



Joint Institute  
of Engineering

SUN YAT-SEN UNIVERSITY

Carnegie Mellon University

# Design and Implementation of Speech Recognition Systems

*Fall 2014*

*Ming Li*

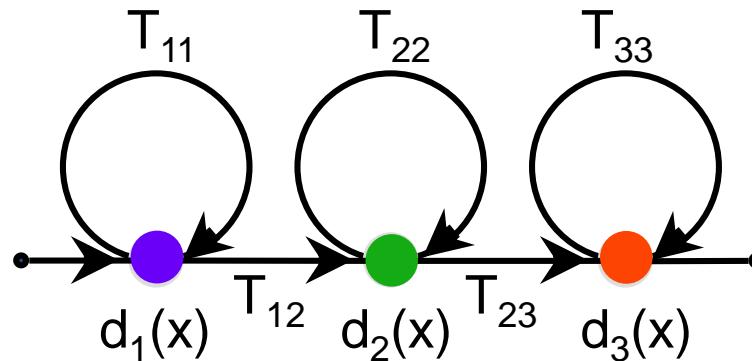
Class 10: Grammars

Oct 30th

*Thanks to Professor Bhiksha Raj for the contribution of the slides*

# Recap: HMMs are Generalized Templates

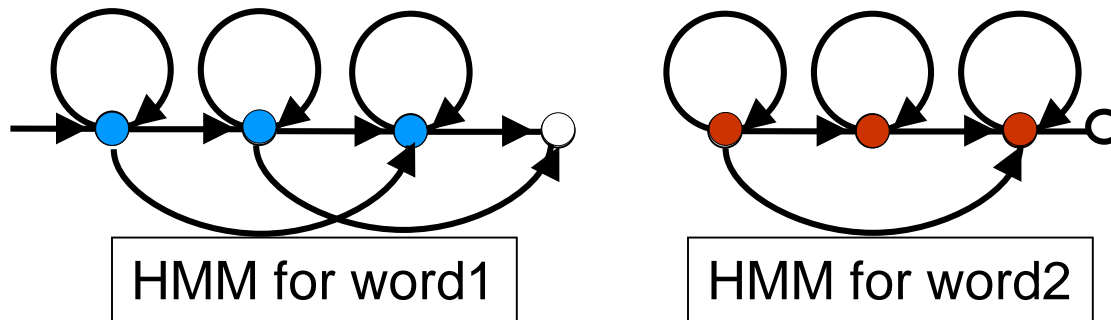
---



- A set of “states”
  - A distance function associated with each state
- A set of transitions
  - Transition-specific penalties

# Recap: Isolated word recognition with HMMs

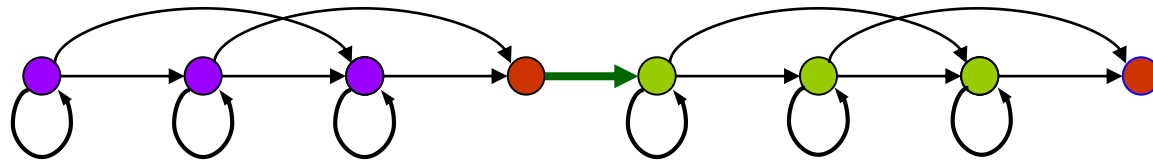
---



- An HMM for each word
- Score incoming speech against each HMM
- Pick word whose HMM scores best
  - Best == lowest cost
  - Best == highest score
  - Best == highest probability

# Recap: Recognizing word sequences

---

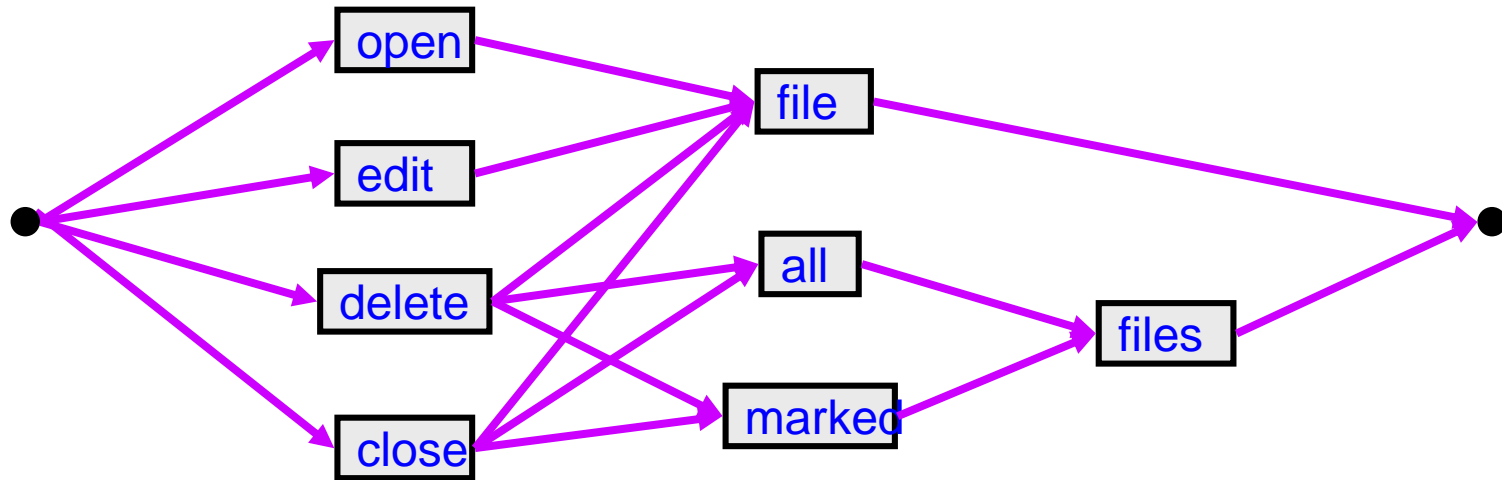


Combined HMM for the sequence **word 1** **word 2**

- Train HMMs for words
- Create HMM for each word sequence
  - Recognize as in isolated word case

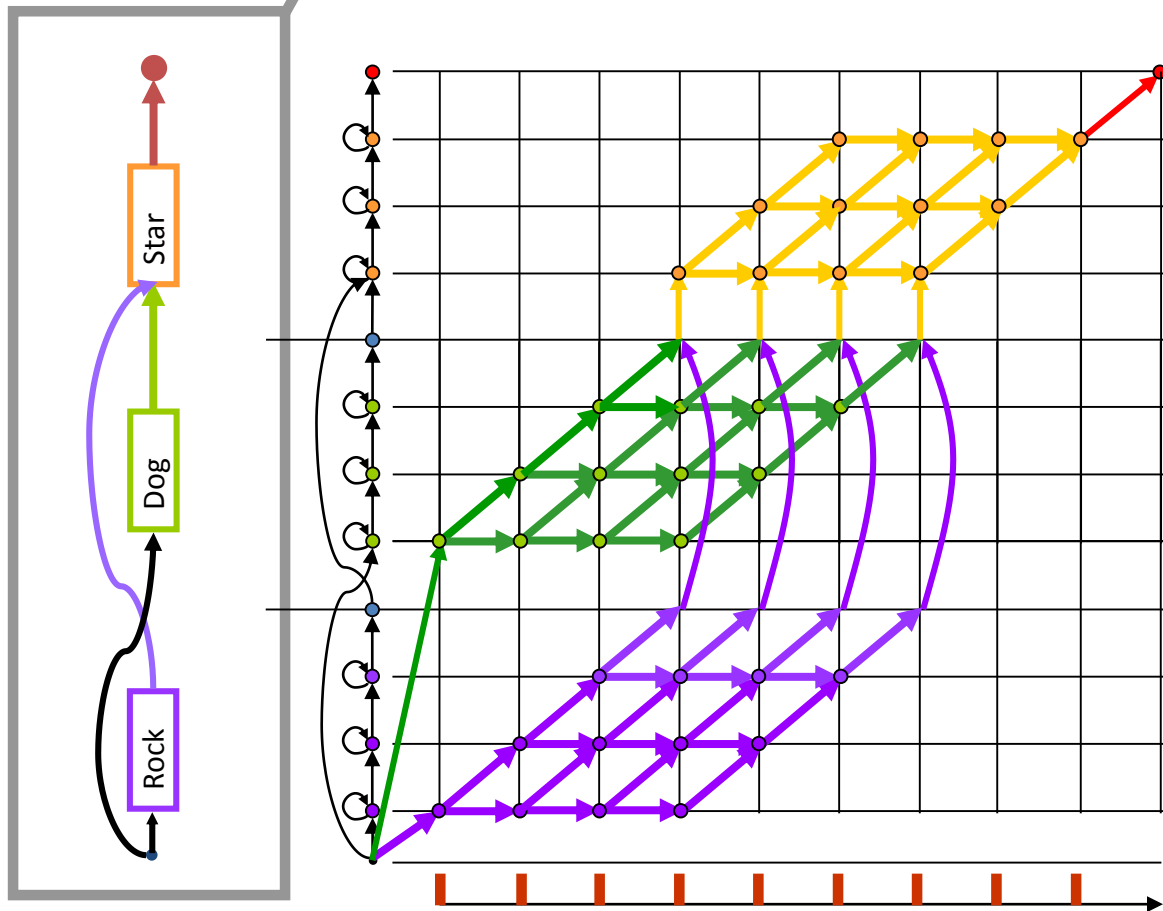
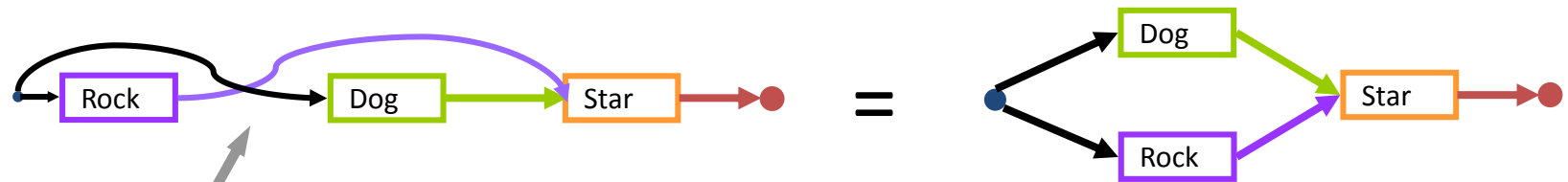
# Recap: Recognizing word sequences

