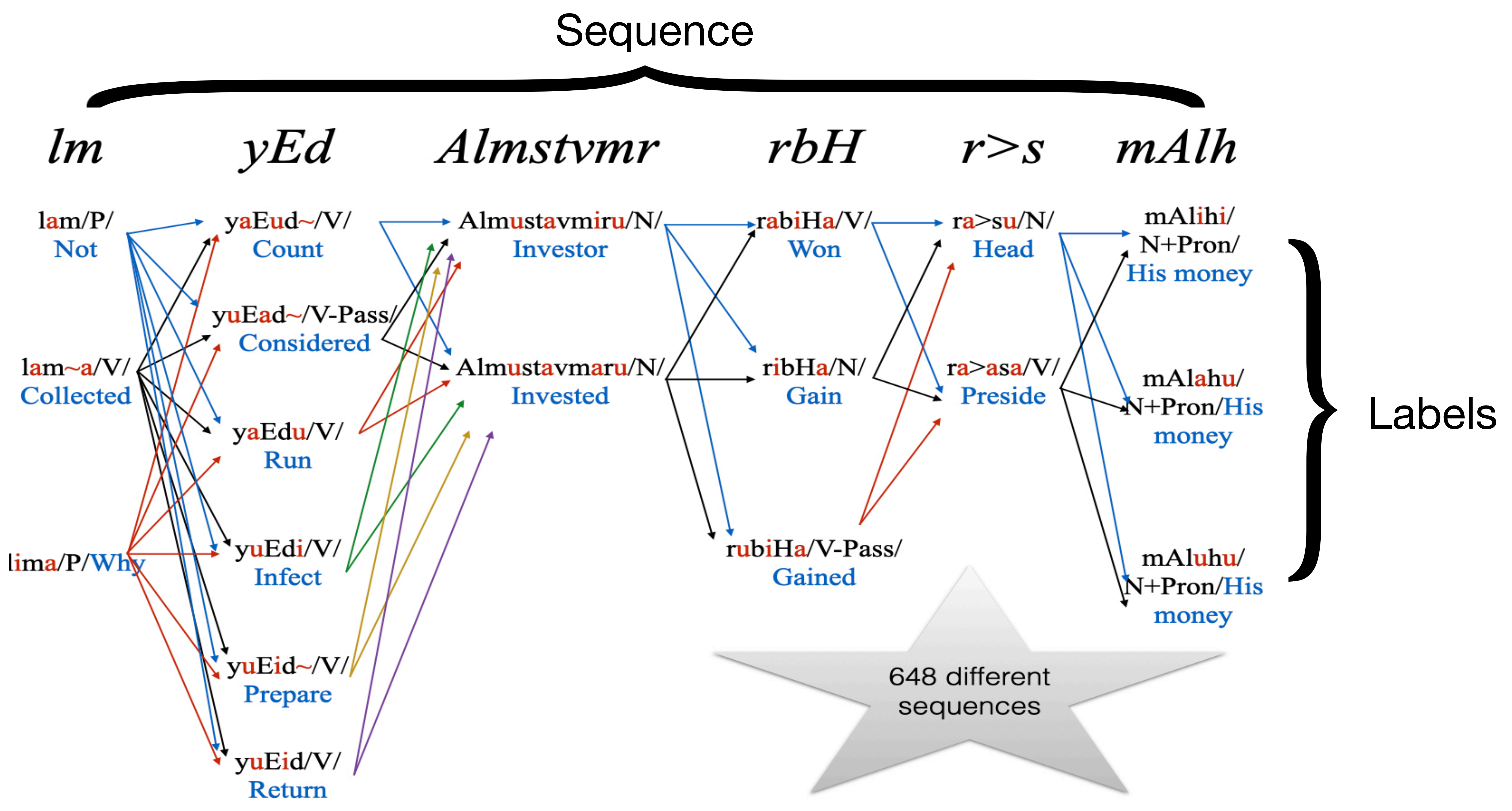
Introduction to Sequence Labeling

Mohamed Elbadrashiny

Some slides are adapted from Mona Diab, and Raymond Mooney

What is the sequence labeling?

