Lecture # 14
Session 2003

Part I: Designing HMM-based ASR systems

Rita Singh

School of Computer Science
Carnegie Mellon University

Table of contents

Statistical Pattern Classification

- Given data X, find which of a number of classes $C_1, C_2, \ldots C_N$ it belongs to, based on known distributions of data from C_1, C_2, etc.

- Bayesian Classification:

  $$\text{Class} = C_i : i = \text{argmax}_j \; P(C_j)P(X|C_j)$$

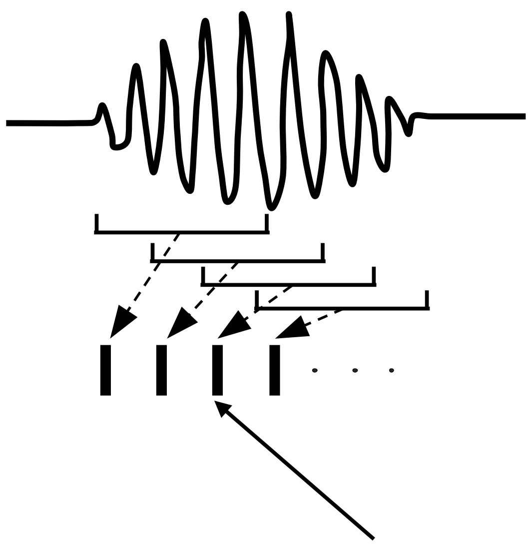
  a priori probability of C_j

  Probability of X as given by the probability distribution of C_j

- The *a priori* probability accounts for the relative proportions of the classes
  - If you never saw any data, you would guess the class based on these probabilities alone
- $P(X|C_j)$ accounts for evidence obtained from observed data X

Statistical Classification of Isolated Words

- Classes are words
- Data are instances of isolated spoken words
  - Sequence of feature vectors derived from speech signal, typically 1 vector from a 25ms frame of speech, with frames shifted by 10ms.



- Bayesian classification:

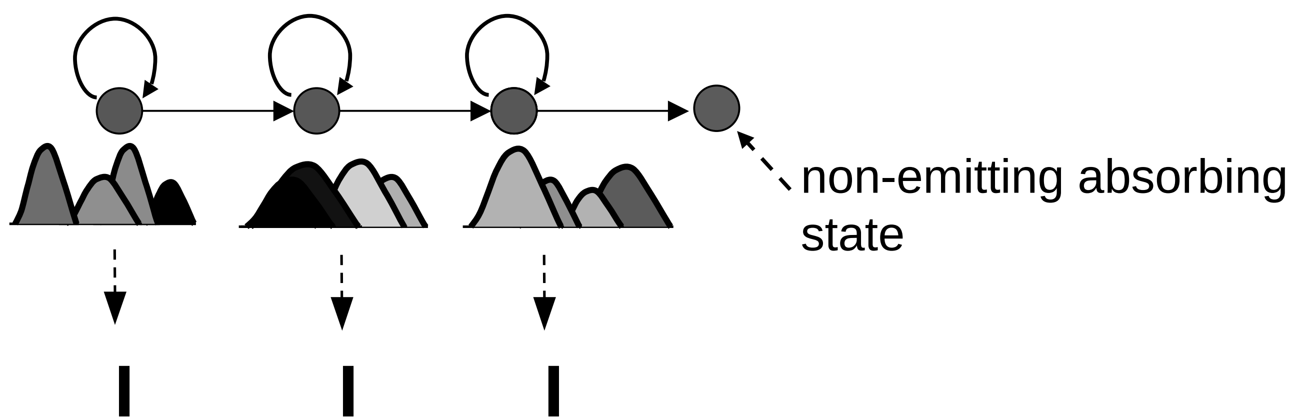
  Recognized_Word = $\text{argmax}_{word}\, P(word)P(X|\, word)$

- $P(word)$ is *a priori* probability of *word*
  - Obtained from our expectation of the relative frequency of occurrence of the word

- $P(X|word)$ is the probability of X computed on the probability distribution function of *word*

Computing P(*X*|*word*)

- To compute P(*X*|*word*), there must be a statistical distribution for *X* corresponding to *word*
  - Each word must be represented by some statistical model.
- We represent each word by an HMM
  - An HMM is really a graphical form of probability density function for time-varying data

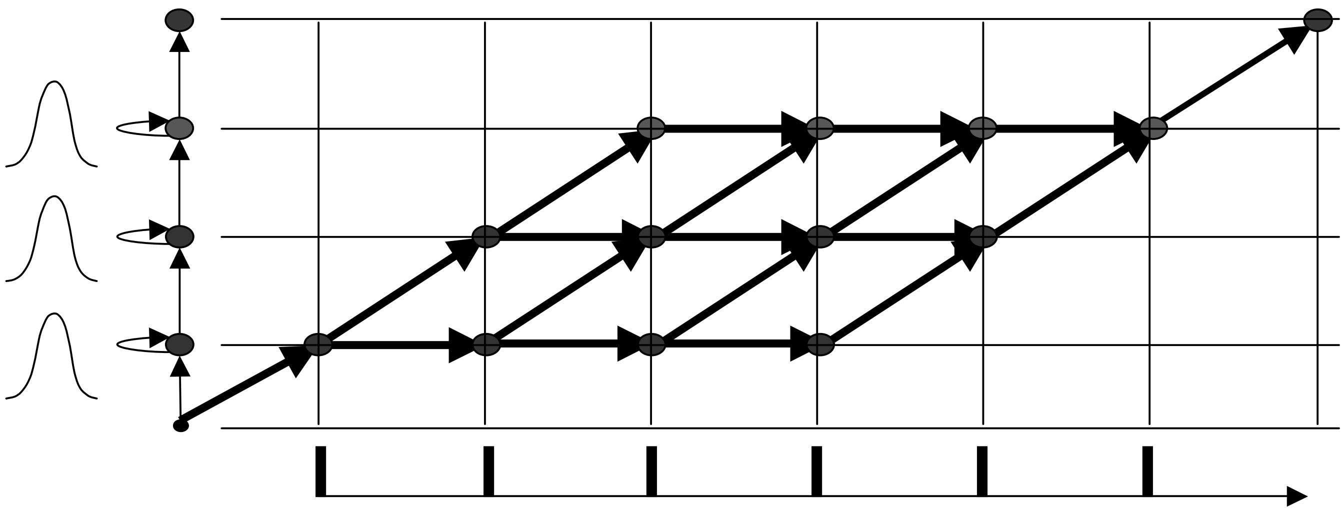


- Each state has a probability distribution function
- Transitions between states are governed by transition probabilities
- At each time instant the model is in some state, and it emits one observation vector from the distribution associated with that state

Computing P(*X*|*word*)

- The actual state sequence that generated *X* is never known.
  - P(*X*|*word*) must therefore consider *all* possible state sequences.

$$P(X \mid word) = \sum_{s \in \{\text{all state sequences}\}} P(X, s \mid word)$$



The probabilities of all possible state sequences must be added to obtain the total probability

Computing P(*X*|*word*)

- The actual state sequence that generated *X* is never known.
  - P(*X*|*word*) must therefore consider *all* possible state sequences.

$$P(X \mid word) = \sum_{s \in \{\text{all state sequences}\}} P(X, s \mid word)$$

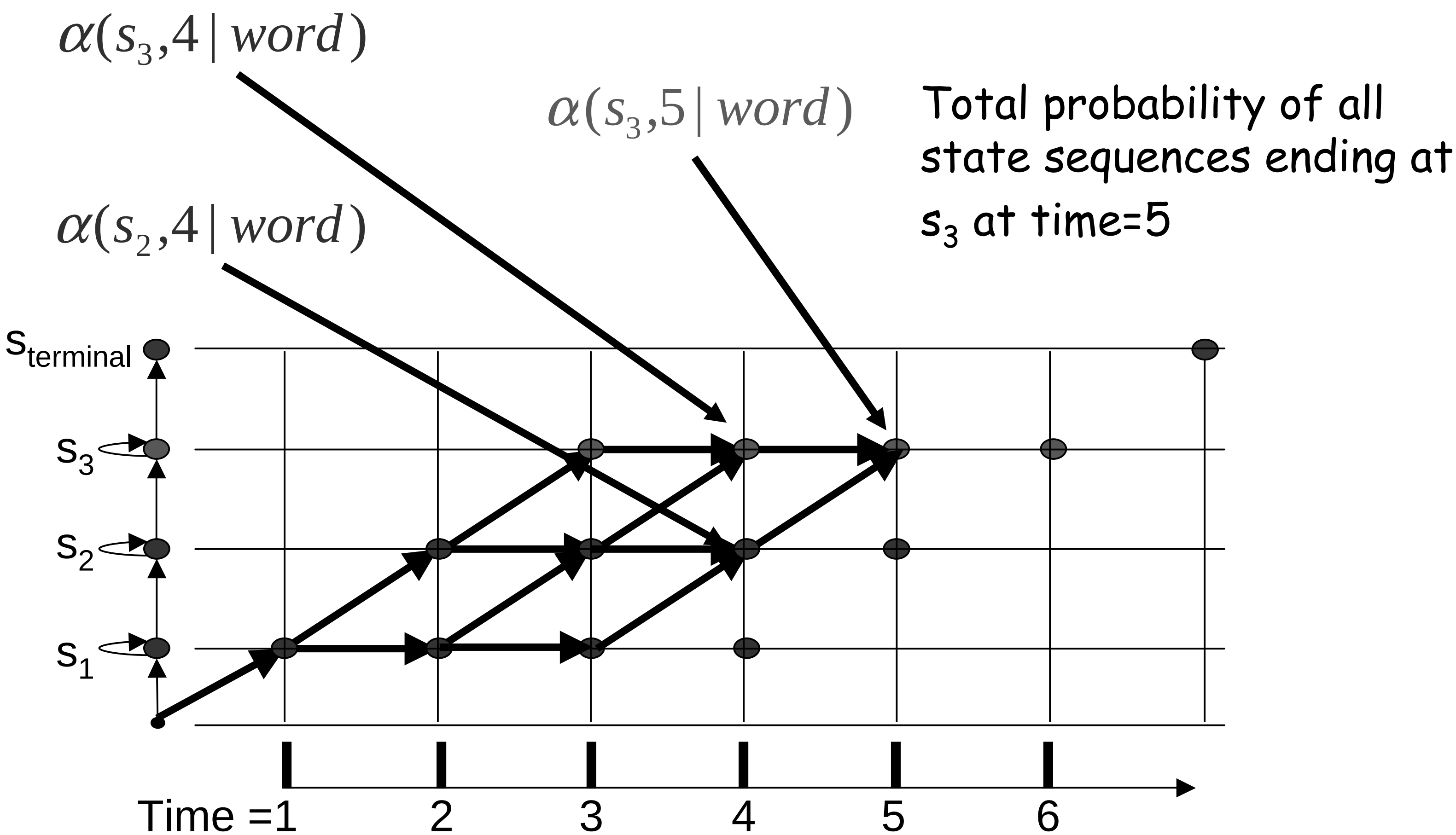
- The actual number of state sequences can be very large
  - Cannot explicitly sum over all state sequences
- P(*X*|*word*) can however be efficiently calculated using the forward recursion

$$\alpha(s,t \mid word) = \sum_{s'} \alpha(s',t-1 \mid word) P(s \mid s') P(X_t \mid s)$$

Total probability of all state sequences ending in state *s* at time *t*

Computing P(X|$word$)

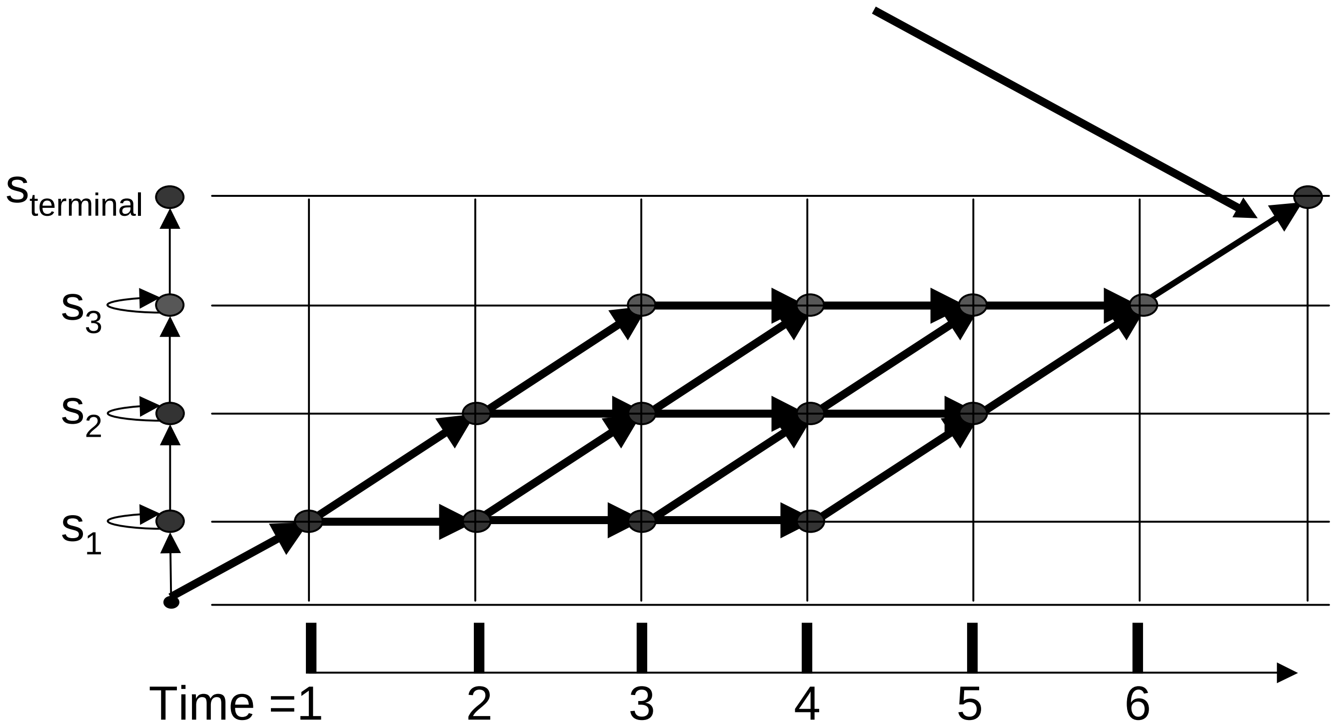
- The forward recursion



Computing P(*X*|*word*)

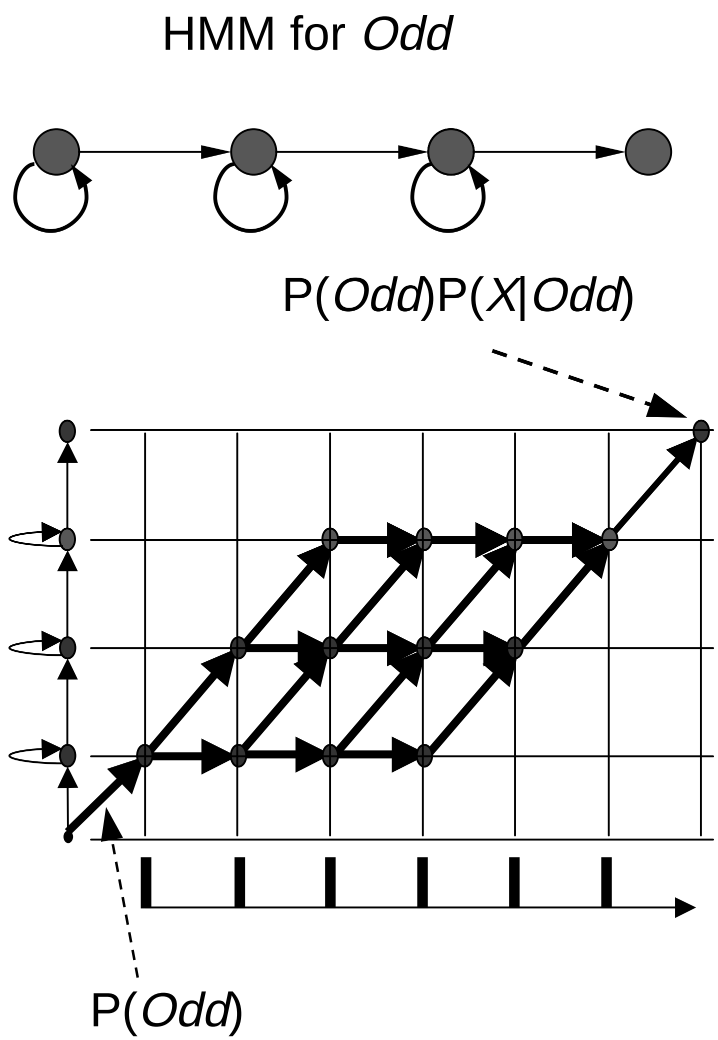
- The total probability of *X*, including the contributions of all state sequences, is the forward probability at the final non-emitting node

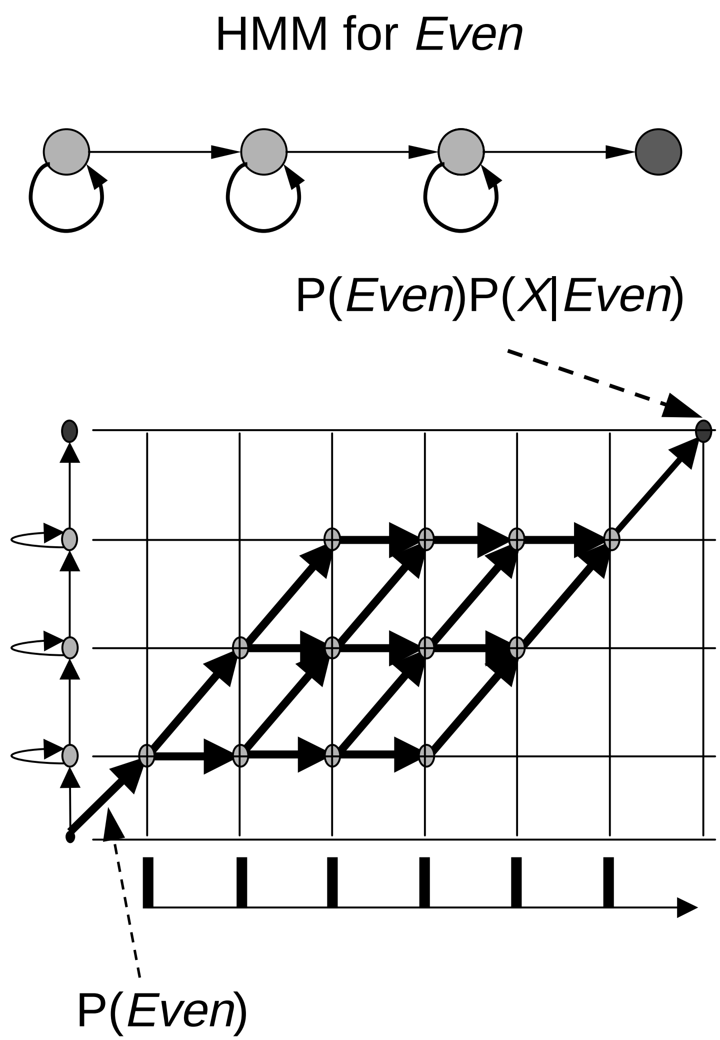
$$P(X \mid word) = \alpha(s_{terminal}, T \mid word)$$



