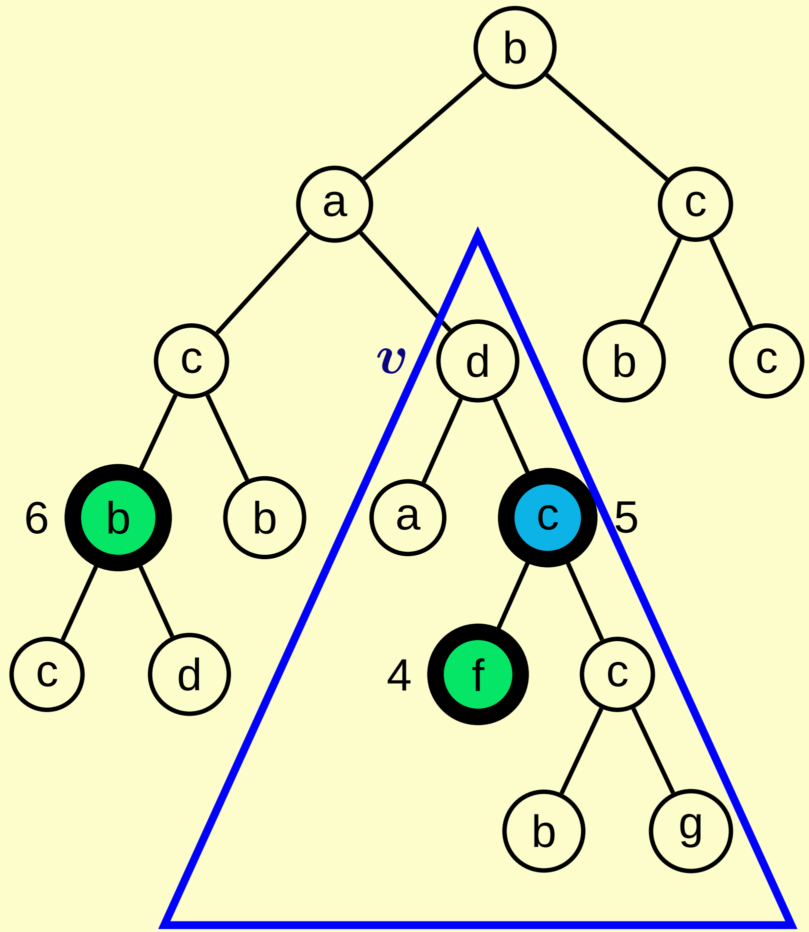
We fix some PA A

Basic question

What is the **behavior** of A on a subtree t_v?

Some definitions

- **i-configuration**: $(v, p, v_{i+1}, \ldots, v_n)$
- **tree i-loop**: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

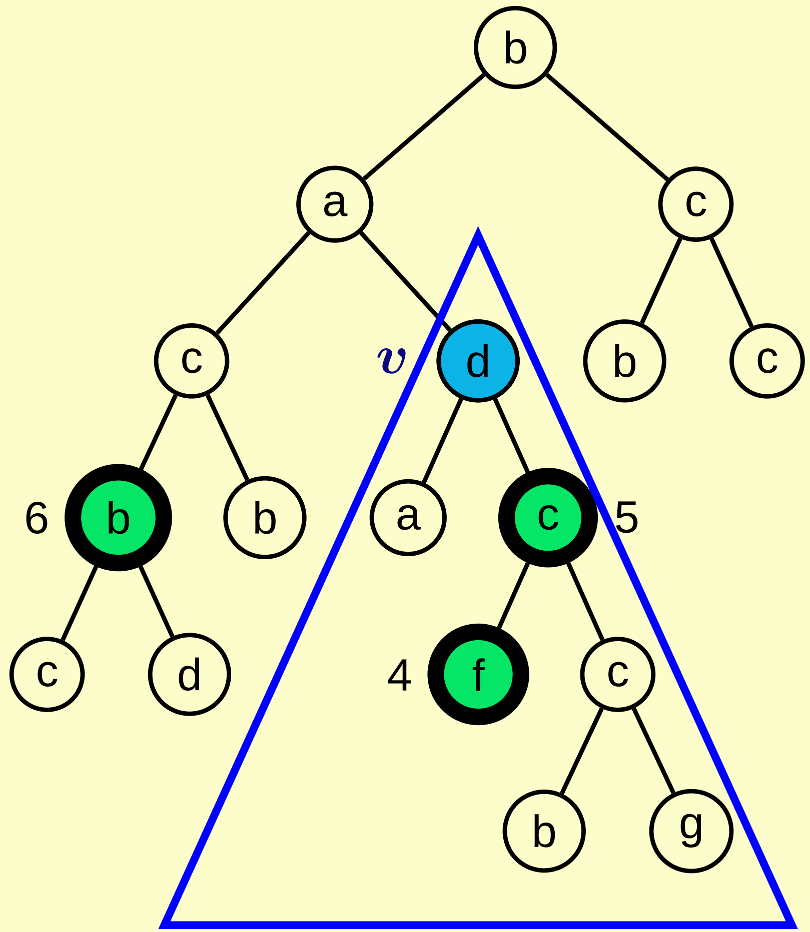
We fix some PA A

Basic question

What is the **behavior** of A on a subtree t_v?

Some definitions

- **i-configuration**: $(v, p, v_{i+1}, \ldots, v_n)$
- **tree i-loop**: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \quad \Rightarrow$ head below v



Tree loops...

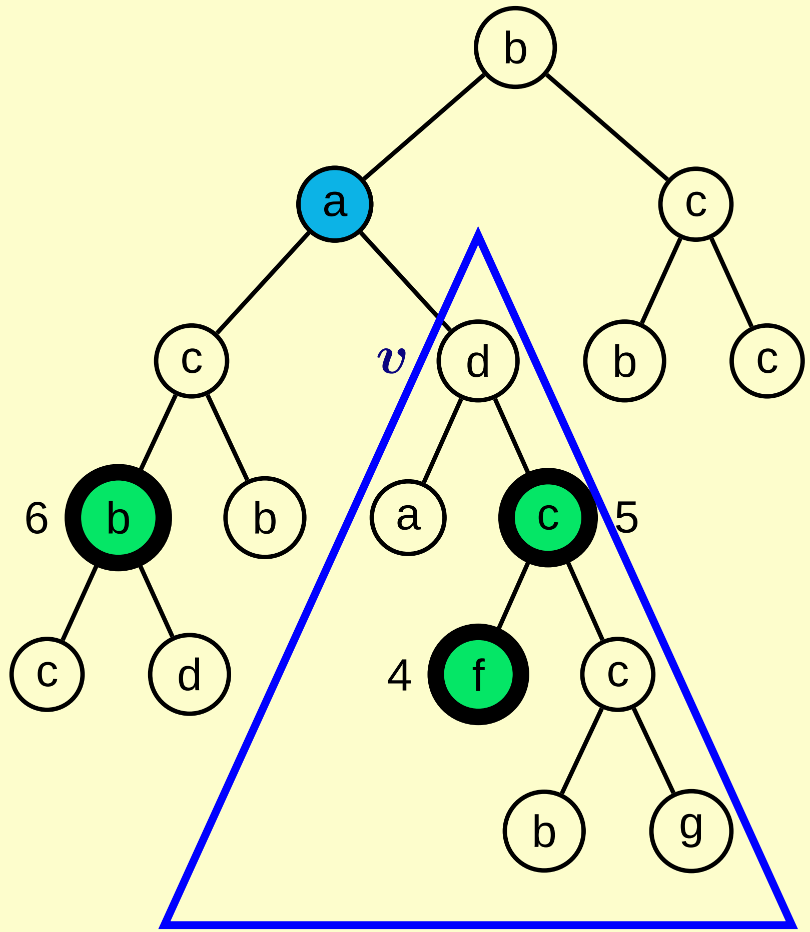
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \ldots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

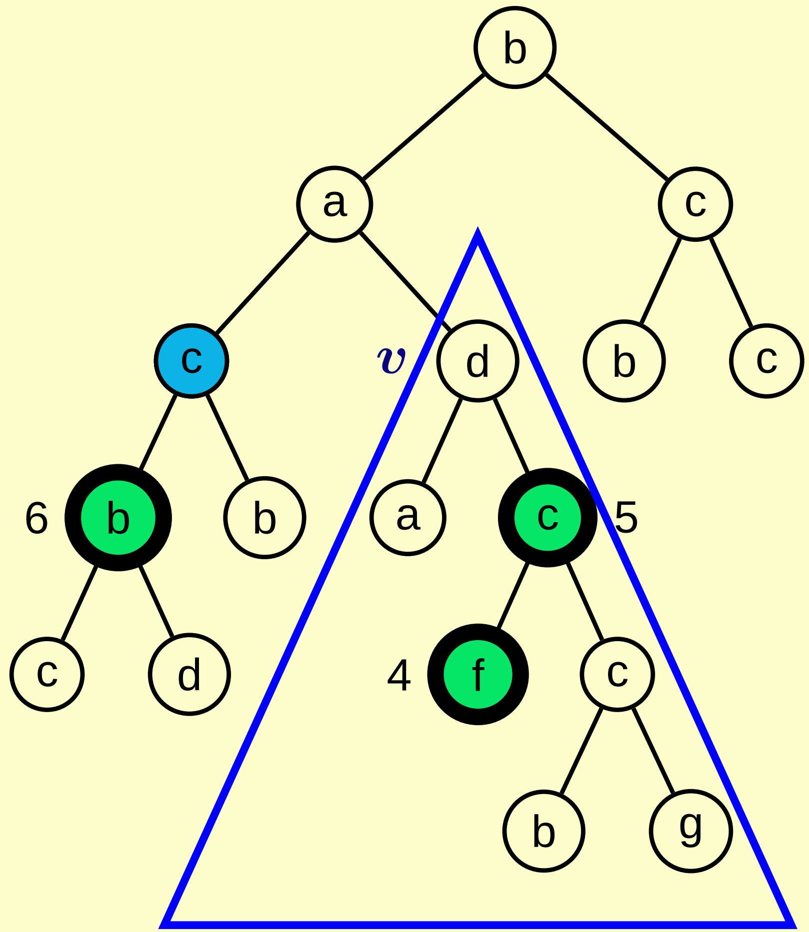
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \dots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \dots, v_n)$ to $(v, q, v_{i+1}, \dots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

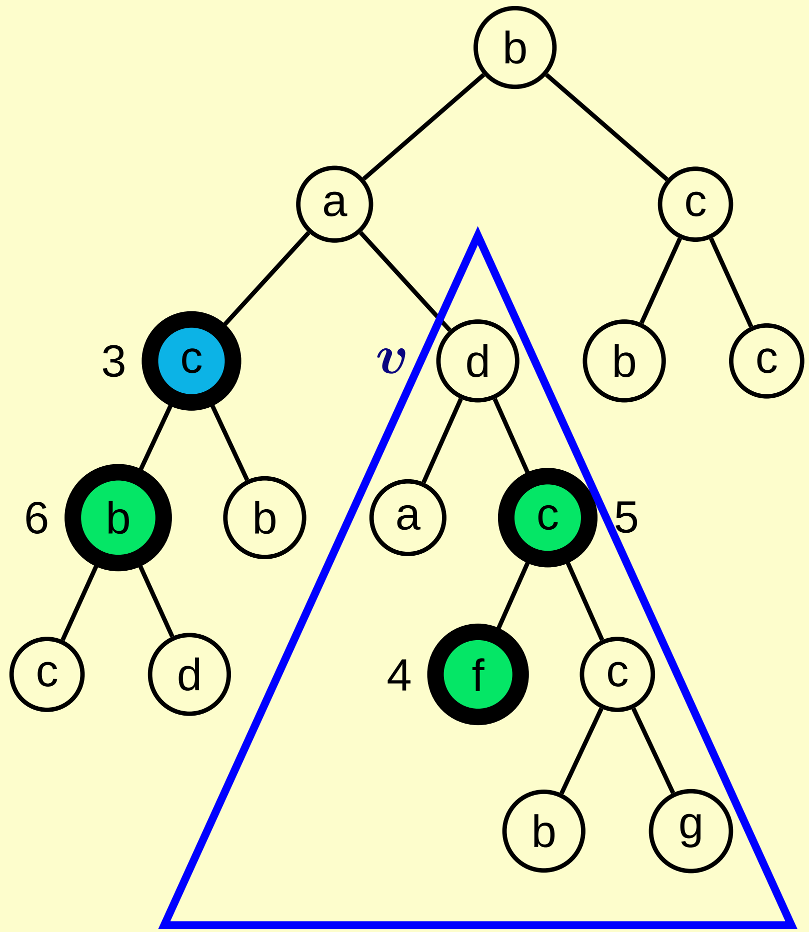
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \ldots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

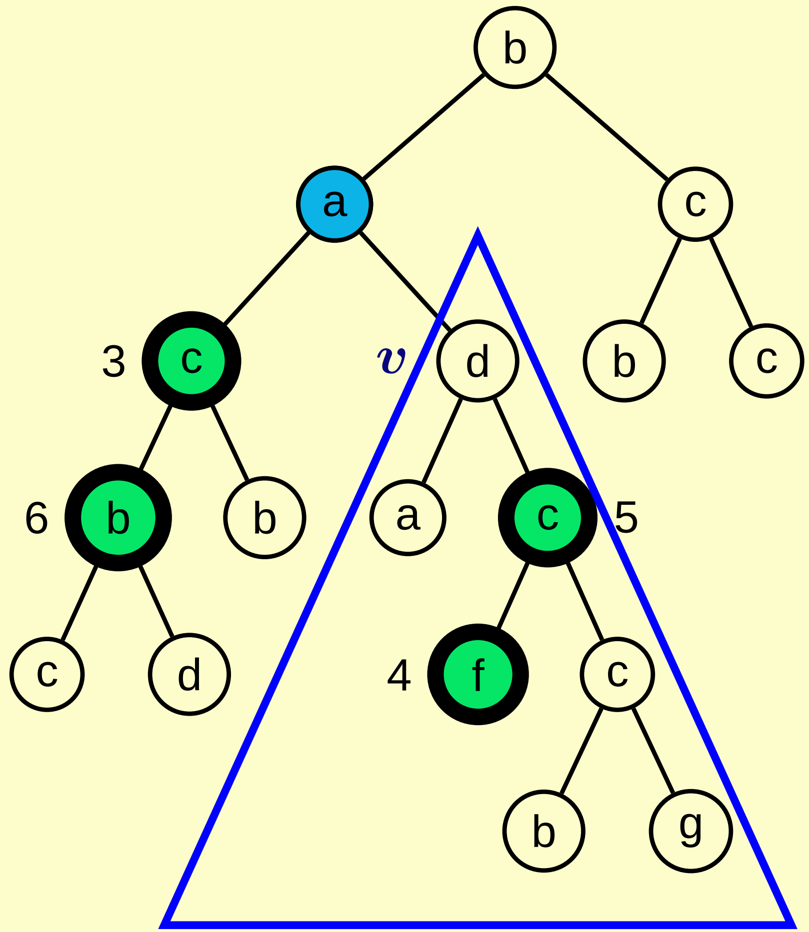
We fix some PA A

Basic question

What is the **behavior** of A on a subtree t_v?