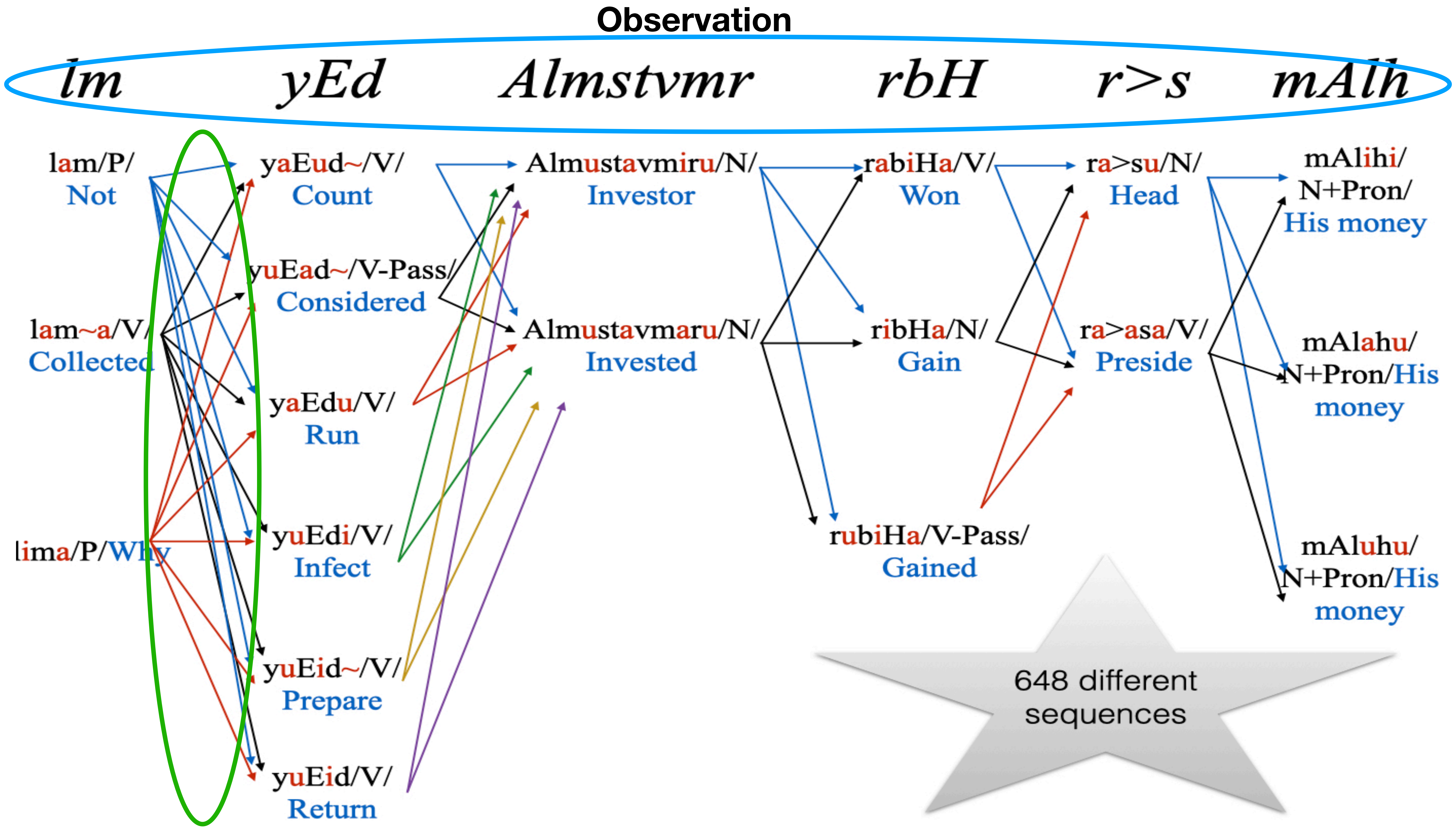
- Many NLP problems can be viewed as sequence labeling
- Each token in a sequence is assigned a label
- Labels of tokens are dependent on the labels of other tokens in the sequence, particularly their neighbors



Hidden Markov Model (HMM)

- A finite state machine with probabilistic state transitions.
- Markov assumptions:
  - The probability of a state depends only on the state that precedes it
    $P(w_i | w_1, w_2, \ldots w_{i-1}) \approx P(w_i | w_{i-1})$ →Transition probability (likelihood)
  - The probability of an output observation depends only on the state that produced the observation
    $P(q_i | q_1^{i-1}, w_1^i) \approx P(q_i | w_i)$ →Prior probability

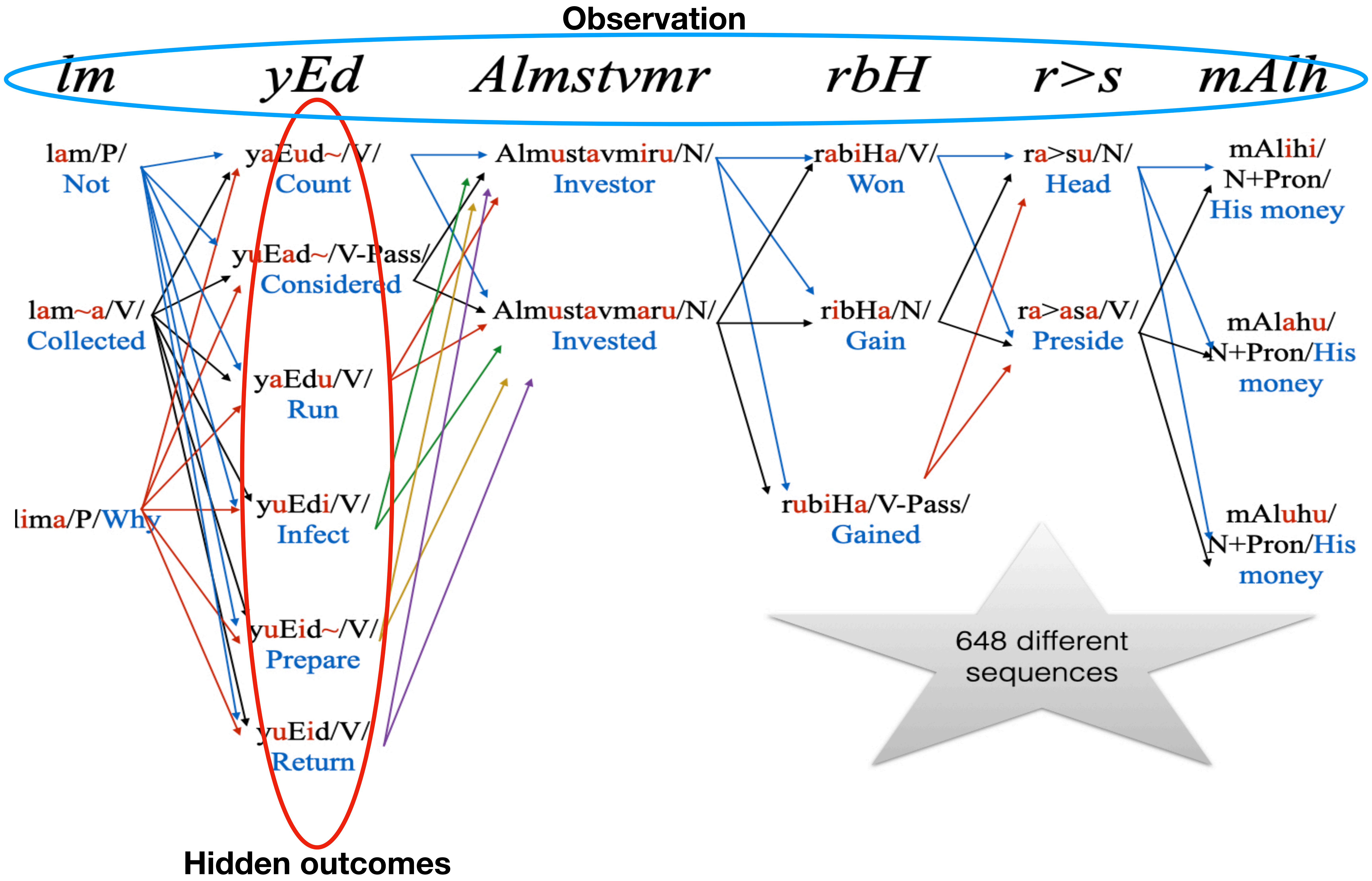
Hidden Markov Model (HMM)