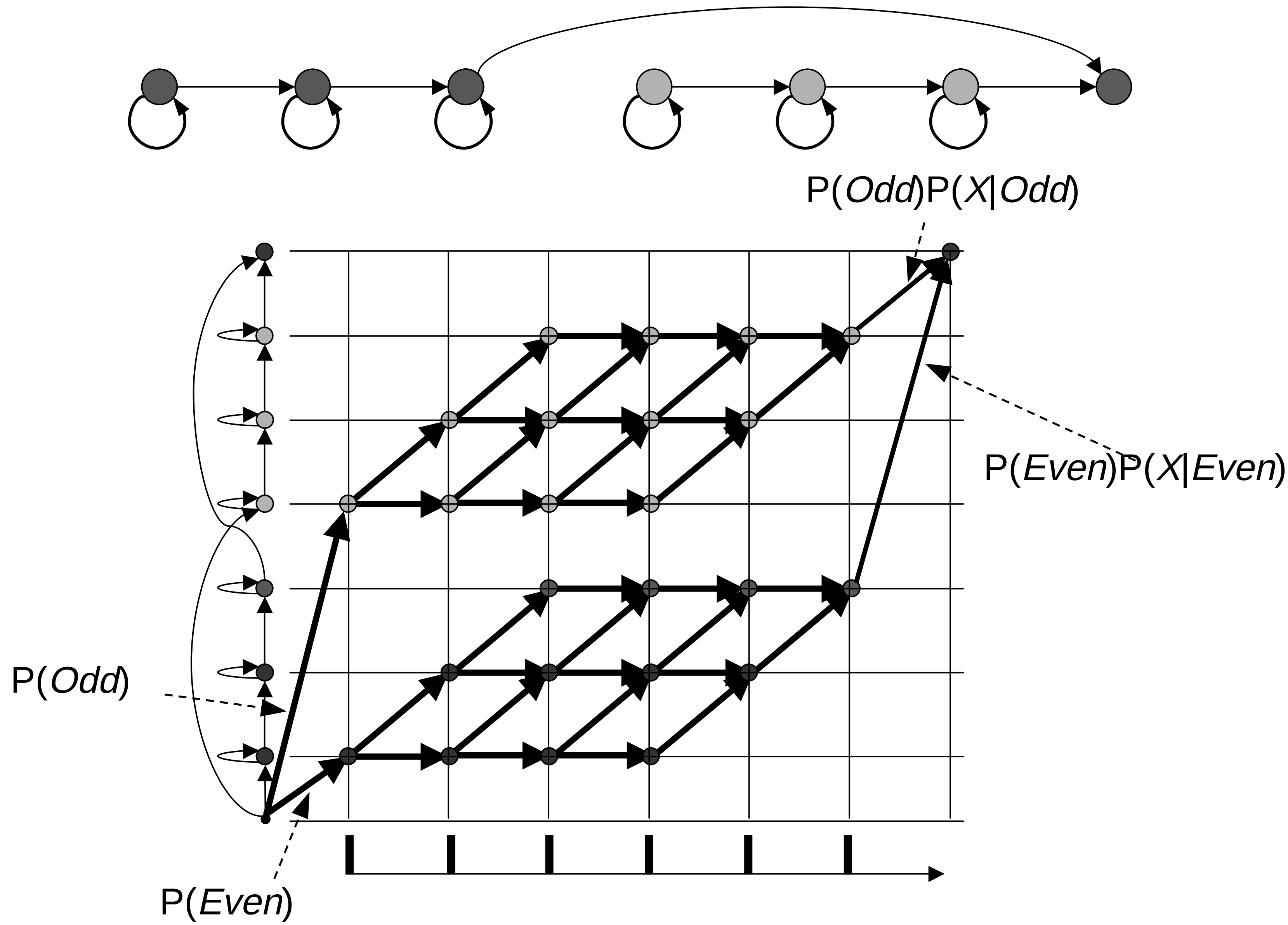
Decoding isolated words

Classifying between two words: *Odd* and *Even*



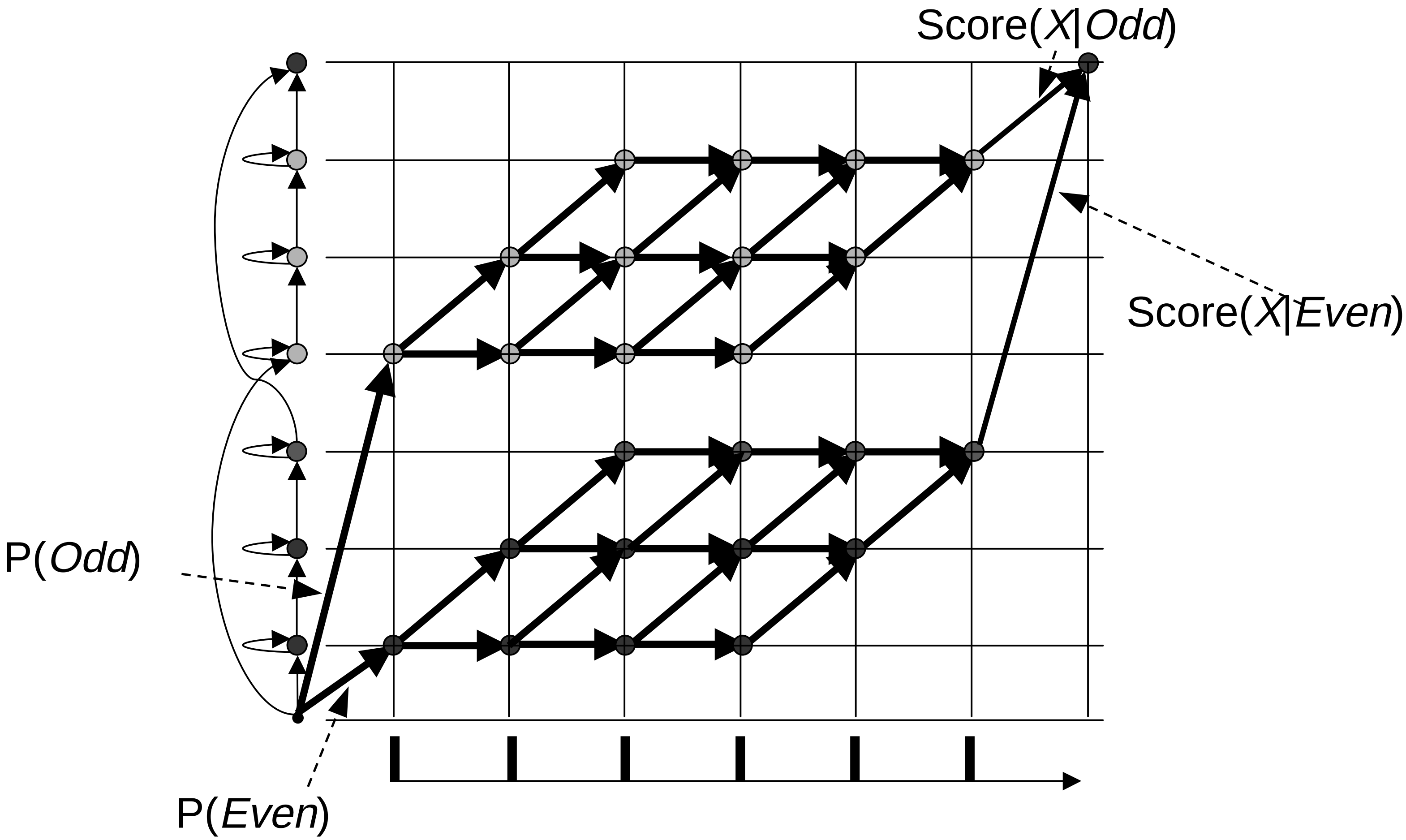


Classifying between *Odd* and *Even*



Decoding to classify between *Odd* and *Even*

- Approximate total probability of all paths with probability of best path
  - Computations can be done in the log domain. Only additions and comparisons are required
    - Cheaper than full-forward where multiplications and additions are required



Computing Best Path Scores for Viterbi Decoding

- The approximate score for a word is

$$P(X \mid word) \approx \max_{s \in \{\text{all state sequences}\}} \{P(X, s \mid word)\}$$

- Written explicity

$$P(X_1..X_t \mid word) \approx \max_{\underbrace{s_1, s_2, \ldots, s_T}_{\text{state sequence}}} \{\pi(s_1) P(X_1 \mid s_1) P(s_2 \mid s_1) P(X_2 \mid s_2) \ldots . P(s_T \mid s_{T-1}) P(X_T \mid s_T)\}$$

- The word score can be be recursively computed using the Viterbi algorithm

$$P_s(t) = \max_{s'} \{P_{s'}(t-1) P(s \mid s') P(X_t \mid s)\}$$

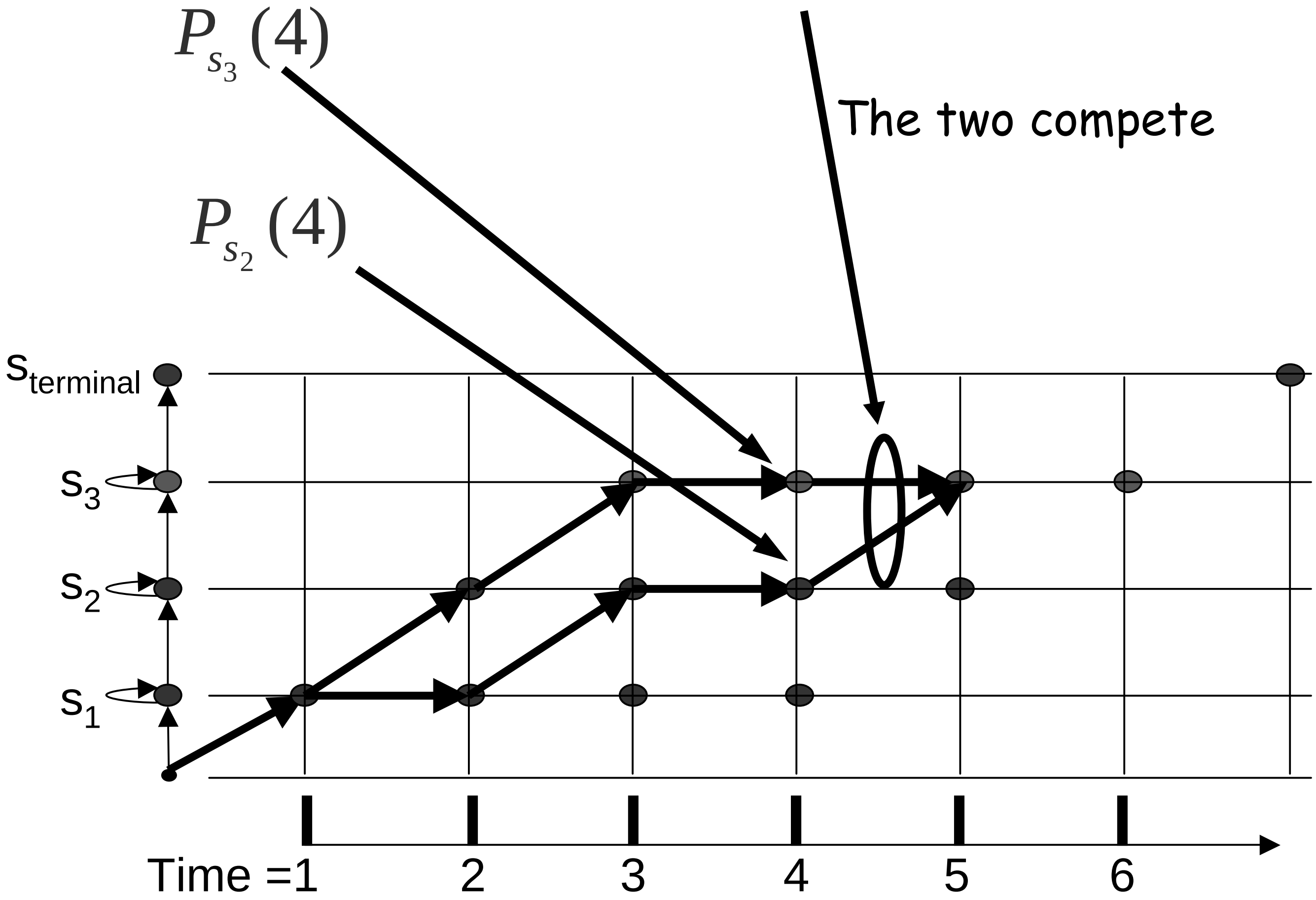
Best path score of all state sequences ending in state *s* at time *t*

Best path score of all state sequences ending in state *s'* at time *t-1*

Computing Best Path Scores for Viterbi Decoding

- The forward recursion

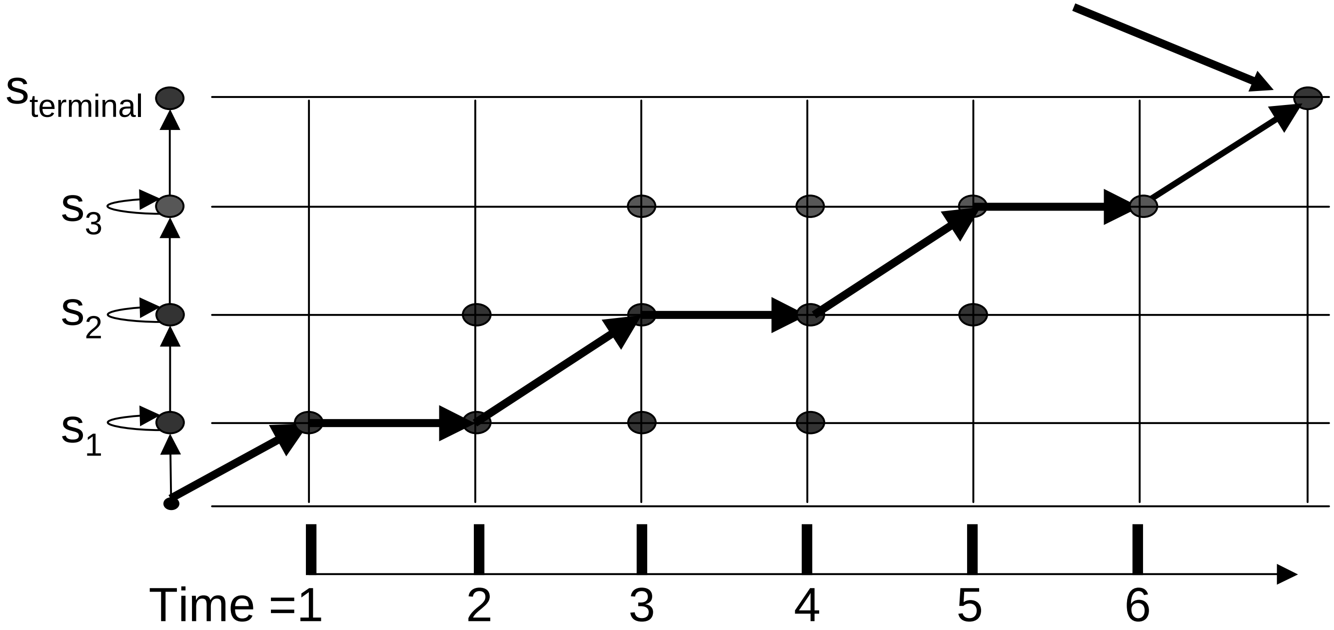
$$\boxed{P_{s_3}(5) = \max\{ P_{s3}(4)P(s_3 \mid s_3)P(X_5 \mid s_3), P_{s_2}(4)P(s_3 \mid s_2)P(X_5 \mid s_3)\}}$$



Computing Best Path Scores for Viterbi Decoding

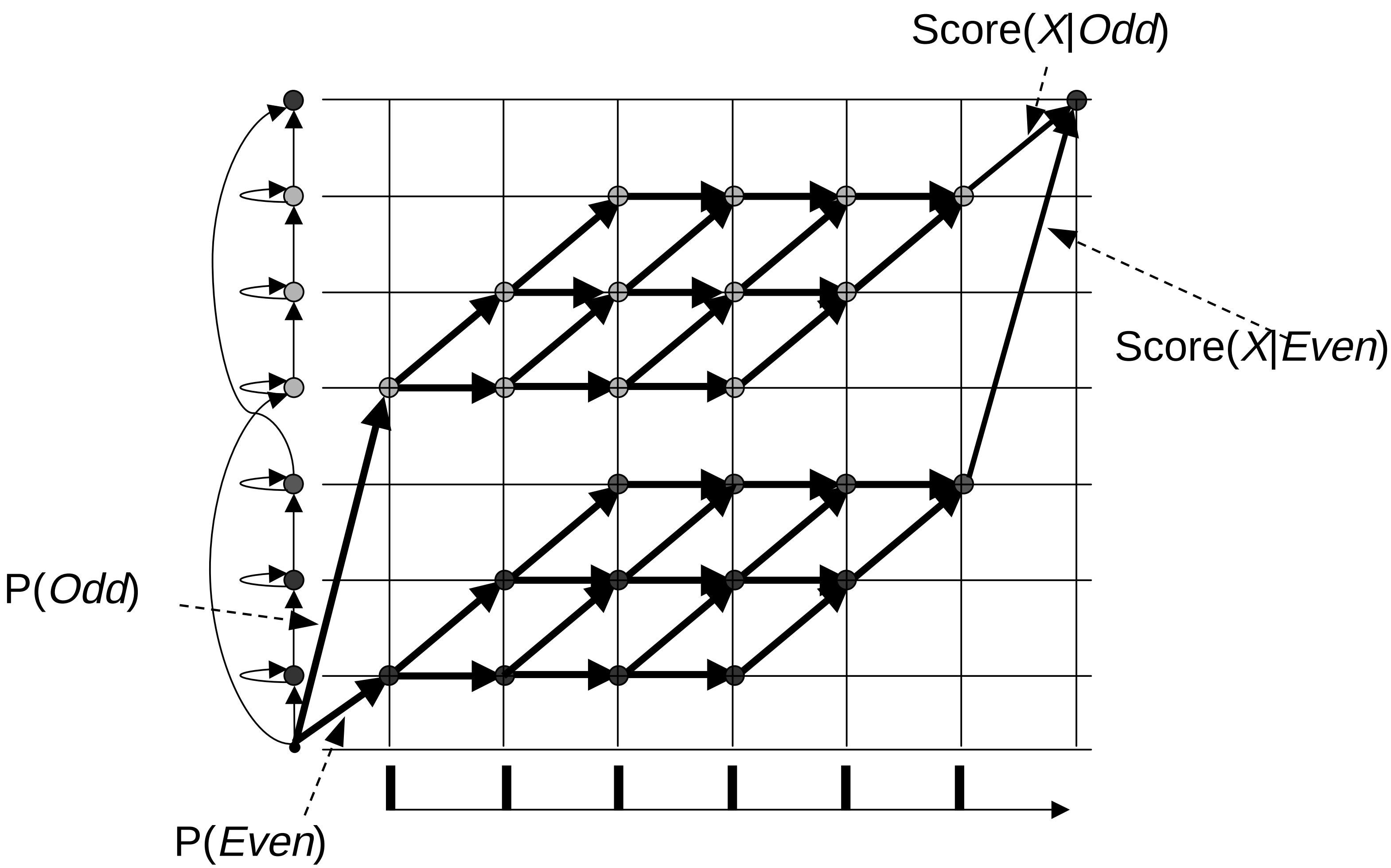
- The forward recursion is termed Viterbi *decoding*
  - The terminology is derived from decoding of error correction codes
- The score for the word is the score of the path that wins through to the terminal state
  - We use the term *score* to distinguish it from the total probability of the word

$$Score(X \mid word) = P_{s_{terminal}}(T)$$



Decoding to classify between *Odd* and *Even*

- Compare scores (best state sequence probabilities) of all competing words



Decoding word sequences

Statistical Classification of Word Sequences

- Classes are word sequences
- Data are spoken recordings of word sequences

- Bayesian classification:

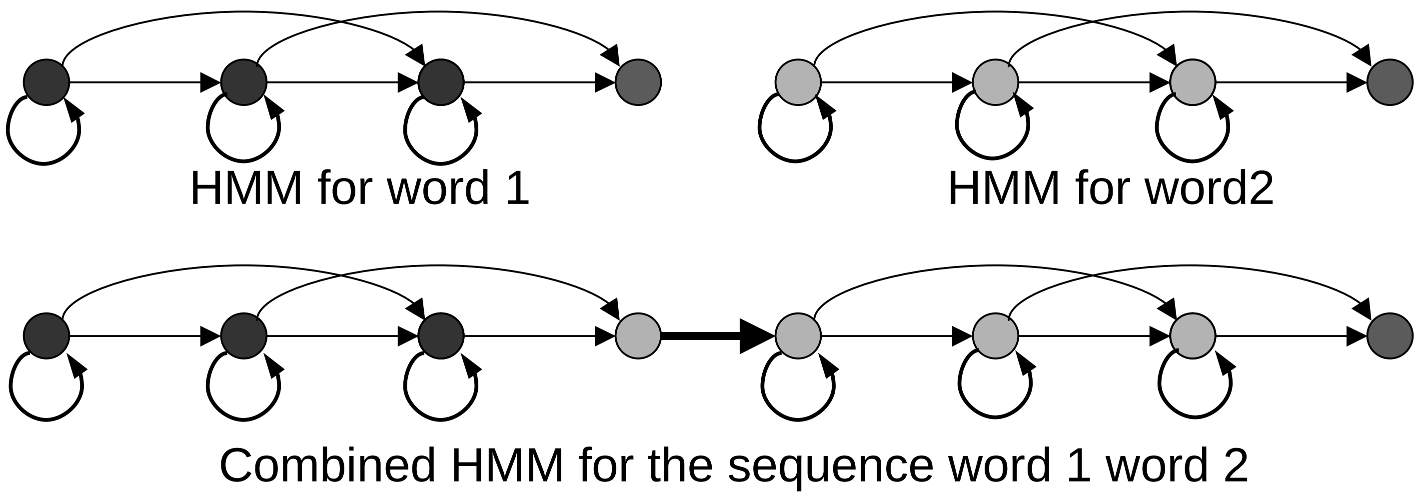
$$word_1, word_2, ..., word_N =$$
$$\arg\max_{wd_1, wd_2, ..., wd_N} \{P(X \mid wd_1, wd_2, ..., wd_N) P(wd_1, wd_2, ..., wd_N)\}$$

- P(wd_1,wd_2,wd_3..) is *a priori* probability of word sequence wd_1,wd_2,wd_3..
  - Obtained from a model of the language

- P(X| wd_1,wd_2,wd_3..) is the probability of X computed on the probability distribution function of the word sequence wd_1,wd_2,wd_3..
  - HMMs now represent probability distributions of word sequences

Constructing HMMs for word sequences

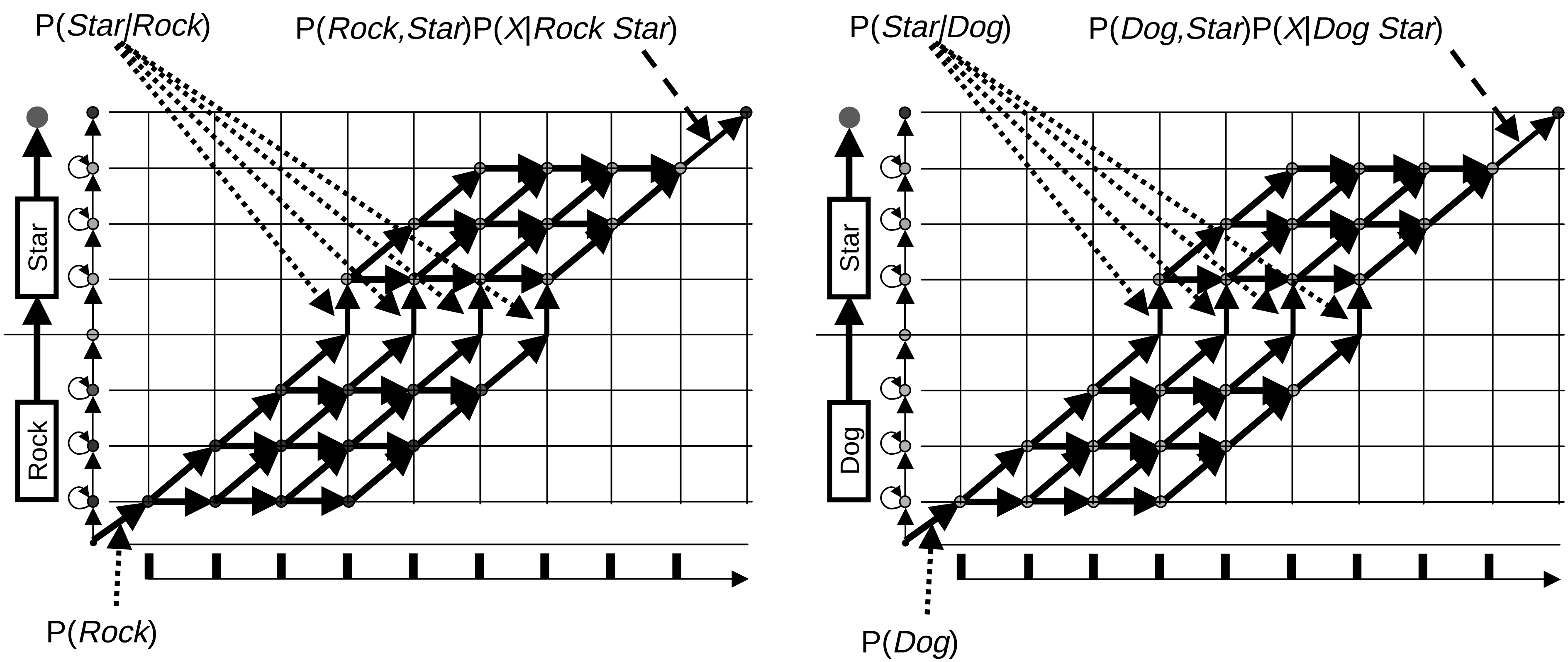
- Conacatenate HMMs for each of the words in the sequence



