Formal language theory from a functional programming perspective

Tim Hunter

22 March 2023

Outline

Outline

Linguistic Knowledge

Which are possible words (say, names for a new invention)?

ptak	thole
hlad	plast
sram	mgla
vlas	flitch
prasp	psapr
traf	ftra

Linguistic Knowledge

Compare this with your knowledge of traffic light colors

Linguistic Knowledge

Compare this with your knowledge of traffic light colors

Which are “possible traffic light colors”?

Linguistic Knowledge

Can ‘him’ refer to Bill in these sentences?

(1) a. Bill thinks Mary likes him

 b. Bill likes him

Linguistic Knowledge

Can 'him' refer to Bill in these sentences?

(1) a. Bill thinks Mary likes him
b. Bill likes him

How about these ones?

(2) a. Bill expected to see him

Linguistic Knowledge

Can ‘him’ refer to Bill in these sentences?

(1) a. Bill thinks Mary likes him
    b. Bill likes him

How about these ones?

(2) a. Bill expected to see him
    b. I wonder who Bill expected to see him
    c. I think that Bill expected to see him

(NB: These last two have zero Google hits.)

Linguistic Knowledge

How do these ‘guess what’ sentences sound?

(3) a. John looked at something
    b. Guess what John looked at

(4) a. John likes stories about something at bedtime
    b. Guess what John likes stories about at bedtime

Linguistic Knowledge

How do these ‘guess what’ sentences sound?

(3) a. John looked at something
b. Guess what John looked at

(4) a. John likes stories about something at bedtime
b. Guess what John likes stories about at bedtime

How about this one?

(5) a. Stories about something please John at bedtime
b. Guess what stories about please John at bedtime

Linguistic Knowledge

Investigate some yourself, using (3) as a model:

(3) a. John looked at <u>something</u>
    b. Guess what John looked at

(6) a. Mary said that John bought <u>something</u> at the store
    b. . . .

(7) a. Mary complained because John bought <u>something</u> at the store
    b. . . .

(8) a. Mary wondered whether John bought <u>something</u> at the store
    b. . . .

Linguistic Knowledge

Investigate some yourself, using (3) as a model:

(3) a. John looked at <u>something</u>
    b. Guess what John looked at

(6) a. Mary said that John bought <u>something</u> at the store
    b. Guess what Mary said that John bought at the store

(7) a. Mary complained because John bought <u>something</u> at the store
    b. . . .

(8) a. Mary wondered whether John bought <u>something</u> at the store
    b. . . .

Linguistic Knowledge

Investigate some yourself, using (3) as a model:

(3) a. John looked at <u>something</u>
    b. Guess what John looked at

(6) a. Mary said that John bought <u>something</u> at the store
    b. Guess what Mary said that John bought at the store

(7) a. Mary complained because John bought <u>something</u> at the store
    b. Guess what Mary complained because John bought at the store

(8) a. Mary wondered whether John bought <u>something</u> at the store
    b. . . .

Linguistic Knowledge

Investigate some yourself, using (3) as a model:

(3) a. John looked at <u>something</u>
    b. Guess what John looked at

(6) a. Mary said that John bought <u>something</u> at the store
    b. Guess what Mary said that John bought at the store

(7) a. Mary complained because John bought <u>something</u> at the store
    b. Guess what Mary complained because John bought at the store

(8) a. Mary wondered whether John bought <u>something</u> at the store
    b. Guess what Mary wondered whether John bought at the store

Linguistic Knowledge

How different are the meanings of these two sentences?

(9) a. Bill persuaded the doctor to examine Mary
    b. Bill persuaded Mary to be examined by the doctor

Linguistic Knowledge

How different are the meanings of these two sentences?

(9) a. Bill persuaded the doctor to examine Mary
    b. Bill persuaded Mary to be examined by the doctor

How about these two?

(10) a. Bill expected the doctor to examine Mary
     b. Bill expected Mary to be examined by the doctor

Linguistic Knowledge

What effect does leaving off 'the apple' have on the meaning here?

(11) a. John ate the apple
b. John ate

Linguistic Knowledge

What effect does leaving off ‘the apple’ have on the meaning here?

(11) a. John ate the apple
b. John ate

What effect does leaving off ‘the professor’ have here?