Transitions

$P(yaEud\sim|\,lam) \triangleq$ transition probability

Hidden Markov Model (HMM)



$P(yEd \mid yaEud \sim) \triangleq$ prior probability

Diacritization as a sequence labeling task

- Given a sentence (Sequence of observations):
  q_1^n: lm yEd Almstvmr rbi r>s mAlh

- What is the best sequence of diacritized words (the "hidden" outcomes) which corresponds to this sequence of observations?
  $\hat{w}_1^n$: lam yaEud~ Almustavmiru ribHa ra>si mAlihi

- Probabilistic Bayesian approach:
  - Consider all possible sequences of tags (diacritizations)
  - Out of all sequences of n tags $w_1 \ldots w_{n,}$, we want the single tag sequence such that $P(w_1 \ldots w_n \mid q_1 \ldots q_n)$ is highest:

$$\hat{w}_1^n = \underset{w_1^n}{\operatorname{argmax}} P(w_1^n \mid q_1^n)$$ →eq(1)

How to compute this?

Diacritization as a sequence labeling task

- Bay's Rule:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

.

Diacritization as a sequence labeling task

- Bay's Rule:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

- Rewrite eq(1):

$$\hat{w}_1^n = \underset{w_1^n}{\operatorname{argmax}} \frac{P(q_1^n | w_1^n) \times P(w_1^n)}{P(q_1^n)}$$

.

Diacritization as a sequence labeling task

- Bay's Rule:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

- Rewrite eq(1):

$$\hat{w}_1^n = \underset{w_1^n}{\operatorname{argmax}} \frac{P(q_1^n|w_1^n) \times P(w_1^n)}{P(q_1^n)}$$

- We can drop the denominator since $P(q_1^n)$ is the same for all $P(w_1^n)$:

$$\hat{w}_1^n = \underset{w_1^n}{\operatorname{argmax}}\, P(q_1^n|w_1^n) \times P(w_1^n)$$ →eq(2)

.

Diacritization as a sequence labeling task

- Apply Markov assumptions:

    1. Assumption 1: $P(w_i)$ depends only on $P(w_{i-1})$

    $$\therefore P(w_1^n) \approx \prod_{i=1}^{n} P(w_i \mid w_{i-1})$$

    2. Assumption 2: $P(q)$ depends only on its own alternative diacritizations

    $$\therefore P(q_1^n \mid w_1^n) \approx \prod_{i=1}^{n} P(q_i \mid w_i)$$

Diacritization as a sequence labeling task

- Apply Markov assumptions:

  1. Assumption 1: $P(w_i)$ depends only on $P(w_{i-1})$

     $$\therefore P(w_1^n) \approx \prod_{i=1}^{n} P(w_i \mid w_{i-1})$$

  2. Assumption 2: $P(q)$ depends only on its own alternative diacritizations

     $$\therefore P(q_1^n \mid w_1^n) \approx \prod_{i=1}^{n} P(q_i \mid w_i)$$

- Substitute in eq(2):

$$\boxed{\hat{w}_1^n = \underset{w_1^n}{\operatorname{argmax}}\, P(w_1^n \mid q_1^n) \approx \underset{w_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(q_i \mid w_i) \times P(w_i \mid w_{i-1})}$$ →eq(3)

Diacritization as a sequence labeling task

- Now we have two probabilities to calculate:

  1. Probability of a word occurring given its diacritization

  2. Probability of a diacritized word occurs given the previous diacritized word

- We can calculate each of these from a bigrams LM built using a diacritized corpus

Diacritization as a sequence labeling task

- Assume we have a sequence of undiacritized T words. And assume each undiacritized word has N different diacritizations. All what we need to do now is:

  1. Generate all possible sequences of diacritized words
  2. Use equation 3 to calculate the sequence probability
  3. Sort all the sequences and select that one with the highest probability

- What is the order of complexity of this solution?

Diacritization as a sequence labeling task

- Assume we have a sequence of undiacritized T words. And assume each undiacritized word has N different diacritizations. All what we need to do now is:

    1. Generate all possible sequences of diacritized words

    2. Use equation 3 to calculate the sequence probability

    3. Sort all the sequences and select that one with the highest probability

- What is the order of complexity of this solution?

    $O(TN^T)$ Too expensive. Can we do better?