Some definitions

- **i-configuration**: $(v, p, v_{i+1}, \ldots, v_n)$
- **tree i-loop**: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \quad \Rightarrow$ head below v



Tree loops...

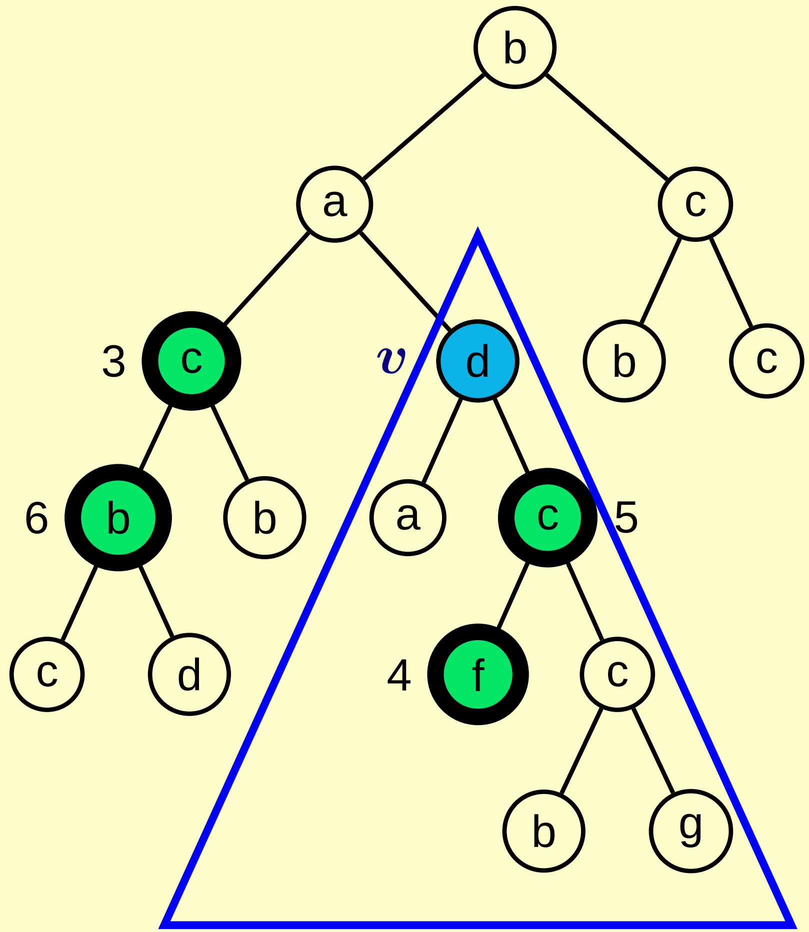
We fix some PA A

Basic question

What is the **behavior** of A on a subtree t_v?

Some definitions

- **i-configuration**: $(v, p, v_{i+1}, \ldots, v_n)$
- **tree i-loop**: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

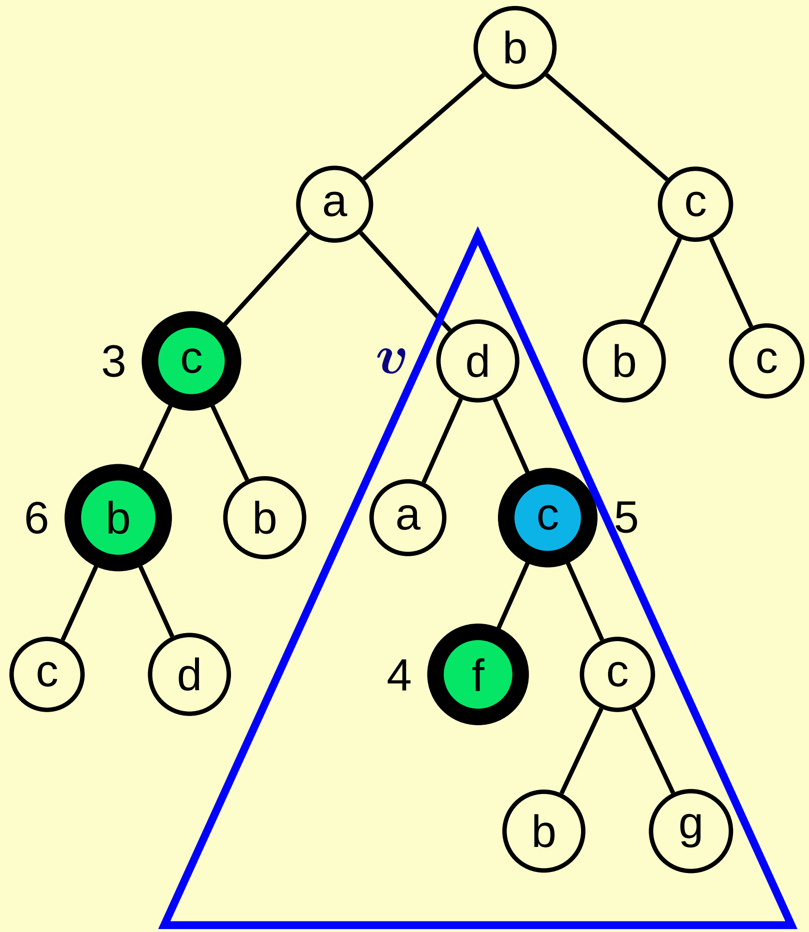
We fix some PA A

Basic question

What is the **behavior** of A on a subtree t_v?

Some definitions

- **i-configuration**: $(v, p, v_{i+1}, \ldots, v_n)$
- **tree i-loop**: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \quad \Rightarrow \quad$ head below v



Tree loops...

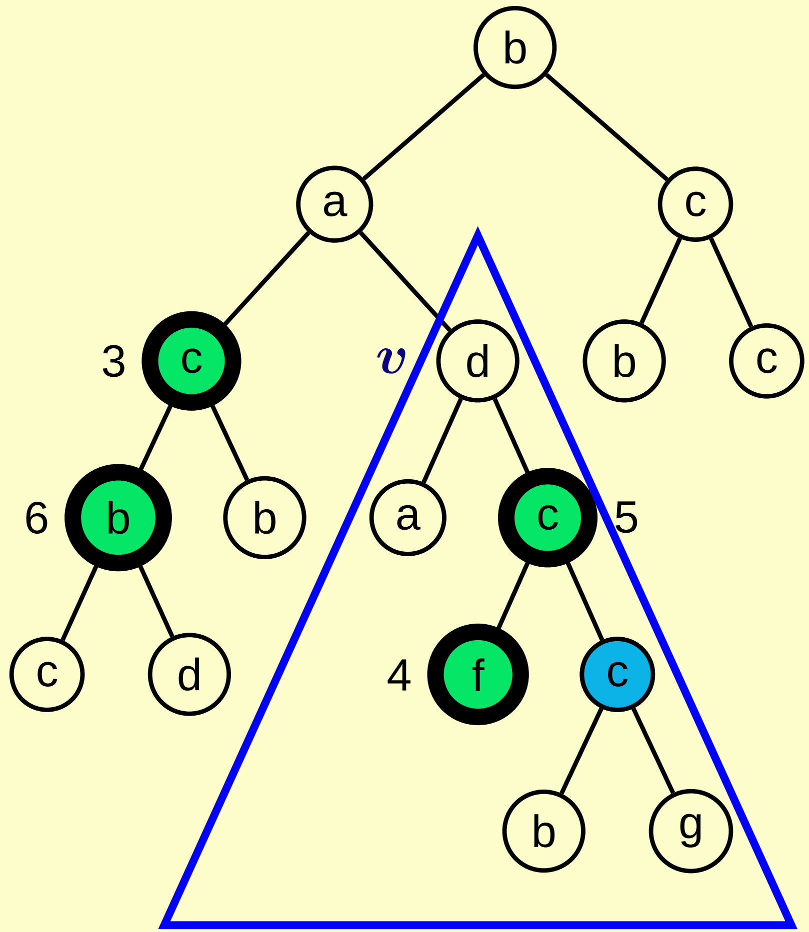
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \ldots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

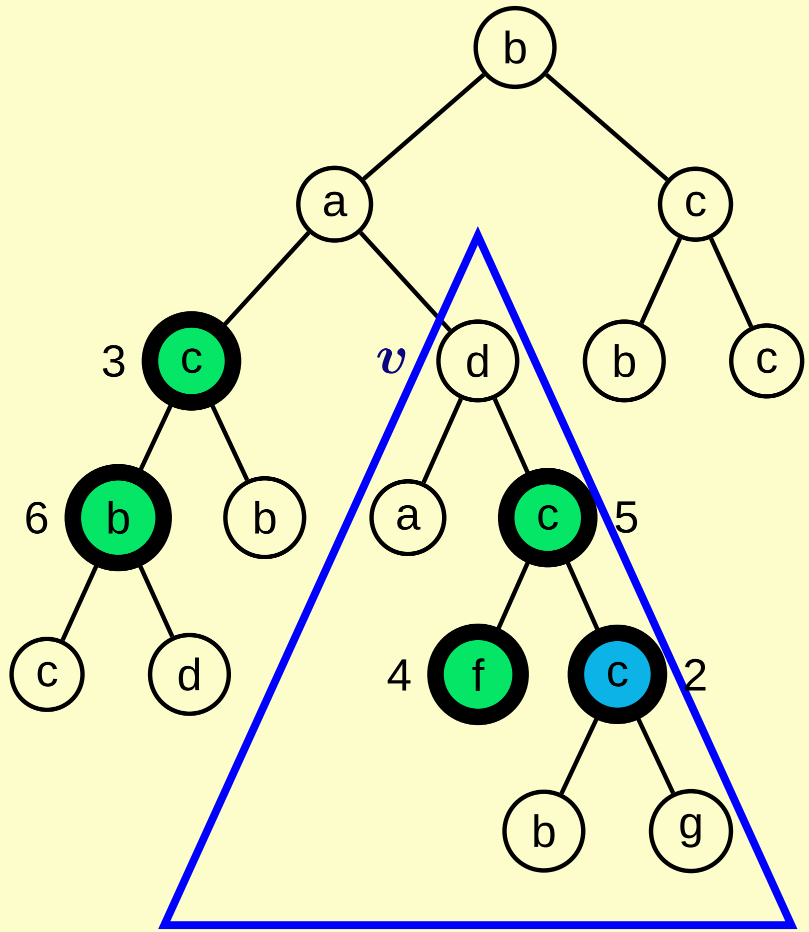
We fix some PA A

Basic question

What is the **behavior** of A on a subtree t_v?

Some definitions

- **i-configuration**: $(v, p, v_{i+1}, \ldots, v_n)$
- **tree i-loop**: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

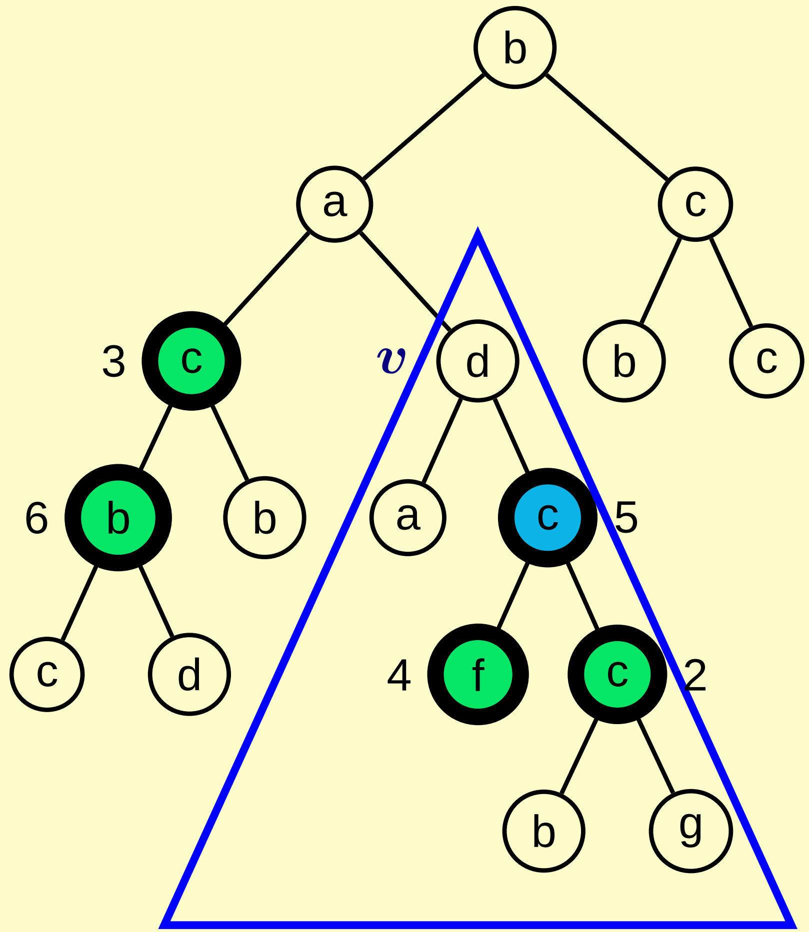
We fix some PA A

Basic question

What is the **behavior** of A on a subtree t_v?

Some definitions

- **i-configuration**: $(v, p, v_{i+1}, \ldots, v_n)$
- **tree i-loop**: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \quad \Rightarrow \quad$ head below v



Tree loops...