(12) a. John is too stubborn to talk to the professor
b. John is too stubborn to talk to

The big question

What is the unconscious "knowledge" underlying these judgements (and how did we acquire it)?

A **grammar** is a hypothesis about what that knowledge looks like.

A **grammar formalism** is a hypothesis about what the acquisition task's search space looks like, i.e. a hypothesis about what constitutes a "possible human language".

Potential distractions: Other questions

We can distinguish these puzzles from some **very different** questions which might, at first glance, seem similar:

Potential distractions: Other questions

We can distinguish these puzzles from some **very different** questions which might, at first glance, seem similar:

- Why do we say 'the teacher taught' but not 'the preacher praught'?
- Why is it that we call a place where one drives a parkway, and we call a place where one parks a driveway?

Potential distractions: Other questions

We can distinguish these puzzles from some **very different** questions which might, at first glance, seem similar:

- Why do we say ‘the teacher taught’ but not ‘the preacher praught’?
- Why is it that we call a place where one drives a parkway, and we call a place where one parks a driveway?

- In what situation would a person be likely to say to someone ‘Guess what John likes stories about’?

Potential distractions: Other questions

We can distinguish these puzzles from some **very different** questions which might, at first glance, seem similar:

- Why do we say ‘the teacher taught’ but not ‘the preacher praught’?
- Why is it that we call a place where one drives a parkway, and we call a place where one parks a driveway?

- In what situation would a person be likely to say to someone ‘Guess what John likes stories about’?
- How are those situations different from situations where a person would say ‘John likes certain stories’?
- How often do such situations arise?

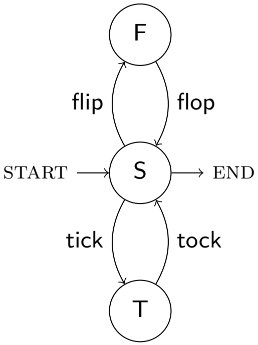
Outline

Serial dependencies

(13)
a. flip flop
b. tick tock
c. flip flop tick tock
d. tick tock flip flop flip flop
e. tick tock flip flop tick tock tick tock

Serial dependencies

(13)
a. flip flop
b. tick tock
c. flip flop tick tock
d. tick tock flip flop flip flop
e. tick tock flip flop tick tock tick tock



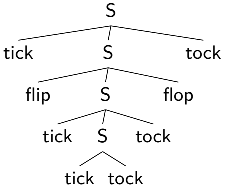
Nesting dependencies

(14)
a. flip flop
b. tick tock
c. flip tick tock flop
d. tick flip flip flop flop tock
e. tick flip tick tick tock tock flop tock

Nesting dependencies

(14)
a. flip flop
b. tick tock
c. flip tick tock flop
d. tick flip flip flop flop tock
e. tick flip tick tick tock tock flop tock

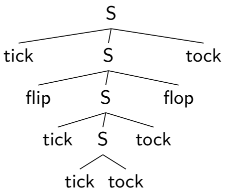
S → flip (S) flop
S → tick (S) tock



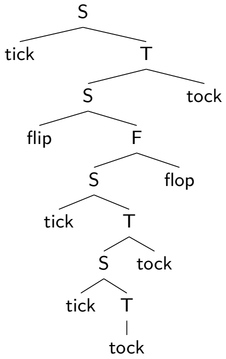
Nesting dependencies

(14)
  a. flip flop
  b. tick tock
  c. flip tick tock flop
  d. tick flip flip flop flop tock
  e. tick flip tick tick tock tock flop tock

S → flip (S) flop
S → tick (S) tock



S → flip F
F → (S) flop
S → tick T
T → (S) tock

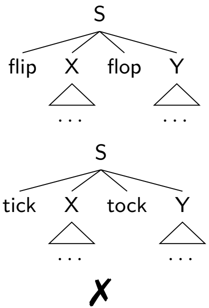


Crossing dependencies

(15)
a. flip flop
b. tick tock
c. flip tick flop tock
d. tick flip flip tock flop flop
e. tick flip tick tick tock flop tock tock