Diacritization as a sequence labeling task

- Assume we have a sequence of undiacritized T words. And assume each undiacritized word has N different diacritizations. All what we need to do now is:

  1. Generate all possible sequences of diacritized words
  2. Use equation 3 to calculate the sequence probability
  3. Sort all the sequences and select that one with the highest probability

- What is the order of complexity of this solution?

  $O(TN^T)$ Too expensive. Can we do better?

  Yes, use optimized lattice search algorithms like Viterbi or A* via beam search

Exercise

- Given a corpus, write a python class to calculate $P(q_i | w_i)$
    - The constructor takes the path of the training corpus and the path of the output map file in the following format
      لم لَمْ 0.6 لَمَّ 0.4
      يعد يَعُدَّ 0.3 يُعِدْ 0.2 يُعَدّ 0.1 يُعِدّ 0.4
    - The class has a function “get_w_given_q(w,q)” to return the $P(q_i | w_i)$.
      EX: get_q_given_w(‘لم’,’لَمْ’) returns 0.6

Viterbi

- Due to the Markov assumption, the probability of being in any state at any given time t only relies on the probability of being in each of the possible states at time t–1

- Forward step: Uses dynamic programming to exploit this fact to efficiently compute observation likelihood in $O(TN^2)$ time.

- Back-tracing: At state s_j at time t, store a back pointer to the state at t-1 that maximizes the transition probability to the current state s_j at time t

Forward step

SOP

lm	*yEd*	*Almstvmr*	*rbH*	*r>s*	*mAlh*
lam/P/ Not	yaEud~/V/ Count	Almustavmiru/N/ Investor	rabiHa/V/ Won	ra>su/N/ Head	mAlihi/ N+Pron/ His money
lam~a/V/ Collected	yuEad~/V-Pass/ Considered	Almustavmaru/N/ Invested	ribHa/N/ Gain	ra>asa/V/ Preside	mAlahu/ N+Pron/His money
⋮	yaEdu/V/ Run		rubiHa/V-Pass/ Gained		mAluhu/ N+Pron/His money
lima/P/Why	yuEdi/V/ Infect				
	yuEid~/V/ Prepare				
	yuEid/V/ Return				

EOP

Forward step

lm $\quad$ yEd

SOP

s_1 lam/P/ Not $\alpha(s_1)_{t-1}$

s_2 lam~a/V/ Collected $\alpha(s_2)_{t-1}$

⋮

s_N lima/P/Why $\alpha(s_N)_{t-1}$

a_{11} $\quad$ a_{21} $\quad$ a_{N1}

yaEud~/V/ Count

yuEad~/V-Pass/ Considered

yaEdu/V/ Run

yuEdi/V/ Infect

yuEid~/V/ Prepare

yuEid/V/ Return

$$\alpha(s_1)_t = \max((a_{11} + \alpha(s_1)_{t-1}), \ldots (a_{N1} + \alpha(s_N)_{t-1})) + log(P(yEd \,|\, yaEud \sim))$$

Forward step

lm *yEd*

s_1 lam/P/ Not $\alpha(s_1)_{t-1}$

a_{12}

yaEud~/V/ Count

$\alpha(s_1)_t$

yuEad~/V-Pass/ Considered

$$\alpha(s_2)_t = \max((a_{12} + \alpha(s_1)_{t-1}), \ldots (a_{N2} + \alpha(s_N)_{t-1})) + log(P(yEd \,|\, yuEad \sim))$$

a_{22}

s_2 lam~a/V/ Collected $\alpha(s_2)_{t-1}$

SOP

a_{N2}

⋮

yaEdu/V/ Run

yuEdi/V/ Infect

s_N lima/P/Why $\alpha(s_N)_{t-1}$

yuEid~/V/ Prepare

yuEid/V/ Return

Forward step

lm yEd

s_1 lam/P/ Not $\alpha(s_1)_{t-1}$

yaEud~/V/ Count $\alpha(s_1)_t$

yuEad~/V-Pass/ Considered $\alpha(s_2)_t$

s_2 lam~a/V/ Collected $\alpha(s_2)_{t-1}$

a_{1j}

SOP

yaEdu/V/ Run

⋮

a_{2j}

yuEdi/V/ Infect

$$\alpha(s_j)_t = \max((a_{1j} + \alpha(s_1)_{t-1}), \ldots (a_{Nj} + \alpha(s_N)_{t-1})) + log(P(yEd \,|\, yuEdi))$$

a_{Nj}

s_N lima/P/Why $\alpha(s_N)_{t-1}$

yuEid~/V/ Prepare

yuEid/V/ Return

Back-tracing

	lm	*yEd*	*Almstvmr*	*rbH*	*r>s*	*mAlh*	
	lam/P/ $\alpha(s_1)_{t-1}$	yaEud~/V/ $\alpha(s_1)_t$	Almustavmiru/N/	rabiHa/V/	ra>su/N/	mAlihi/N+Pron/ $\alpha(s_1)_T$	
		yuEad~/V-Pass/ $\alpha(s_2)_t$				mAlahu/N+Pron/ $\alpha(s_2)_T$	
	lam~a/V/ $\alpha(s_2)_{t-1}$		Almustavmaru/N/	ribHa/N/	ra>asi/N/		
SOP		yaEdu/V/		⋮	⋮	⋮	EOP $\alpha(EOP)$
	⋮	yuEdi/V/ $\alpha(s_j)_t$					
	lima/P/ $\alpha(s_N)_{t-1}$	yuEid~/V/			ra>asa/V/	mAluhu/N+Pron/ $\alpha(s_N)_T$	
				rubiHa/V-Pass/			
		yuEid/V/ $\alpha(s_N)_t$					