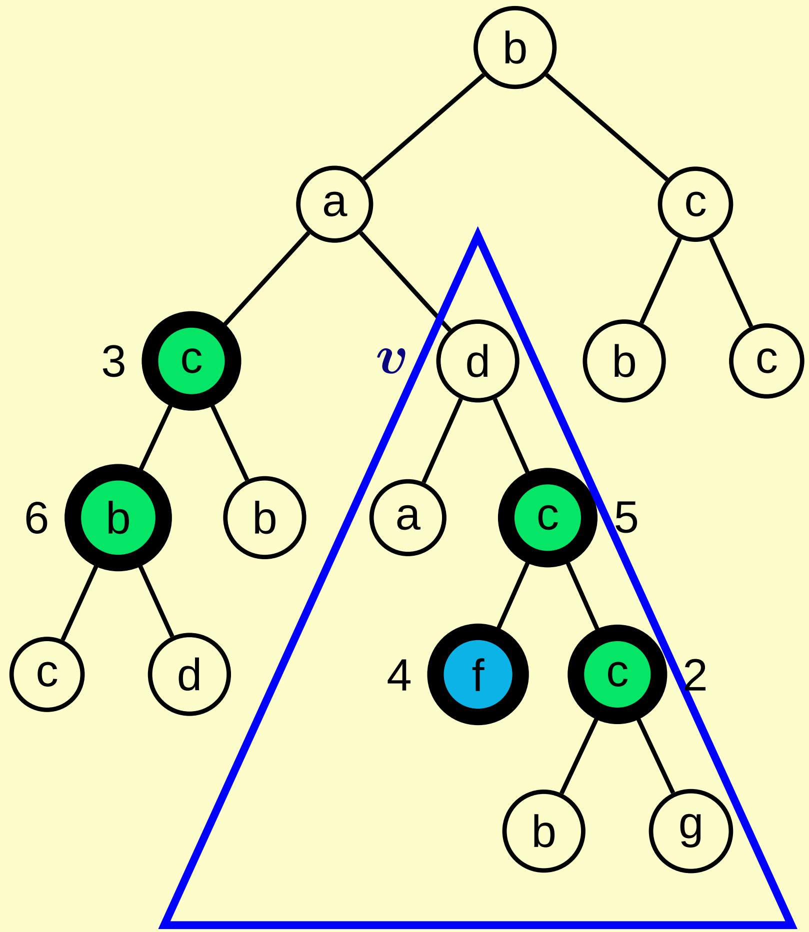
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \ldots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

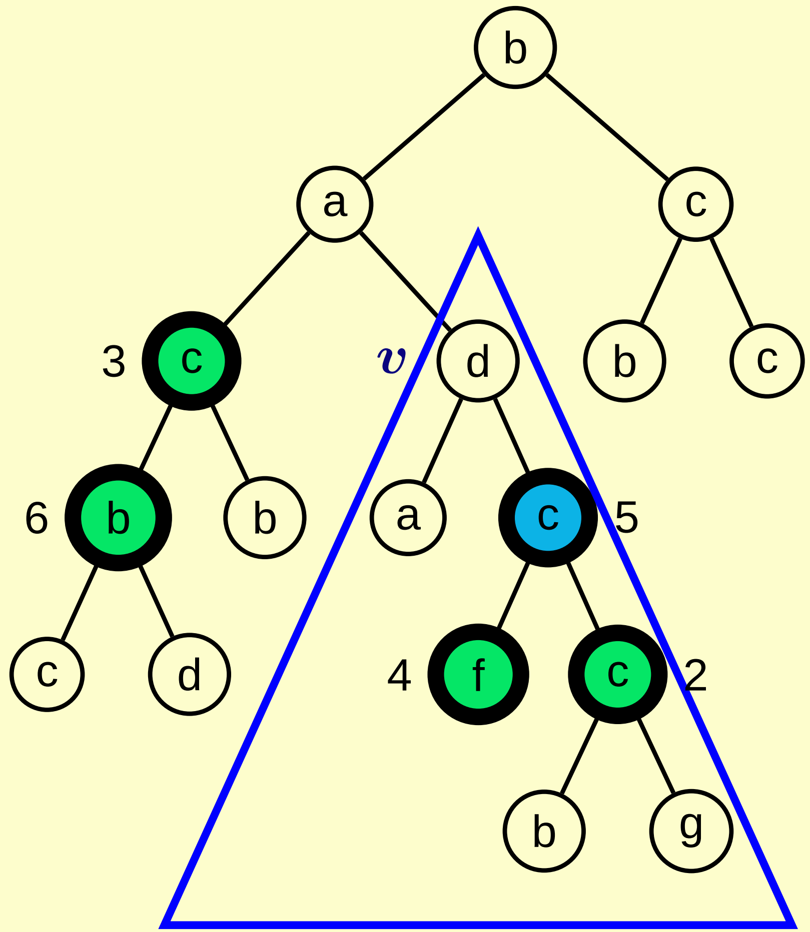
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \dots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \dots, v_n)$ to $(v, q, v_{i+1}, \dots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

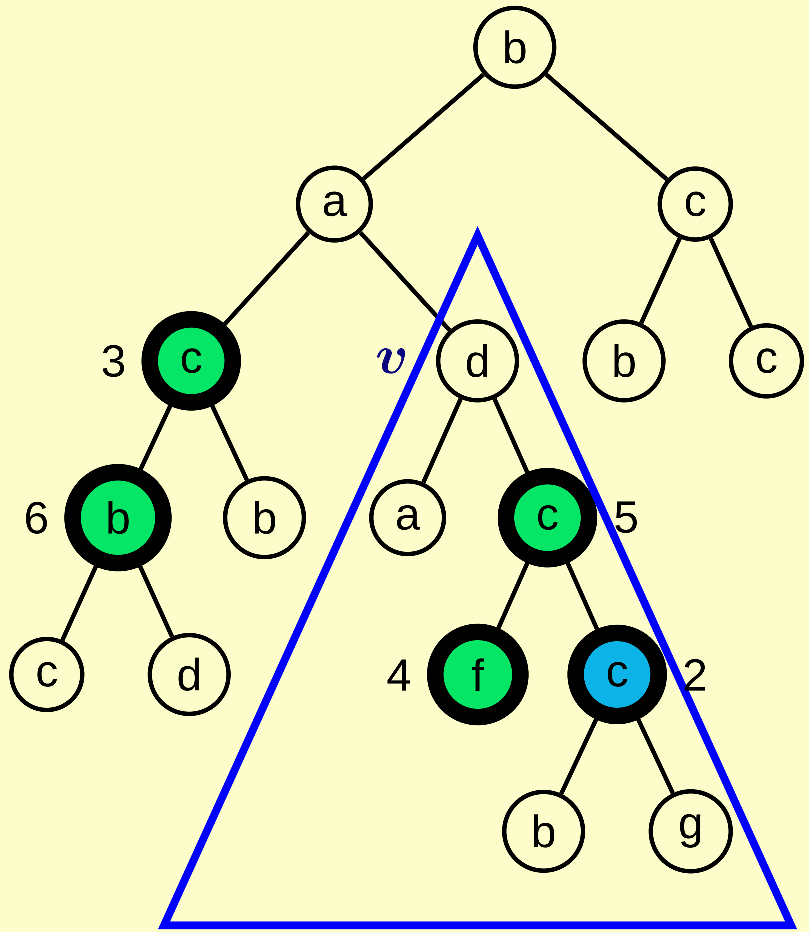
We fix some PA A

Basic question

What is the **behavior** of A on a subtree t_v?

Some definitions

- **i-configuration**: $(v, p, v_{i+1}, \ldots, v_n)$
- **tree i-loop**: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

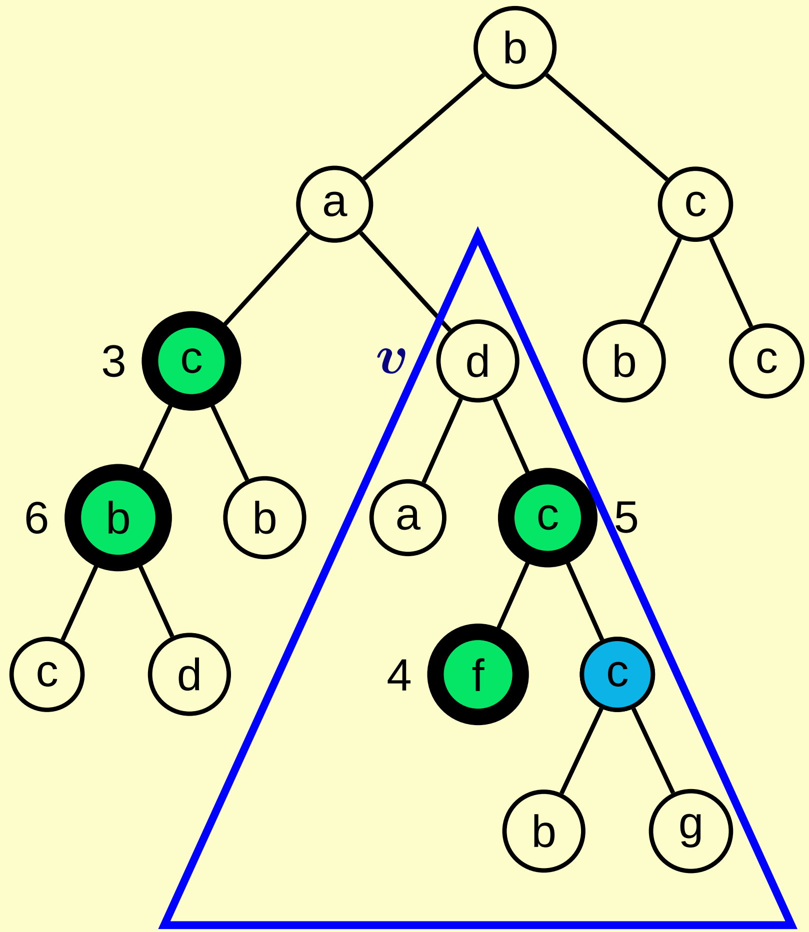
We fix some PA A

Basic question

What is the **behavior** of A on a subtree t_v?

Some definitions

- **i-configuration**: $(v, p, v_{i+1}, \dots, v_n)$
- **tree i-loop**: subcomputation from $(v, p, v_{i+1}, \dots, v_n)$ to $(v, q, v_{i+1}, \dots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

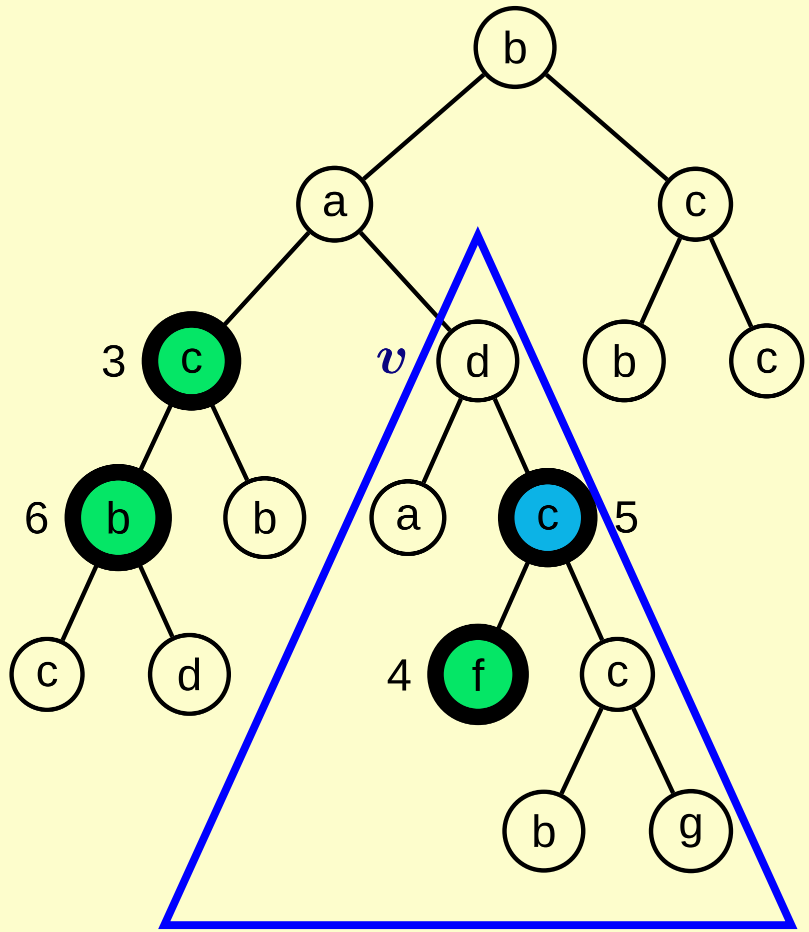
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \dots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \dots, v_n)$ to $(v, q, v_{i+1}, \dots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

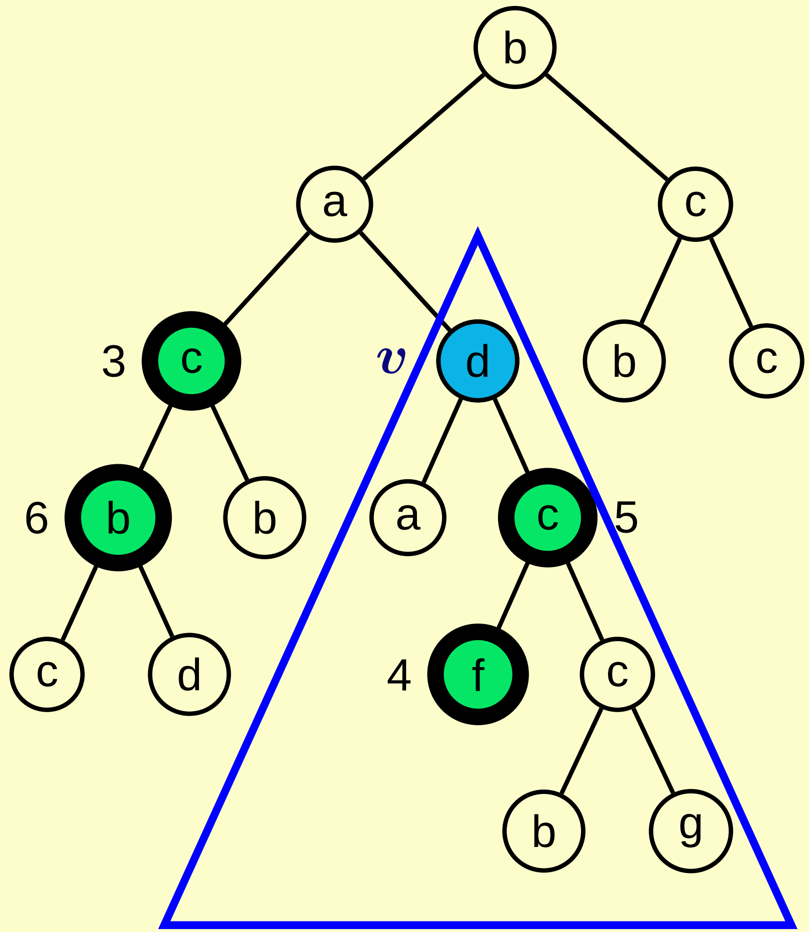
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \ldots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