---



- Create word graph HMM representing all word sequences
  - Word sequence obtained from best state sequence

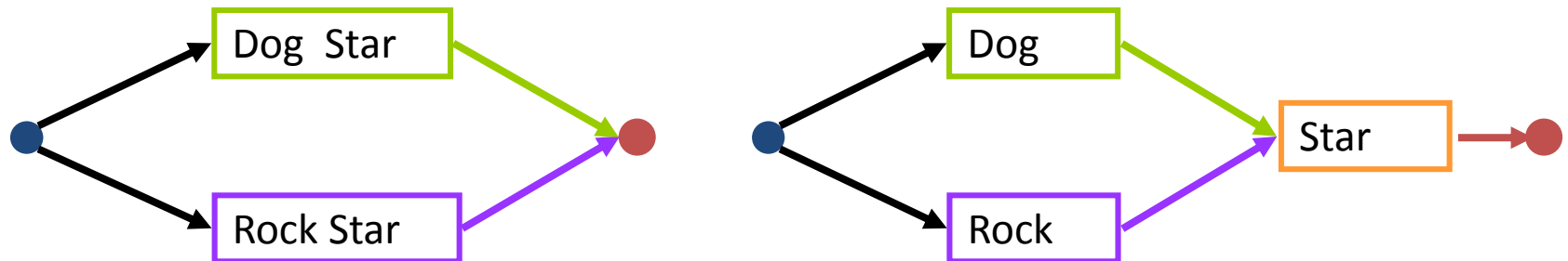
# Language-HMMs for fixed length word sequences



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

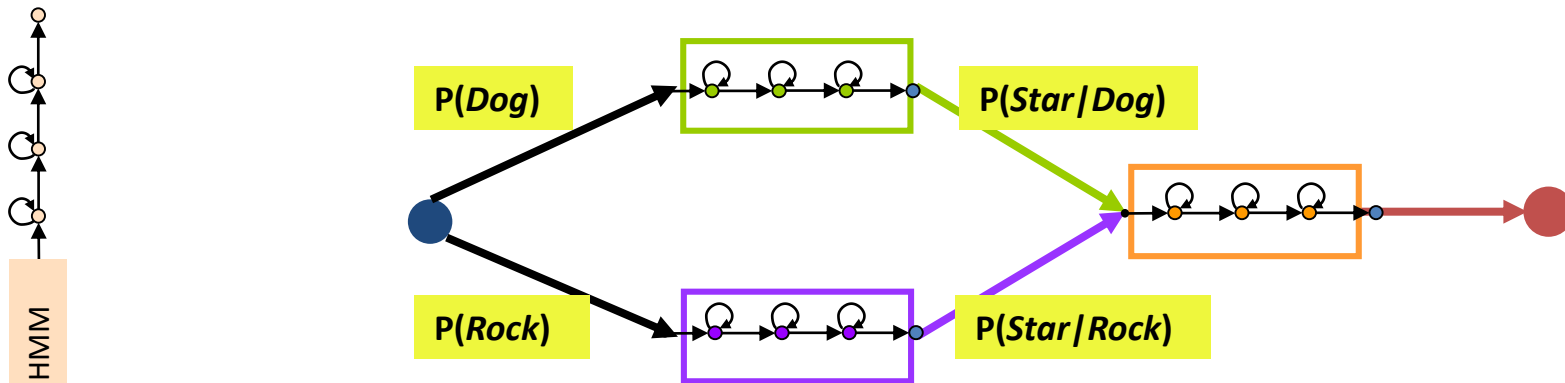
# The Real “Classes”

---



- The actual recognition is DOG STAR vs. ROCK STAR
  - i.e. the two items that form our “classes” are entire phrases
- The reduced graph to the right is merely an engineering reduction obtained by utilizing commonalities in the two phrases (STAR)
  - **Only possible because we use the best path score and not the entire forward probability**
- This distinction affects the design of the recognition system

# 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

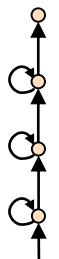
- Recognizing one of four lines from “charge of the light brigade”