Back-tracing

SOP

lm

lam/P/
$\alpha(s_1)_{t-1}$

lam~a/V/
$\alpha(s_2)_{t-1}$

⋮

lima/P/
$\alpha(s_N)_{t-1}$

yEd

yaEud~/V/
$\alpha(s_1)_t$

yuEad~/V-Pass/
$\alpha(s_2)_t$

yaEdu/V/

yuEdi/V/
$\alpha(s_j)_t$

yuEid~/V/

yuEid/V/
$\alpha(s_N)_t$

Almstvmr

Almustavmiru/N/

Almustavmaru/N/

rbH

rabiHa/V/

ribHa/N/

⋮

rubiHa/V-Pass/

r>s

ra>su/N/

ra>asi/N/

⋮

ra>asa/V/

mAlh

$\boxed{\text{mAlihi/N+Pron/}}$
$\alpha(s_1)_T$

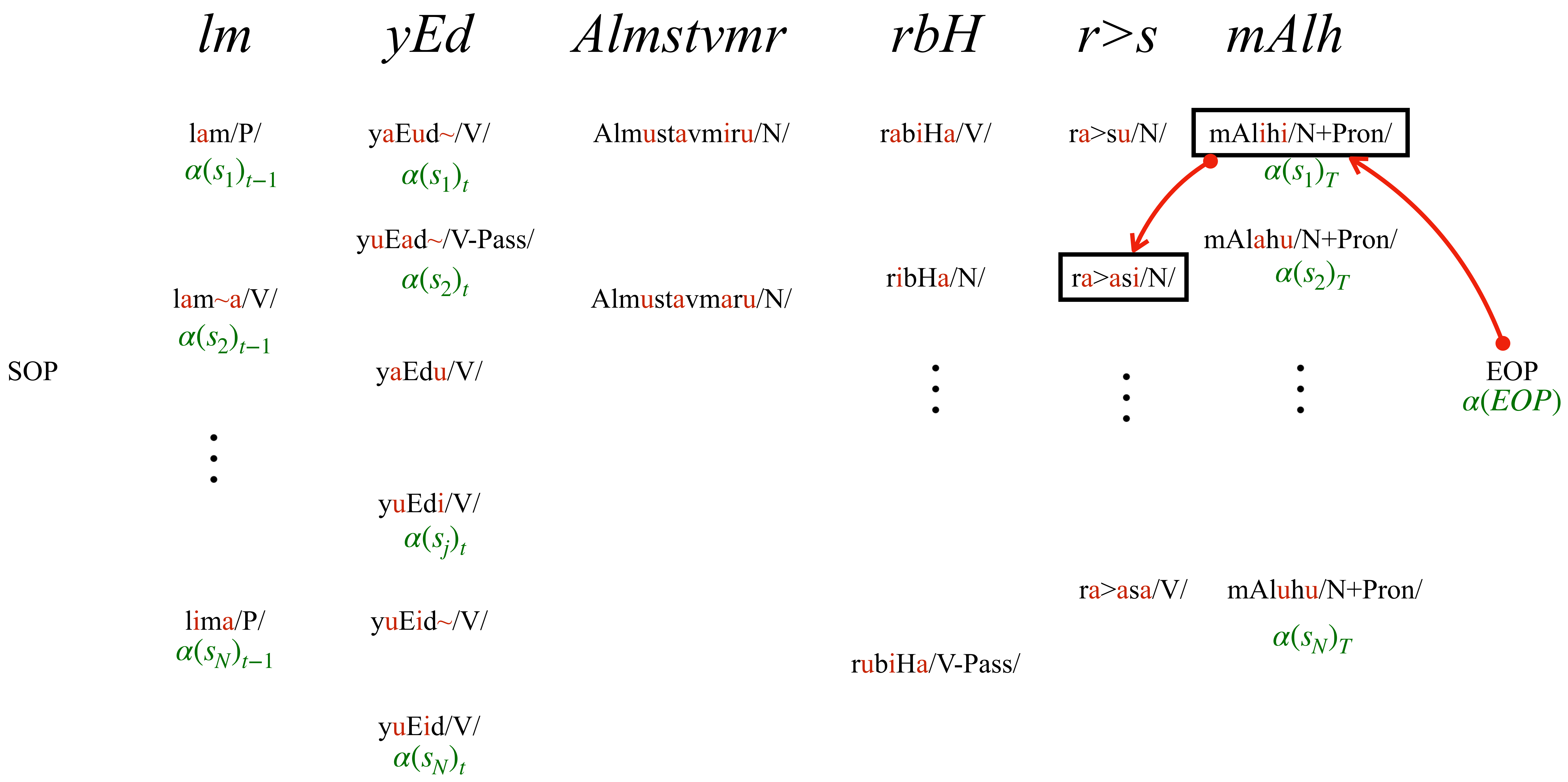
mAlahu/N+Pron/
$\alpha(s_2)_T$

⋮

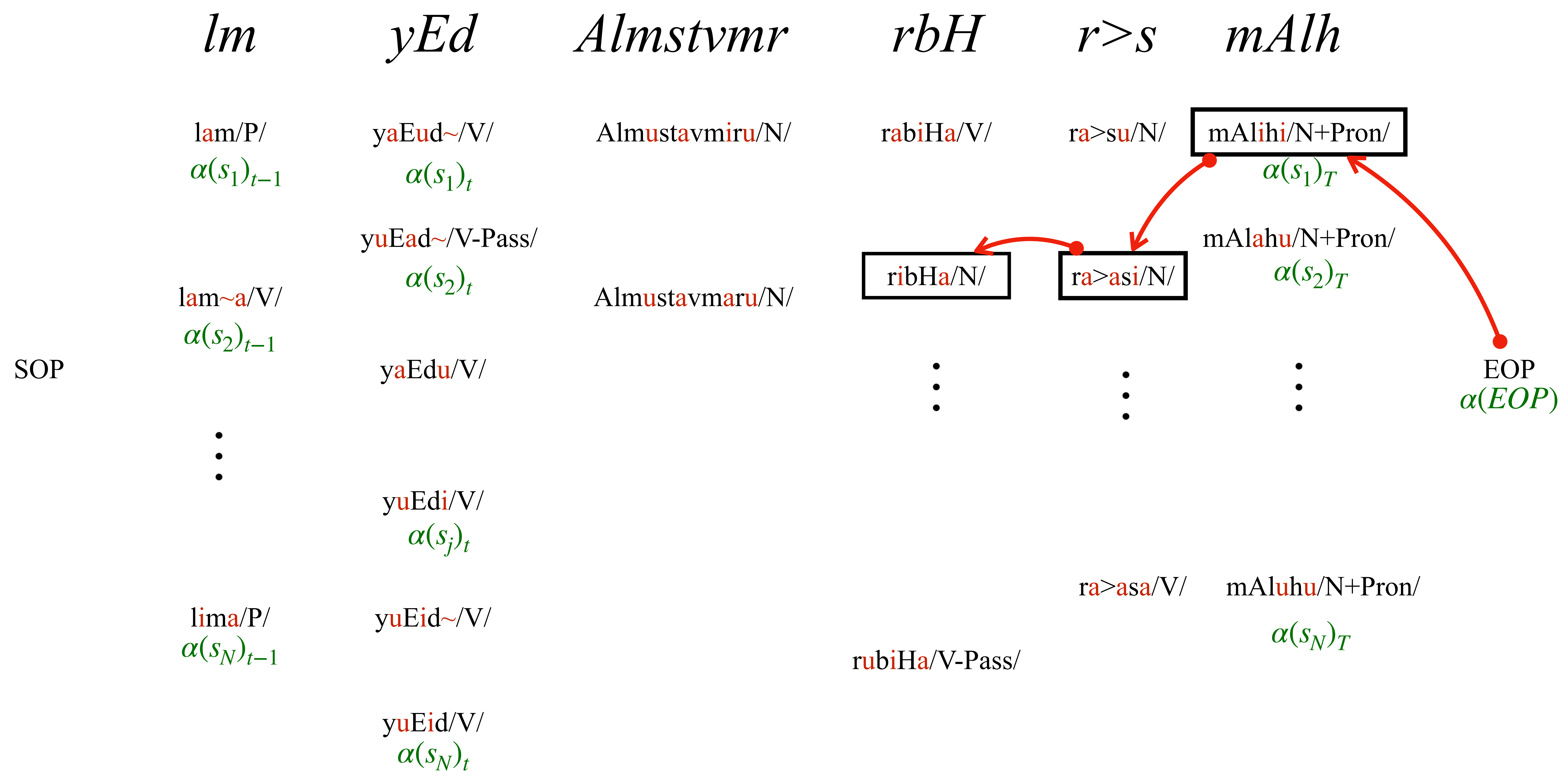
mAluhu/N+Pron/