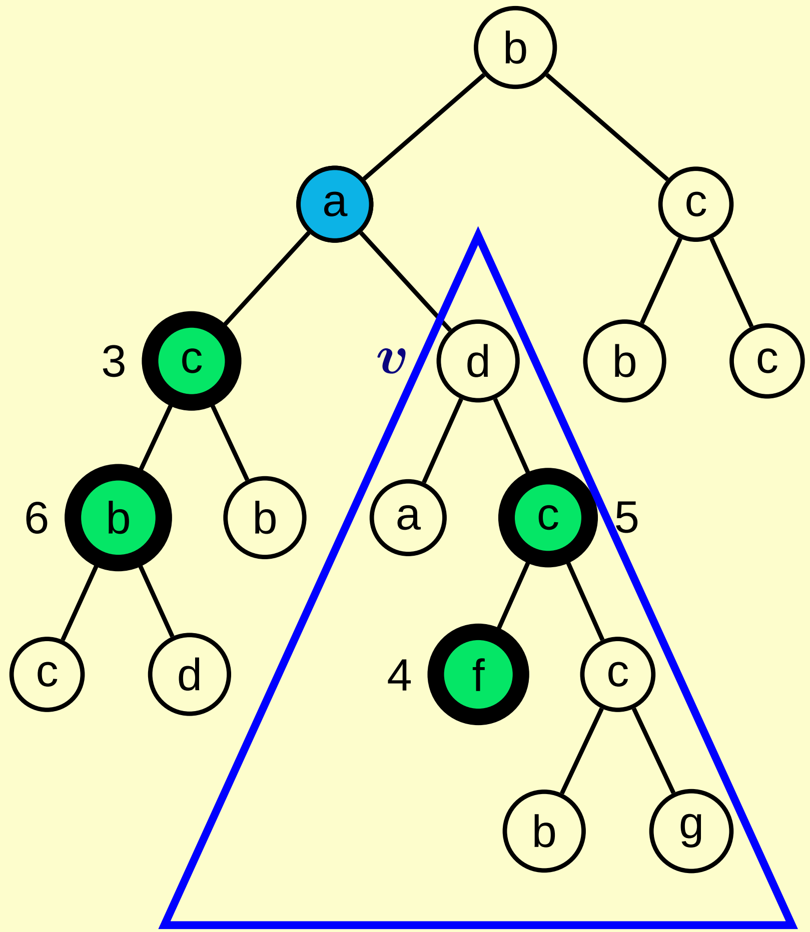
We fix some PA A

Basic question

What is the **behavior** of A on a subtree t_v?

Some definitions

- **i-configuration**: $(v, p, v_{i+1}, \ldots, v_n)$
- **tree i-loop**: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \quad \Rightarrow \quad$ head below v



Tree loops...

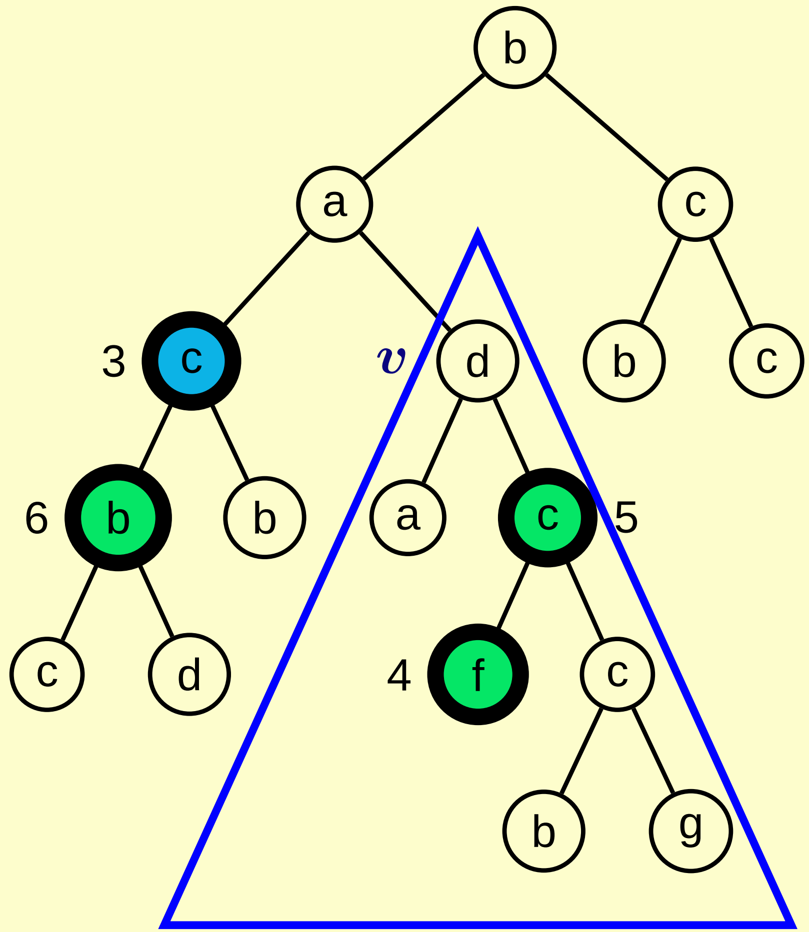
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \ldots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

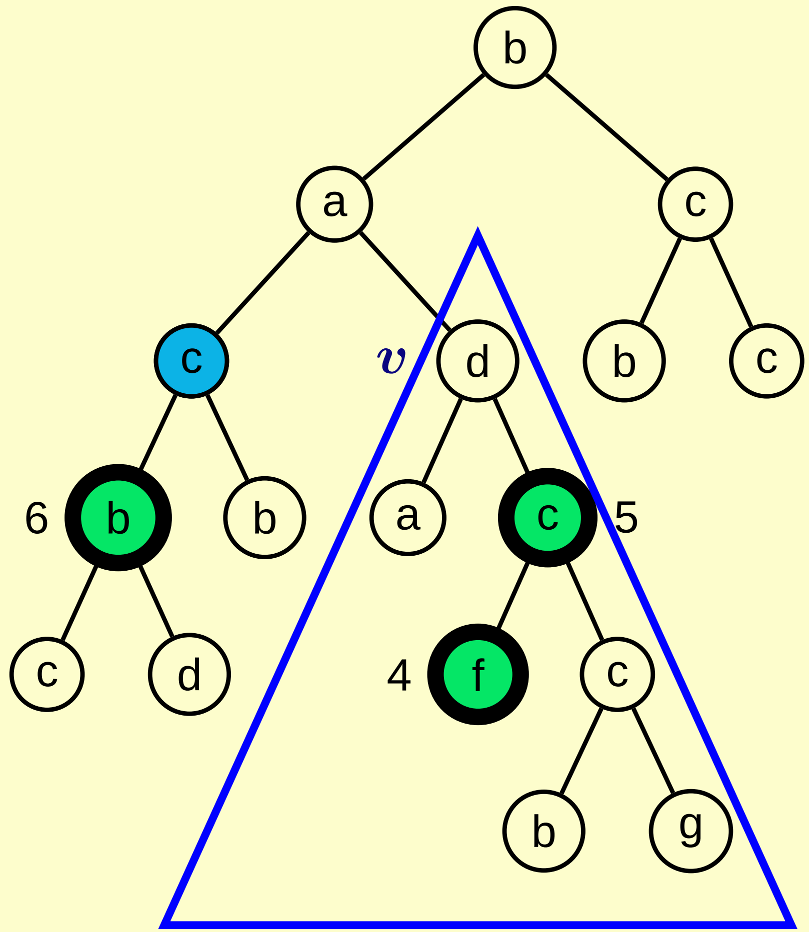
We fix some PA A

Basic question

What is the **behavior** of A on a subtree t_v?

Some definitions

- **i-configuration**: $(v, p, v_{i+1}, \ldots, v_n)$
- **tree i-loop**: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \quad \Rightarrow \quad$ head below v



Tree loops...

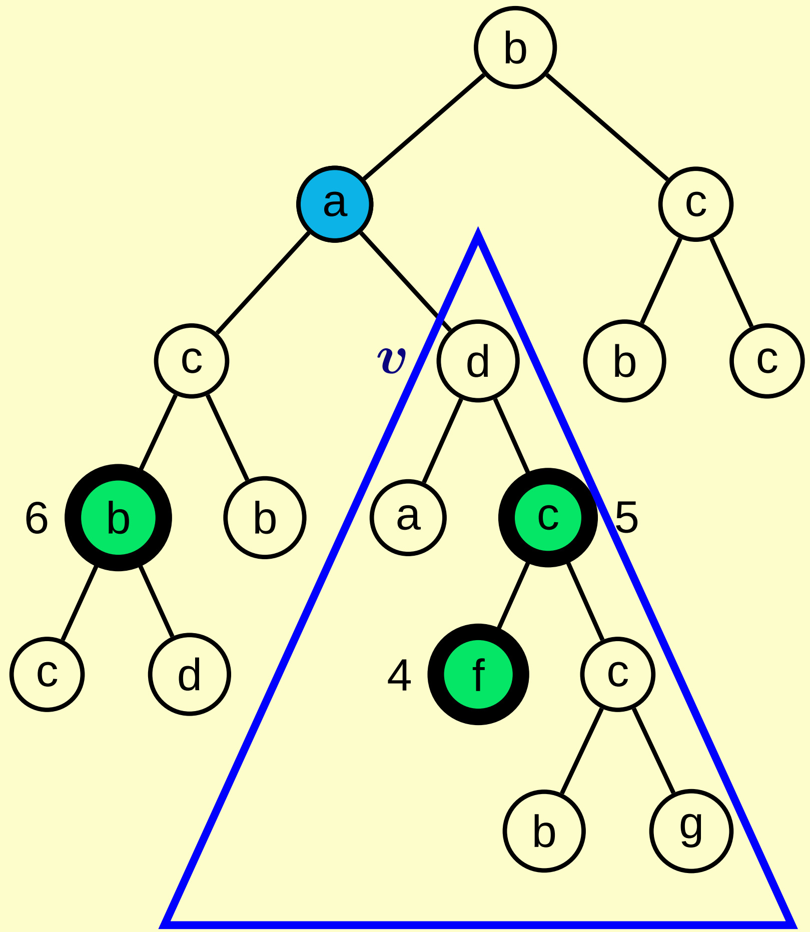
We fix some PA A

Basic question

What is the **behavior** of A on a subtree t_v?

Some definitions

- **i-configuration**: $(v, p, v_{i+1}, \ldots, v_n)$
- **tree i-loop**: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \Rightarrow$ head below v



Tree loops...

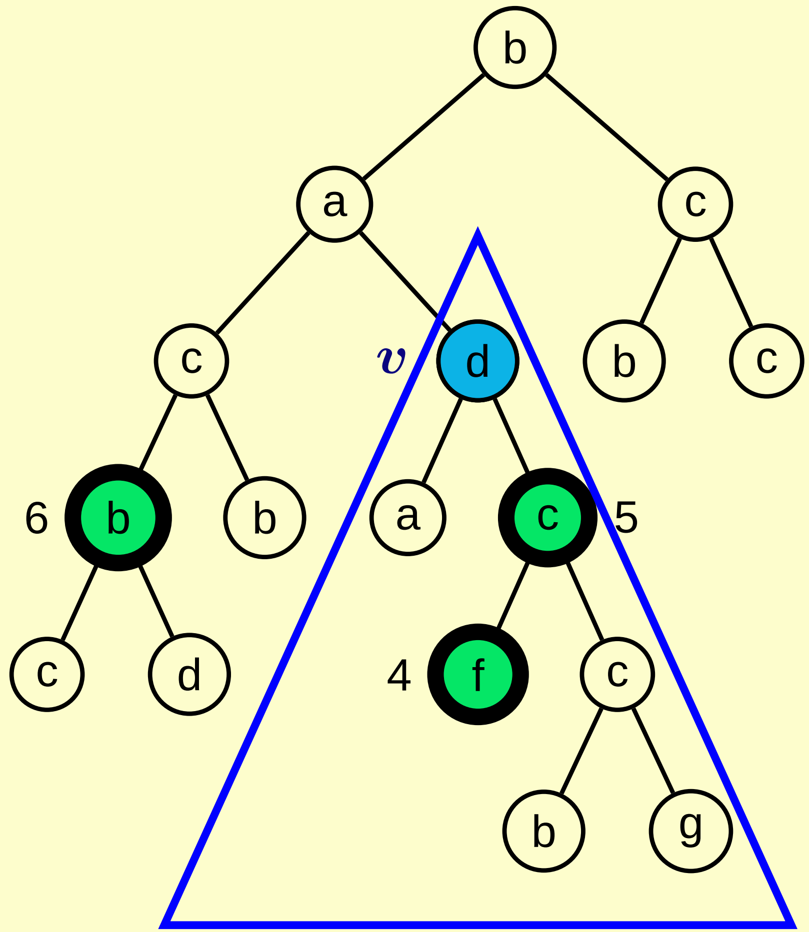
We fix some PA A

Basic question

What is the **behavior** of A on a subtree t_v?

Some definitions

- **i-configuration**: $(v, p, v_{i+1}, \dots, v_n)$
- **tree i-loop**: subcomputation from $(v, p, v_{i+1}, \dots, v_n)$ to $(v, q, v_{i+1}, \dots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \quad \Rightarrow \quad$ head below v



Tree loops...

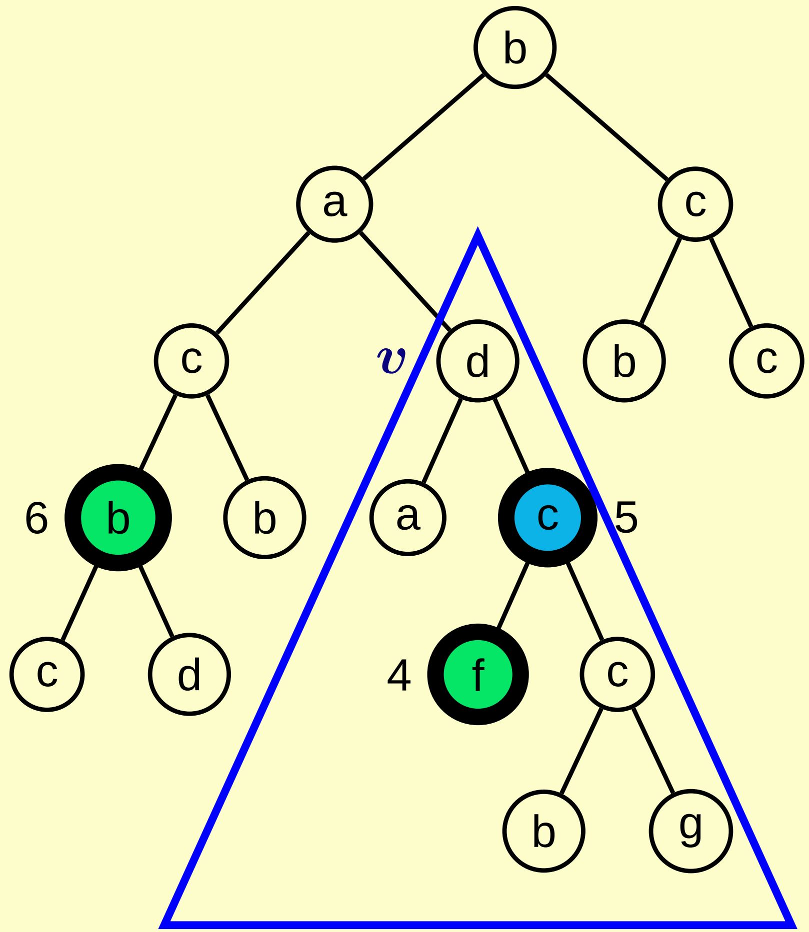
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \ldots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \quad \Rightarrow \quad$ head below v



Tree loops...