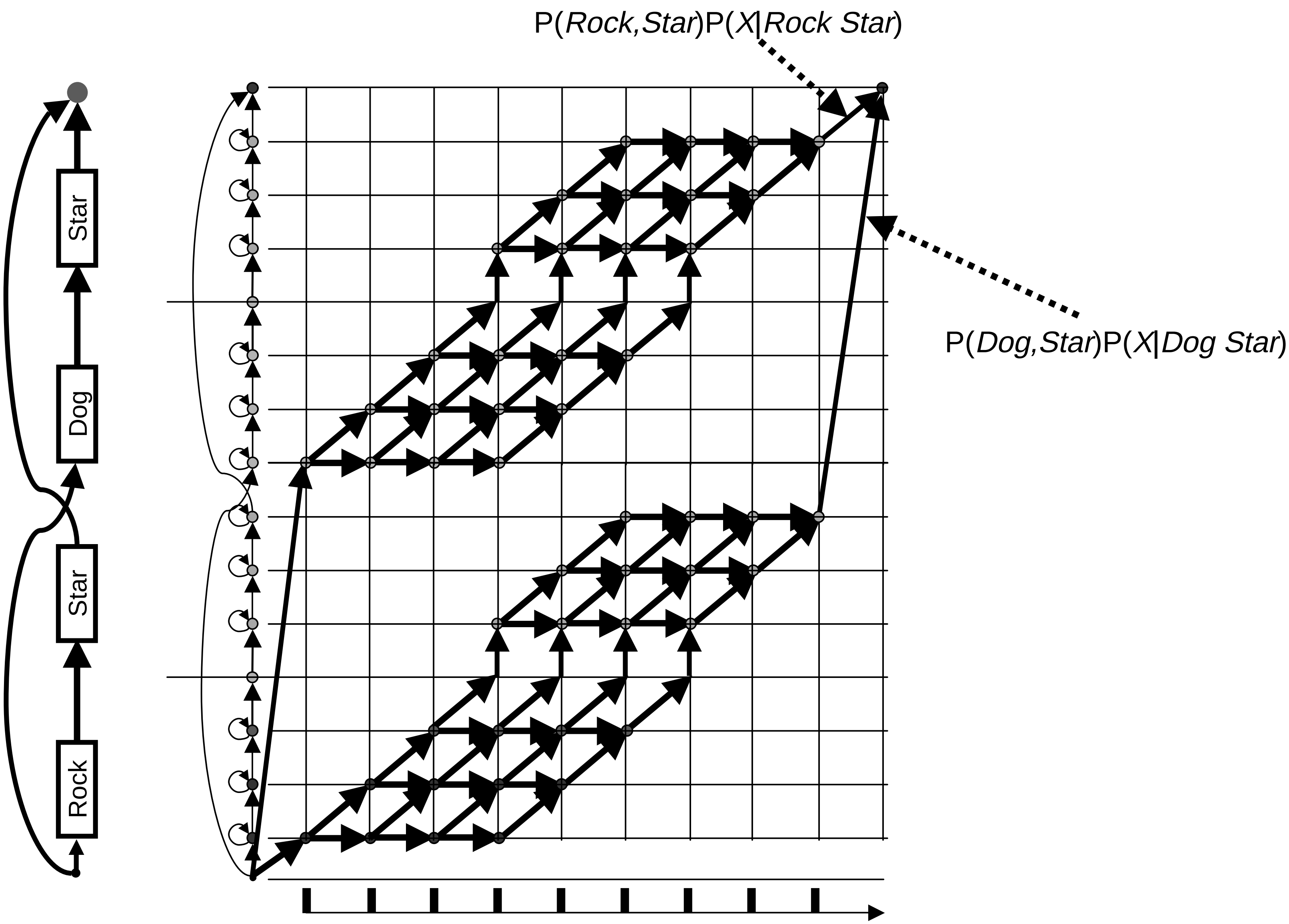
- In fact, word HMMs themselves are frequently constructed by concatenating HMMs for phonemes
  - Phonemes are far fewer in number than words, and occur more frequently in training data
  - Words that were never seen in the training data can be constructed from phoneme HMMs

Bayesian Classification between word sequences

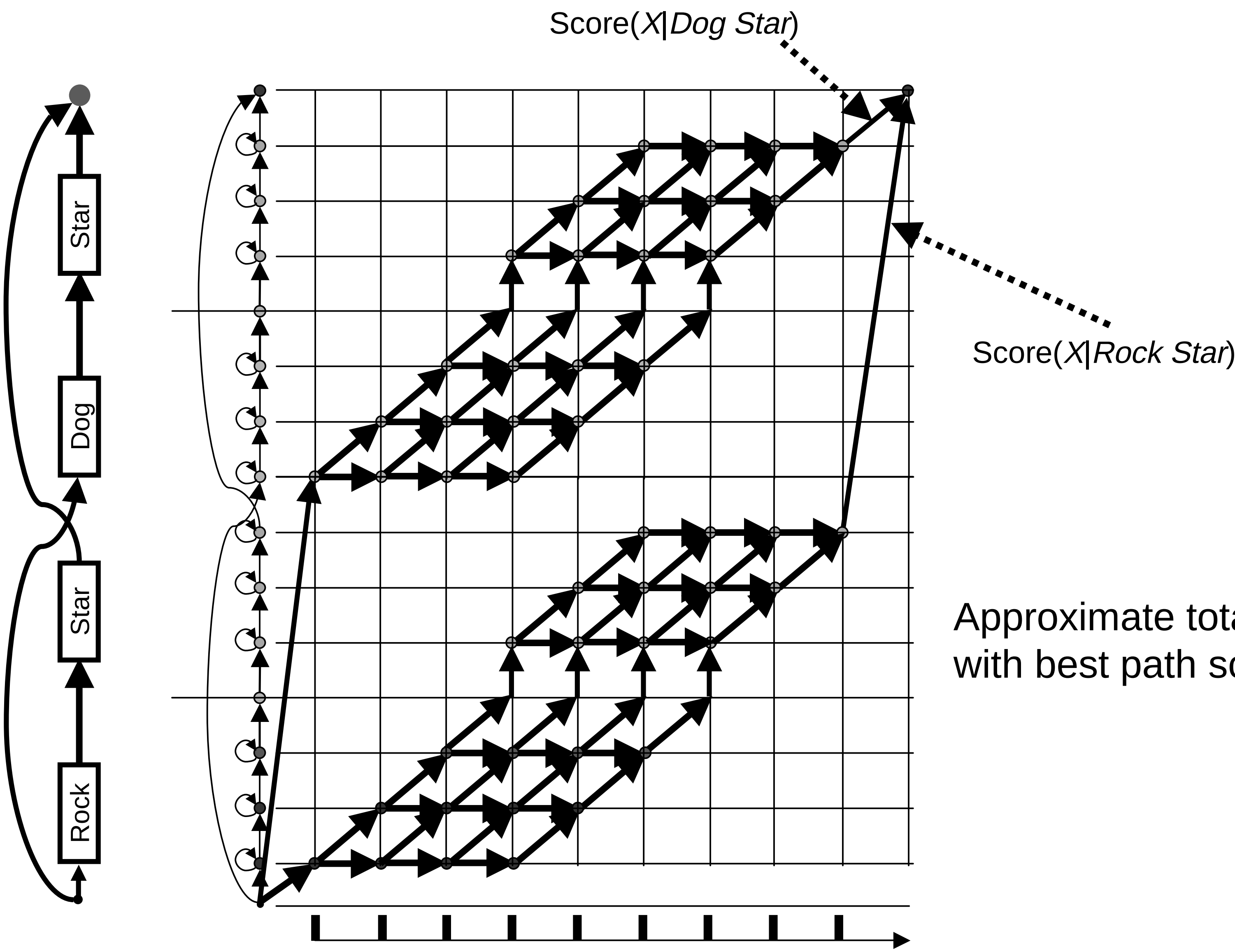
- Classifying an utterance as either “Rock Star” or “Dog Star”
  - Must compare P(Rock,Star)P(X|Rock Star) with P(Dog,Star)P(X|Dog Star)



Bayesian Classification between word sequences

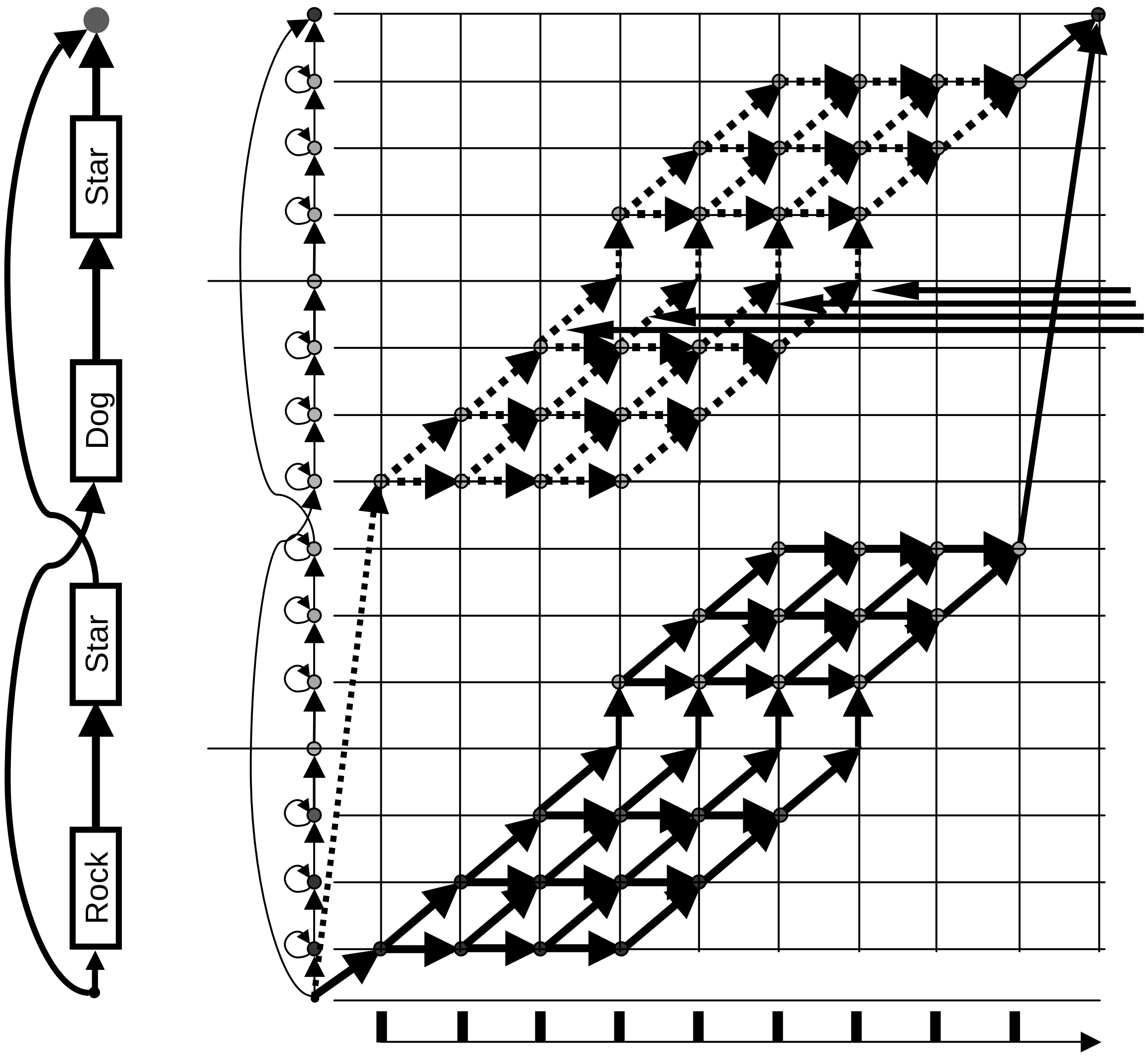


Decoding word sequences



Approximate total probability with best path score

Decoding word sequences



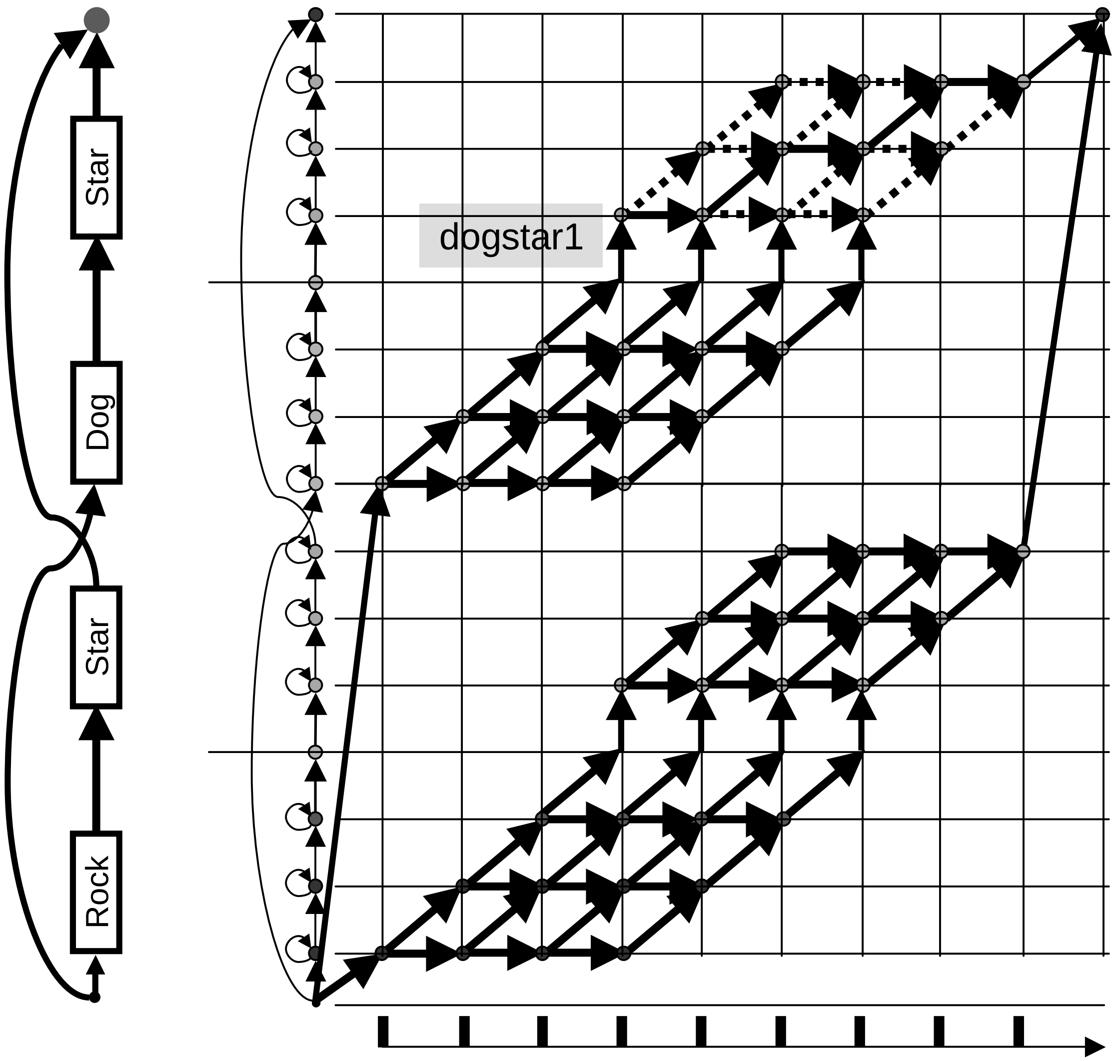
The best path through *Dog Star* lies within the dotted portions of the trellis

There are four transition points from *Dog* to *Star* in this trellis

There are four different sets paths through the dotted trellis, each with its own best path

Decoding word sequences

SET 1 and its best path



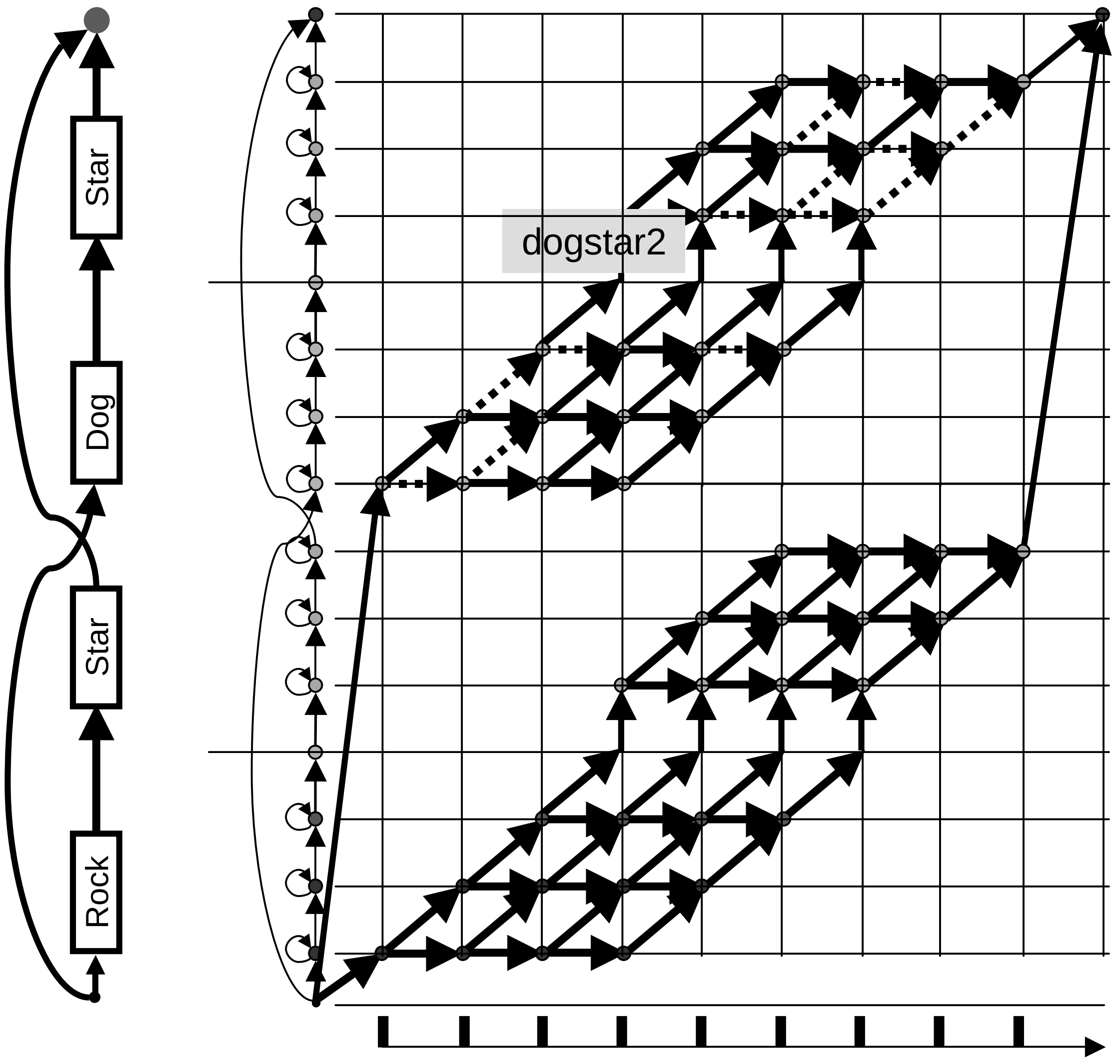
The best path through *Dog Star* lies within the dotted portions of the trellis

There are four transition points from Dog to Star in this trellis

There are four different sets paths through the dotted trellis, each with its own best path

Decoding word sequences

SET 2 and its best path



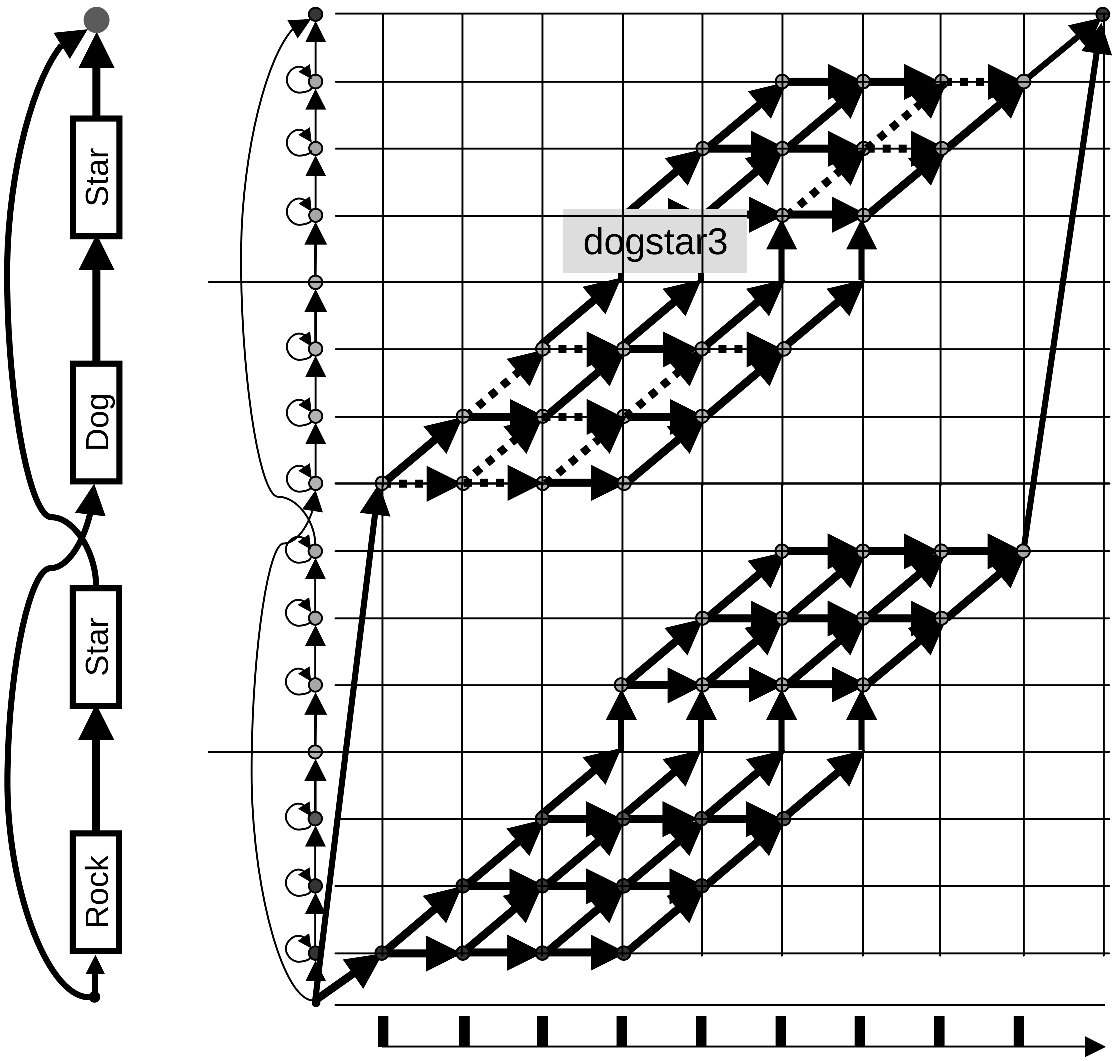
The best path through *Dog Star* lies within the dotted portions of the trellis