Cannon to right of them

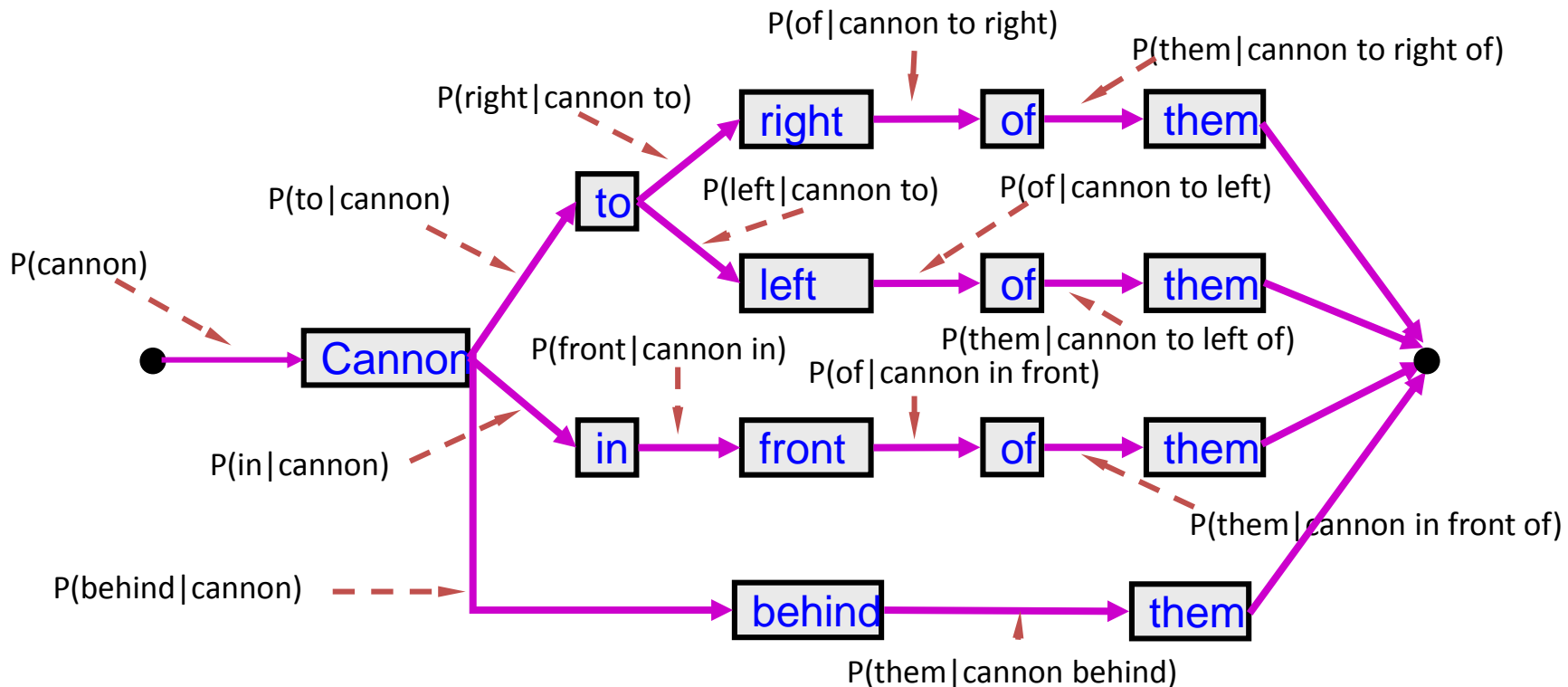
Cannon to left of them

Cannon in front of them

Cannon behind them



Each word is an HMM



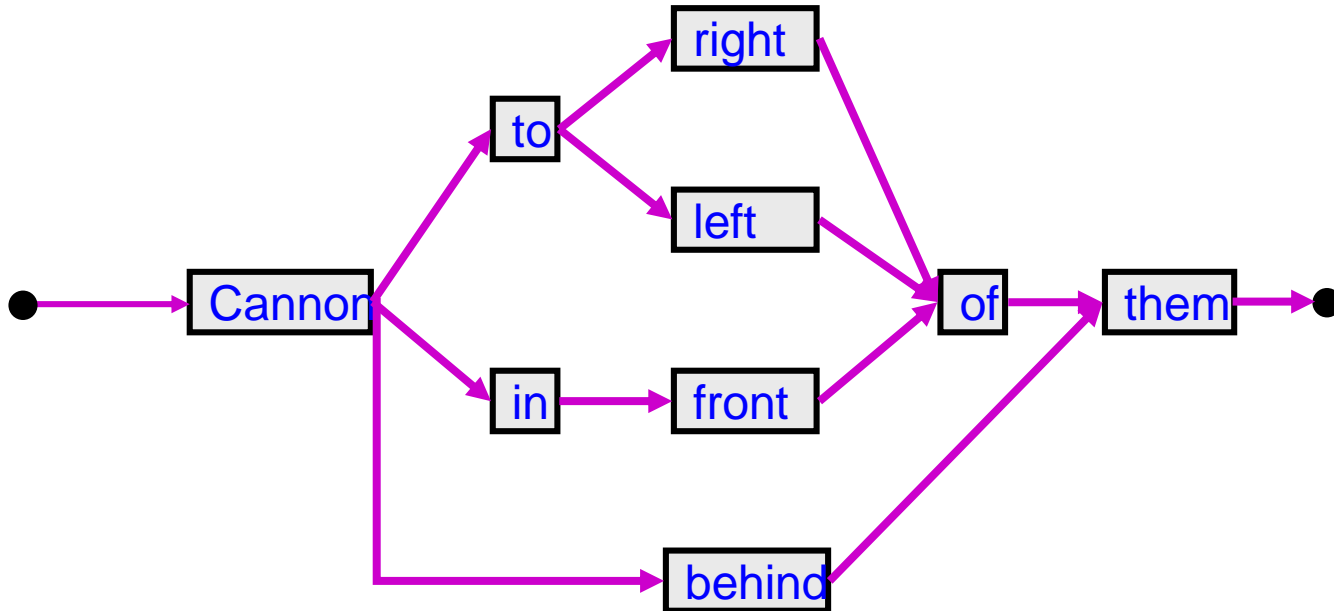
# Where does the graph come from

---

- The graph must be specified to the recognizer
  - What we are actually doing is to specify the complete set of “allowed” sentences in graph form
- May be specified as an FSG or a Context-Free Grammar
  - CFGs and FSG do not have probabilities associated with them
  - We could factor in prior biases through *probabilistic* FSG/CFGs
  - In probabilistic variants of FSGs and CFGs we associate probabilities with options
    - E.g. in the last graph

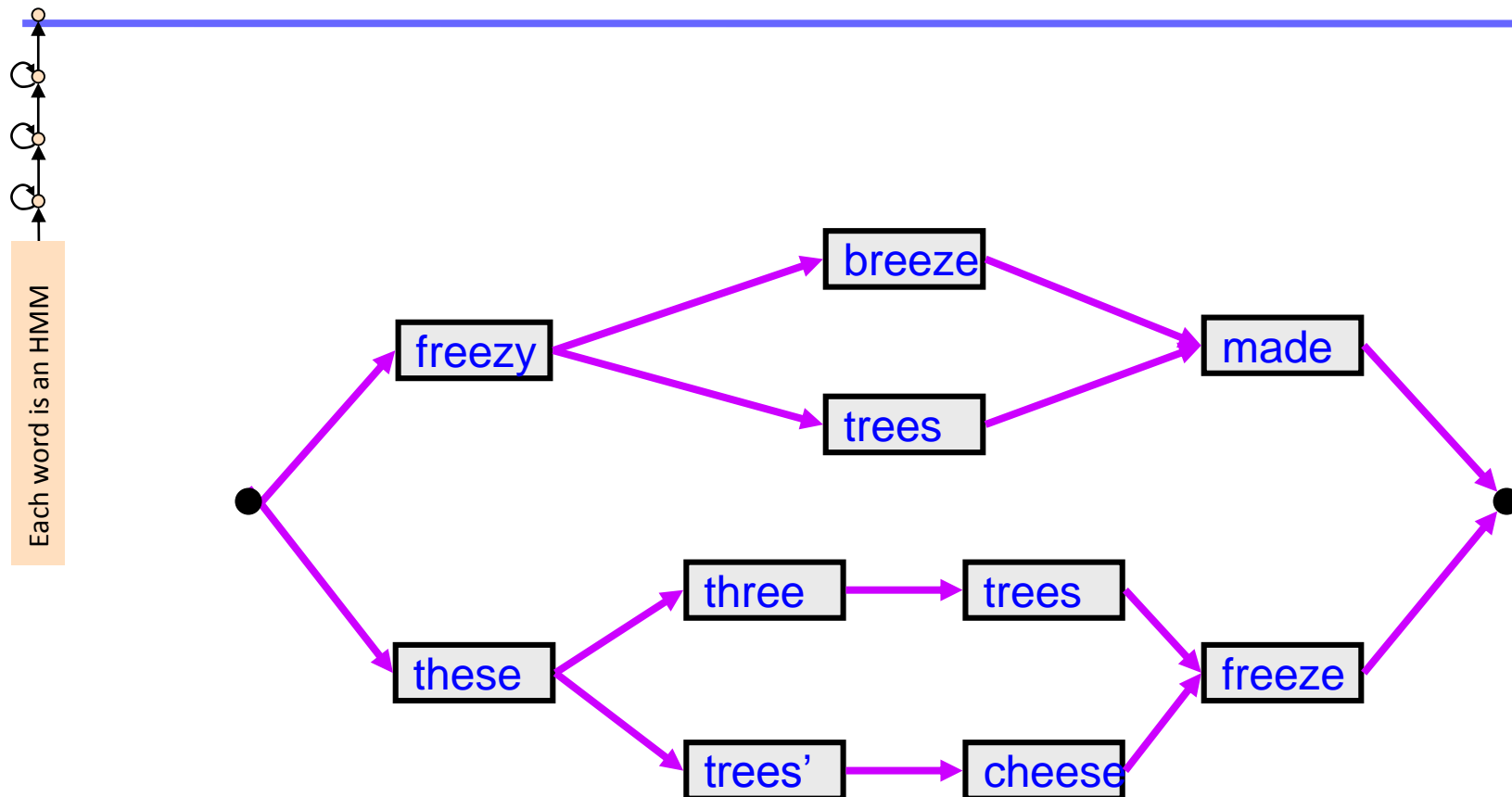
# Simplification of the language HMM through lower context language models

- Recognizing one of four lines from “charge of the light brigade”
- If we do not associate probabilities with FSG rules/transitions