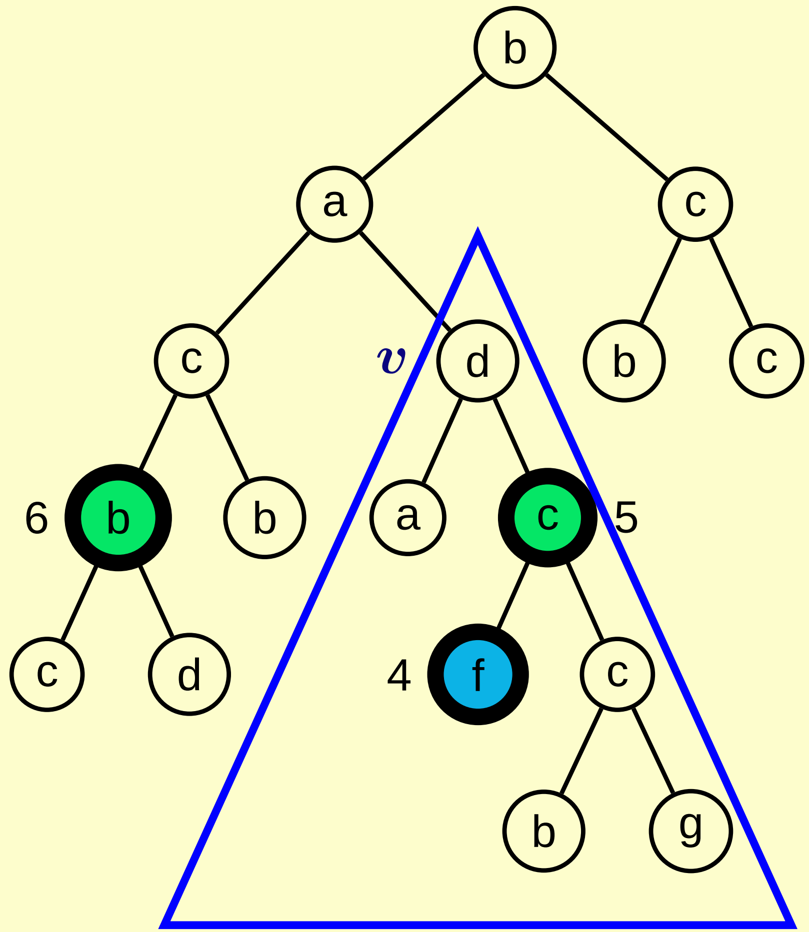
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \ldots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \quad \Rightarrow \quad$ head below v



Tree loops...

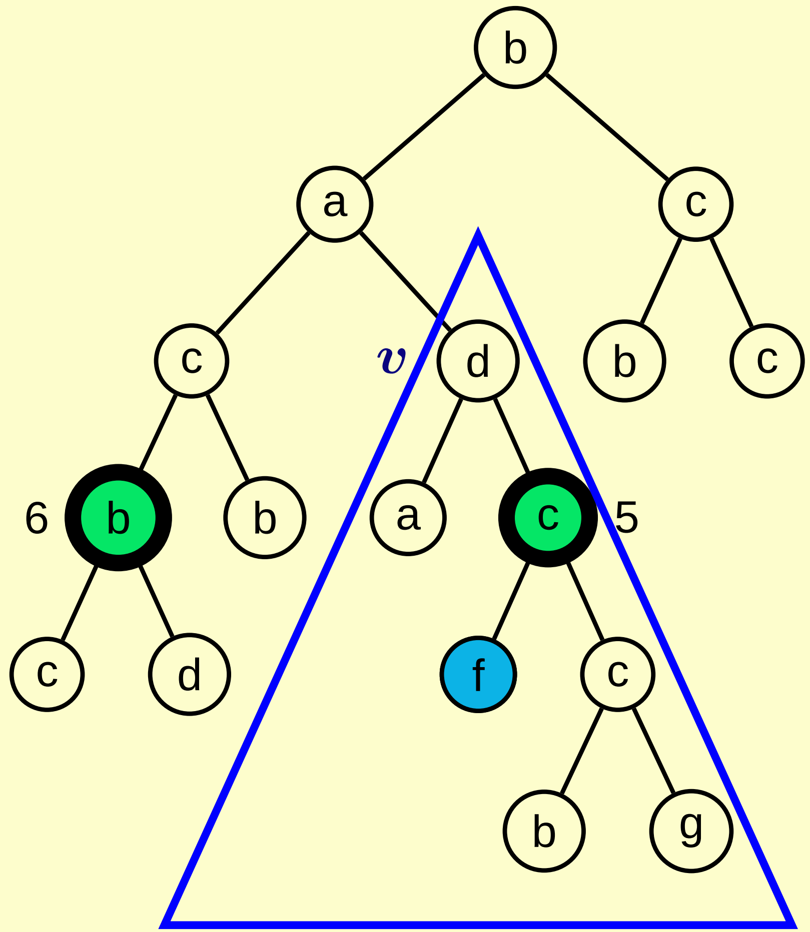
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \ldots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \quad \Rightarrow \quad$ head below v



Tree loops...

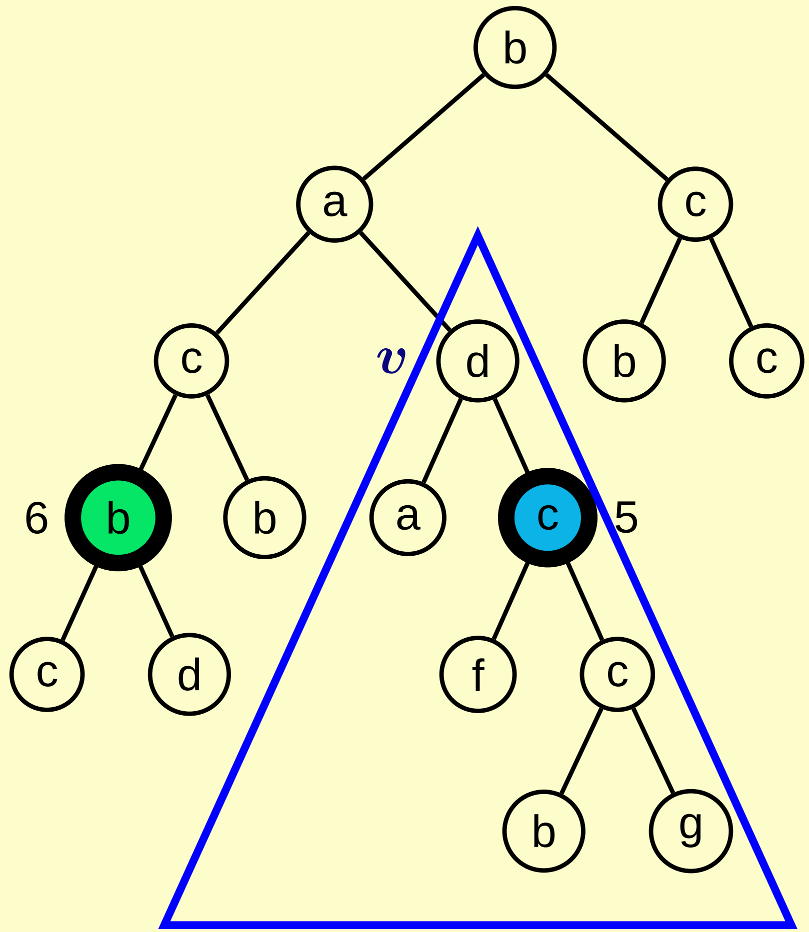
We fix some PA A

Basic question

What is the behavior of A on a subtree t_v?

Some definitions

- i-configuration: $(v, p, v_{i+1}, \ldots, v_n)$
- tree i-loop: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \;\; \Rightarrow$ head below v



Tree loops...

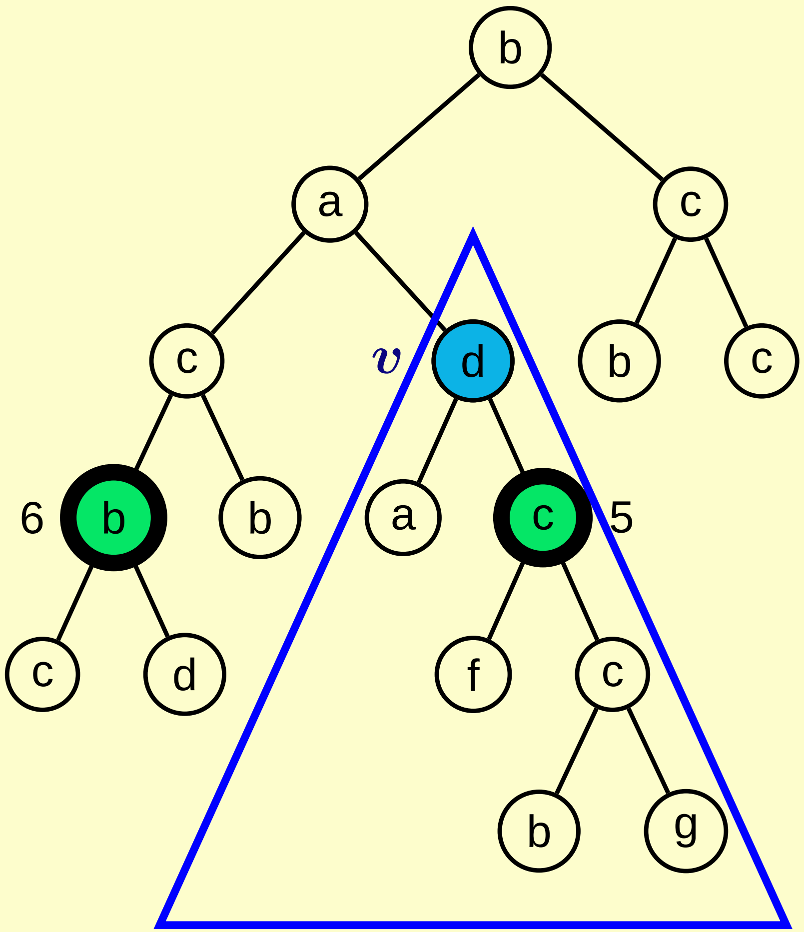
We fix some PA A

Basic question

What is the **behavior** of A on a subtree t_v?

Some definitions

- **i-configuration**: $(v, p, v_{i+1}, \ldots, v_n)$
- **tree i-loop**: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \quad \Rightarrow$ head below v



Tree loops...

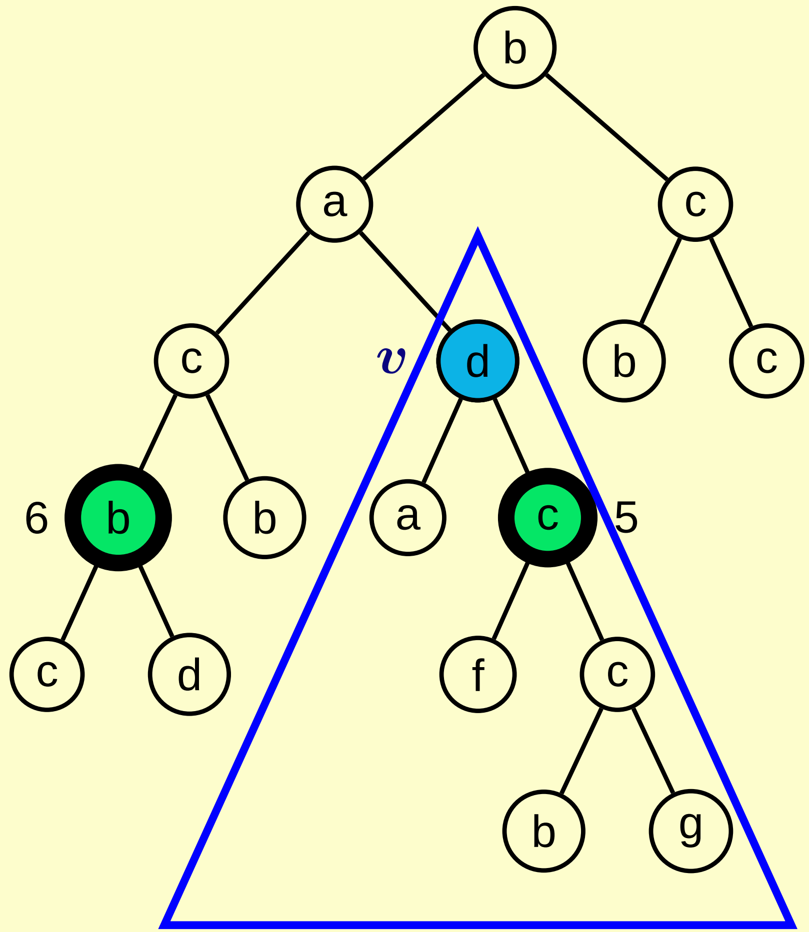
We fix some PA $\mathcal{A}$

Basic question

What is the **behavior** of $\mathcal{A}$ on a subtree t_v?