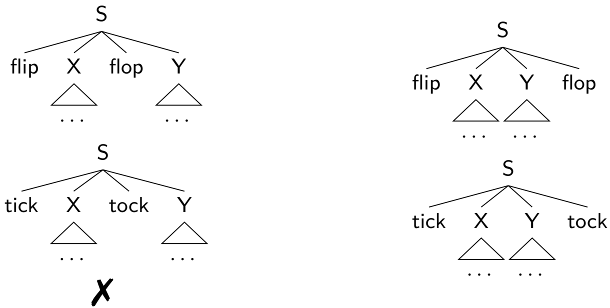
Crossing dependencies

(15)
a. flip flop
b. tick tock
c. flip tick flop tock
d. tick flip flip tock flop flop
e. tick flip tick tick tock flop tock tock



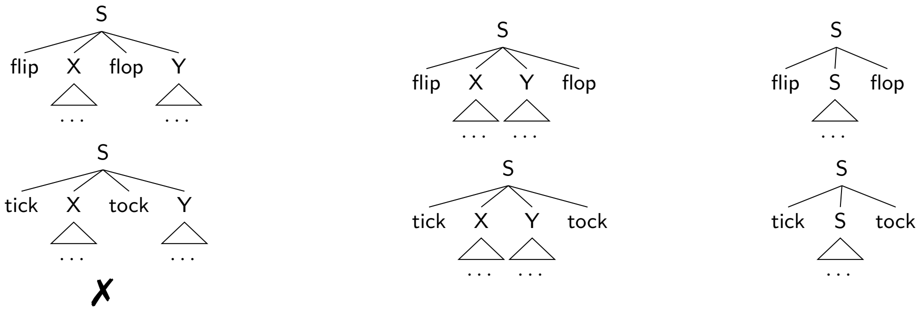
Crossing dependencies

(15)
a. flip flop
b. tick tock
c. flip tick flop tock
d. tick flip flip tock flop flop
e. tick flip tick tick tock flop tock tock



Crossing dependencies

(15)
a. flip flop
b. tick tock
c. flip tick flop tock
d. tick flip flip tock flop flop
e. tick flip tick tick tock flop tock tock



Serial dependencies (English):

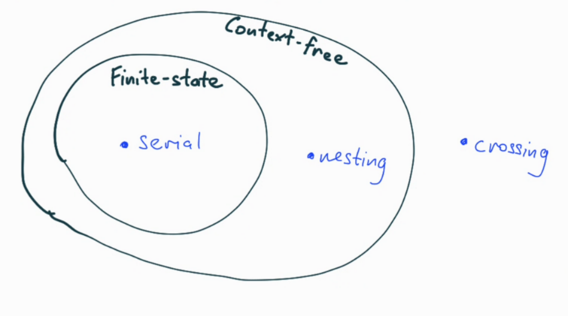
(16) John saw Peter let Marie swim

Nesting dependencies (German):

(17) . . . dass Hans Peter Marie schwimmen lassen sah
that Hans Peter Marie swim let saw

Crossing dependencies (Dutch):

(18) . . . dat Jan Piet Marie zag laten zwemmen
that Jan Piet Marie saw let swim


Context-free
Finite-state
serial
nesting
crossing

Outline

What Linguists Worry About

Starting points: Finite-state and context-free grammars

Recursion on strings

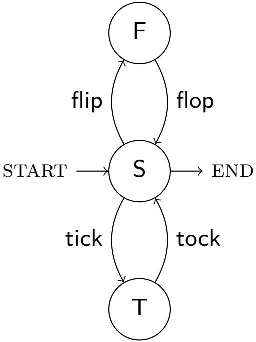
- Finite memory
- Unbounded stack memory

Recursion on trees

- Finite memory
- Unbounded stack memory

Outline

Finite-state (string) automata



A finite-state automaton (FSA) (over an alphabet Σ) is a four-tuple (Q, I, F, Δ) where:

- Q is a finite set of states;
- $I \subseteq Q$ is the set of initial states;
- $F \subseteq Q$ is the set of ending states; and
- $\Delta \subseteq Q \times \Sigma \times Q$ is the set of transitions.