There are four transition points from Dog to Star in this trellis

There are four different sets paths through the dotted trellis, each with its own best path

Decoding word sequences

SET 3 and its best path



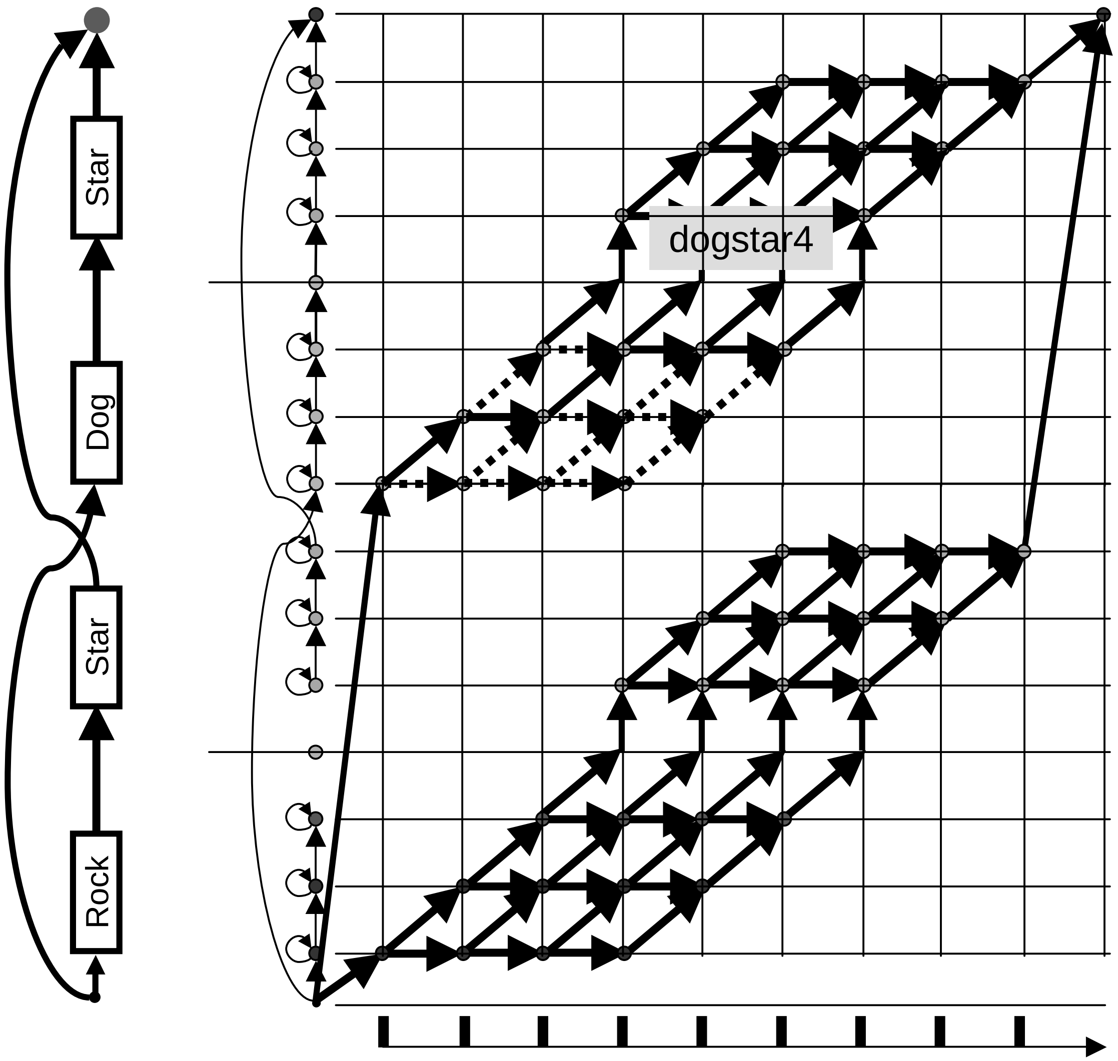
The best path through *Dog Star* lies within the dotted portions of the trellis

There are four transition points from Dog to Star in this trellis

There are four different sets paths through the dotted trellis, each with its own best path

Decoding word sequences

SET 4 and its best path

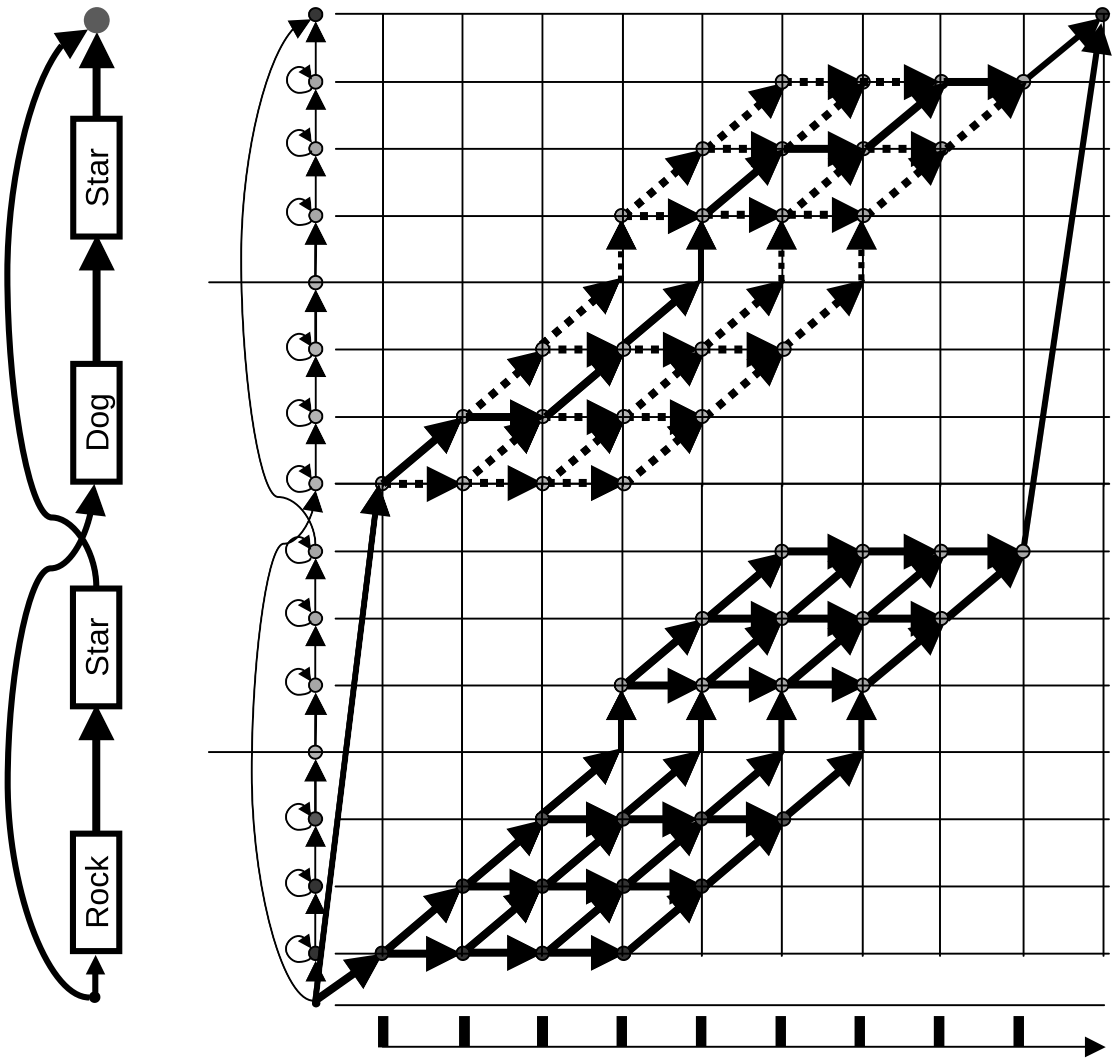


The best path through *Dog Star* lies within the dotted portions of the trellis

There are four transition points from Dog to Star in this trellis

There are four different sets paths through the dotted trellis, each with its own best path

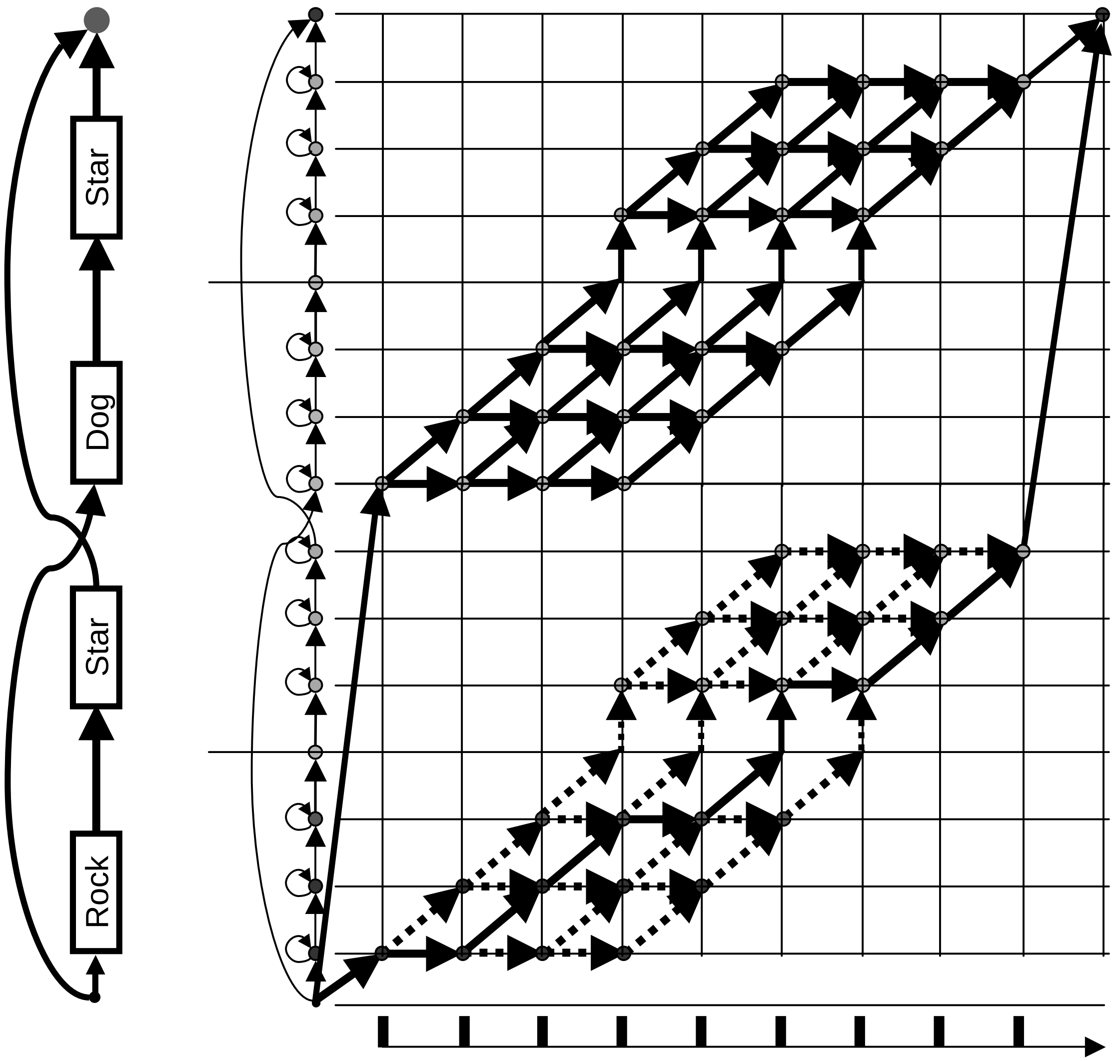
Decoding word sequences



The best path through *Dog Star* is the best of the four transition-specific best paths

max(dogstar) =
max (dogstar1, dogstar2,
dogstar3, dogstar4)

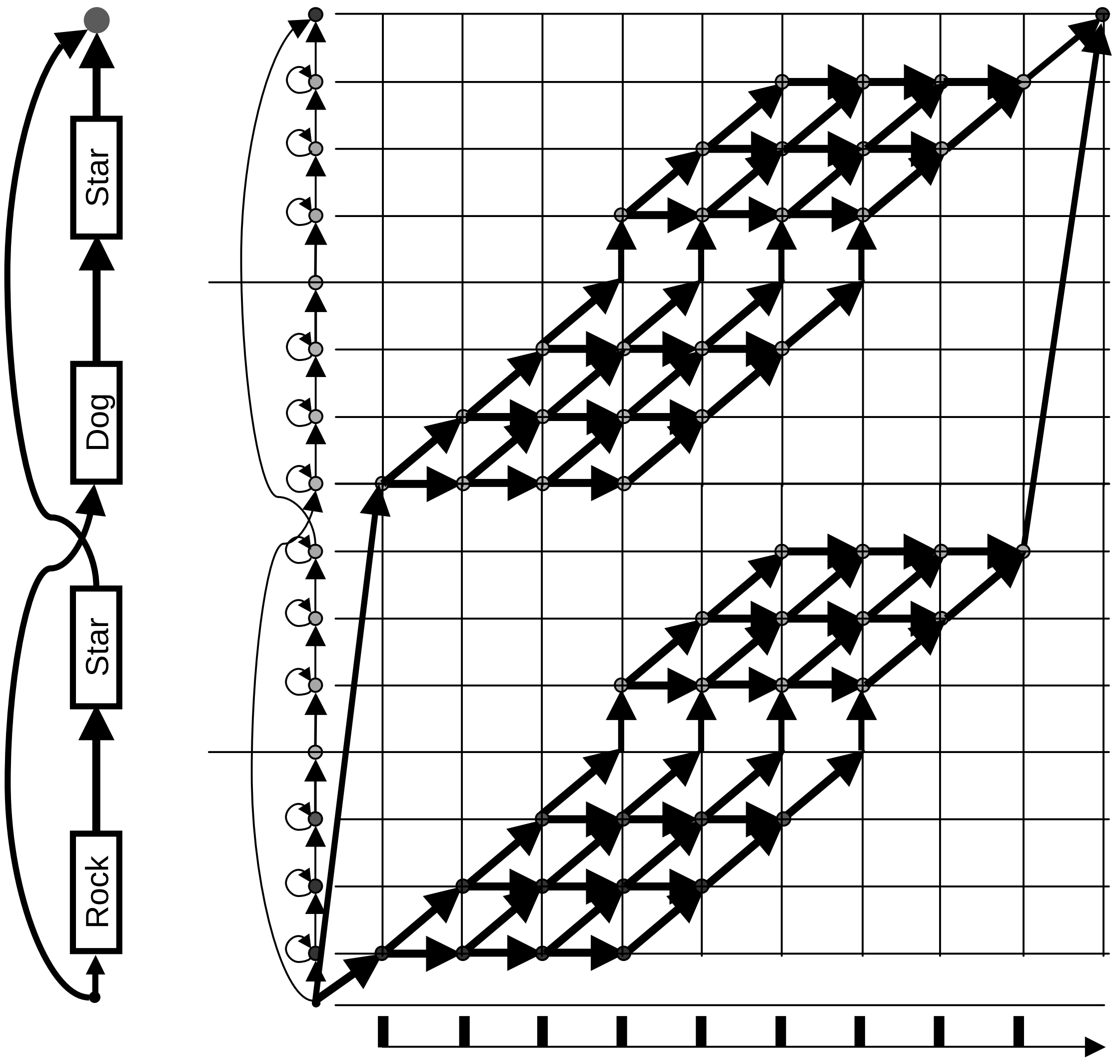
Decoding word sequences



Similarly, for *Rock Star* the best path through the trellis is the best of the four transition-specific best paths

max(rockstar) =
max (rockstar1, rockstar2,
rockstar3, rockstar4)

Decoding word sequences



Then we'd compare the best paths through *Dog Star* and *Rock Star*

max(dogstar) =

max (dogstar1, dogstar2,
dogstar3, dogstar4)

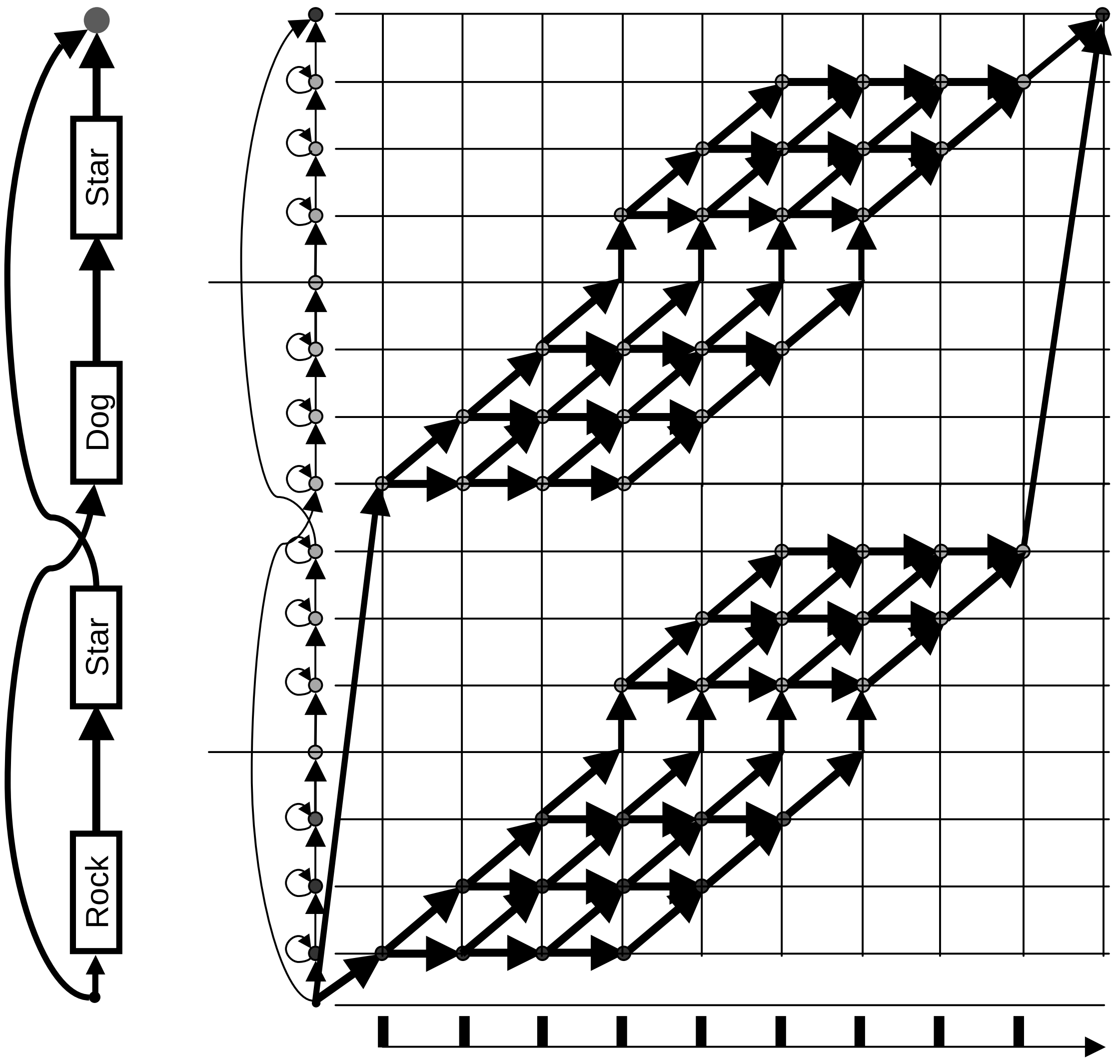
max(rockstar) =

max (rockstar1, rockstar2,
rockstar3, rockstar4)

Viterbi =

max(max(dogstar),
max(rockstar))

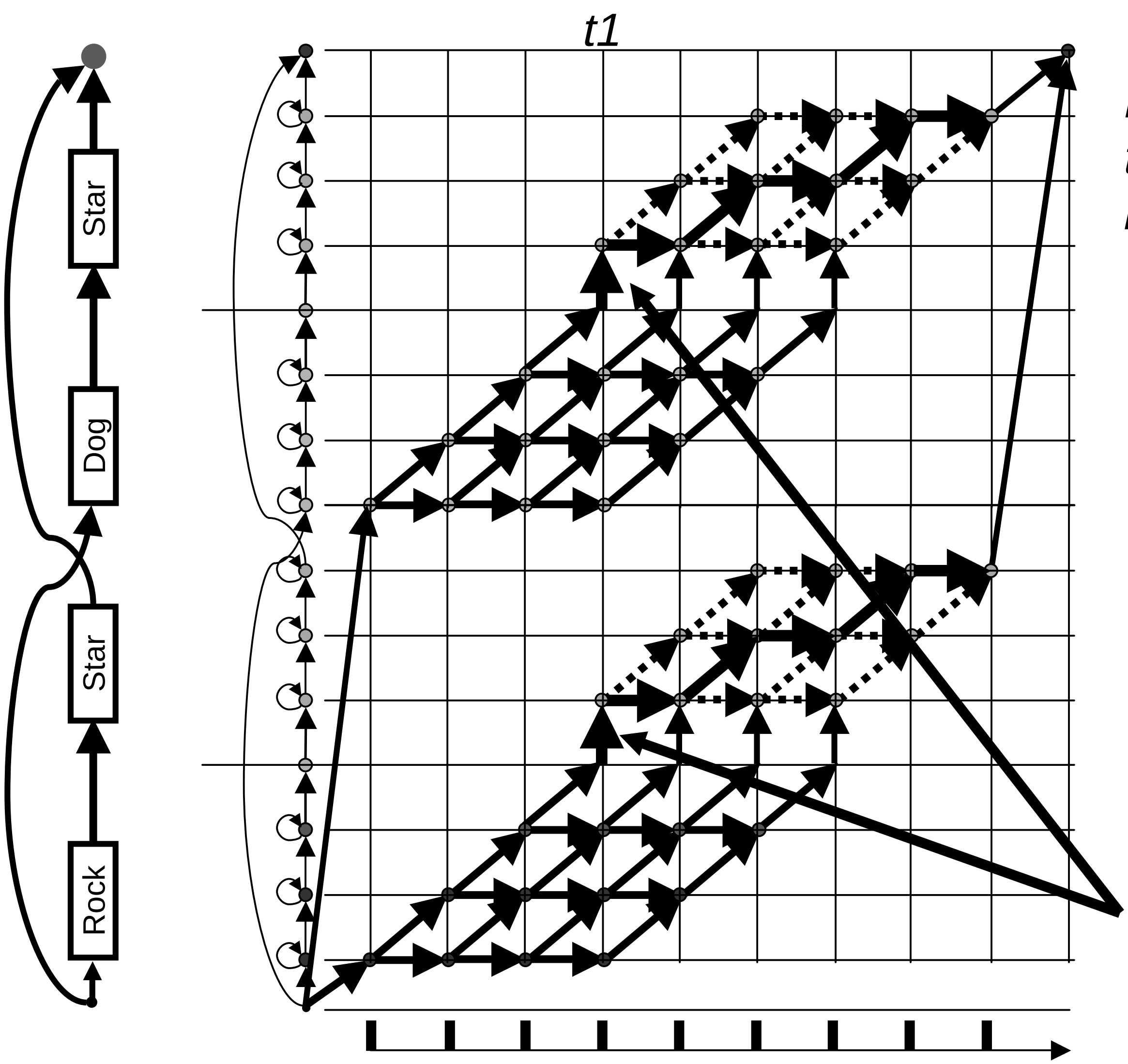
Decoding word sequences



argmax is commutative:

max(max(dogstar), max(rockstar))
=
max (
max(dogstar1, rockstar1),
max (dogstar2, rockstar2),
max (dogstar3,rockstar3),
max(dogstar4,rockstar4)
)