$\alpha(s_N)_T$

EOP
$\alpha(EOP)$

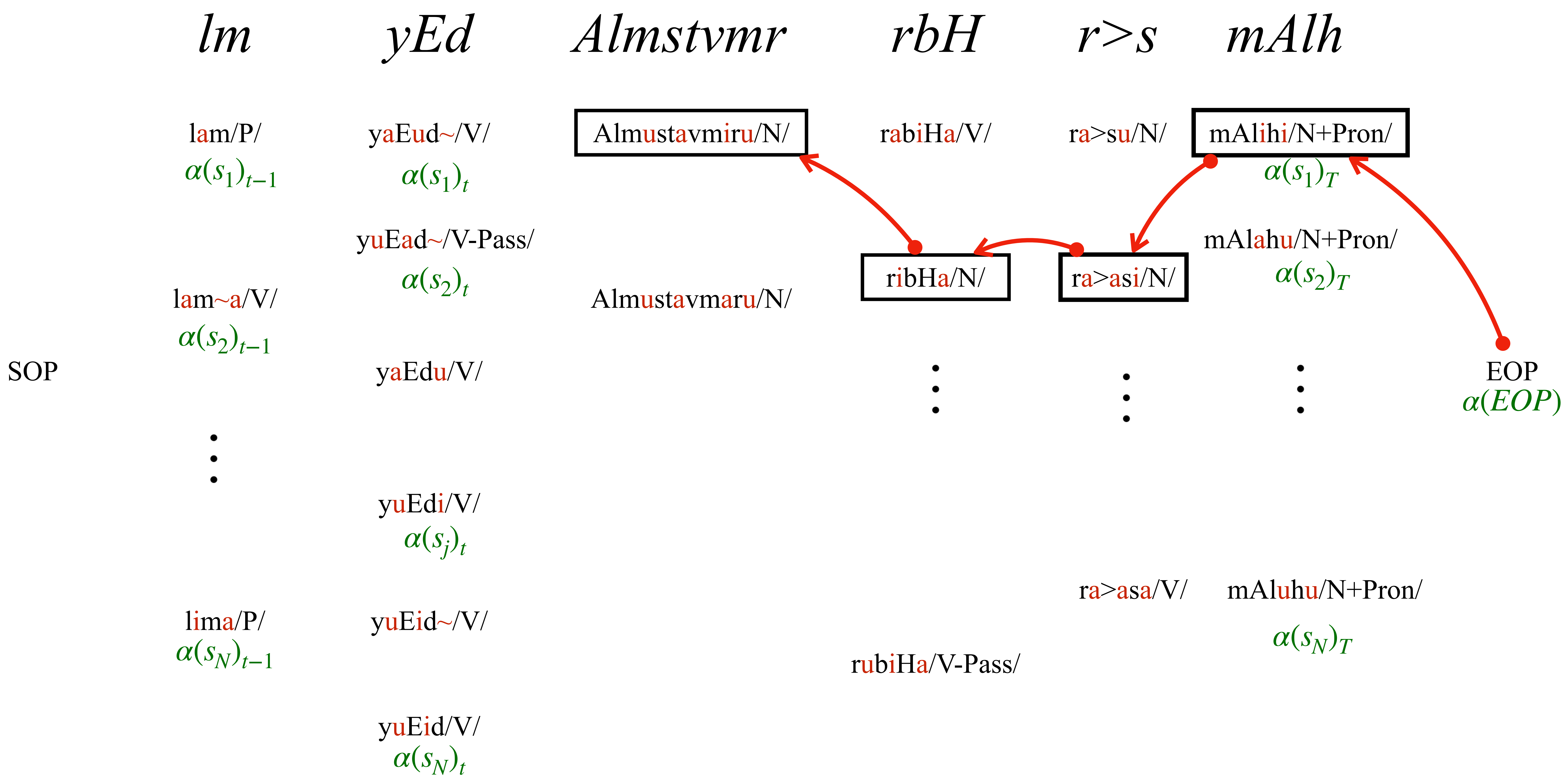
Back-tracing



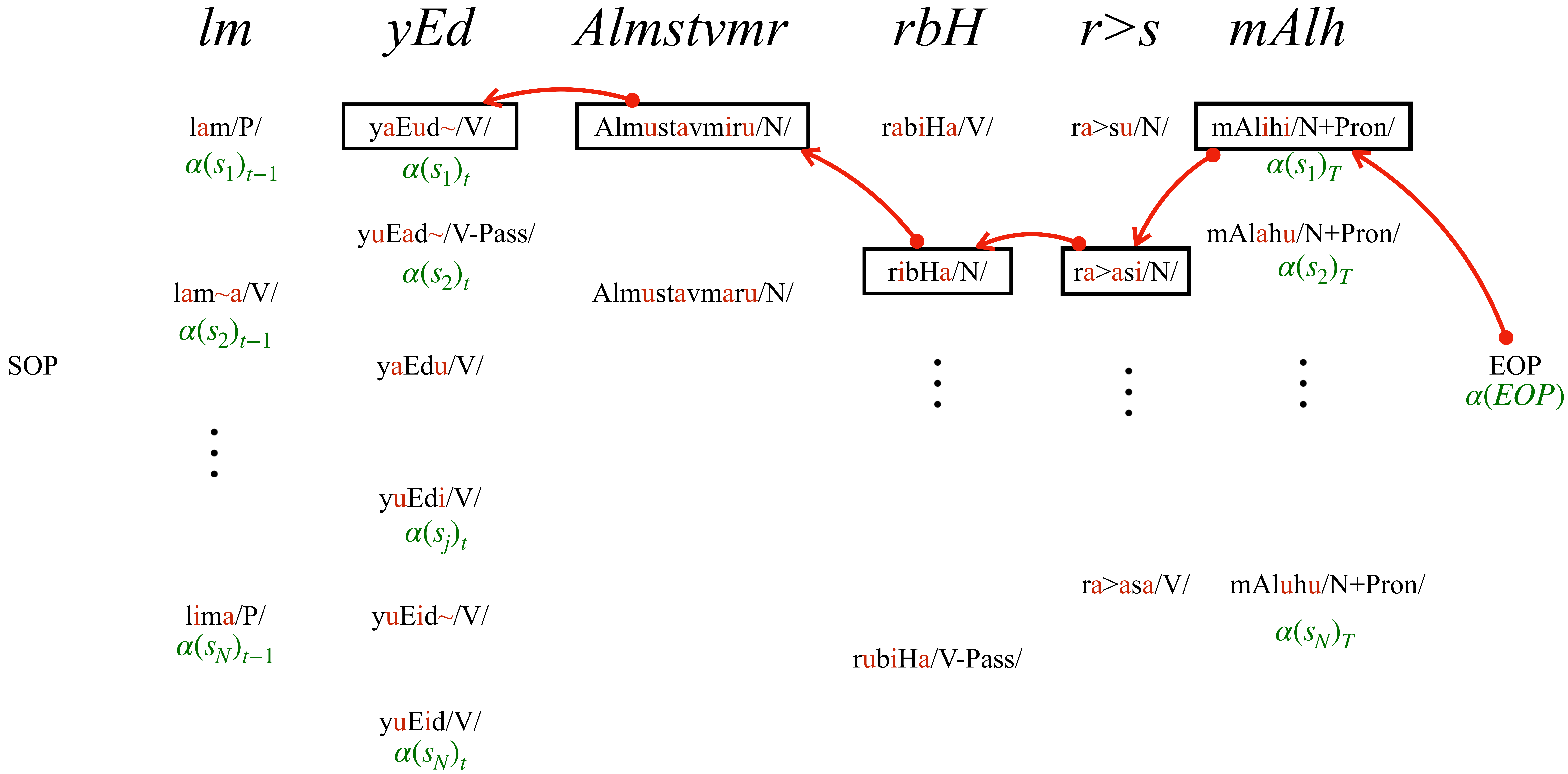
Back-tracing



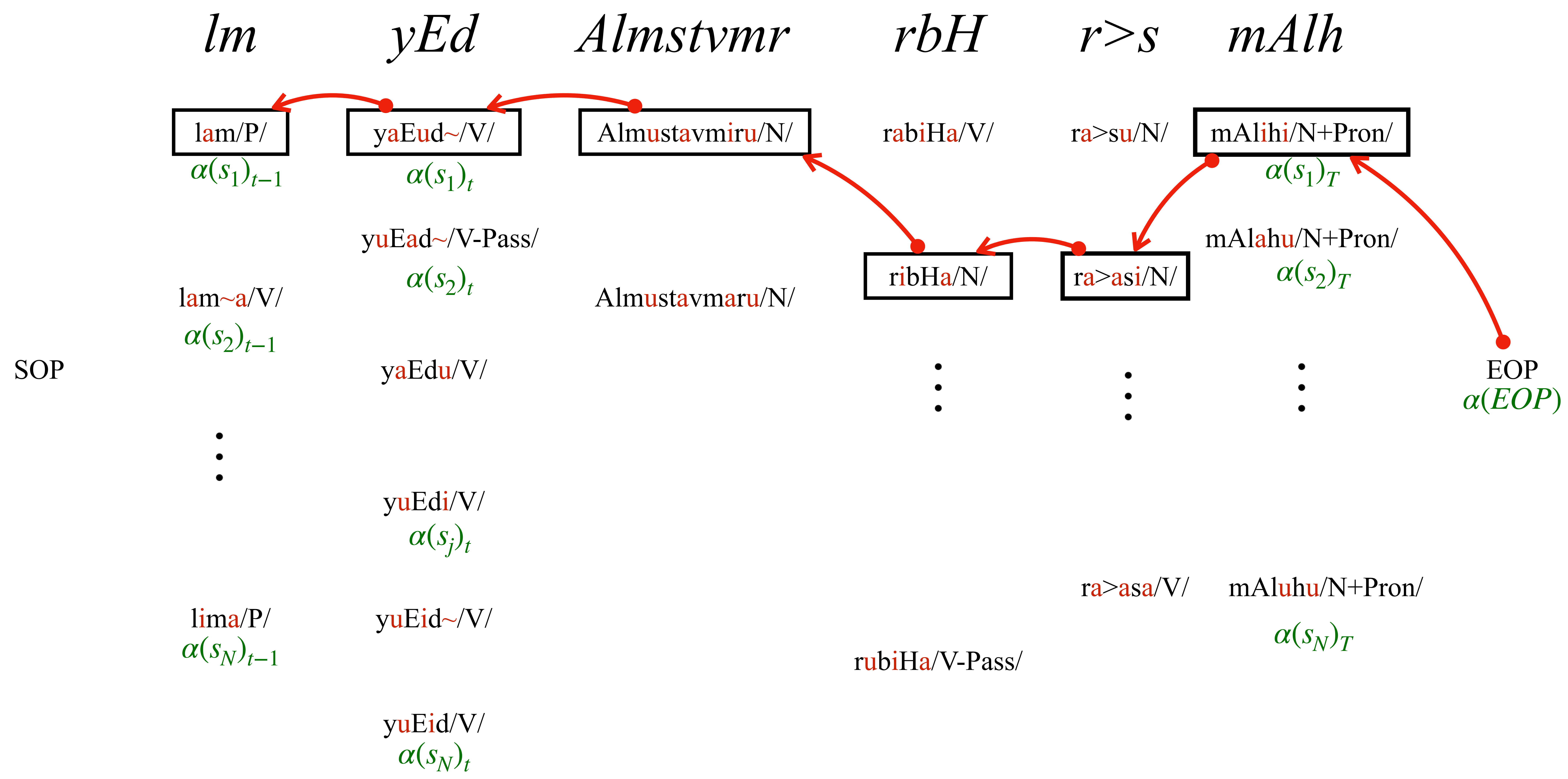
Back-tracing



Back-tracing



Back-tracing



Viterbi Algorithm

```
function  viterbi(observations, states_graph) ⟶ best path
    num_states ← num_states(states_graph)
    T ← len(observations)
    Create transition matrix A[num_states, T]
    Create back-pointers matrix B[num_states, T]
    Create best path array O[T]
    A ← 0
    for each time step t=0; t<T; t++:
        for each state s at time step t:
            for each state e at time step (t+1):
                new_score ← A[s, t] + log(P(e|s)) + log(P(q_{t+1}|e))
                if A[e, t+1] < new_score:
                    A[e, t+1] ← new_score
                    B[e, t+1] ← index(s)
    k ← argmax(A[:, T-1])
    O[T-1] ← s_k at state T-1
    for each time step t=T-2; t>=0; t--:
        k ← B[k, t+1]
        O[t] ← s_k at state t
    return O
```

Build LM:

```
bin/lmplz -o 2 </path/to/training/data.txt  >/path/to/output/lm.arpa

from LanguageModel import LanguageModel

lm = LanguageModel(‘/path/to/output/lm.arpa’, 'TEXT')
```

Build Map:

```
from LMDisambigMapBuilder import LMDisambigMapBuilder
from LMDisambigMap import LMDisambigMap


LMDisambigMapBuilder.build(‘/path/to/training/corpus.txt’, 'path/to/output/map.txt')
mapping = LMDisambigMap('path/to/output/map.txt', 'TEXT', lm)
```

Diacritization

```
from LMDisambig import LMDisambig

disambiguaty = LMDisambig(lm, mapping)
sequence = ‘undiacritized_line’.split()
dicaritized_line = disambiguaty.disambig(sequence, decoder =
'VITERBI').get_output_sequence()
```