$(\{\mathrm{S}, \mathrm{F}, \mathrm{T}\}, \{\mathrm{S}\}, \{\mathrm{S}\}, \{(\mathrm{S}, \mathrm{flip}, \mathrm{F}), (\mathrm{F}, \mathrm{flop}, \mathrm{S}), (\mathrm{S}, \mathrm{tick}, \mathrm{T}), (\mathrm{T}, \mathrm{tock}, \mathrm{S})\})$

```
type FSA st sym = ([st], [st], [st], [(st,sym,st)])

fsa5 :: FSA Char String
fsa5 = (['S','F','T'], ['S'], ['S'], [ ('S',"flip",'F'), ('F',"flop",'S')
                                     , ('S',"tick",'T'), ('T',"tock",'S') ])
```

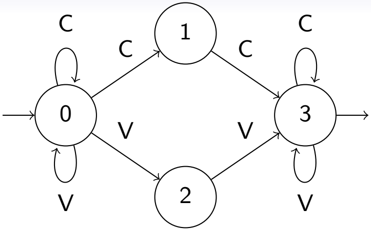
Forward values

We'll define a function $\text{fwd}_M : \Sigma^* \times Q \to \{0, 1\}$, for any FSA M, such that $\text{fwd}_M(w)(q)$ is true iff there's a path through M from some initial state to the state q, emitting the string w.

$$w \in \mathcal{L}(M) \iff \bigvee_{q \in Q} \Big[\text{fwd}_M(w)(q) \wedge F(q) \Big]$$

$$\text{fwd}_M(\varepsilon)(q) = I(q)$$

$$\text{fwd}_M(x_1 \dots x_n)(q) = \bigvee_{q' \in Q} \Big[\text{fwd}_M(x_1 \dots x_{n-1})(q') \wedge \Delta(q', x_n, q) \Big]$$



$$\mathsf{fwd}_M(\varepsilon)(q) = I(q)$$

$$\mathsf{fwd}_M(x_1 \dots x_n)(q) = \bigvee_{q' \in Q} \left[\mathsf{fwd}_M(x_1 \dots x_{n-1})(q') \wedge \Delta(q', x_n, q) \right]$$

Table of forward values:

State		C	V	C	C	V
0	1	1	1	1	1	1
1	0	1	0	1	1	0
2	0	0	1	0	0	1
3	0	0	0	0	1	1

Forward values

We'll define a function $\mathrm{fwd}_M : \Sigma^* \times Q \to \{0,1\}$, for any FSA M, such that $\mathrm{fwd}_M(w)(q)$ is true iff there's a path through M from some initial state to the state q, emitting the string w.

$$w \in \mathcal{L}(M) \iff \bigvee_{q \in Q} \Big[\mathrm{fwd}_M(w)(q) \wedge F(q) \Big]$$

$$\mathrm{fwd}_M(\varepsilon)(q) = I(q)$$

$$\mathrm{fwd}_M(x_1 \dots x_n)(q) = \bigvee_{q' \in Q} \Big[\mathrm{fwd}_M(x_1 \dots x_{n-1})(q') \wedge \Delta(q', x_n, q) \Big]$$

Forward values

We'll define a function $\text{fwd}_M : \Sigma^* \times Q \to \{0,1\}$, for any FSA M, such that $\text{fwd}_M(w)(q)$ is true iff there's a path through M from some initial state to the state q, emitting the string w.

$$w \in \mathcal{L}(M) \iff \bigvee_{q \in Q} \Big[\text{fwd}_M(w)(q) \wedge F(q) \Big]$$

$$\text{fwd}_M(\varepsilon)(q) = I(q)$$

$$\text{fwd}_M(x_1 \dots x_n)(q) = \bigvee_{q' \in Q} \Big[\text{fwd}_M(x_1 \dots x_{n-1})(q') \wedge \Delta(q', x_n, q) \Big]$$

```
fsa_forward :: FSA st sym -> SnocList sym -> st -> Bool
fsa_forward fsa@(states, istates, fstates, delta) str q =
    case str of
    Nil -> istates q
    Snoc xs x -> or [fsa_forward fsa xs q' && delta (q',x,q) | q' <- states]

fsa_fwd :: FSA st sym -> [sym] -> st -> Bool
fsa_fwd fsa@(states, istates, fstates, delta) =
    foldl (\acc -> \x -> \q -> or [acc q' && delta (q',x,q) | q' <- states]) istates
```

Backward values

We'll define a function $\text{bwd}_M : \Sigma^* \times Q \to \{0, 1\}$, for any FSA M, such that $\text{bwd}_M(w)(q)$ is true iff there's a path through M from the state q to some ending state, emitting the string w.