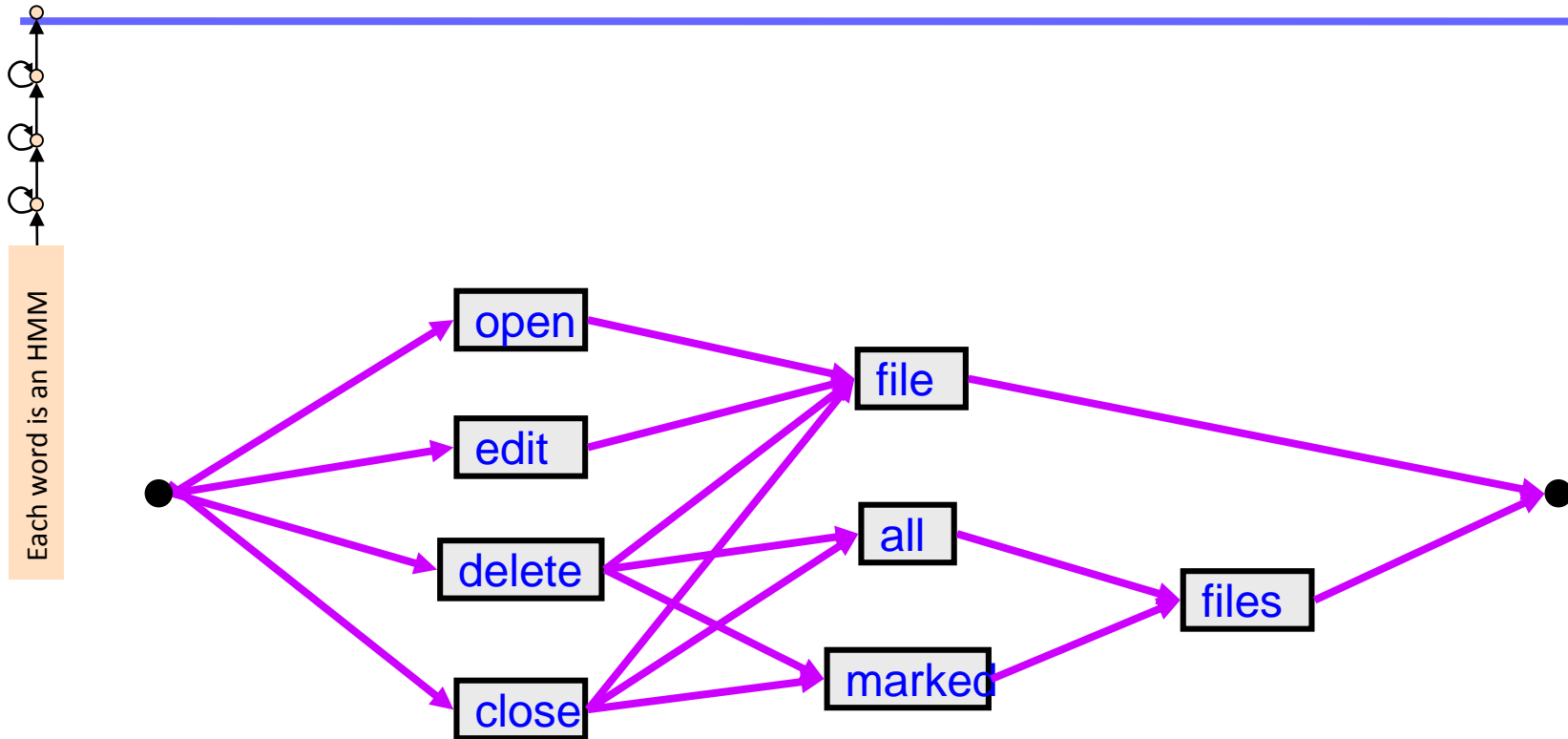
Each word is an HMM

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



No probabilities specified – a person may utter any of these phrases at any time

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



**No probabilities specified – a person may utter any of these phrases at any time**

# 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**

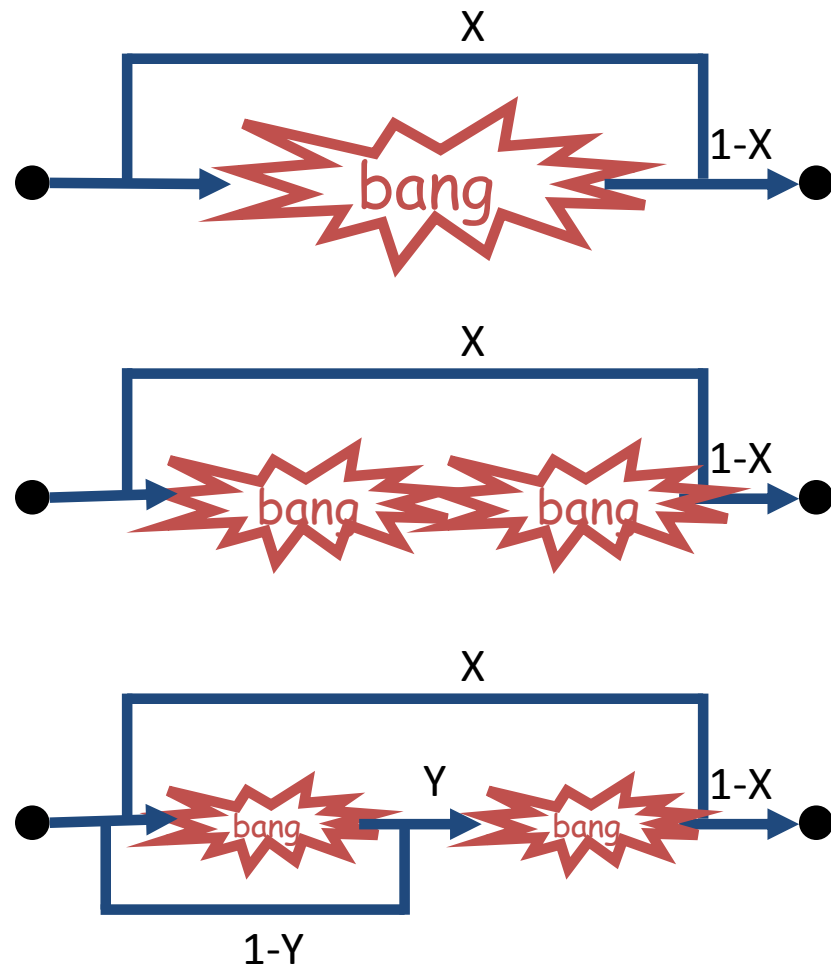
**bang bang bang bang**

.....

- 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

- Each word is an HMM
- Arbitrary word sequences can be modeled with loops under some assumptions. E.g.:
  - A “bang” can be followed by another “bang” with probability  $P(\text{“bang”})$ .
    - $P(\text{“bang”}) = X$ ;
    - $P(\text{Termination}) = 1-X$ ;
  - 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 other words in addition to “bang”
    - e.g. “bang bang you’re dead”



# Motivation

---

- So far, we have looked at speech recognition without worrying about *language structure*
  - i.e. we've treated all word sequences as being equally plausible
  - But this is rarely the case
- Using language knowledge is crucial for recognition accuracy
  - Humans use a tremendous amount of *context* to “fill in holes” in what they hear, and to disambiguate between confusable words
  - Speech recognizers should do so too!
- Such knowledge used in a decoder is called a *language model* (LM)



# Impact of Language Models on ASR

---

- Example with a 20K word vocabulary system:
  - Without an LM (“any word is equally likely” model):

AS COME ADD TAE ASIAN IN THE ME AGE OLE FUND IS MS. GROWS INCREASING ME IN  
TENTS MAR PLAYERS AND INDUSTRY A PAIR WILLING TO SACRIFICE IN TAE GRITTY IN THAN  
ANA IF PERFORMANCE

- With an appropriate LM (“knows” what word sequences make sense):

AS COMPETITION IN THE MUTUAL FUND BUSINESS GROWS INCREASINGLY INTENSE MORE  
PLAYERS IN THE INDUSTRY APPEAR WILLING TO SACRIFICE INTEGRITY IN THE NAME OF  
PERFORMANCE

# Syntax and Semantics

---

- However, human knowledge about context is far too rich to capture in a formal model
  - In particular, humans rely on *meaning*
- Speech recognizers only use models relating to word sequences
  - *i.e.* focus on *syntax* rather than *semantics*