Some definitions

- **i-configuration**: $(v, p, v_{i+1}, \ldots, v_n)$
- **tree i-loop**: subcomputation from $(v, p, v_{i+1}, \ldots, v_n)$ to $(v, q, v_{i+1}, \ldots, v_n)$, where for all intermediate j-configurations:
  - $j \leq i$, and
  - $j = i \quad \Rightarrow \quad$ head below v



Observations

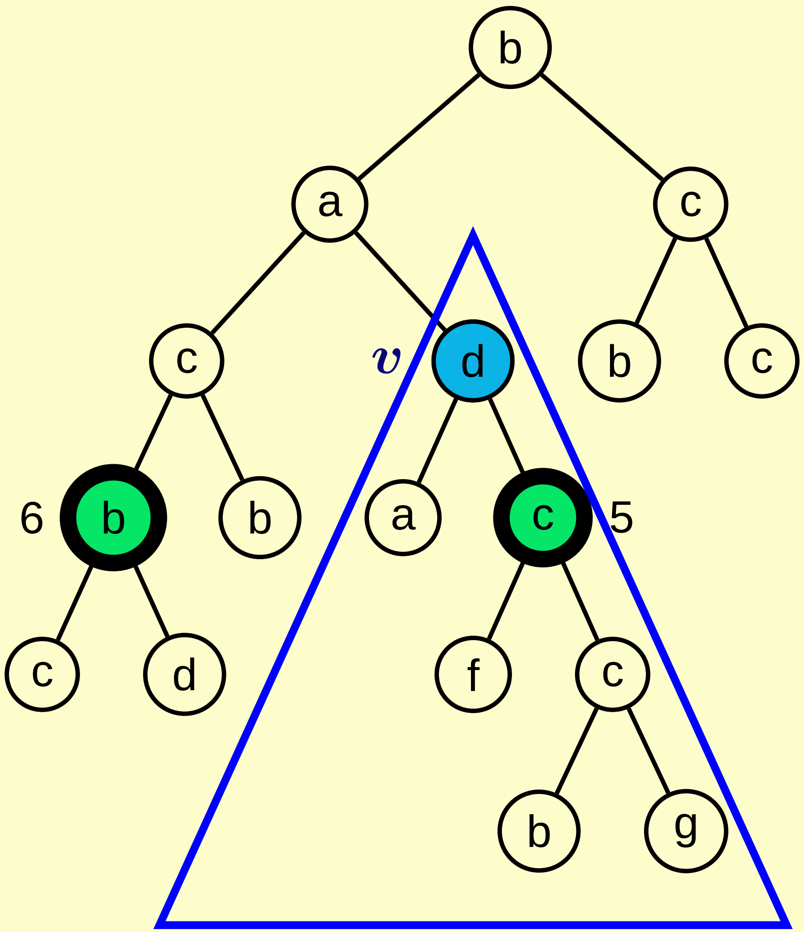
- During an i-loop the head can leave t_v
- Subcomputations on the context C_v might depend on t_v and on further pebbles in t_v

$\longrightarrow$ What do we need to know of t_v to simulate $\mathcal{A}$?

... and types

Definition

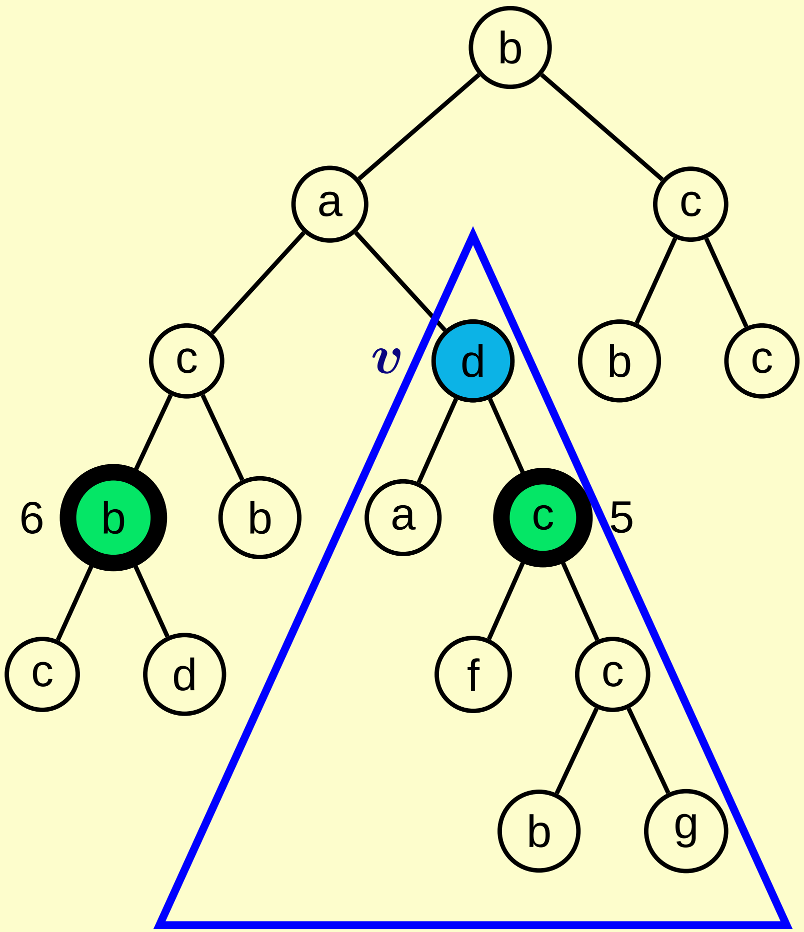
- $\boxed{\text{0-type of } t_v}$: all pairs (p, q) such that there is a tree $\mathbf{0}$-loop ...



... and types

Definition

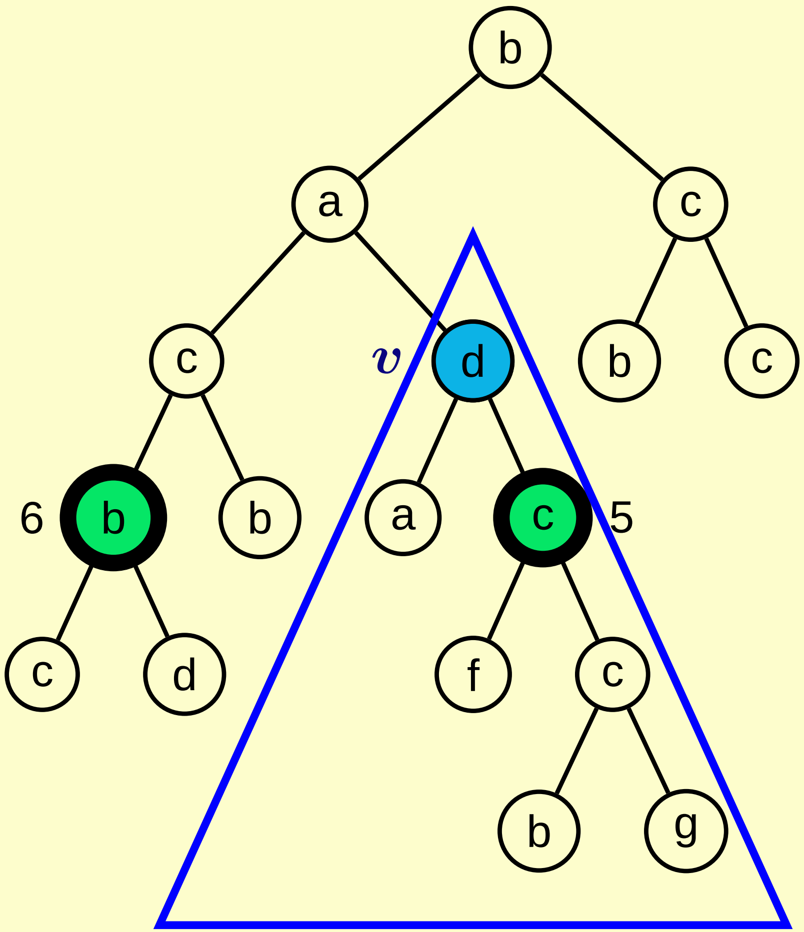
- **0-type of t_v**: all pairs (p, q) such that there is a tree 0-loop ...
- **1-type of t_v**: for each 0-type B_0 of a context C_v all pairs (p, q) such that there is a tree 1-loop ...



... and types

Definition

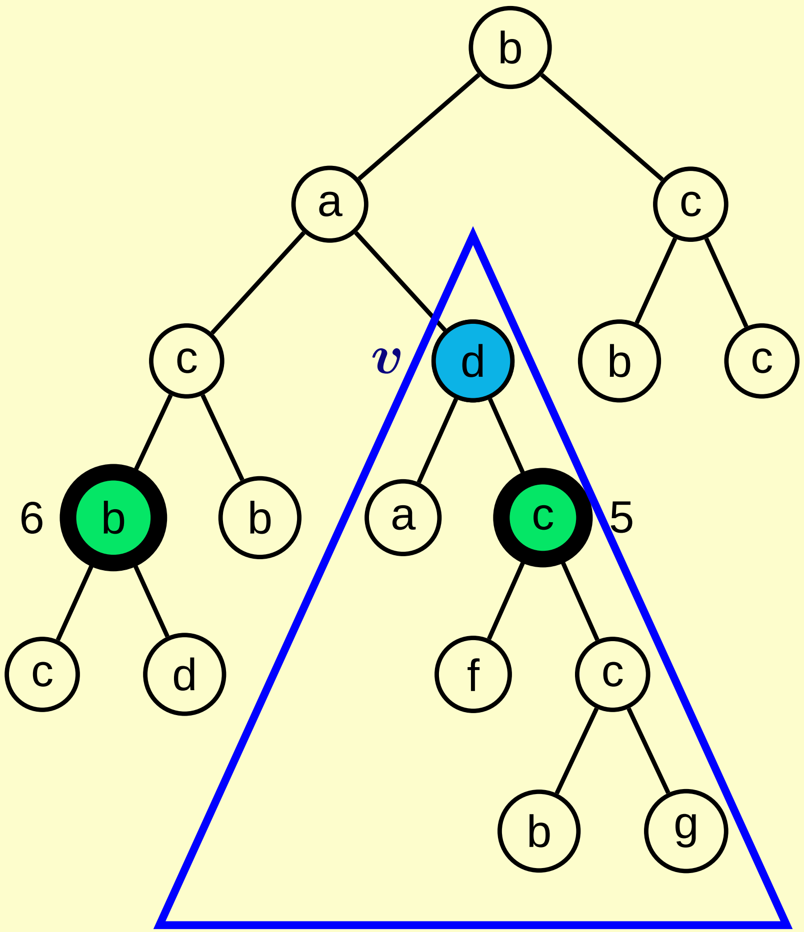
- **0-type of t_v**: all pairs (p, q) such that there is a tree $\mathbf{0}$-loop ...
- **1-type of t_v**: for each $\mathbf{0}$-type B_0 of a context C_v all pairs (p, q) such that there is a tree $\mathbf{1}$-loop ...
- **2-type of t_v**: for each $\mathbf{0}$-type B_0 and $\mathbf{1}$-type B_1 of a context C_v all pairs (p, q) such that there is a tree $\mathbf{2}$-loop ...



... and types

Definition

- 0-type of t_v: all pairs (p, q) such that there is a tree 0-loop ...
- 1-type of t_v: for each 0-type B_0 of a context C_v all pairs (p, q) such that there is a tree 1-loop ...
- 2-type of t_v: for each 0-type B_0 and 1-type B_1 of a context C_v all pairs (p, q) such that there is a tree 2-loop ...
- i-type of t_v: for each 0-type B_0,..., $(i-1)$-type B_{i-1} of a context C_v all pairs (p, q) such that there is a tree i-loop ...



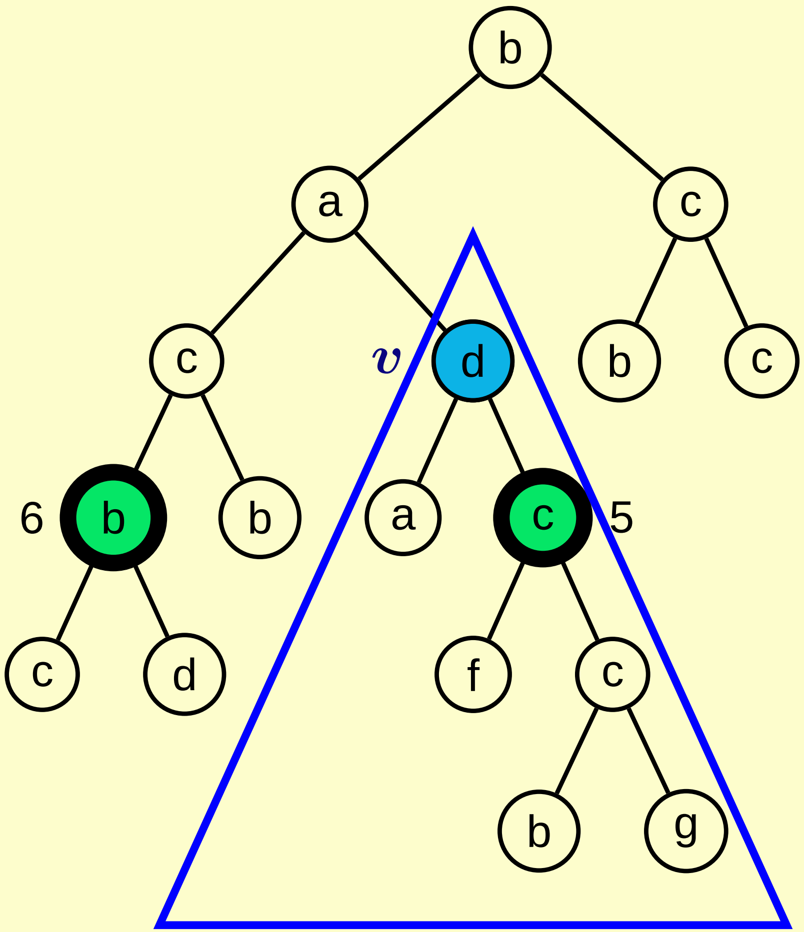
... and types

Definition

- 0-type of t_v: all pairs (p, q) such that there is a tree 0-loop ...
- 1-type of t_v: for each 0-type B_0 of a context C_v all pairs (p, q) such that there is a tree 1-loop ...
- 2-type of t_v: for each 0-type B_0 and 1-type B_1 of a context C_v all pairs (p, q) such that there is a tree 2-loop ...
- i-type of t_v: for each 0-type B_0,..., $(i-1)$-type B_{i-1} of a context C_v all pairs (p, q) such that there is a tree i-loop ...