$$w \in \mathcal{L}(M) \iff \bigvee_{q \in Q} \Big[I(q) \wedge \text{bwd}_M(w)(q) \Big]$$

$$\text{bwd}_M(\varepsilon)(q) = F(q)$$

$$\text{bwd}_M(x_1 \dots x_n)(q) = \bigvee_{q' \in Q} \Big[\Delta(q, x_1, q') \wedge \text{bwd}_M(x_2 \dots x_n)(q') \Big]$$

```
fsa_backward :: FSA st sym -> [sym] -> st -> Bool
fsa_backward fsa@(states, istates, fstates, delta) str q =
    case str of
    []   -> fstates q
    x:xs -> or [delta (q,x,q') && fsa_backward fsa xs q' | q' <- states]

fsa_bwd :: FSA st sym -> [sym] -> (st -> Bool)
fsa_bwd fsa@(states, istates, fstates, delta) =
    foldr (\x -> \acc -> \q -> or [delta (q,x,q') && acc q' | q' <- states]) fstates
```

Outline

What Linguists Worry About

Starting points: Finite-state and context-free grammars

Recursion on strings

- Finite memory
- Unbounded stack memory

Recursion on trees

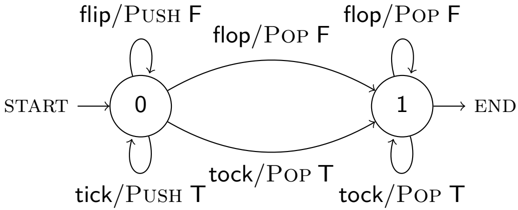
- Finite memory
- Unbounded stack memory

Pushdown (string) automata

A pushdown automaton (PDA) (over an alpabet Σ) is a five-tuple $(Q, \Gamma, I, F, \Delta)$ where:

- Q is a finite set of states;
- Γ, the stack alphabet, is a finite set of symbols disjoint from Σ;
- $I \subseteq Q$ is the set of initial states;
- $F \subseteq Q$ is the set of ending states; and
- $\Delta \subseteq Q \times \Sigma \times O(\Gamma) \times Q$ is the set of transitions.

Stack operations: $O(\Gamma) = \{\textsc{Push}\ x \mid x \in \Gamma\} \cup \{\textsc{Pop}\ x \mid x \in \Gamma\} \cup \{\textsc{Nop}\}$




```
pda2 :: PDA Int Char String
pda2 = ([0,1], ['F','T'], (`elem` [0]), (`elem` [1]),
            (`elem` [(0, "flip", Push 'F', 0)
                    ,(0, "tick", Push 'T', 0)
                    ,(0, "flop", Pop 'F', 1)
                    ,(0, "tock", Pop 'T', 1)
                    ,(1, "flop", Pop 'F', 1)
                    ,(1, "tock", Pop 'T', 1)
                    ]))
```

Forward values

For a PDA M, we define a function $\text{fwd}_M : \Sigma^* \times (Q \times \Gamma^*) \to \{0, 1\}$, such that $\text{fwd}_M(w)(q, s)$ is true iff there's a path through M from some initial state with empty stack to the state q with stack s, emitting the string w.

```
type PDA st stksym sym = ([st], [stksym], st -> Bool, st -> Bool,
                          (st, sym, StackOp stksym, st) -> Bool)

pda_forward :: PDA st stksym sym -> SnocList sym -> (st,[stksym]) -> Bool
pda_forward m@(states, stksyms, istates, fstates, delta) str (q,stk) =
    case str of
    Nil       -> istates q && null stk
    Snoc xs x -> or [pda_forward m xs (q',stk') && delta (q', x, op, q)
                            | q' <- states, (stk',op) <- stkpredecessors stksyms stk]

pda_fwd :: PDA st stksym sym -> [sym] -> (st,[stksym]) -> Bool
pda_fwd (states, stksyms, istates, fstates, delta) =
    foldl (\acc -> \x -> \(q,stk) -> or [acc (q',stk') && delta (q', x, op, q)
                            | q' <- states, (stk',op) <- stkpredecessors stksyms stk])
          (\(q,stk) -> istates q && null stk)
```

So here a symbol from Σ corresponds to binary relation on (state,stack) pairs.
Or a function that maps a (stack,stack) pair to a set of such pairs.