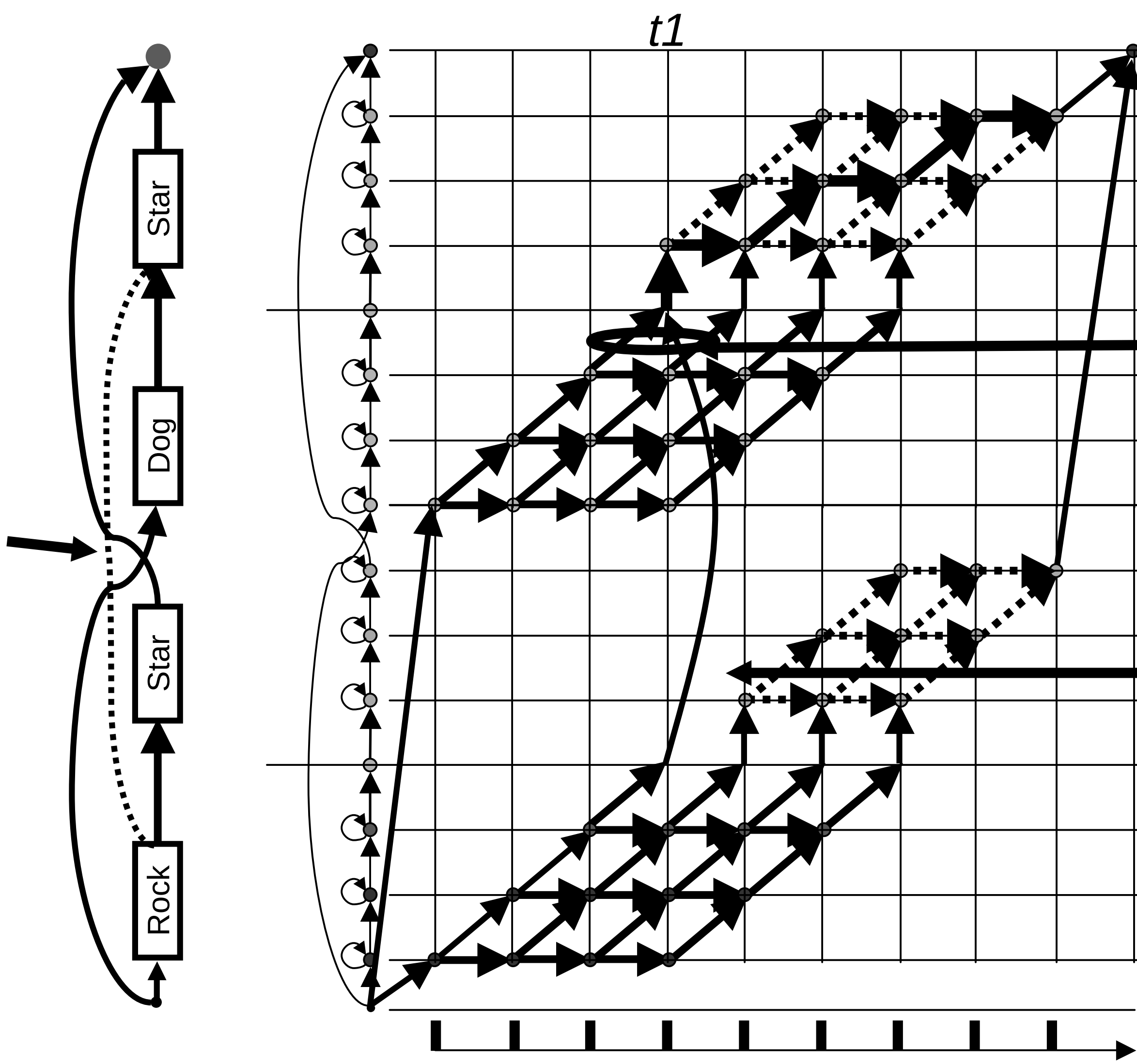
Decoding word sequences



For a given entry point the best path through STAR is the same for both trellises

We can choose between Dog and Rock right here because the futures of these paths are identical

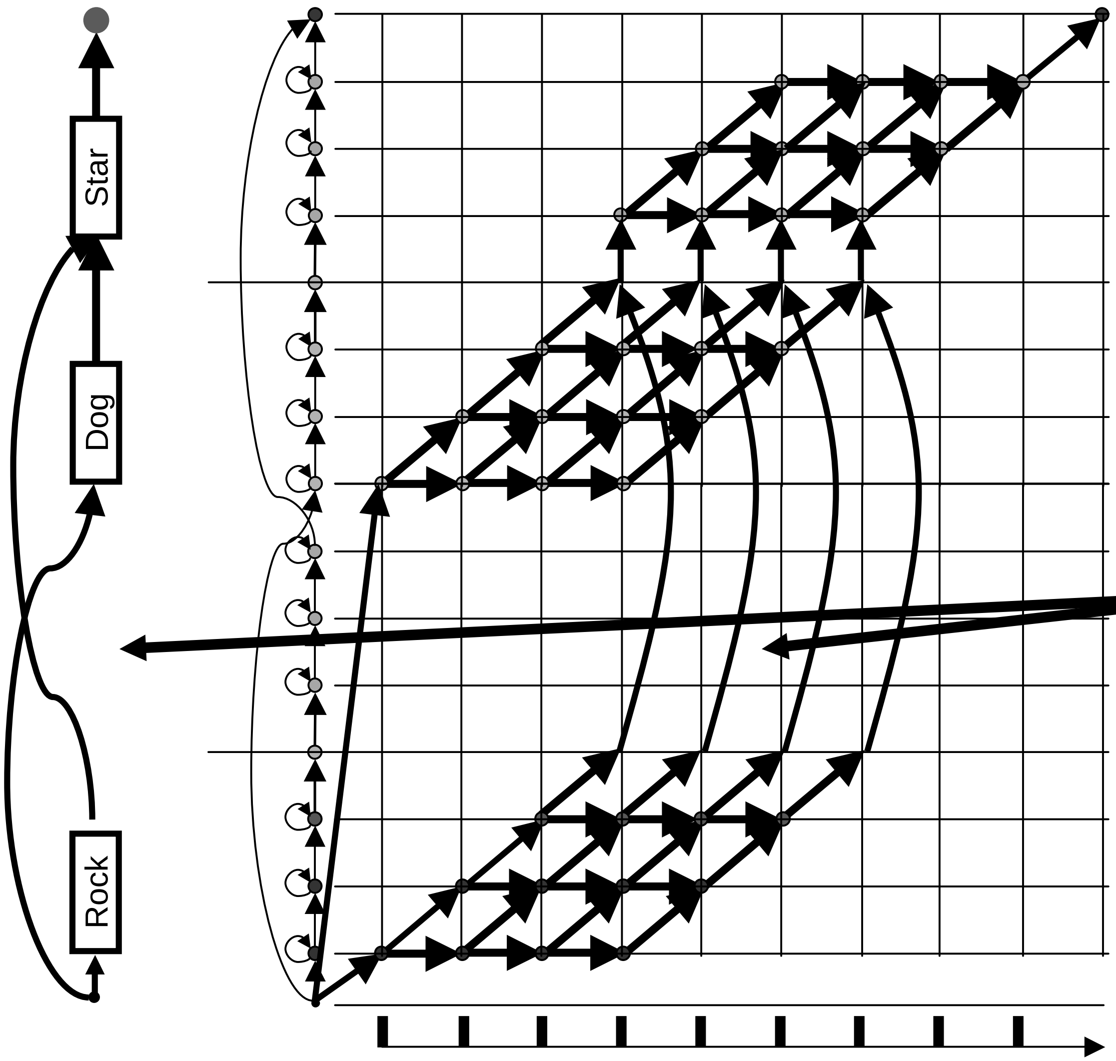
Decoding word sequences



We select the higher scoring of the two incoming edges here

This portion of the trellis is now deleted

Decoding word sequences

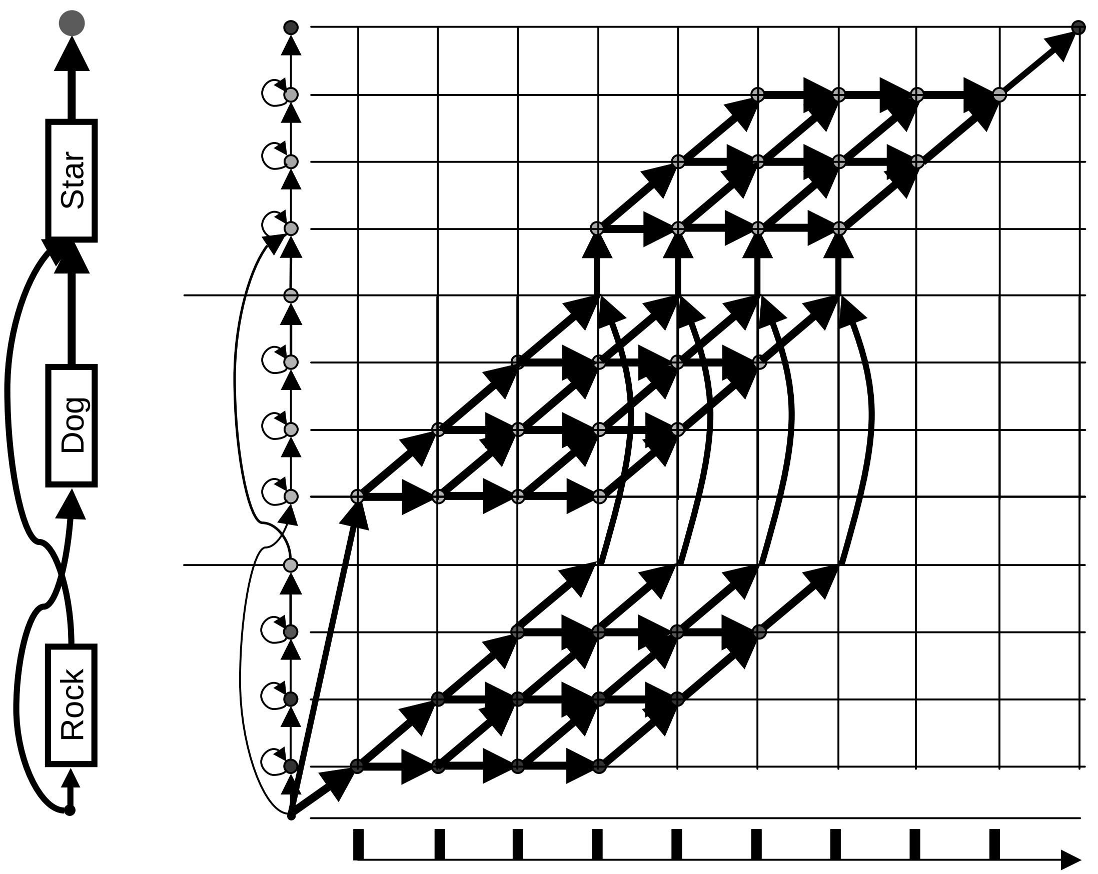


Similar logic can be applied at other entry points to *Star*

This copy of the trellis for *STAR* is completely removed

Decoding word sequences

- The two instances of *Star* can be collapsed into one to form a smaller trellis
  - This is possible because we are decoding based on best path score, instead of full forward score

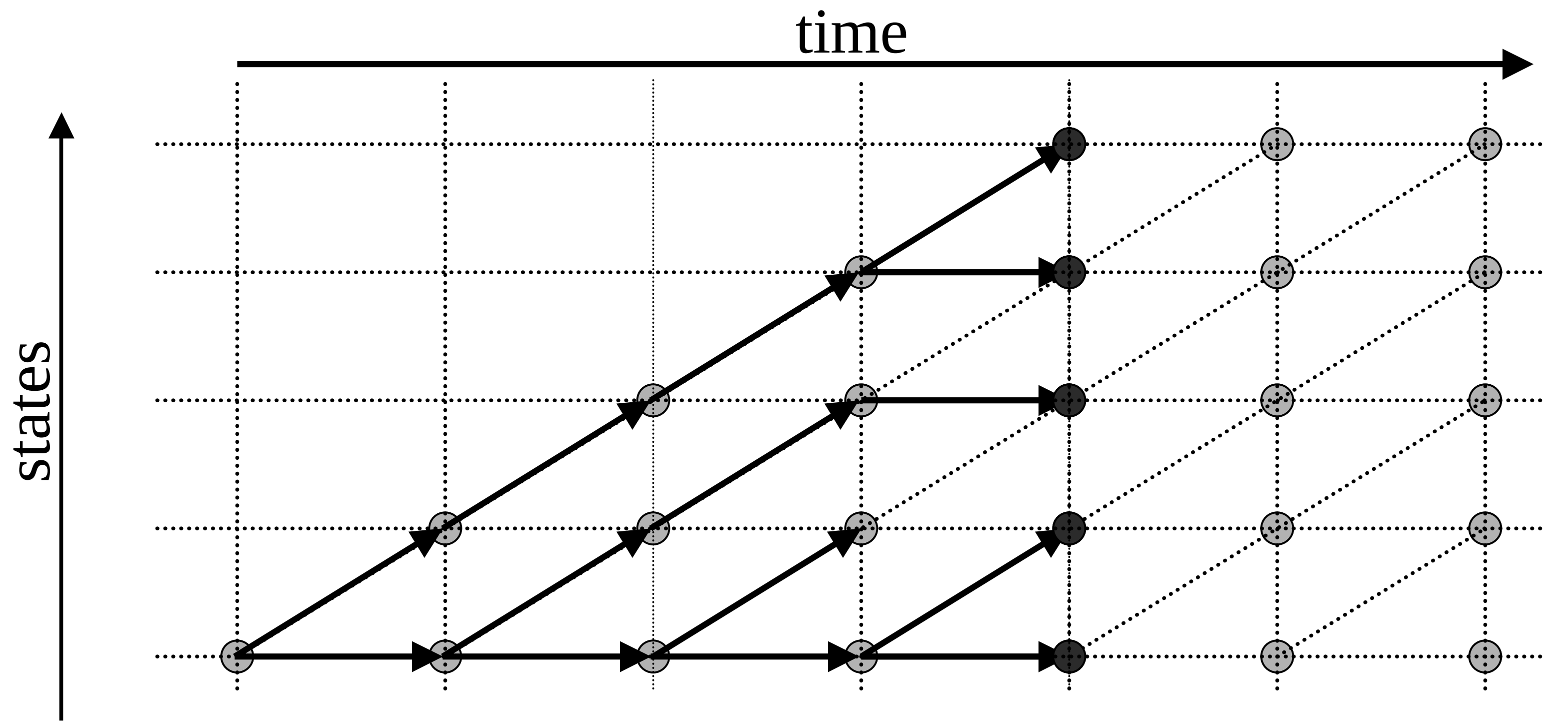


We can only do Viterbi decoding on this collapsed graph.

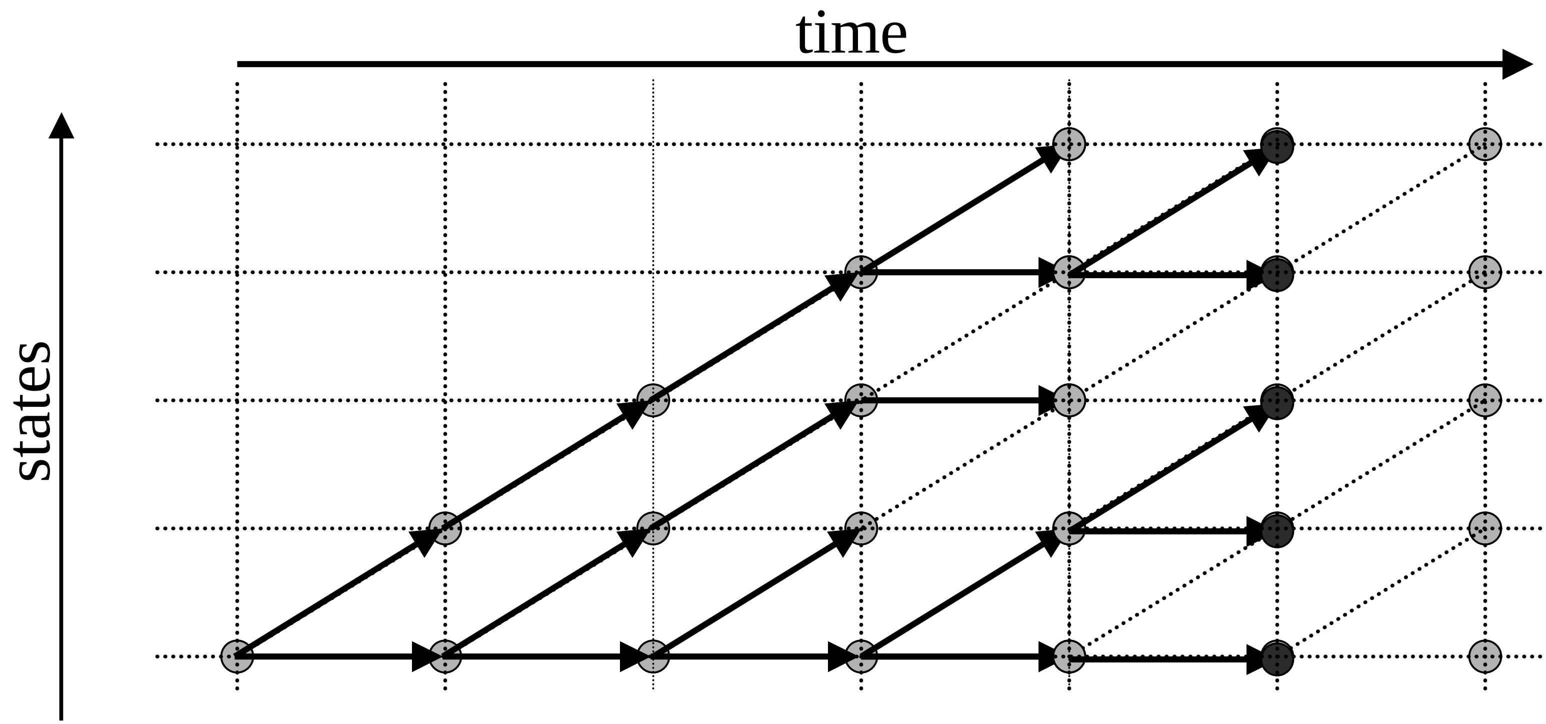
Full forward decoding is no longer possible