Lemma

- There are only finitely many i-types
- i-types are compositional
- i-types can be "computed" by an i-pebble PA



Contents

Strong pebbles aren't that strong

Theorem 1

For each $n \geq 0$:

(a) $\mathbf{sPA}_n = \mathbf{PA}_n$

(b) $\mathbf{sDPA}_n = \mathbf{DPA}_n$

Proof idea

- Intermediate model: k-weak-PA
  - pebbles $k, \ldots, n$ are strong
  - pebbles $1, \ldots, k-1$ are weak
- Induction on k.
  - Each k-weak n-pebble automaton A has an equivalent $(k+1)$-weak automaton A'

$\longrightarrow$ we basically have to show how to simulate a strong pebble k by a weak pebble k (where strong pebbles $k+1, \ldots, n$ are fixed)

Strong pebbles aren't that strong

Theorem 1

For each $n \geq 0$:

(a) $\mathbf{sPA}_n = \mathbf{PA}_n$

(b) $\mathbf{sDPA}_n = \mathbf{DPA}_n$

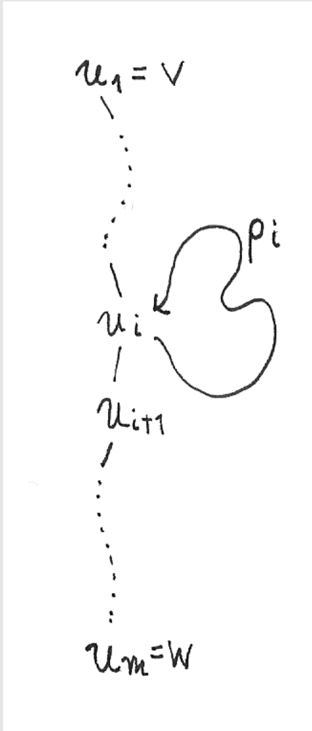
Proof idea

- Intermediate model: k-weak-PA
  - pebbles $k, \ldots, n$ are strong
  - pebbles $1, \ldots, k-1$ are weak
- Induction on k.
  - Each k-weak n-pebble automaton A has an equivalent $(k+1)$-weak automaton A'

$\longrightarrow$ we basically have to show how to simulate a strong pebble k by a weak pebble k (where strong pebbles $k+1, \ldots, n$ are fixed)

Proof idea (cont.)

- Assume A drops strong pebble k at v and lifts it when its head is at w (say: below v)
- Let $u_1 = v, u_2, \ldots, v_m = w$ path from v to w
- Idea: A' moves pebble k towards w
  - When k-configuration at u_i is reached, pebble k is moved to it
  - The head never moves above pebble k
  - Behavior of C_{u_i} is maintained inductively



- The deterministic case requires more care

Contents

Hierarchy theorems

Goal of this part:

Theorem 2

$\mathbf{PA} \subsetneq \mathbf{REG}$

We first show:

Theorem 3

For each $n \geq 0$, $\mathbf{PA}_n \subsetneq \mathbf{PA}_{n+1}$

We build on

Theorem [Bojańczyk, Colcombet 05]

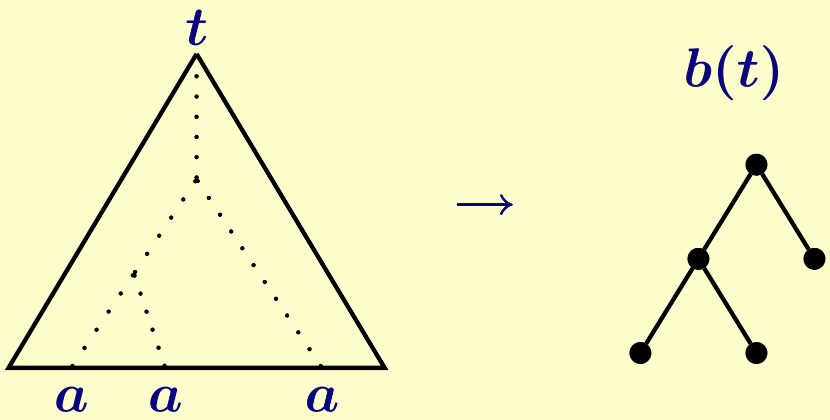
TWA $\subsetneq$ REG

A look at "TWA $\subsetneqq$ REG"

- Quasi-blank trees:
  Alphabet $\{a, b\}$, a appears only in leaves

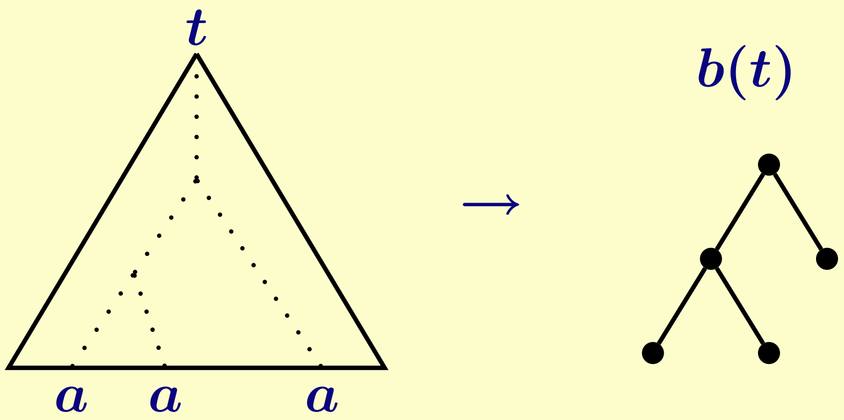
A look at "TWA $\subsetneq$ REG"

- **Quasi-blank trees**:
  Alphabet $\{a, b\}$, a appears only in leaves
- **Branching structure**:



A look at "TWA $\subsetneq$ REG"

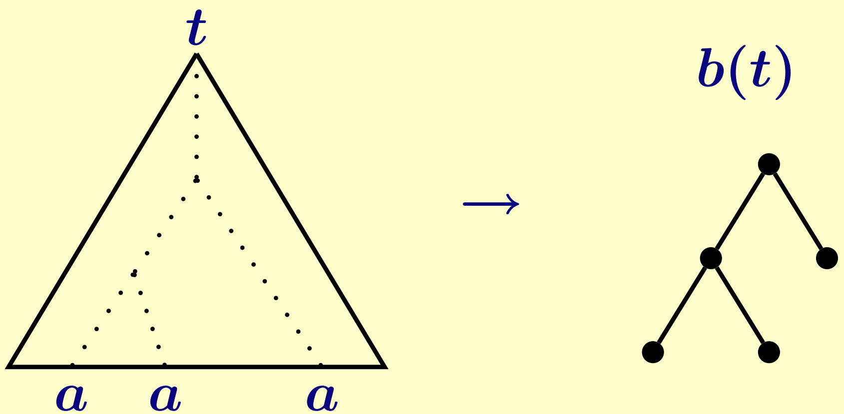
- **Quasi-blank trees**:
  Alphabet $\{a, b\}$, a appears only in leaves
- **Branching structure**:



- L_{branch}: all trees t, where in $b(t)$ all root-to-leaf paths have even length

A look at "TWA $\subsetneq$ REG"

- **Quasi-blank trees**:
  Alphabet $\{a, b\}$, a appears only in leaves
- **Branching structure**:



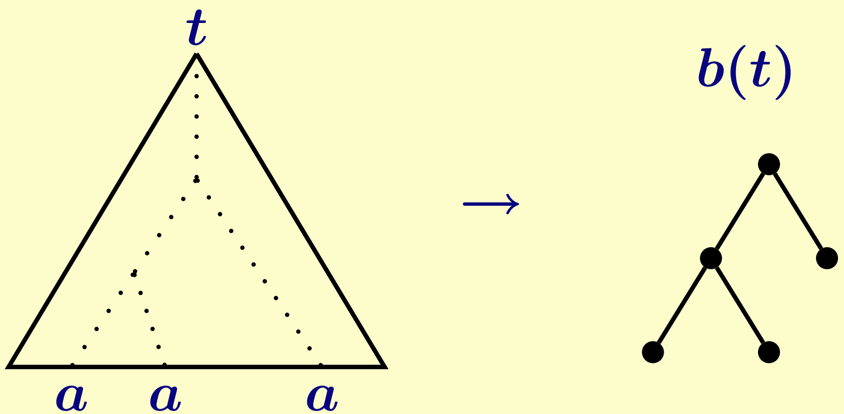
- **L_{branch}**: all trees t, where in $b(t)$ all root-to-leaf paths have even length

Theorem [Bojańczyk, Colcombet 05]

$L_{\text{branch}} \in$ REG $-$ TWA

A look at "TWA $\subsetneq$ REG"

- **Quasi-blank trees**:
  Alphabet $\{a, b\}$, a appears only in leaves
- **Branching structure**:



- L_{branch}: all trees t, where in $b(t)$ all root-to-leaf paths have even length

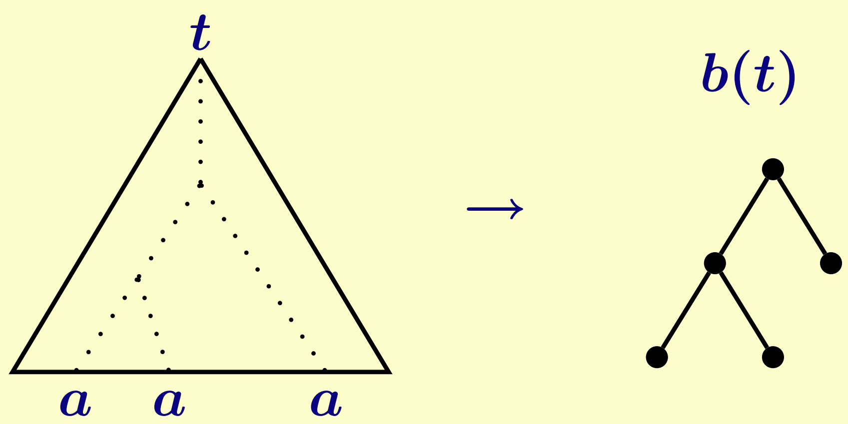
Theorem [Bojańczyk, Colcombet 05]

$L_{\text{branch}} \in$ REG $-$ TWA

- Even more: for each A there are $s \in L_{\text{branch}}$ and $t \notin L_{\text{branch}}$ such that each root-to-root loop of A on s also exists on t

A look at "TWA $\subsetneq$ REG"

- **Quasi-blank trees**:
  Alphabet $\{a, b\}$, a appears only in leaves
- **Branching structure**:



- **L_{branch}**: all trees t, where in $b(t)$ all root-to-leaf paths have even length

Theorem [Bojańczyk, Colcombet 05]

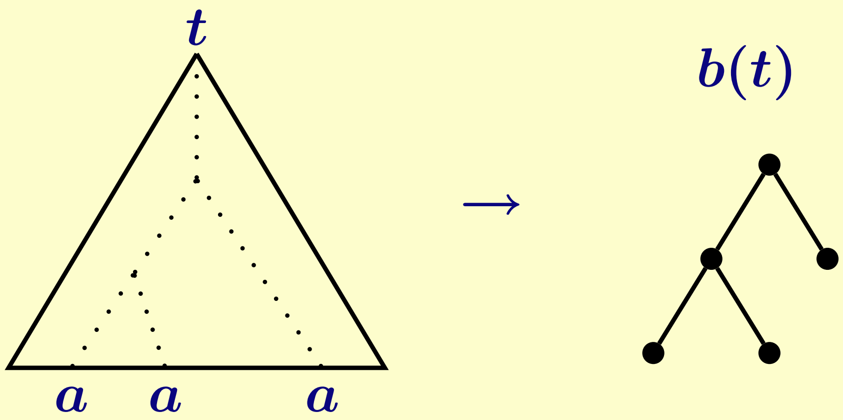
$L_{\text{branch}} \in \text{REG} - \text{TWA}$

- Even more: for each A there are $s \in L_{\text{branch}}$ and $t \notin L_{\text{branch}}$ such that each root-to-root loop of A on s also exists on t

- We need a stronger statement: the root-to-root behavior of A on s and t should be exactly the same
- **L_{even}**: Number of 0^*1-nodes v in $b(t)$ whose subtree only has even length branches is even

A look at "TWA $\subsetneq$ REG"

- **Quasi-blank trees**:
  Alphabet $\{a, b\}$, a appears only in leaves
- **Branching structure**:



- **L_{branch}**: all trees t, where in $b(t)$ all root-to-leaf paths have even length

Theorem [Bojańczyk, Colcombet 05]

$L_{\text{branch}} \in$ REG $-$ TWA

- Even more: for each A there are $s \in L_{\text{branch}}$ and $t \notin L_{\text{branch}}$ such that each root-to-root loop of A on s also exists on t

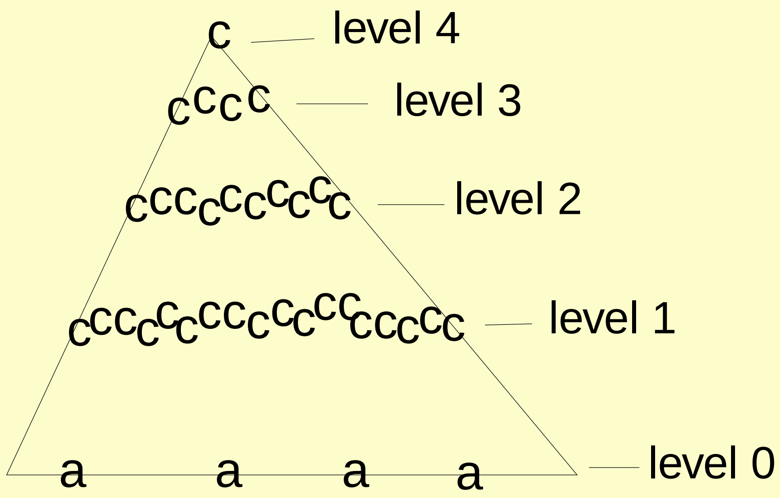
- We need a stronger statement: the root-to-root behavior of A on s and t should be exactly the same
- **L_{even}**: Number of 0^*1-nodes v in $b(t)$ whose subtree only has even length branches is even

Proposition

For each A there are $s \in L_{\text{even}}$ and $t \notin L_{\text{even}}$ such that A has the same root-to-root behavior on s and t

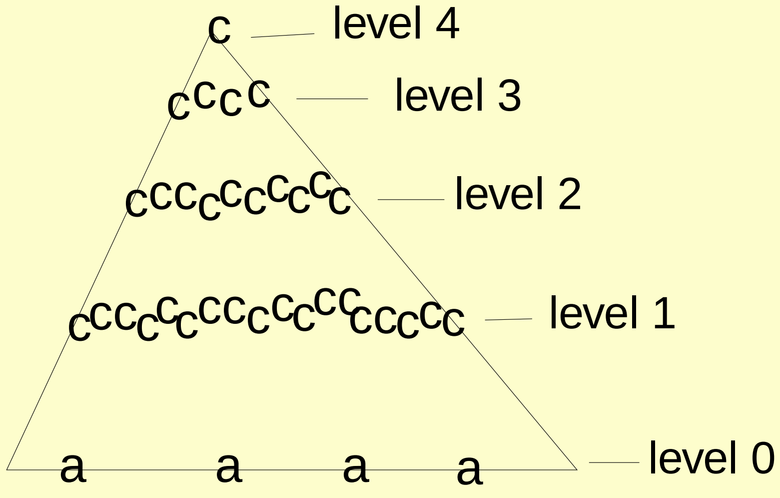
A language separating $\mathbf{PA}_n$ from $\mathbf{PA}_{n-1}$