# Importance of Semantics

---

- From *Spoken Language Processing*, by Huang, Acero and Hon:
  - Normal language, 5K word vocabulary:
    - ASR: 4.5% word error rate (WER)
    - Humans: 0.9% WER
  - Synthetic language generated from a *trigram* LM, 20K word vocabulary:
    - Example: *BECAUSE OF COURSE AND IT IS IN LIFE AND ...*
    - ASR: 4.4% WER
    - Humans: 7.6% WER
- Still, we will focus only on the syntactic level

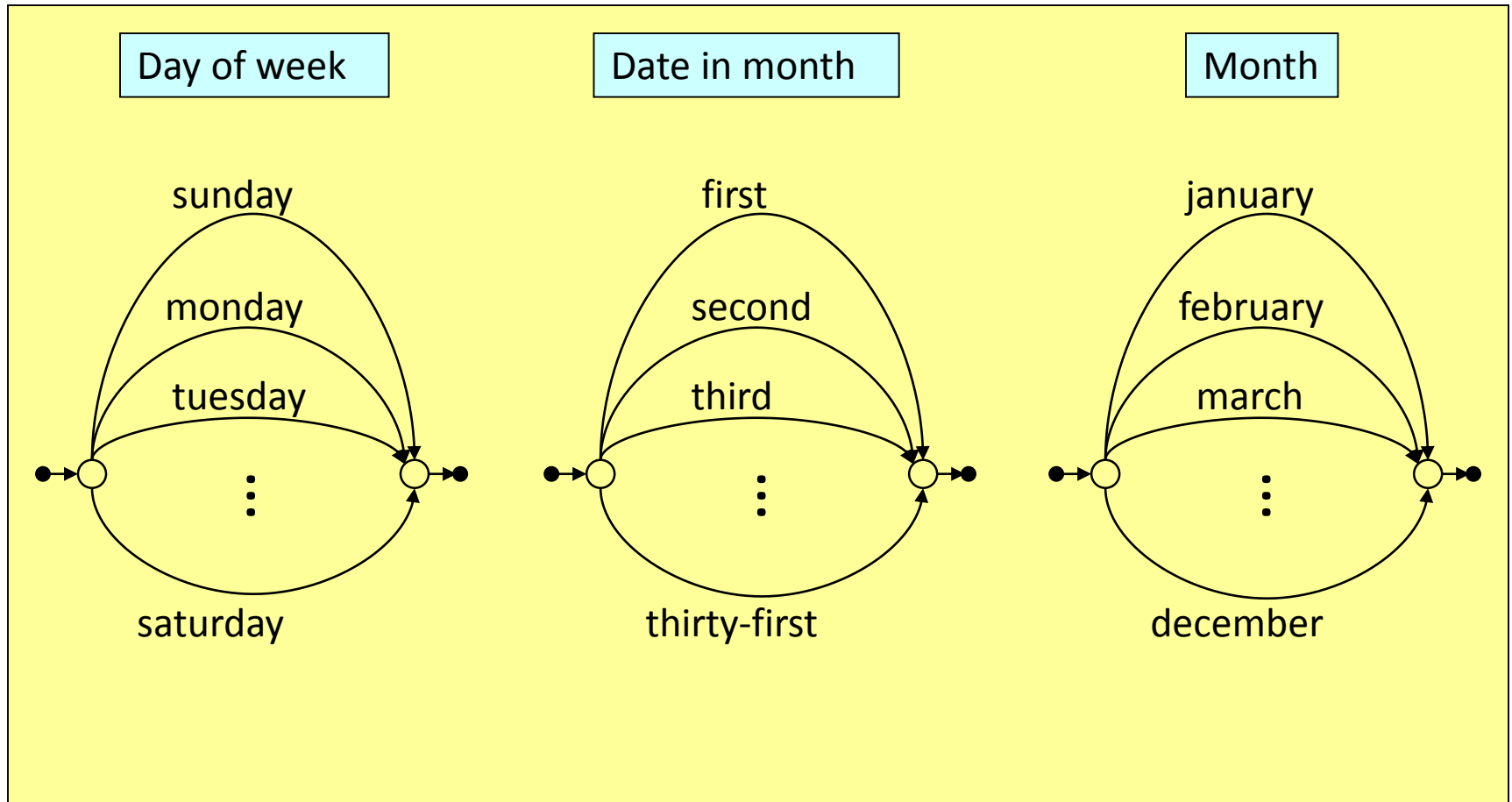
# Types of LMs

---

- We will use *grammars* or LMs to constrain the search algorithm
- This gives the decoder a *bias*, so that not all word sequences are equally likely
- Our topics include:
  - *Finite state* grammars (FSGs)
  - *Context free* grammars (CFGs)
  - Decoding algorithms using them
- These are suitable for small/medium vocabulary systems, and highly structured systems
- For large vocabulary applications, we use *N-gram* LMs, which will be covered later

# Finite State Grammar Examples

- Three simple finite state grammars (FSGs):



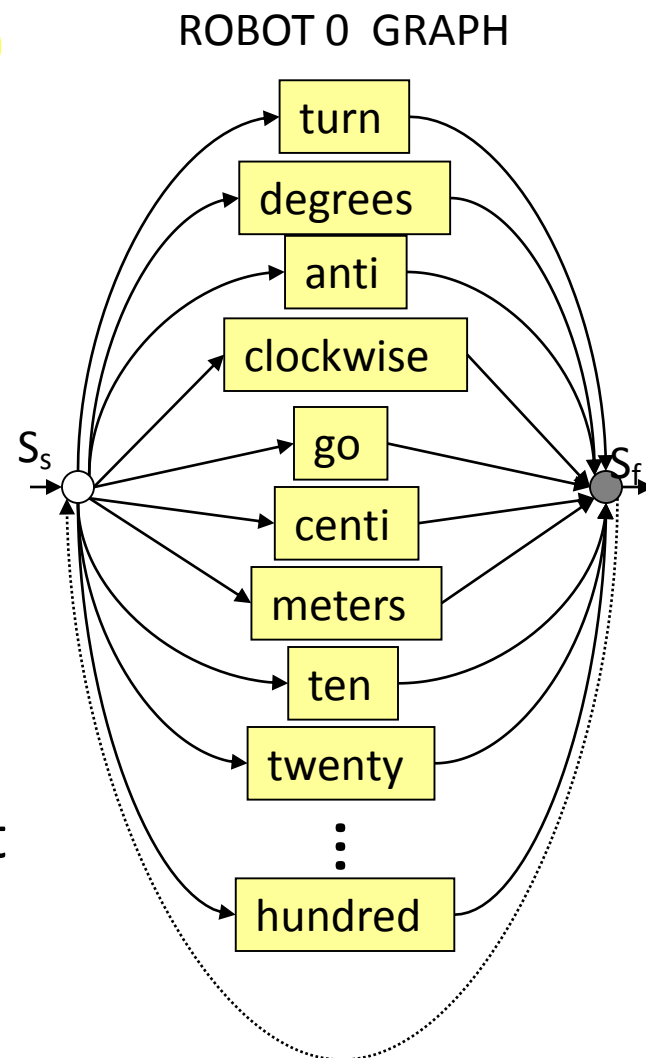
# A More Complex Example

---

- A robot control application:
  - TURN 10 DEGREES CLOCKWISE
  - TURN 30 DEGREES ANTI CLOCKWISE
  - GO 10 METERS
  - GO 50 CENTI METERS
  - Allowed angles: 10 20 30 40 50 60 70 80 90 (clk/anticlk)
  - Allowed distances: 10 20 30 40 50 60 70 80 90 100 (m/cm)
- Vocabulary of this application = 17 words:
  - TURN DEGREES CLOCKWISE ANTI  
GO METERS CENTI and TEN TWENTY ... HUNDRED
  - Assume we have word HMMs for all 17 words
- How can we build a continuous speech recognizer for this application?