Extending paths through the trellis: Breadth First vs. Depth First Search

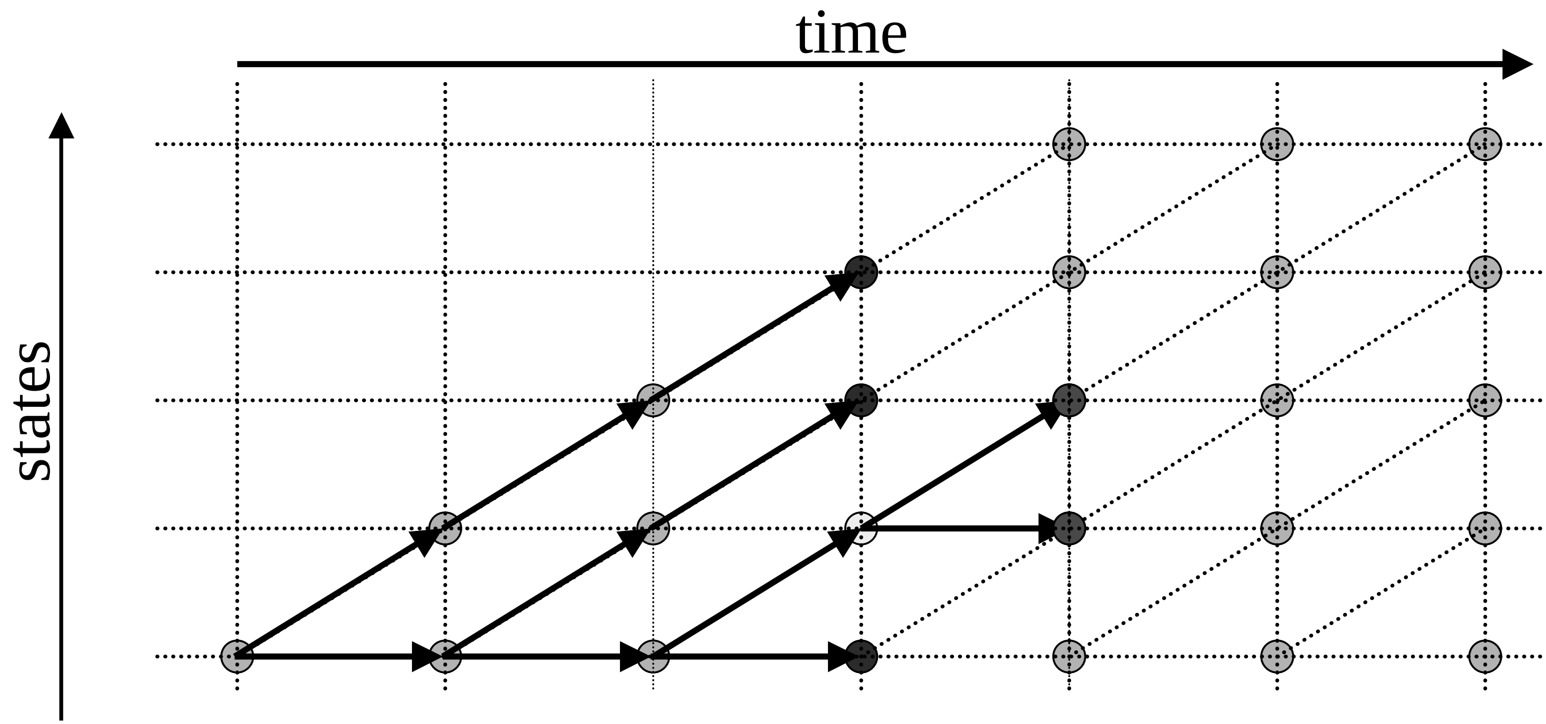
Breadth First Search



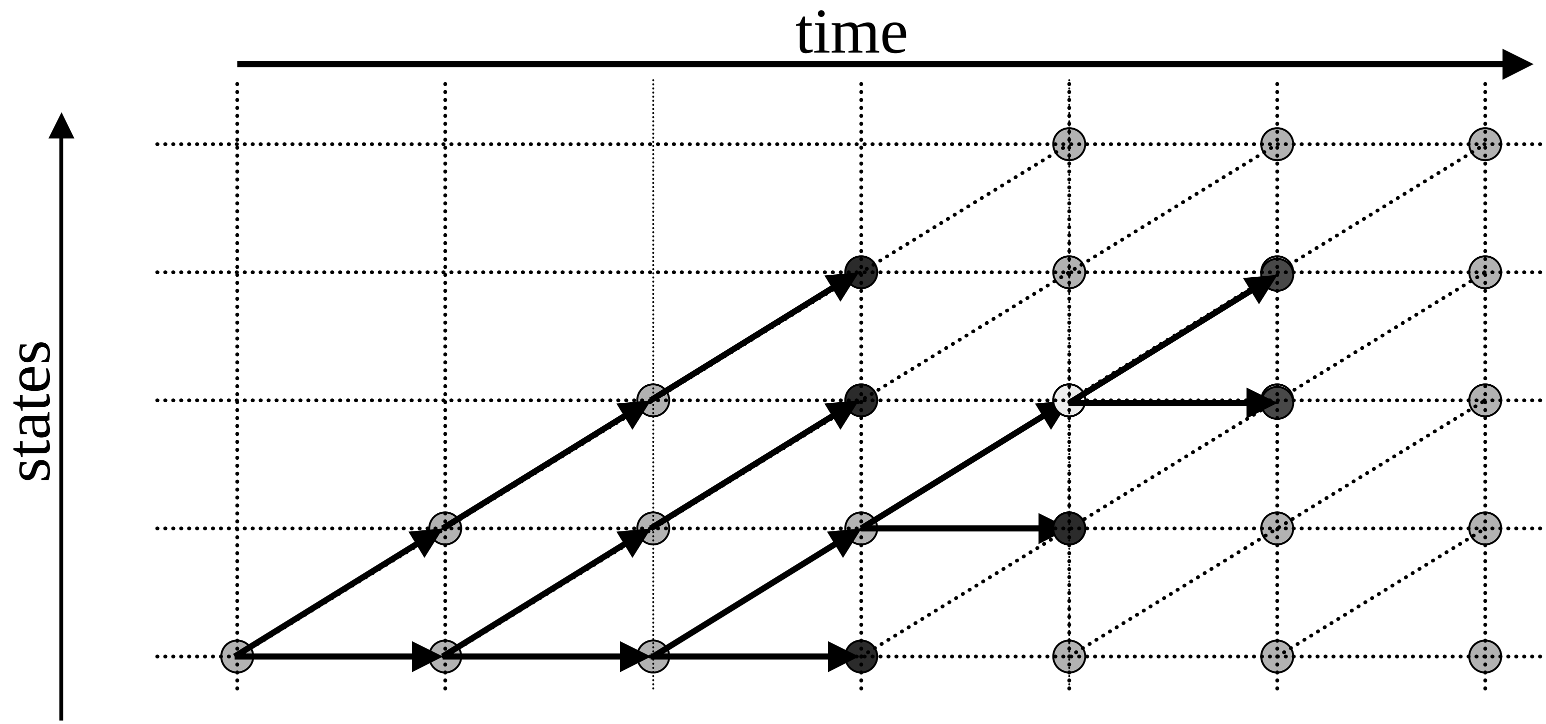
Breadth First Search



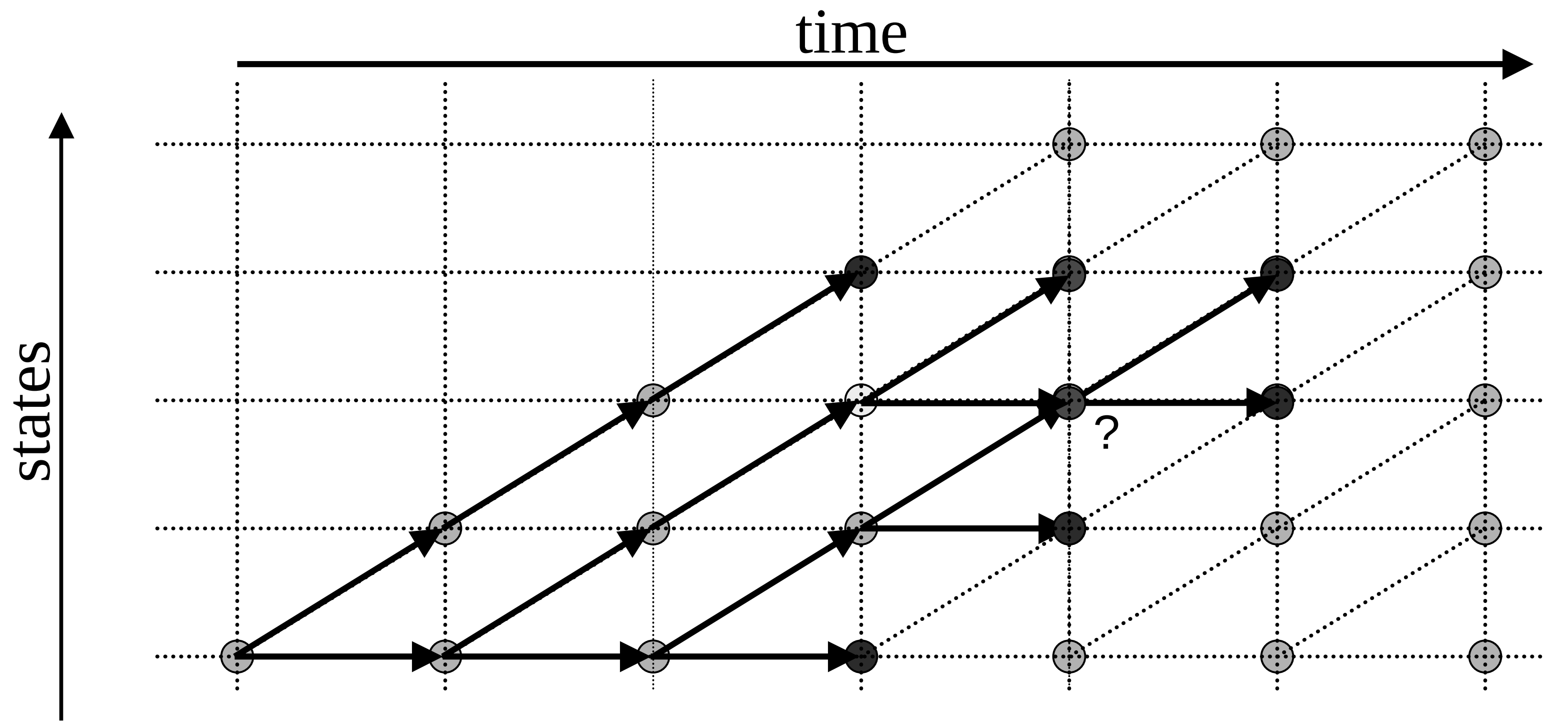
Depth First Search



Depth First Search



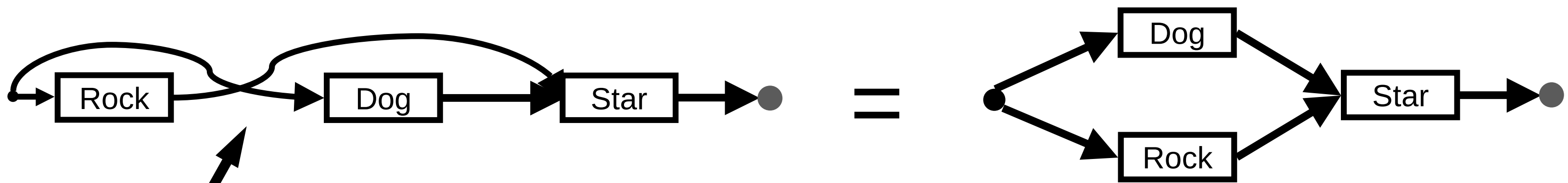
Depth First Search

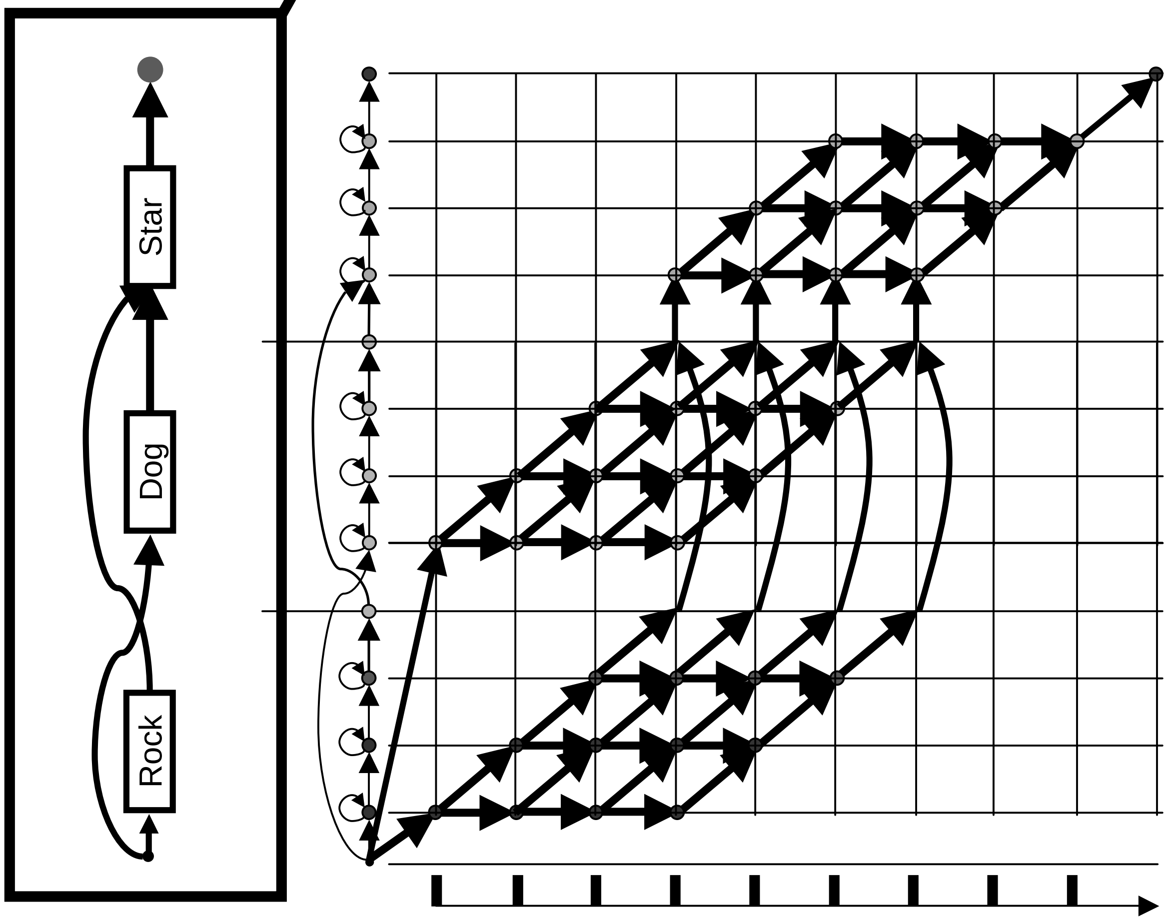


- No inconsistencies arise if path scores change monotonically with increasing path length
- This can be guaranteed by normalizing all state output density values for a vector at a time *t* with respect to the highest valued density at *t*.

Designing optimal graph structures for the language HMM

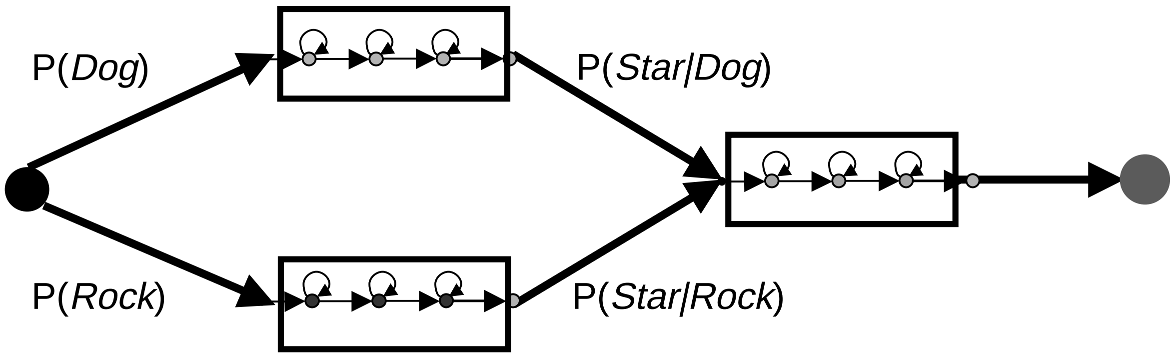
Language HMMs for fixed-length word sequences





We will represent the= vertical axis of the = trellis in this simplified= manner=

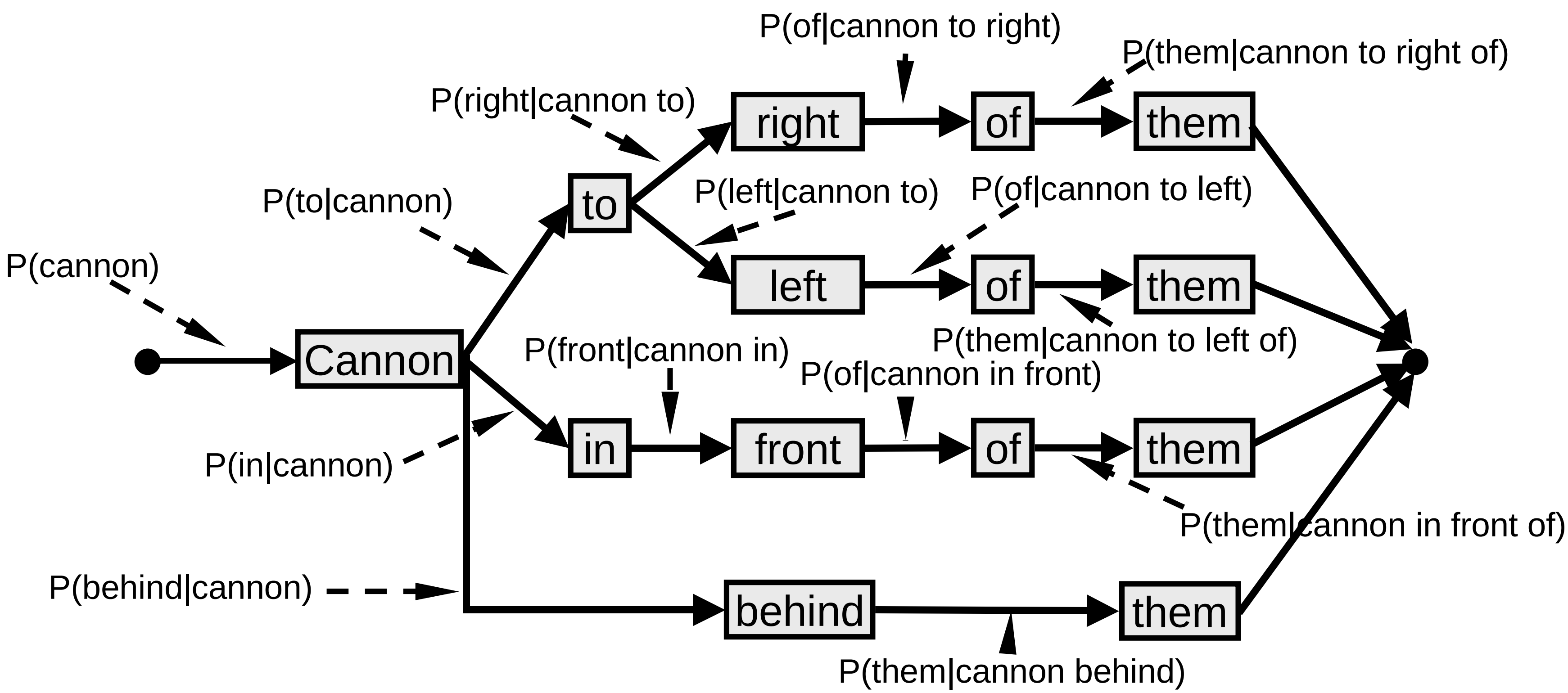
Language HMMs for fixed-length word sequences



- The word graph represents all allowed word sequences in our example
  - The set of all allowed word sequences represents the allowed "language"
- At a more detailed level, the figure represents an HMM composed of the HMMs for all words in the word graph
  - This is the "Language HMM" – the HMM for the entire allowed language
- The language HMM represents the vertical axis of the trellis
  - It is the trellis, and NOT the language HMM, that is searched for the best path

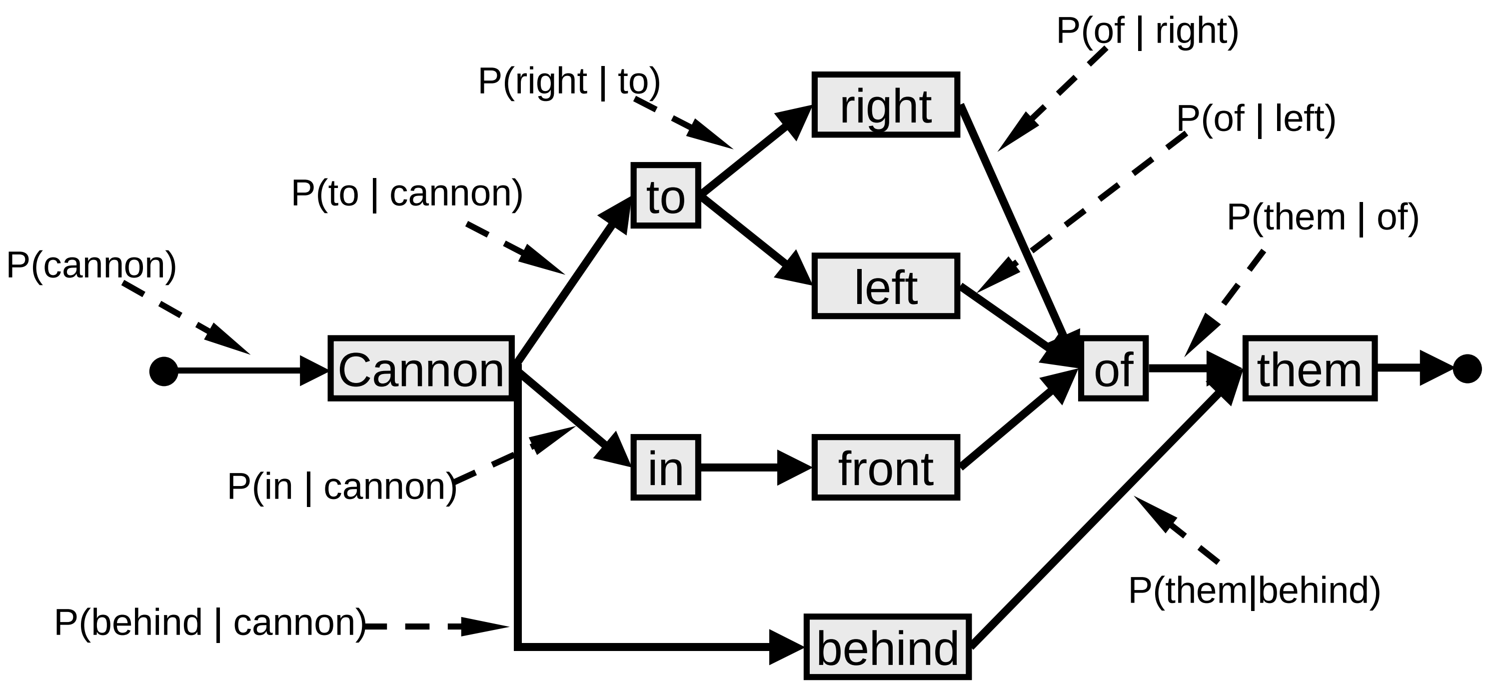
Language HMMs for fixed-length word sequences

- Recognizing one of four lines from “charge of the light brigade”
  - Cannon to right of them
  - Cannon to left of them
  - Cannon in front of them
  - Cannon behind them



Simplification of the language HMM through lower context language models

- Recognizing one of four lines from “charge of the light brigade”
- If the probability of a word only depends on the preceding word, the graph can be collapsed:
  - e.g. P(them | cannon to right of) = P(them | cannon to left of) = P(cannon | of)



Refactored recognition problem for language HMMs which incorporate with lower context language models

- Our new statement of the problem with a smaller language model requirement:

$word1, word2, \ldots =$

$$argmax_{wd_1, wd_2, \ldots}\{argmax_{s_1, s_2, \ldots, s_T} P(wd_1, wd_2, \ldots) P(X_1, X_2, ..X_T, s_1, s_2, \ldots, s_T | wd_1, wd_2, \ldots)\}$$