Outline

Outline

What Linguists Worry About

Starting points: Finite-state and context-free grammars

Recursion on strings

- Finite memory
- Unbounded stack memory

Recursion on trees

- Finite memory
- Unbounded stack memory

Finite-state tree automata

For an alphabet Σ, we define T_Σ as the smallest set such that:

- if $x \in \Sigma$, then $x[] \in T_\Sigma$, and
- if $x \in \Sigma$ and $t_1, t_2, \ldots, t_k \in T_\Sigma$, then $x[t_1, t_2, \ldots, t_k] \in T_\Sigma$.

A finite-state tree automaton (FSTA) (over an alphabet Σ) is a four-tuple (Q, F, Δ) where:

- Q is a finite set of states;
- $F \subseteq Q$ is the set of ending states; and
- $\Delta \subseteq Q^* \times \Sigma \times Q$ is the set of transitions, which must be finite.

Finite-state tree automata

For an alphabet Σ, we define T_Σ as the smallest set such that:

- if $x \in \Sigma$, then $x[] \in T_\Sigma$, and
- if $x \in \Sigma$ and $t_1, t_2, \ldots, t_k \in T_\Sigma$, then $x[t_1, t_2, \ldots, t_k] \in T_\Sigma$.

A finite-state tree automaton (FSTA) (over an alphabet Σ) is a four-tuple (Q, F, Δ) where:

- Q is a finite set of states;
- $F \subseteq Q$ is the set of ending states; and
- $\Delta \subseteq Q^* \times \Sigma \times Q$ is the set of transitions, which <u>must be finite</u>.

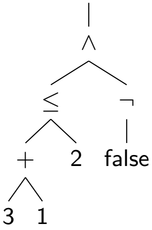
For any FSTA $M = (Q, \Sigma, F, \Delta)$, under_M is a function from $T_\Sigma \times Q$ to booleans:

$$\text{under}_M(x[])(q) = \Delta([], x, q)$$

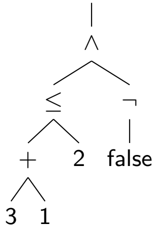
$$\text{under}_M(x[t_1, \ldots, t_k])(q) = \bigvee_{q_1 \in Q} \cdots \bigvee_{q_k \in Q} \Big[\Delta([q_1, \ldots, q_k], x, q) \wedge \text{under}_M(t_1)(q_1) \wedge \cdots \wedge \text{under}_M(t_k)(q_k)\Big]$$

(When $k = 1$, this is just like strings!)

$$t \in \mathcal{L}(M) \iff \bigvee_{q \in Q} \Big[\text{under}_M(t)(q) \wedge F(q)\Big]$$



$(3 + 1 \leq 2) \wedge (\neg \text{false})$

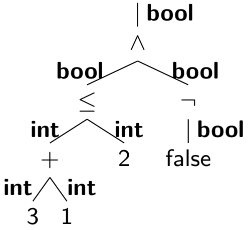


$(3 + 1 \leq 2) \wedge (\neg\text{false})$

$$\Sigma = \{\text{false}, \text{true}, 1, 2, 3, \ldots, \neg, +, \leq, \vee, \wedge, \ldots\}$$

$$Q = \{\textbf{int}, \textbf{bool}\}$$

$$\Delta = \left\{ \begin{array}{lcccl} (& (), & 1, & \textbf{int} &), \\ (& (), & 2, & \textbf{int} &), \\ (& (), & 3, & \textbf{int} &), \\ (& (), & \text{true}, & \textbf{bool} &), \\ (& (), & \text{false}, & \textbf{bool} &), \\ (& (\textbf{bool}), & \neg, & \textbf{bool} &), \\ (& (\textbf{int}, \textbf{int}), & +, & \textbf{int} &), \\ (& (\textbf{int}, \textbf{int}), & \leq, & \textbf{bool} &), \\ (& (\textbf{bool}, \textbf{bool}), & \wedge, & \textbf{bool} &), \\ (& (\textbf{bool}, \textbf{bool}), & \vee, & \textbf{bool} &) \end{array} \right\}$$