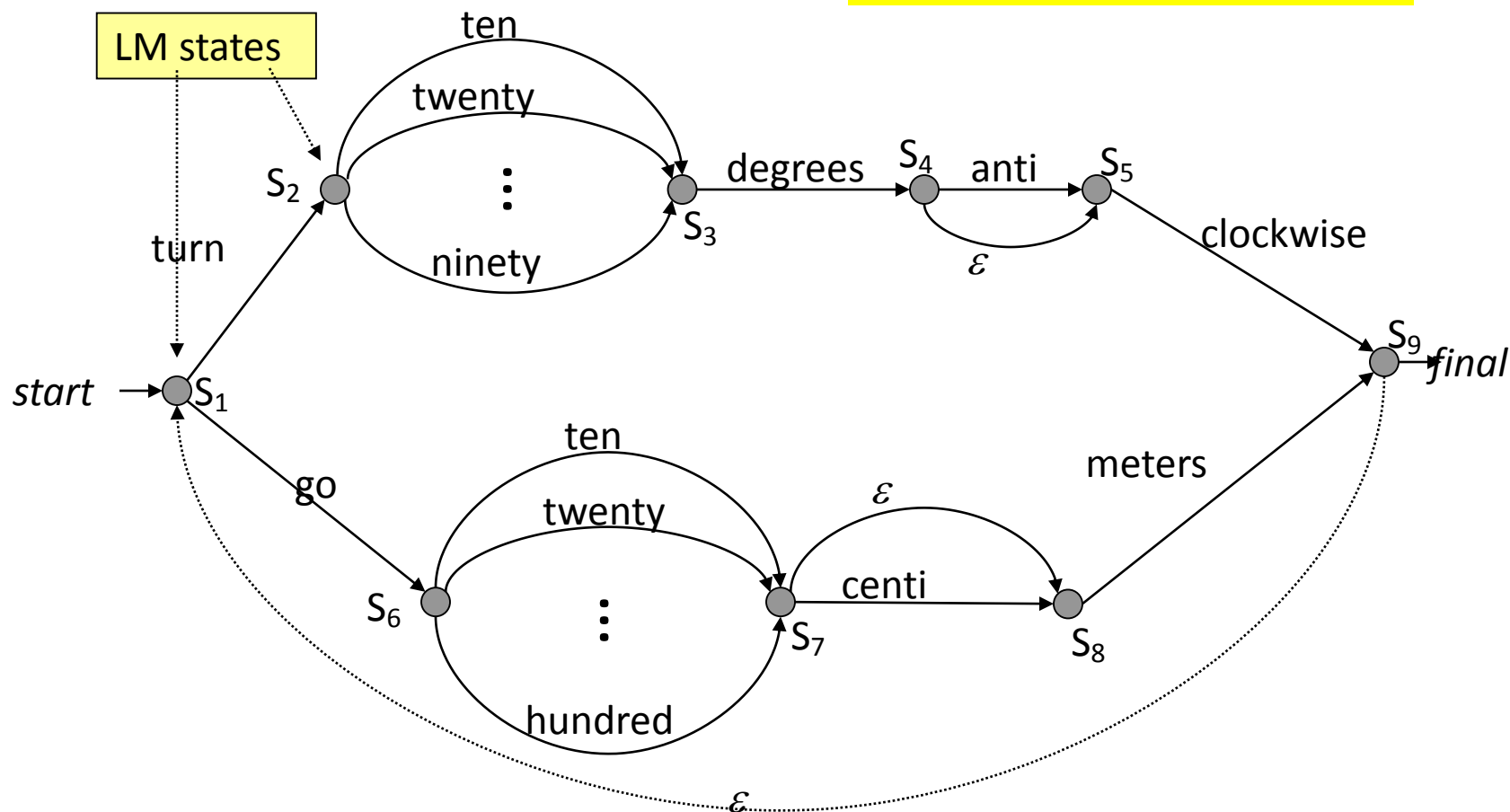
# A More Complex Example

- One possibility: Build an “any word can follow any word” sentence HMM using the word HMMs
- Allows many word sequences that simply do not make any sense!
  - The recognizer would search through many meaningless paths
  - Greater chance of misrecognitions
- Must *tell* the system about the *legal* set of sentences
- We do this using an FSG



# Robot Control FSG (ROBOT1)

NOTE: WORDS ARE ON EDGES





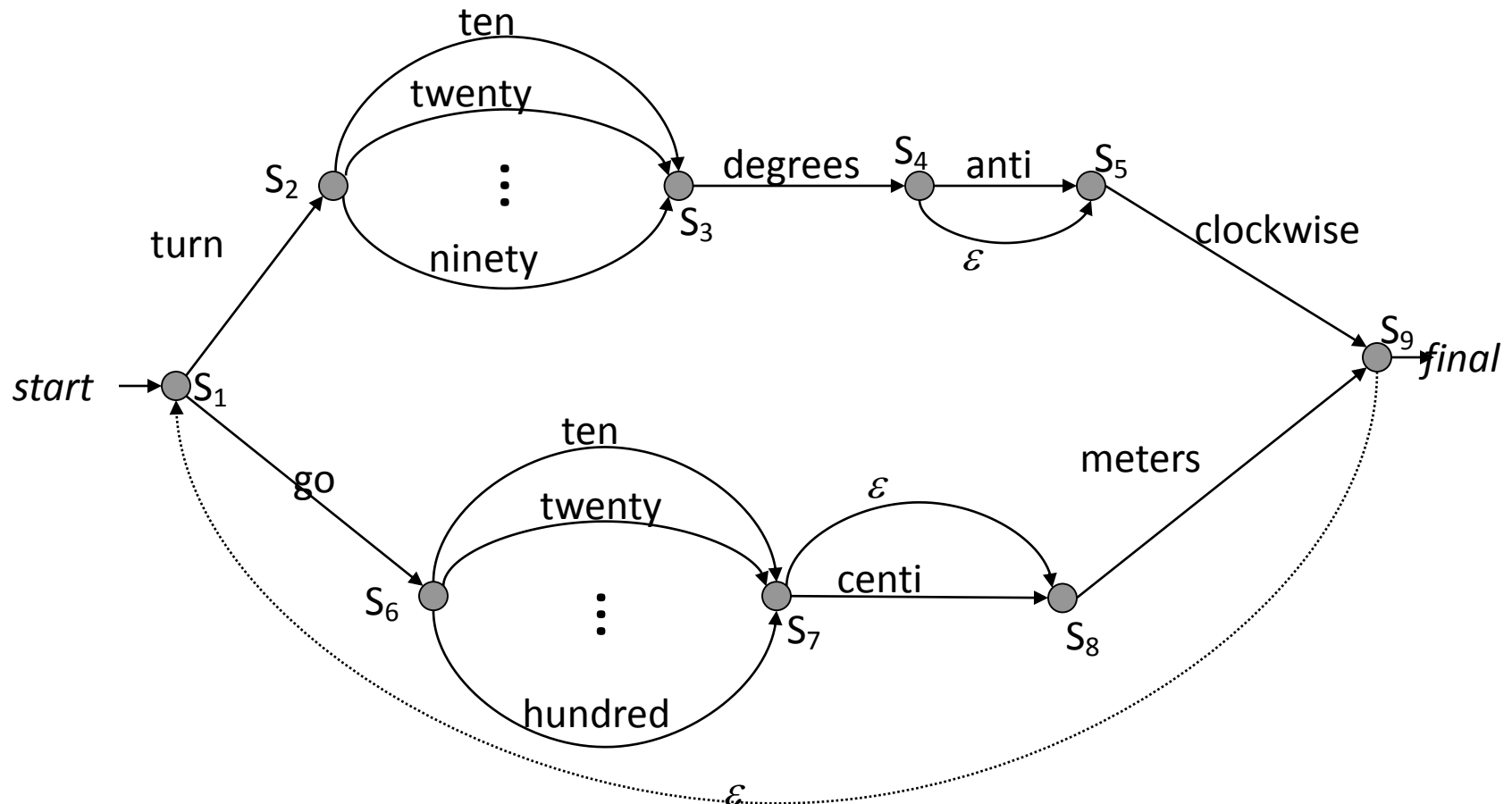
# Elements of Finite State Grammars

---

- FSGs are defined by the following (very much like HMMs):
  - A *finite* set of states
    - These are generically called *LM states*
  - One or more of the states are *initial* or *start* states
  - One or more of the states are *terminal* or *final* states
  - *Transitions* between states, *optionally* labeled with words
    - The words are said to be *emitted* by those transitions
    - Unlabelled transitions are called *null* or *e* transitions
  - PFSG: Transitions have probabilities associated with them, as usual
    - All transitions out of a state without an explicit transition probability are assumed to be equally likely
- Any *path* from a start state to a final state emits a legal word sequence (called a *sentence*)
- The set of all possible sentences produced by the FSG is called its *language*

# Decoding with Finite State Grammars

- How can we incorporate our *ROBOT1* FSG into the Viterbi decoding scheme?