- The probability term can be factored into individual words as

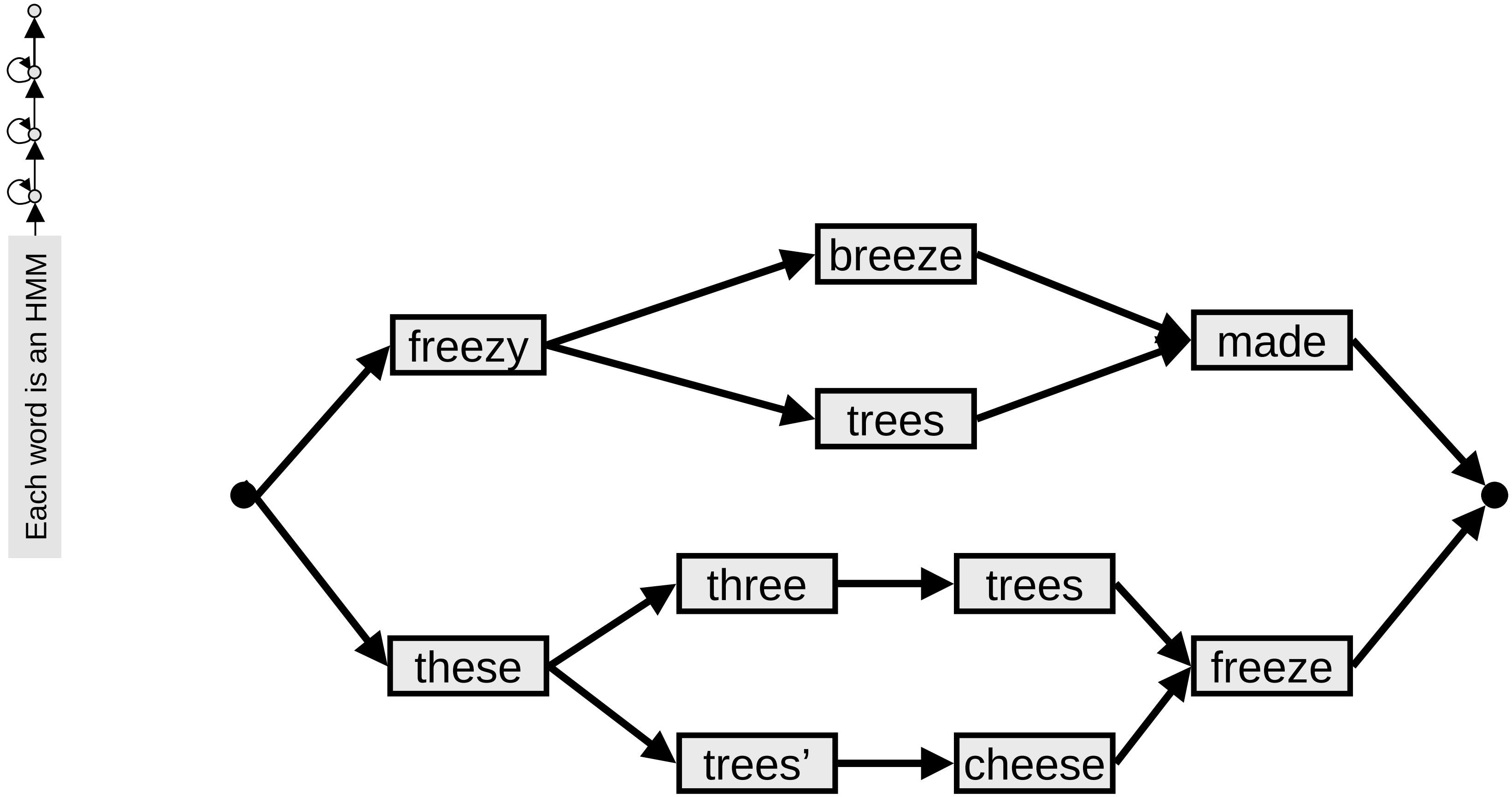
$$P(wd_1)P(X_{wd1}, S_{wd1}|wd_1).P(wd_2|wd_1)P(X_{wd2}, S_{wd2}|wd_2).$$
$$P(wd_3|wd_1, wd2)P(X_{wd3}, S_{wd3}\ |wd_3)\ldots P(wd_N|wd1...wd_{N-1})P(X_{wdN}, S_{wdN}|wd_N)$$

- Assume contexts beyond a given length K do not matter

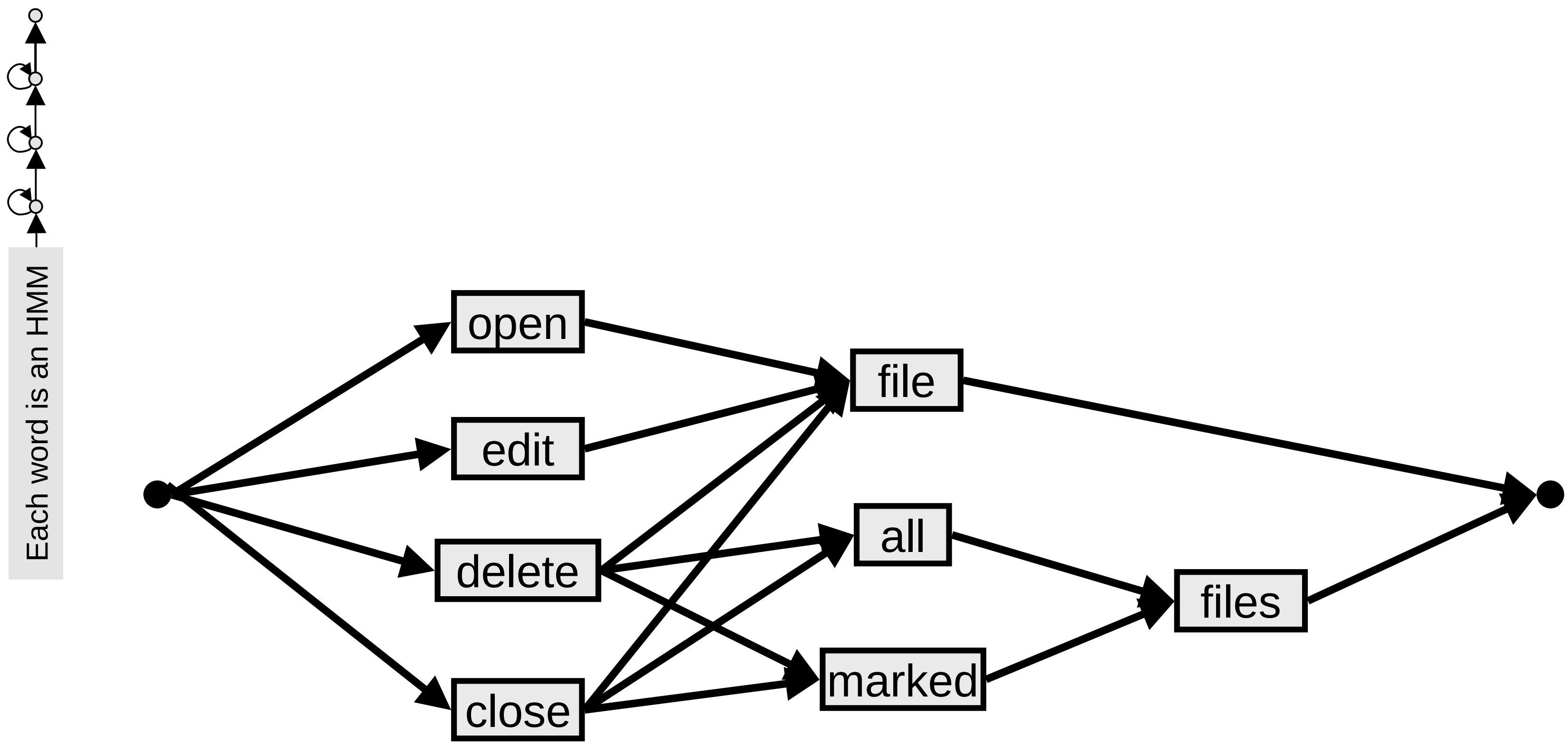
$$P(wd_N|\ wd_{N-1},\ wd_{N-2},\ ..,\ wd_{N-K},\ ..,\ wd_1) = P(wd_{N-}|\ wd_{N-1},\ wd_{N-2}, ..,\ wd_{N-K})$$

- Nodes with the same word history wd_{N-1}, wd_{N-2},.., wd_{N-K} can be collapsed

Language HMMs for fixed-length word sequences: based on a grammar for Dr. Seuss



Language HMMs for fixed-length word sequences: command and control grammar



Language HMMs for arbitrarily long word sequences

- Previous examples chose between a finite set of known word sequences
- Word sequences can be of arbitrary length
  - E.g. set of all word sequences that consist of an arbitrary number of repetitions of the word bang

    bang

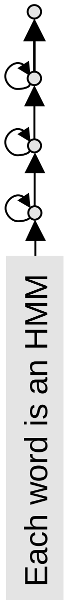
    bang bang

    bang bang bang

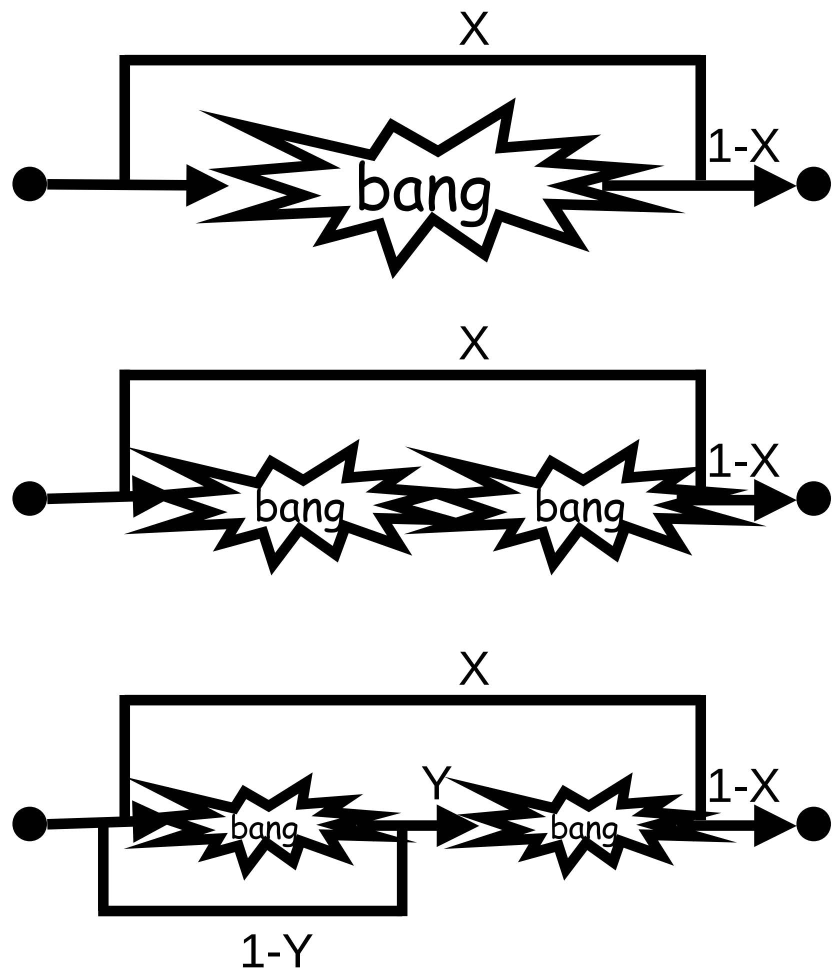
    bang bang bang bang

    ……
  - Forming explicit word-sequence graphs of the type we've seen so far is not possible
    - The number of possible sequences (with non-zero *a-priori* probability) is potentially infinite
    - Even if the longest sequence length is restricted, the graph will still be large

Language HMMs for arbitrarily long word sequences



- Arbitrary word sequences can be modeled with loops under some assumptions. E.g.:
- A “bang” can be followed by another “bang” with probability P(“bang”).
  - P(“bang”) = X; P(Termination) = 1-X;
- Bangs can occur only in pairs with probability X
- A more complex graph allows more complicated patterns
- You can extend this logic to other vocabularies where the speaker says more things than “bang”
  - e.g. “bang bang you’re dead”



Language HMMs for arbitrarily long word sequences

- Constrained set of word sequences with constrained vocabulary are realistic
  - Typically in command-and-control situations
    - Example: operating TV remote
  - Simple dialog systems
    - When the set of permitted responses to a query is restricted
- Unconstrained length word sequences : NATURAL LANGUAGE
  - State-of-art large vocabulary decoders

Language HMMs for Natural language: with unigram representations

- A bag of words model:
  - Any word is possible after any word.
  - The probability of a word is independent of the words preceding or succeeding it.
  - Also called a UNIGRAM model.

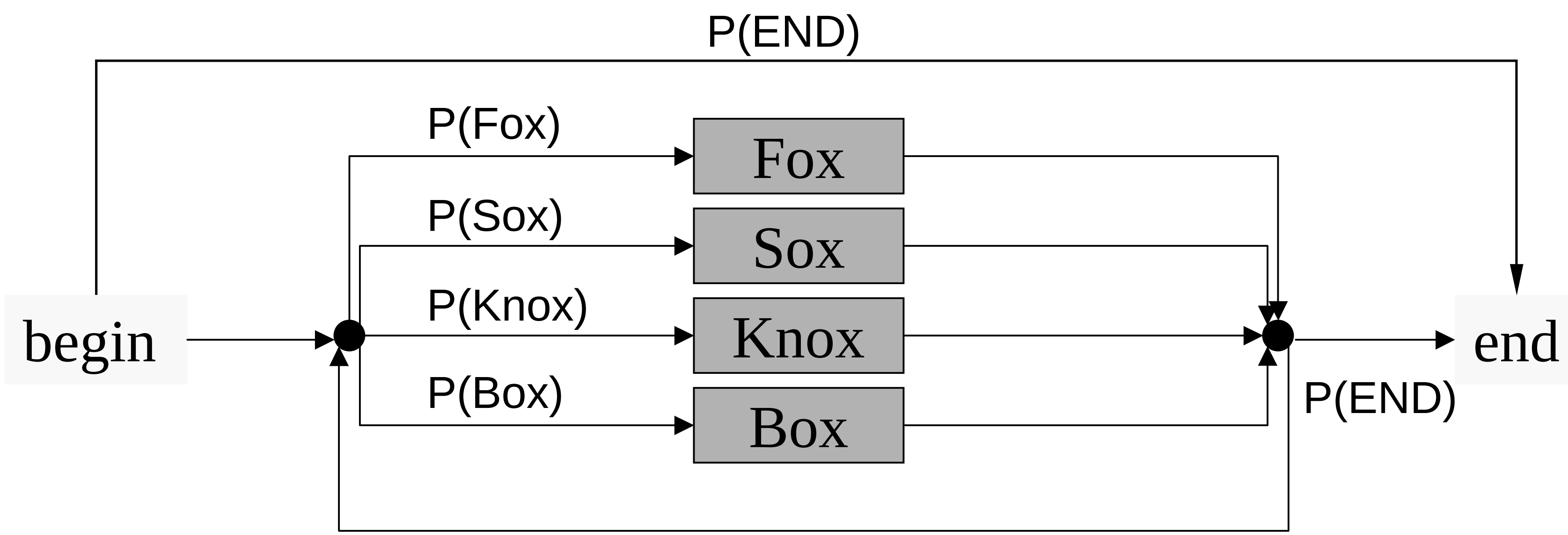
$$P(Star) = P(Star \mid Dog) = P(Star \mid Rock) = P(Star \mid When\ you\ wish\ upon\ a)$$

$$P(When\ you\ wish\ upon\ a\ star) =$$
$$P(When)\ P(you)\ P(wish)\ P(upon)\ P(a)\ P(star)\ P(\mathrm{END})$$

- "END" is a special symbol that indicates the end of the word sequence
- P(END) is necessary – without it the word sequence would never terminate
  - The total probability of all possible word sequences must sum to 1.0
  - Only if P(END) is explicitly defined will the total probability = 1.0

Language HMMs for Natural language: example of a graph which uses unigram representations

- Vocabulary: "fox", "sox", "knox", "box"
- Arbitrary length sequences of arbitrary arrangements of these four words.
- Actual probabilities:
  - P(fox), P(sox), P(knox), P(box), P(END)
  - P(fox) + P(sox) + P(knox) + P(box) + P(END) = 1.0



•The black dots are "non-emitting" states that are not associated with observations

Language HMMs for Natural language: bigram representations

- A unigram model is only useful when no statistical dependency between adjacent words can be assumed
  - Or, alternately, when the data used to learn these dependencies are too small to learn them reliably
    - Learning word dependencies: Later in the program

- In natural language, the probability of a word occurring depends on past words.
- Bigram language model: the probability of a word depends on the previous word
- P(Star | A Rock) = P(Star | The Rock) = P(Star | Rock)
- P(Star | Rock) is not required to be equal to P(Star | Dog)
  - In fact the two terms are assumed to be unrelated.

Language HMMs for Natural language: bigram representations

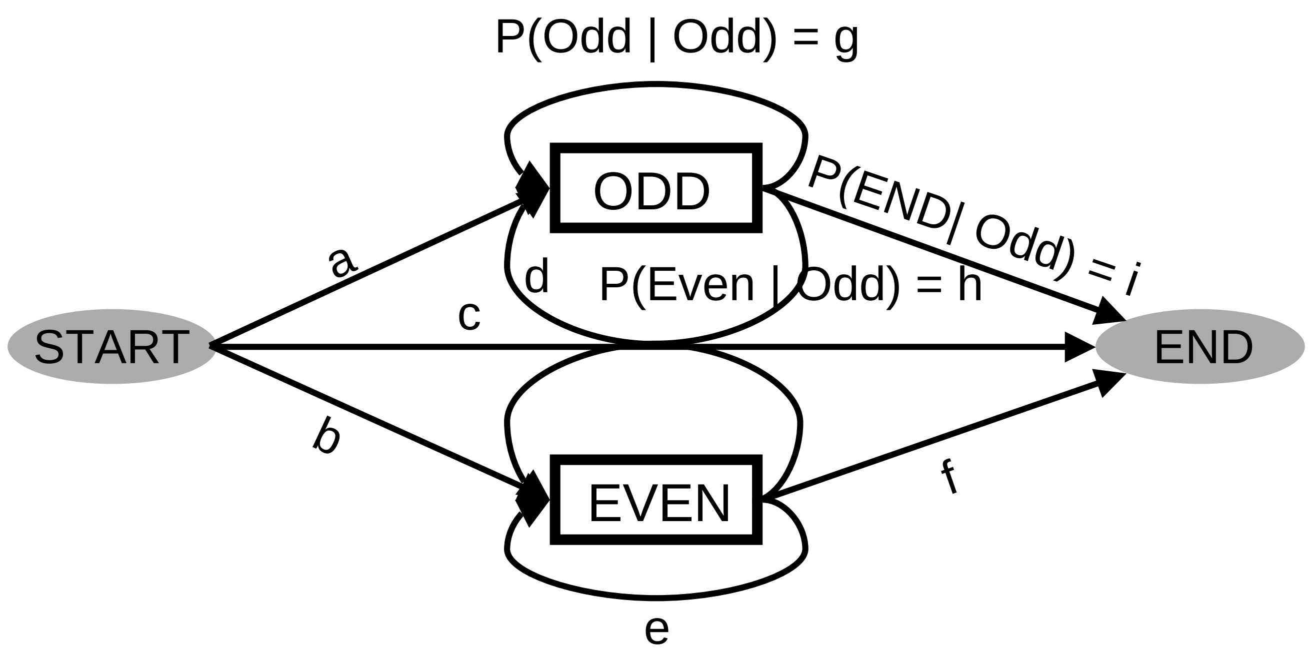
- Simple bigram example:
  - Vocabulary: START, "odd", "even", END
  - P(odd | START) = a, P(even | START) = b, P(END | START) = c
    - a+b+c = 1.0
  - P(odd | even) = d, P(even | even) = e, P(END | even) = f
    - d+e+f = 1.0
  - P(odd | odd) = g, P(even | odd) = h, P(END | odd) = i
    - g+h+i = 1.0

- START is a special symbol, indicating the beginning of the utterance
  - P(word | START) is the probability that the utterance begins with word
  - Prob("odd even even odd") =
    P(odd | START) P(even | odd) P(even | even) P (odd | even) P(END | odd)

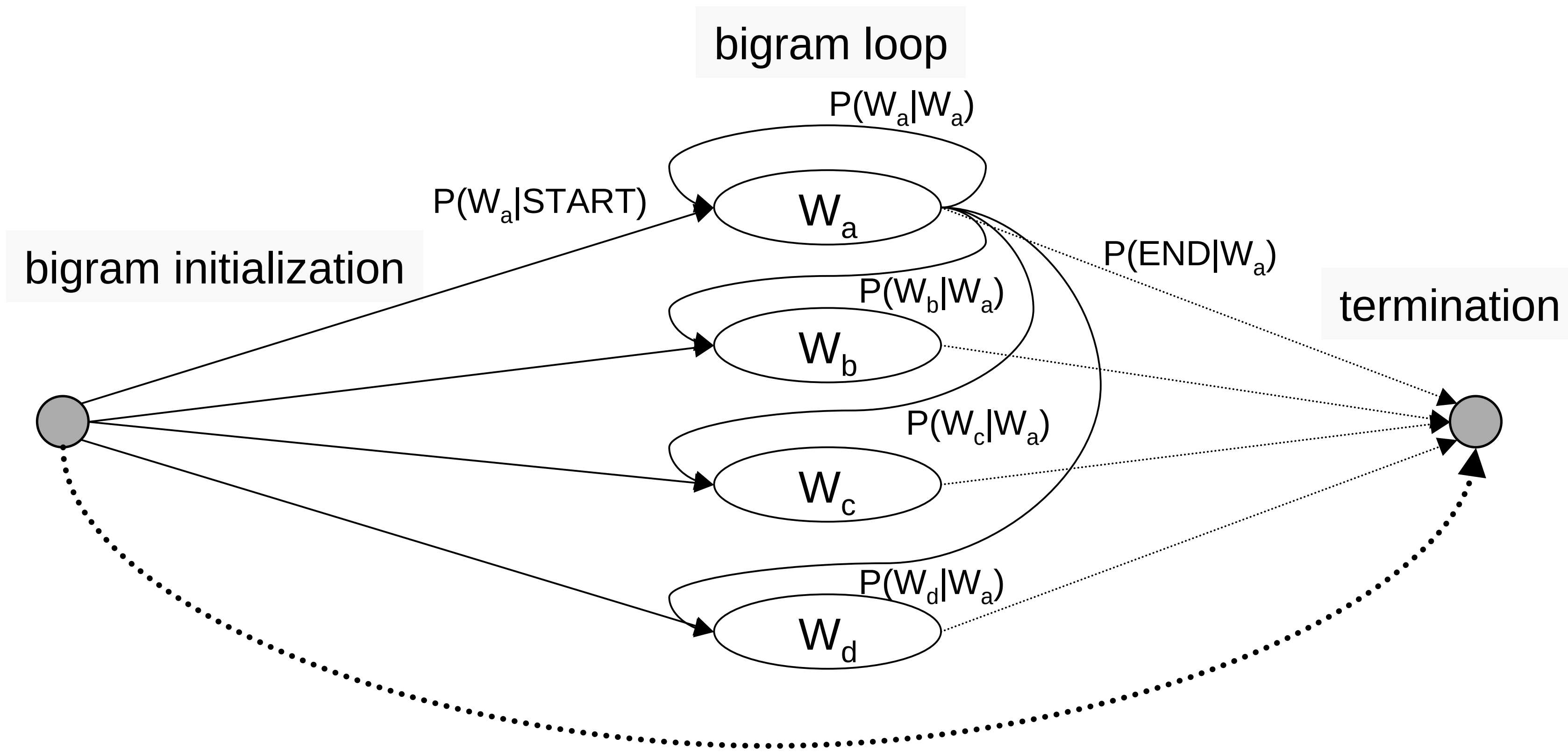
- Can be shown that the total probability of all word sequences of all lengths is 1.0
  - Again, the definition of START and END symbols, and all bigrams involving the two, is crucial

Language HMMs for Natural language: building graphs to incorporate bigram representations

- Edges from “START” contain START dependent word probabilities
- Edges from “Even” contain “Even” dependent word probabilities
- Edges from “Odd” contain “Odd” dependent word probabilities



Language HMMs for Natural language: building graphs to incorporate bigram representations



Language HMMs for Natural language: trigram representations

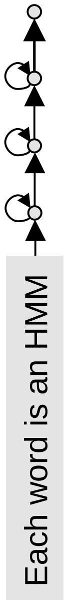
- Assumption: The probability of a word depends on the two preceding words

  P(waltzing | you'll come a) = P(waltzing | who'll come a) =
  P(waltzing | come a)