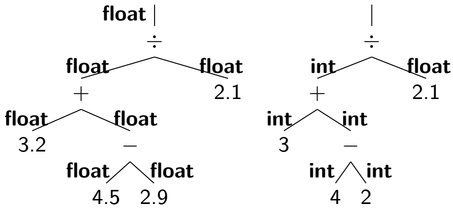
$(3 + 1 \leq 2) \wedge (\neg \text{false})$

$$\Sigma = \{\text{false}, \text{true}, 1, 2, 3, \ldots, \neg, +, \leq, \vee, \wedge, \ldots\}$$

$$Q = \{\mathbf{int}, \mathbf{bool}\}$$

$$\Delta = \left\{ \begin{array}{lcccl}
(& (), & 1, & \mathbf{int} &), \\
(& (), & 2, & \mathbf{int} &), \\
(& (), & 3, & \mathbf{int} &), \\
(& (), & \text{true}, & \mathbf{bool} &), \\
(& (), & \text{false}, & \mathbf{bool} &), \\
(& (\mathbf{bool}), & \neg, & \mathbf{bool} &), \\
(& (\mathbf{int}, \mathbf{int}), & +, & \mathbf{int} &), \\
(& (\mathbf{int}, \mathbf{int}), & \leq, & \mathbf{bool} &), \\
(& (\mathbf{bool}, \mathbf{bool}), & \wedge, & \mathbf{bool} &), \\
(& (\mathbf{bool}, \mathbf{bool}), & \vee, & \mathbf{bool} &)
\end{array} \right\}$$

The states allow us to encode non-local dependencies between tree nodes.



$$\Delta = \left\{ \begin{array}{lccccr} & \ldots & \ldots & \ldots & & \\ (& (\textbf{int}, \textbf{int}), & +, & \textbf{int} &), \\ (& (\textbf{float}, \textbf{float}), & +, & \textbf{float} &), \\ (& (\textbf{int}, \textbf{int}), & -, & \textbf{int} &), \\ (& (\textbf{float}, \textbf{float}), & -, & \textbf{float} &), \\ (& (\textbf{float}, \textbf{float}), & \div, & \textbf{float} &) \end{array} \right\}$$

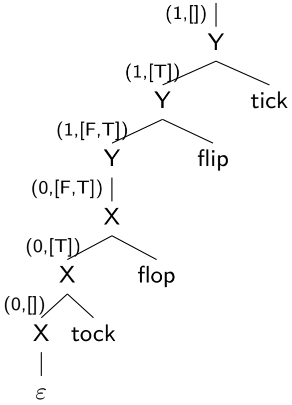
Outline

Some examples of more interesting string languages (with *crossing dependencies*):

- $\{ww \mid w \in \{\mathrm{a}, \mathrm{b}\}^*\}$
- $\{\mathrm{a}^n\mathrm{b}^n\mathrm{c}^n \mid n \geq 0\}$
- $\{\mathrm{a}^n\mathrm{b}^n\mathrm{c}^n\mathrm{d}^n \mid n \geq 0\}$
- $\{\mathrm{a}^i\mathrm{b}^j\mathrm{c}^i\mathrm{d}^j \mid i \geq 0, j \geq 0\}$

Pushdown tree automata

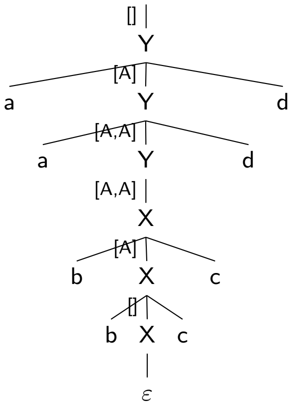
With a stack as our memory as we process a **tree**, we can generate palindrome patterns along the leaves of a strictly left-branching (or strictly right-branching) tree.



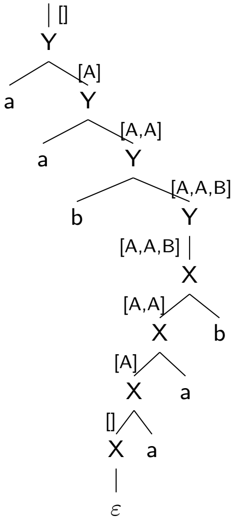
Pushdown tree automata

But when we combine this stack-based memory with center-embedding structures, magic happens!

Here's the rough idea for how we can generate $\{a^n b^n c^n d^n \mid n \geq 0\}$.



Here's the rough idea for how we can generate $\{ww \mid w \in \{\text{a}, \text{b}\}^*\}$.



(19)	daß	mer	d'chind	em Hans	es huus	lond	hälfe	aastrüche
	that	we	the children.ACC	Hans.DAT	the house.ACC	let	help	paint

"that we let the children help Hans paint the house"