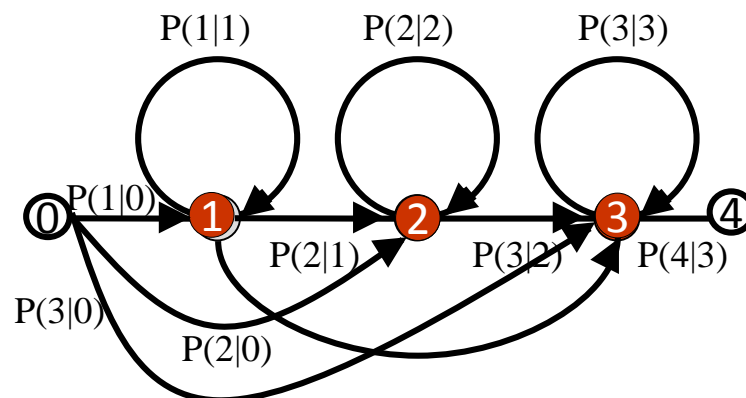
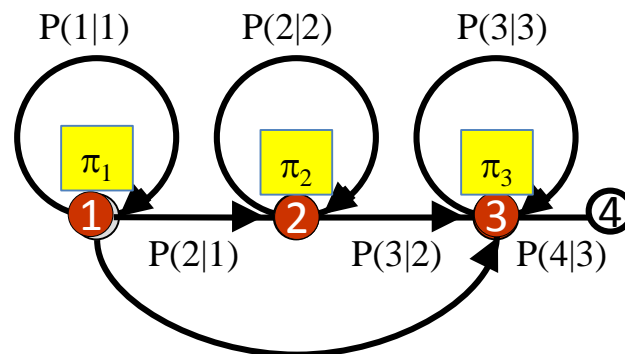
# Decoding with Finite State Grammars

---

- Construct a *language HMM* from the given FSG
  - Replace edges in the FSG with the HMMs for words
  - We are now in familiar territory
    - Apply the standard time synchronous Viterbi search
  - Only modification needed: need to distinguish between LM states (see later)
- First, how do we construct the language HMM for an FSG?

# A Note on the Start State

- We have assumed that HMMs have an initial state probability for states
  - Represented  $\pi_s$  in figure to right
- This is identical to assuming an initial non-emitting state
  - Also more convenient in terms of segmentation and training
- Specification in terms of initial non-emitting state also enables control over permitted initial states
  - By fixing the topology



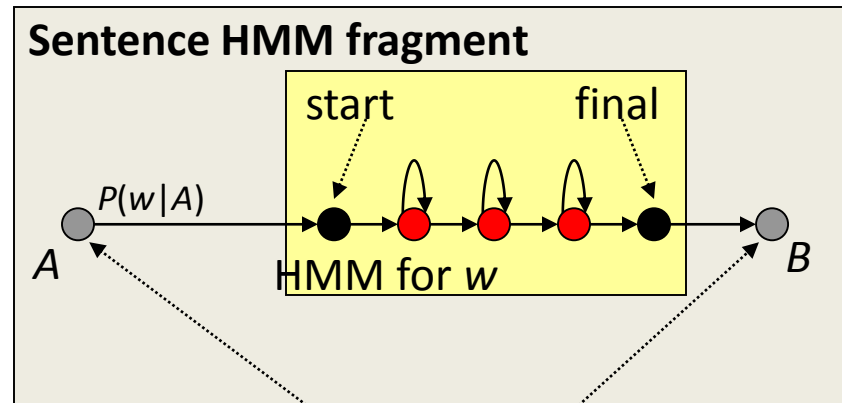
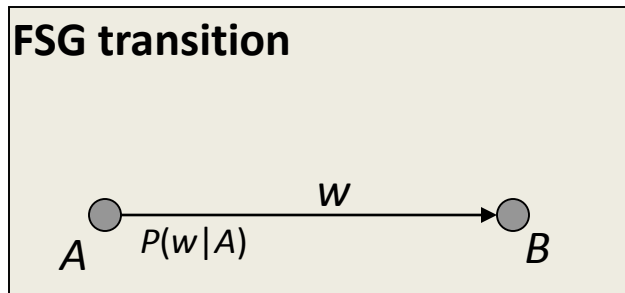
$$P(1|0) = \pi_1$$

$$P(2|0) = \pi_2$$

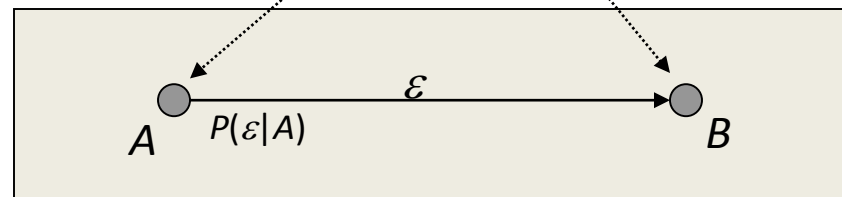
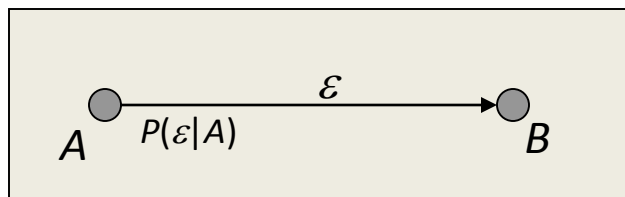
$$P(3|0) = \pi_3$$

# Language HMMs from FSGs

- To construct a language HMM, using word HMMs, we will assume each word HMM has:
  - Exactly one non-emitting start and one non-emitting final state
- Replace each non-null FSG transition by a word HMM:



Non-emitting states created to represent FSG states



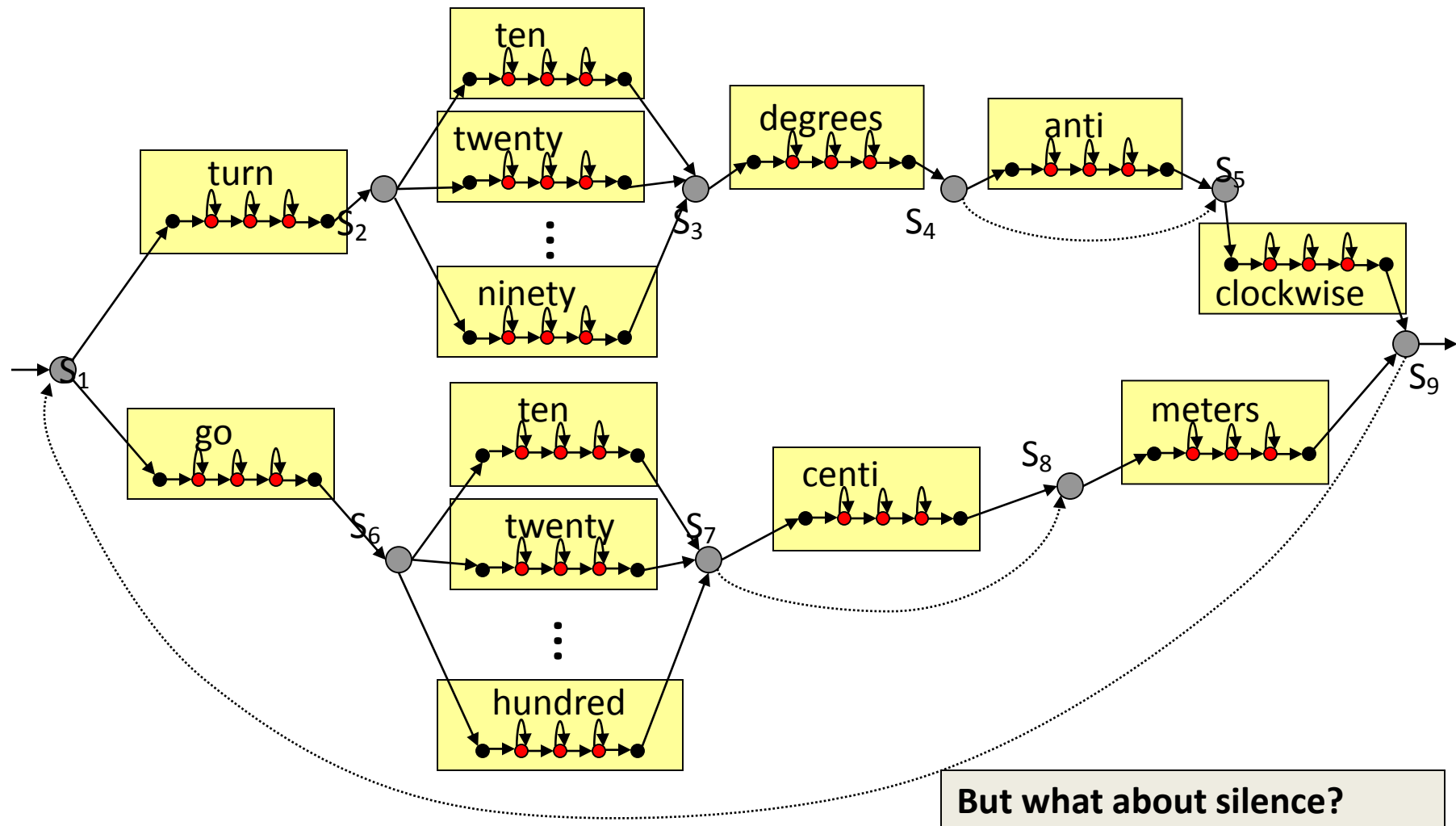
# Language HMMs from FSGs (contd.)