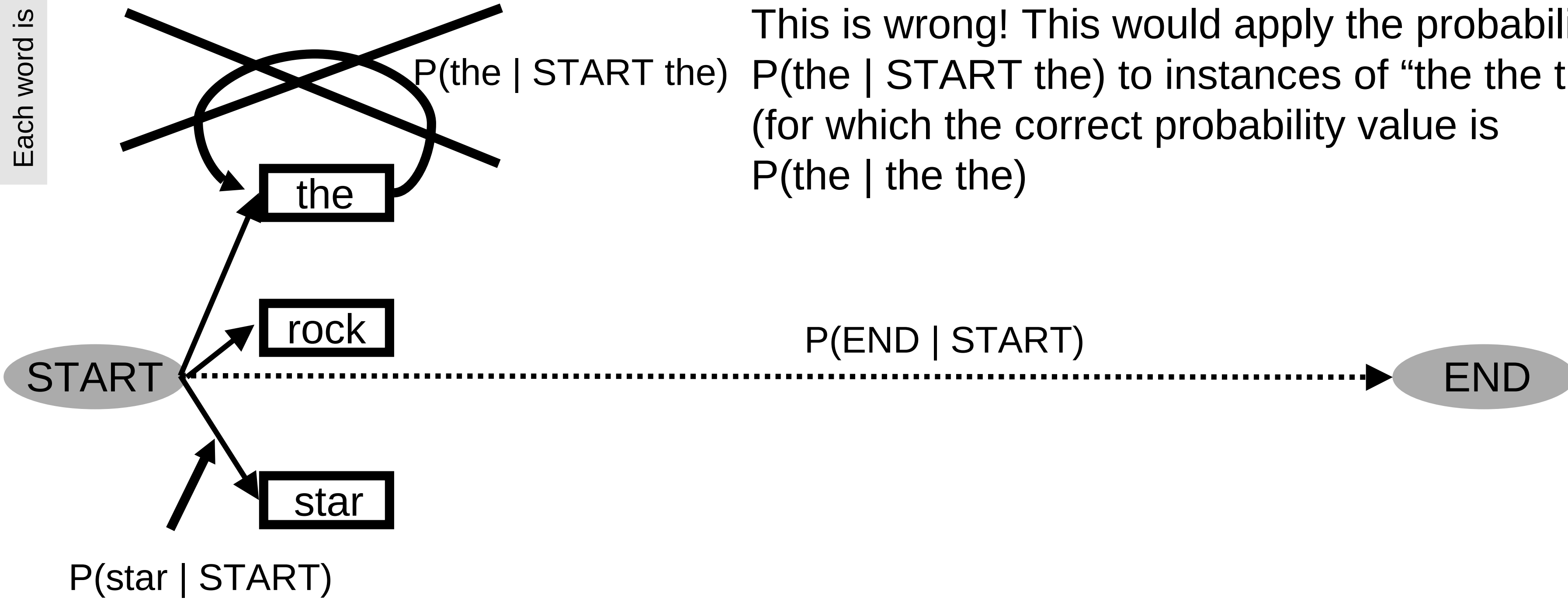
  P(you'll come a waltzing matilda with me) =
  P(you'll | START) * P(come | START you'll) *
  P(a | you'll come) * P(waltzing | come a) *
  P(matilda | a waltzing) * P(with | waltzing matilda) *
  P(me | matilda with) * P(END | with me)

- For any word sequence $w_1, w_2, w_3, w_4, w_5 \ldots w_N$
- $P(w_1, w_2 \ldots w_N) = P(w_1|START)\ P(w_2 \mid START\ w_1)\ P(w_3 \mid w_1\ w_2) \ldots P(END \mid w_{N-1}\ w_N)$
  - Note that the FIRST term $P(w_1 \mid START)$ is a *bigram* term. All the rest are trigrams.

Language HMMs for Natural language: building graphs to incorporate trigram representations



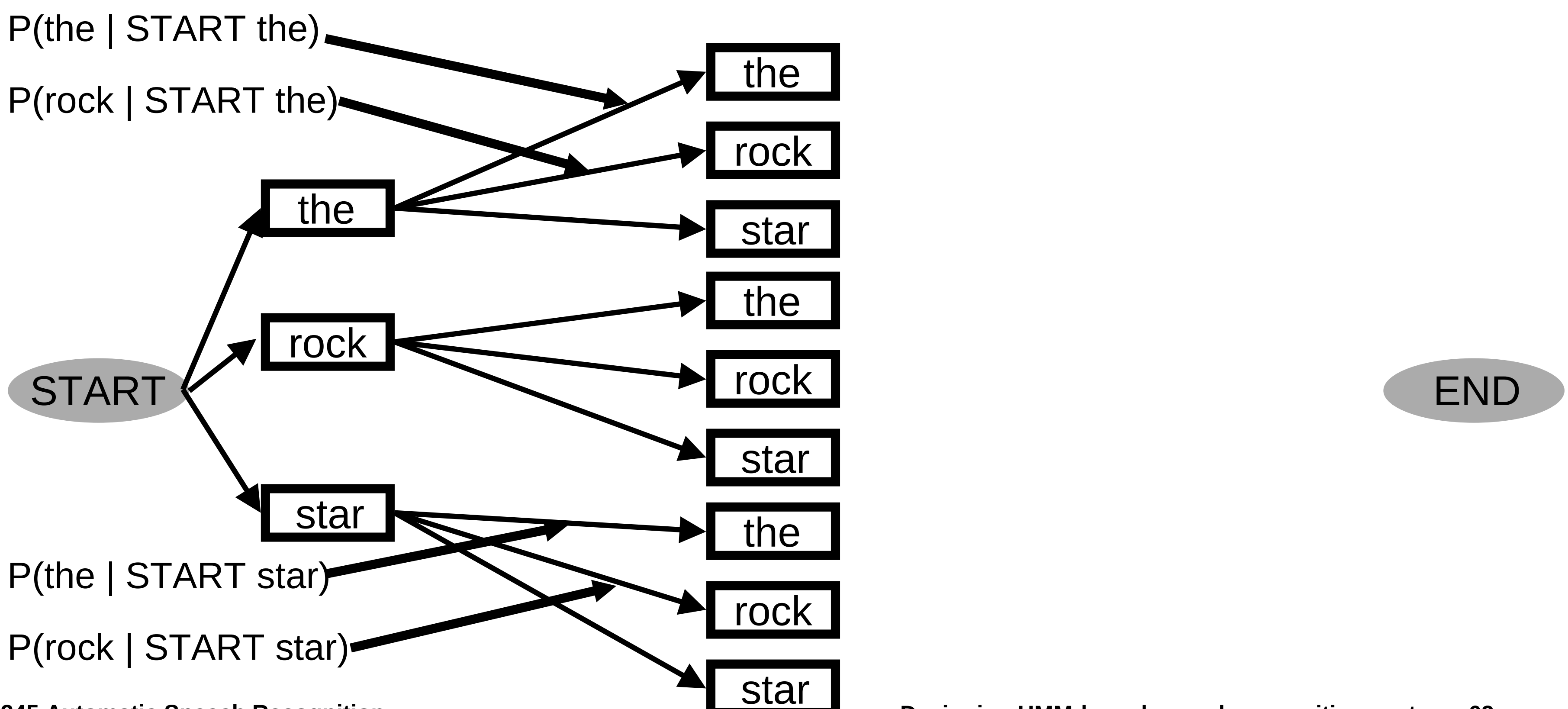
- Explanation with example: three word vocabulary “the”, “rock”, “star”
  - The graph initially begins with bigrams of START, since nothing precedes START
  - Trigrams of “START the”..



This is wrong! This would apply the probability P(the | START the) to instances of “the the the” (for which the correct probability value is P(the | the the)

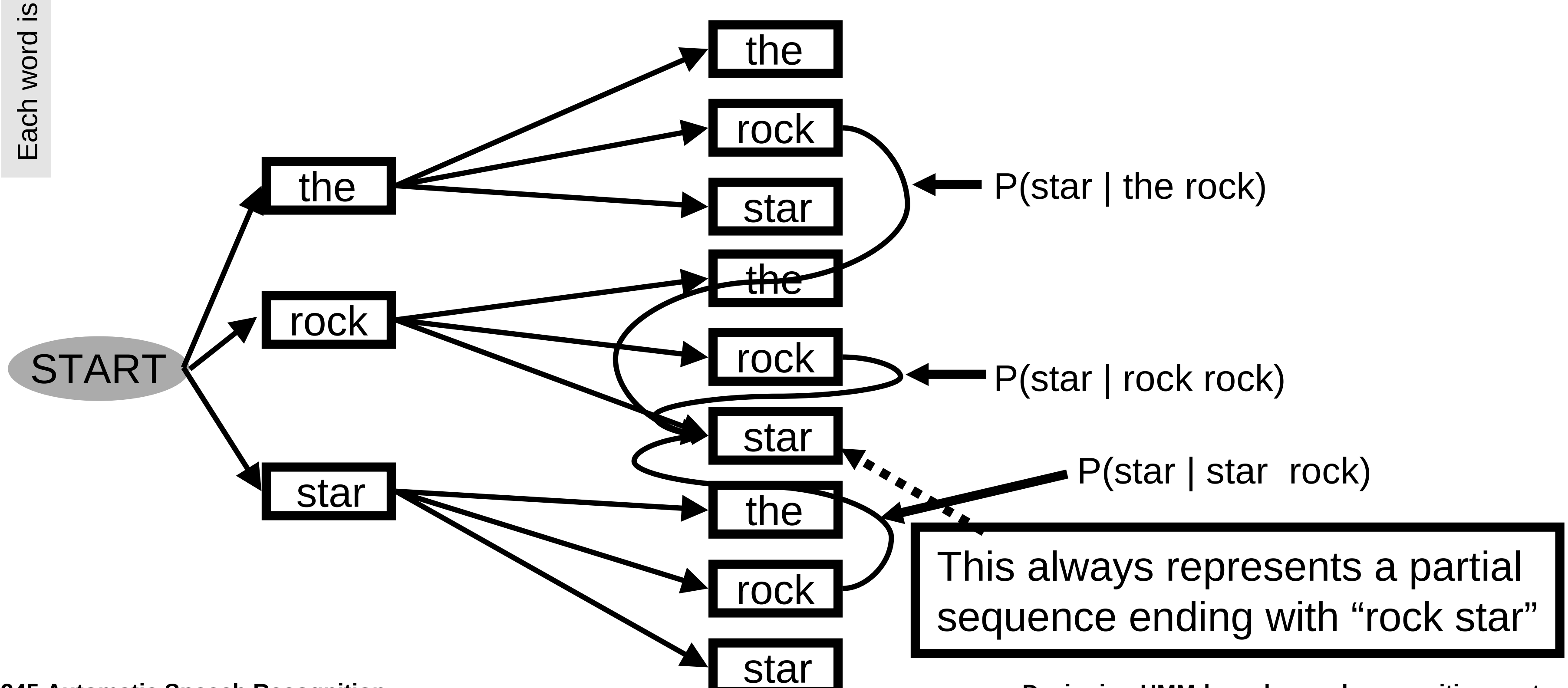
Language HMMs for Natural language: building graphs to incorporate trigram representations

- Explanation with example: three words vocab “the”, “rock”, “star”
  - The graph initially begins with bigrams of START, since nothing precedes START
  - Trigrams for all “START word” sequences
    - A new instance of every word is required to ensure that the two preceding symbols are “START word”



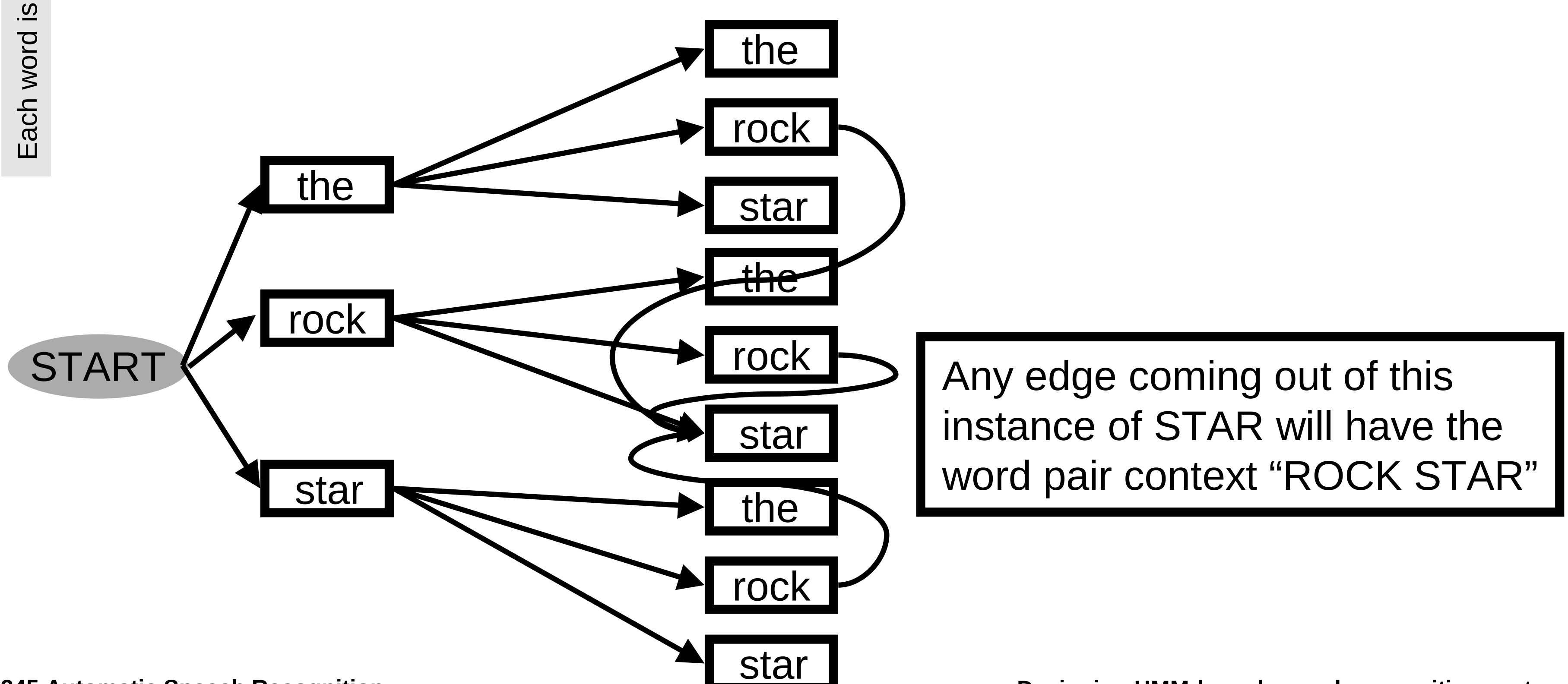
Language HMMs for Natural language: building graphs to incorporate trigram representations

- Explanation with example: three words vocab “the”, “rock”, “star”
  - The graph initially begins with bigrams of START, since nothing precedes START
  - Each word in the second level represents a specific set of two terminal words in a partial word sequence



Language HMMs for Natural language: building graphs to incorporate trigram representations

- Explanation with example: three words vocab "the", "rock", "star"
  - The graph initially begins with bigrams of START, since nothing precedes START
  - Each word in the second level represents a specific set of two terminal words in a partial word sequence

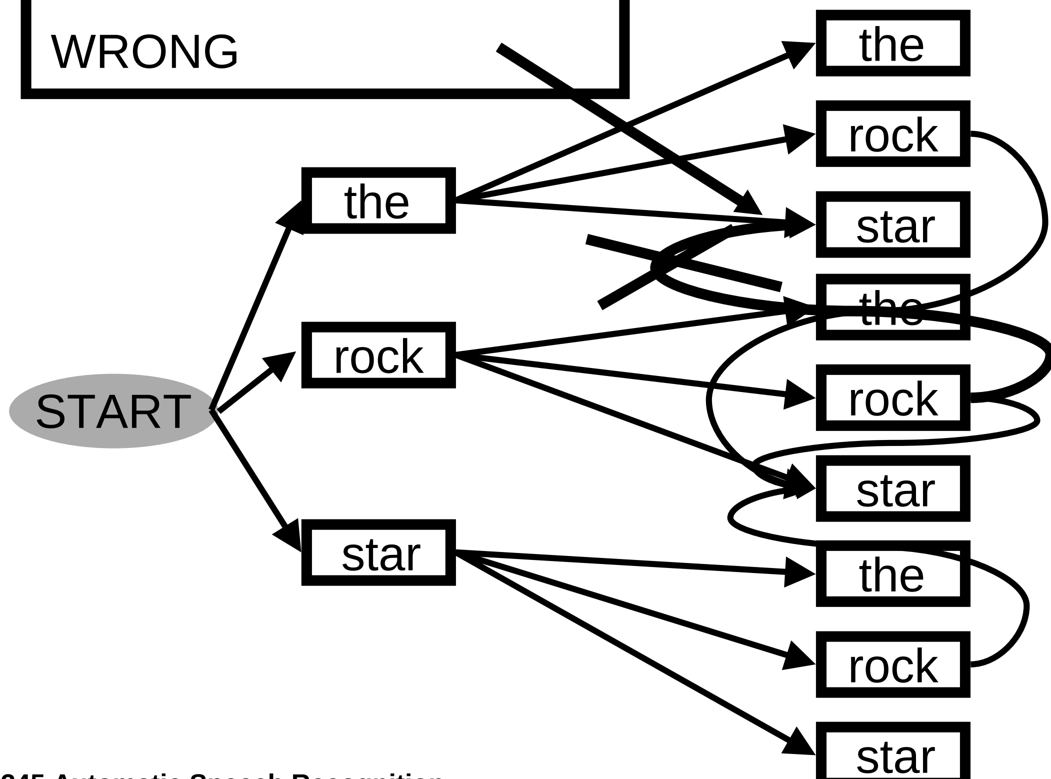


Language HMMs for Natural language: building graphs to incorporate trigram representations

- Explanation with example: three words vocab “the”, “rock”, “star”
  - The graph initially begins with bigrams of START, since nothing precedes START
  - Each word in the second level represents a specific set of two terminal words in a partial word sequence

This STAR has input from “the” and “rock”.

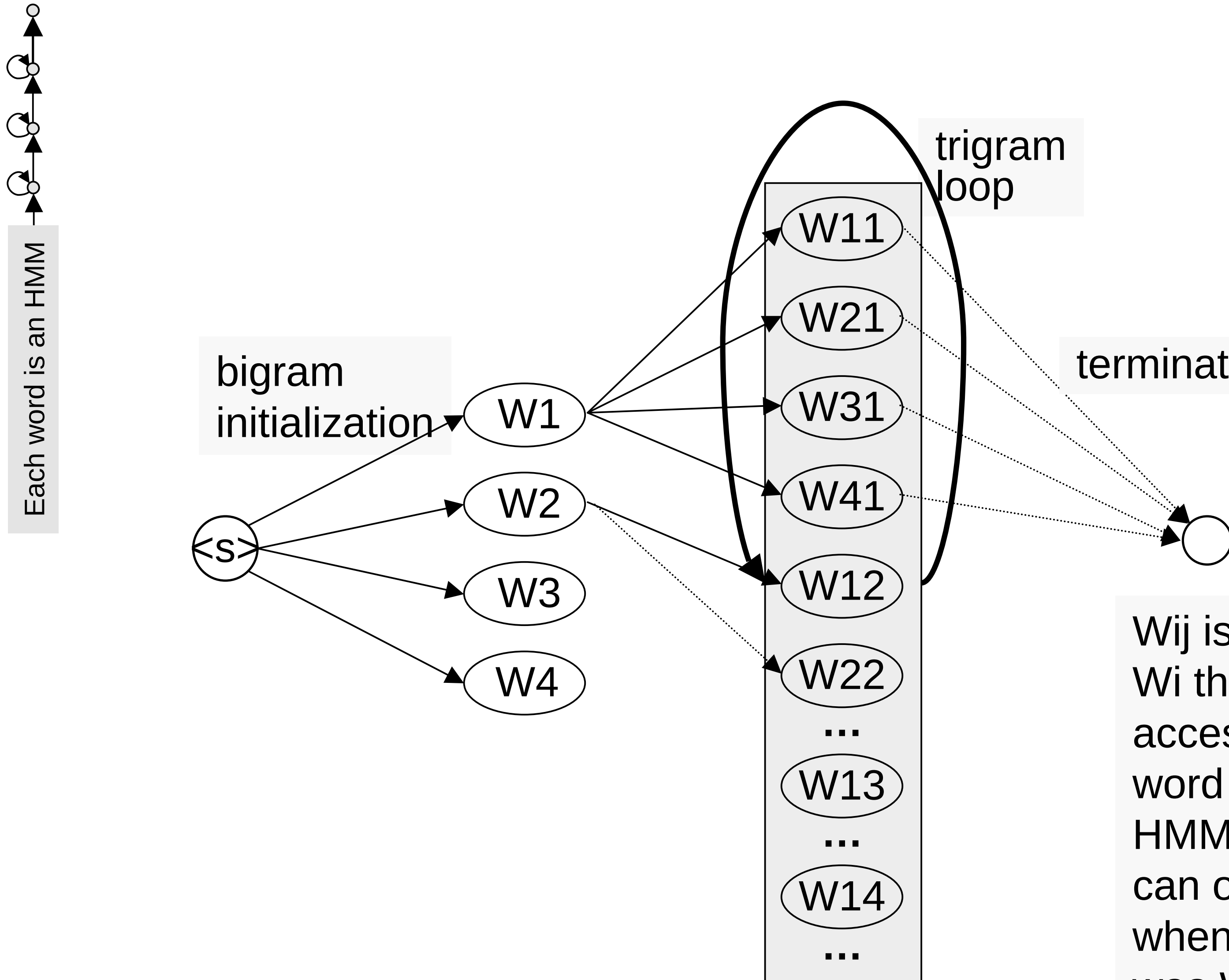
WRONG



Edges coming out of this wrongly connected STAR could have word pair contexts that are either “THE STAR” or “ROCK STAR”.

This is amibiguous. A word cannot have incoming edges from two or more different words

Language HMMs for Natural language: building graphs to incorporate trigram representations



Wij is an HMM for word Wi that can only be accessed from HMMs for word Wj. E.g. W12 is the HMM for word W1 that can only be used when the previous word was W2

Language HMMs for Natural language: Generic N-gram representations

- The logic can be extended
- A trigram decoding structure for a vocabulary of D words needs D word instances at the first level and D^2 word instances at the second level
  - Total of D(D+1) word models must be instantiated
  - Other, more expensive structures are also possible
- An N-gram decoding structure will need
  - $D + D^2 + D^3 \ldots D^{N-1}$ word instances
  - Arcs must be incorporated such that the exit from a word instance in the $(N-1)^{th}$ level always represents a word sequence with the same trailing sequence of N-1 words

Next Class

- How the use of sub-word units complicates the structure of language HMMs in decoders