- n-leveled tree:
  - Alphabet $\{a, b, c\}$
  - All root-to-leaf paths are in $(cb^*)^n(a + b)$

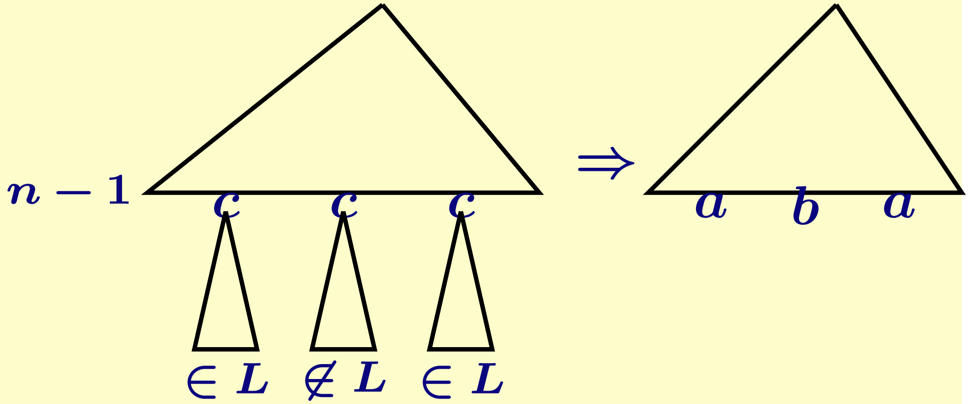


A language separating $\mathbf{PA}_n$ from $\mathbf{PA}_{n-1}$

- **n-leveled tree**:
  - Alphabet $\{a, b, c\}$
  - All root-to-leaf paths are in $(cb^*)^n(a+b)$



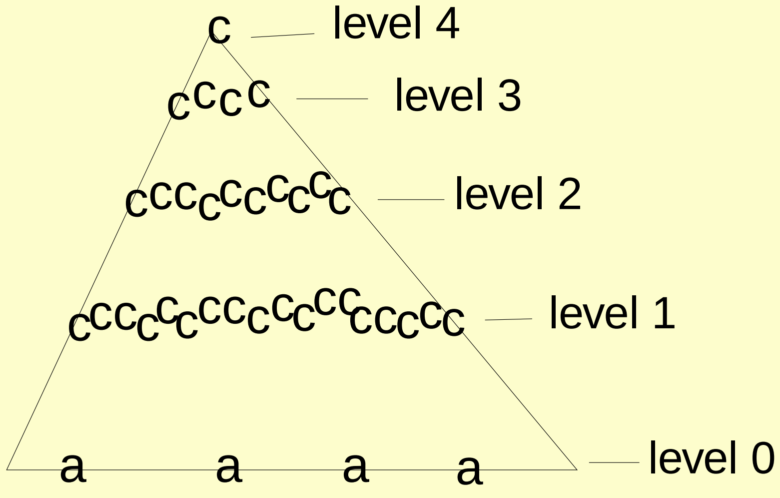
- **L-folding**:



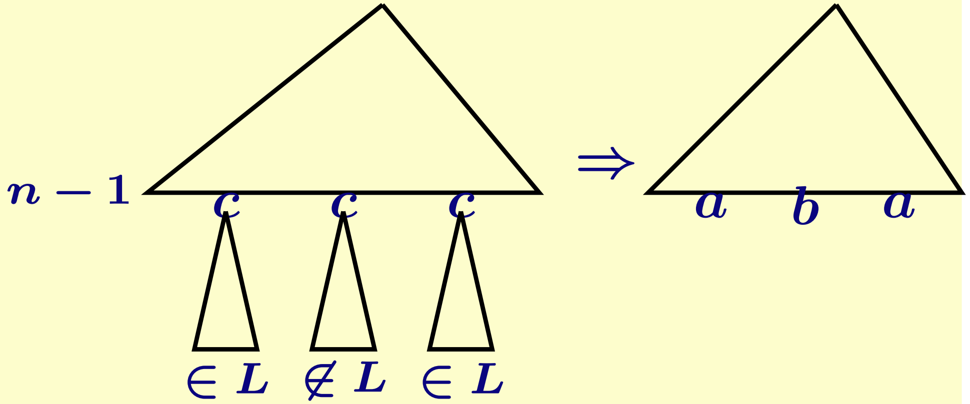
- **L_n**: all n-leveled trees whose L_{n-1}-folding is in L_{even}

A language separating $\mathbf{PA}_n$ from $\mathbf{PA}_{n-1}$

- **n-leveled tree**:
  - Alphabet $\{a, b, c\}$
  - All root-to-leaf paths are in $(cb^*)^n(a+b)$



- **L-folding**:



- **L_n**: all n-leveled trees whose L_{n-1}-folding is in L_{even}

Main Lemma

- $L_n \in \mathbf{DPA}_n$
- $L_n \notin \mathbf{PA}_{n-1}$

Proof of Main Lemma

Main Lemma

- $L_n \in \mathrm{DPA}_n$
- $L_n \notin \mathrm{PA}_{n-1}$

Proof

- Towards a contradiction, assume $L(A) = L_n$, for some $(n-1)$-pebble automaton A
- We inductively construct
  - $s_1, \ldots, s_n \in L_n$
  - $t_1, \ldots, t_n \notin L_n$

  such that s_i and t_i have equivalent i-type wrt A
- At level $i-1$, both s_i, t_i have only subtrees s_{i-1}, t_{i-1}

Proof of Main Lemma

Main Lemma

- $L_n \in \mathrm{DPA}_n$
- $L_n \notin \mathrm{PA}_{n-1}$

Proof

- Towards a contradiction, assume $L(A) = L_n$, for some $(n-1)$-pebble automaton A
- We inductively construct
  - $s_1, \ldots, s_n \in L_n$
  - $t_1, \ldots, t_n \notin L_n$

  such that s_i and t_i have equivalent i-type wrt A
- At level $i-1$, both s_i, t_i have only subtrees s_{i-1}, t_{i-1}

Proof (cont.)

Intuition:

- If pebble i is not in the tree, A acts like a TWA
- If pebble i is placed above level $i-1$, A can not distinguish s_{i-1}-subtrees from t_{i-1}-subtrees
- Subcomputations taking pebble i below level $i-1$ do not learn more than the type of the subtree

Proof of Main Lemma

Main Lemma

- $L_n \in \mathrm{DPA}_n$
- $L_n \notin \mathrm{PA}_{n-1}$

Proof

- Towards a contradiction, assume $L(A) = L_n$, for some $(n-1)$-pebble automaton A
- We inductively construct
  - $s_1, \ldots, s_n \in L_n$
  - $t_1, \ldots, t_n \notin L_n$

  such that s_i and t_i have equivalent i-type wrt A
- At level $i-1$, both s_i, t_i have only subtrees s_{i-1}, t_{i-1}

Proof (cont.)

Intuition:

- If pebble i is not in the tree, A acts like a TWA
- If pebble i is placed above level $i-1$, A can not distinguish s_{i-1}-subtrees from t_{i-1}-subtrees
- Subcomputations taking pebble i below level $i-1$ do not learn more than the type of the subtree
- Not very precise

$\longrightarrow$ Formalization through “oracle automata”

Oracle automata

- Oracle automaton:
  - basis: tree-walking automaton
  - no pebbles!
  - works on folding
  - structure oracle:
    * MSO-formulas $\varphi_1(x), \ldots, \varphi_l(x)$ which do not test labels
    * transition at node v depends on $(\varphi_1(v), \ldots, \varphi_l(v))$

Proposition

For each m there are $s \in L_1$ and $t \notin L_1$ such that each oracle automaton of size $\leq m$ has the same root-to-root loops on s and t

Proof idea for proposition

Slight extension of TWA-proof

Oracle automata

- Oracle automaton:
  - basis: tree-walking automaton
  - no pebbles!
  - works on folding
  - structure oracle:
    - MSO-formulas $\varphi_1(x),\dots,\varphi_l(x)$ which do not test labels
    - transition at node v depends on $(\varphi_1(v),\dots,\varphi_l(v))$

Proposition

For each m there are $s \in L_1$ and $t \notin L_1$ such that each oracle automaton of size $\le m$ has the same root-to-root loops on s and t

Proof idea for proposition

Slight extension of TWA-proof

Proof of Main Lemma (cont.)

- $i = 0$: follows from Proposition (even TWA-case sufficient)
- $i-1 \to i$:
  - Choose m large enough wrt A, i
  - Pick $s \in L_1$,$t \notin L_1$ such that no oracle automaton of size $\le m$ can distinguish s from t
  - $s_{i+1} := s$, where each a-leaf is replaced by s_i and each b-leaf by t_i
  - (t_{i+1} accordingly)

Oracle automata

- **Oracle automaton**:
  - basis: tree-walking automaton
  - no pebbles!
  - works on folding
  - **structure oracle**:
    - ∗ MSO-formulas $\varphi_1(x), \ldots, \varphi_l(x)$ which do not test labels
    - ∗ transition at node v depends on $(\varphi_1(v), \ldots, \varphi_l(v))$

Proposition

For each m there are $s \in L_1$ and $t \notin L_1$ such that each oracle automaton of size $\leq m$ has the same root-to-root loops on s and t

Proof idea for proposition

Slight extension of TWA-proof

Proof of Main Lemma (cont.)

- $i = 0$: follows from Proposition (even TWA-case sufficient)
- $i - 1 \rightarrow i$:
  - Choose m large enough wrt A, i
  - Pick $s \in L_1$,$t \notin L_1$ such that no oracle automaton of size $\leq m$ can distinguish s from t
  - $s_{i+1} := s$, where each a-leaf is replaced by s_i and each b-leaf by t_i
  - (t_{i+1} accordingly)

  - Assumption: i-type of A distinguishes s_{i+1} from t_{i+1}

Oracle automata

- Oracle automaton:
  - basis: tree-walking automaton
  - no pebbles!
  - works on folding
  - structure oracle:
    - MSO-formulas $\varphi_1(x), \ldots, \varphi_l(x)$ which do not test labels
    - transition at node v depends on $(\varphi_1(v), \ldots, \varphi_l(v))$

Proposition

For each m there are $s \in L_1$ and $t \notin L_1$ such that each oracle automaton of size $\leq m$ has the same root-to-root loops on s and t

Proof idea for proposition

Slight extension of TWA-proof

Proof of Main Lemma (cont.)

- $i = 0$: follows from Proposition (even TWA-case sufficient)
- $i - 1 \to i$:
  - Choose m large enough wrt A, i
  - Pick $s \in L_1$,$t \notin L_1$ such that no oracle automaton of size $\leq m$ can distinguish s from t
  - $s_{i+1} := s$, where each a-leaf is replaced by s_i and each b-leaf by t_i
  - (t_{i+1} accordingly)

  - Assumption: i-type of A distinguishes s_{i+1} from t_{i+1}

$\Rightarrow$ there is an oracle automaton O of size $\leq m$ which distinguishes s from t

$\Rightarrow$ contradiction

Finale

Proof of Main Lemma (cont.)

Simulation of $\boldsymbol{A}$ by $\boldsymbol{O}$:

- 3 cases:
  - all pebbles lifted, $\boldsymbol{A}$'s head above level $\boldsymbol{i}$:
    $\boldsymbol{O}$ just does the same as $\boldsymbol{A}$
  - all pebbles lifted, $\boldsymbol{A}$'s head moves below level $\boldsymbol{i}$:
    $\longrightarrow$ $\boldsymbol{O}$ reads $\boldsymbol{a}$ iff subtree is $\boldsymbol{s_i}$
    $\longrightarrow$ transition corresponding to $\boldsymbol{i}$-loops of $\boldsymbol{A}$ on $\boldsymbol{s_i}$ and $\boldsymbol{t_i}$
  - $\boldsymbol{A}$ drops pebble $\boldsymbol{i}$ (above level $\boldsymbol{i}$):
    subcomputation can be simulated by $\boldsymbol{(i-1)}$-pebble automaton $\boldsymbol{A'}$ which does not read labels

$\longrightarrow$ behavior of $\boldsymbol{A'}$ can be described by formulas $\boldsymbol{\varphi_1(x), \ldots, \varphi_l(x)}$

Finale

Proof of Main Lemma (cont.)

Simulation of A by O:

- 3 cases:
  - all pebbles lifted, A's head above level i:
    O just does the same as A
  - all pebbles lifted, A's head moves below level i:
    $\longrightarrow$ O reads a iff subtree is s_i
    $\longrightarrow$ transition corresponding to i-loops of A on s_i and t_i
  - A drops pebble i (above level i):
    subcomputation can be simulated by $(i-1)$-pebble automaton A' which does not read labels

$\longrightarrow$ behavior of A' can be described by formulas $\varphi_1(x), \ldots, \varphi_l(x)$

Conclude $\mathbf{PA} \subsetneq$ REG

$\bigcup_n L_n \notin \mathbf{PA}$ but also

$\bigcup_n L_n \notin \mathbf{REG}$

Finale

Proof of Main Lemma (cont.)

Simulation of A by O:

- 3 cases:
  - all pebbles lifted, A's head above level i:
    O just does the same as A
  - all pebbles lifted, A's head moves below level i:
    $\longrightarrow$ O reads a iff subtree is s_i
    $\longrightarrow$ transition corresponding to i-loops of A on s_i and t_i
  - A drops pebble i (above level i):
    subcomputation can be simulated by $(i-1)$-pebble automaton A' which does not read labels

$\longrightarrow$ behavior of A' can be described by formulas $\varphi_1(x), \ldots, \varphi_l(x)$

Conclude $\mathbf{PA} \subsetneq$ REG

$$\bigcup_n L_n \notin \mathbf{PA} \text{ but also}$$

$$\bigcup_n L_n \notin \mathbf{REG}$$

$\longrightarrow$ Choose L as the "closure of L_{even} under partial folding"

$\Rightarrow$ $L \in \mathbf{REG} - \mathbf{PA}$ and all $s_n \in L, t_n \notin L$

Further results and conclusion

Theorem

- For each $n \geq 0$,
  - $\mathbf{DPA}_n \subsetneq \mathbf{DPA}_{n+1}$
  - $\mathbf{TWA} \not\subseteq \mathbf{DPA}_n$
  - $\mathbf{sDPA}_n = \mathbf{DPA}_n$
  - $\mathbf{sDPA}_n$ is closed under complement

Open problems

- **FO** $+$ DTC $\subsetneq$ **FO** $+$ posTC?
- **FO** $+$ posTC $\subsetneq$ **FO** $+$ TC?
- **FO** $+$ TC $\subsetneq$ $\mathbf{REG}$?