---

- Every FSG state becomes a non-emitting state in the language HMM
- Every *non-null* FSG transition is replaced by a word HMM as shown previously
- Start and final states of sentence HMM = start and final states of FSG

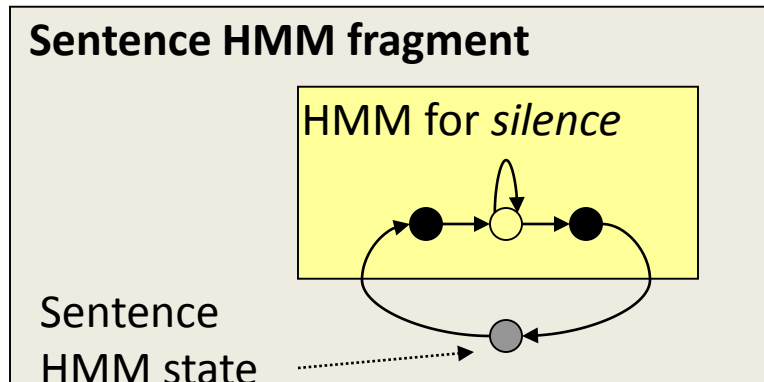
# Robot Control (*ROBOT1*) Language HMM

- The robot control FSG *ROBOT1* becomes this language HMM:



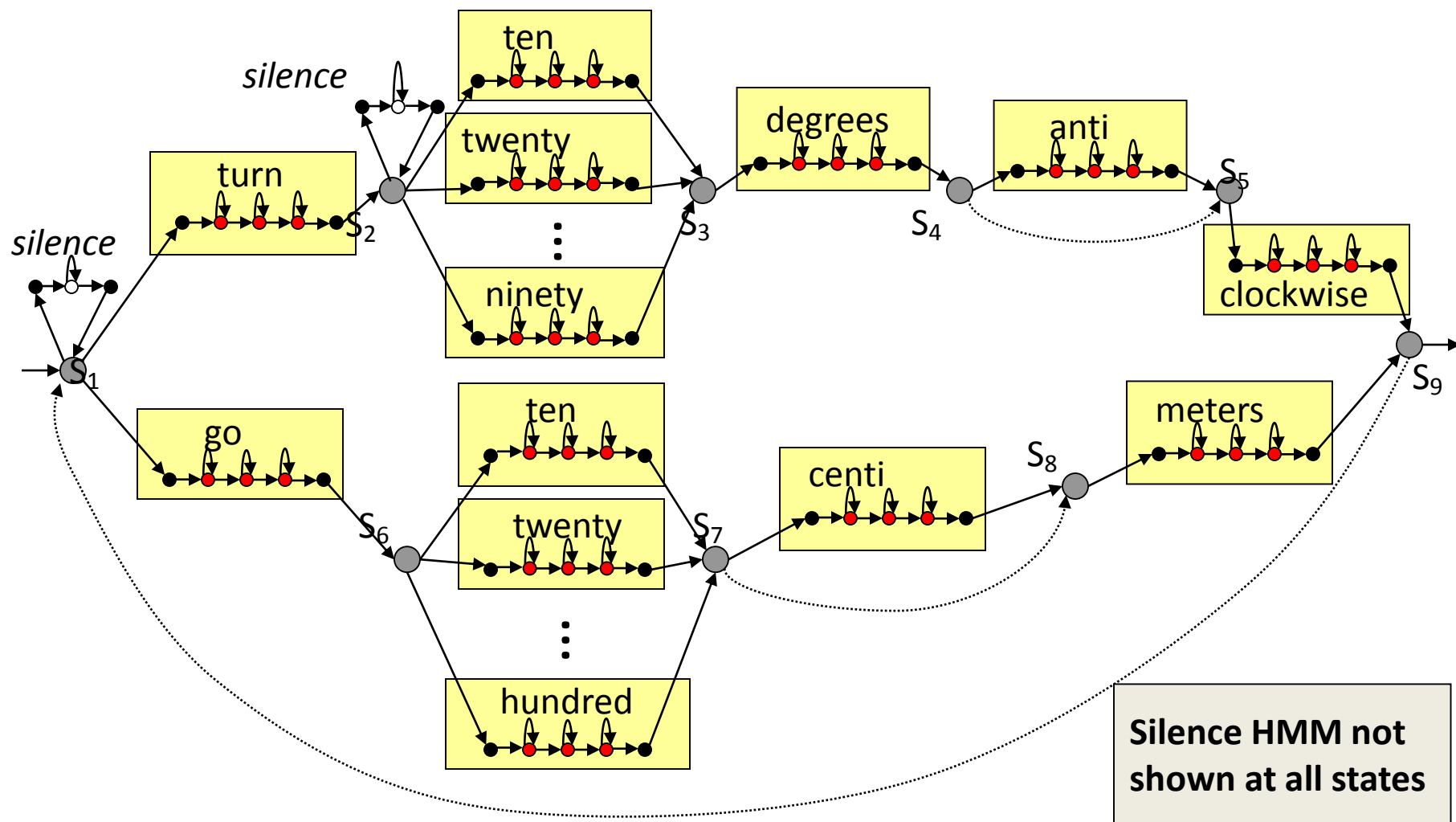
# Language HMMs from FSGs (contd.)

- People may pause between words
  - Unpredictably
- Solution: Add optional silence HMM at each sentence HMM state:



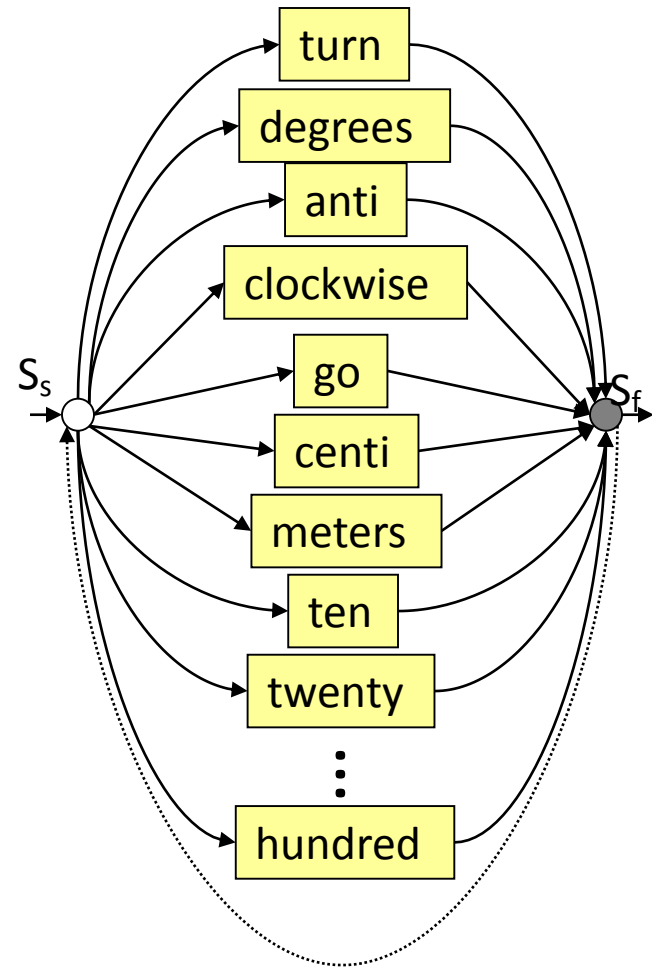


# ROBOT1 with Optional Silences

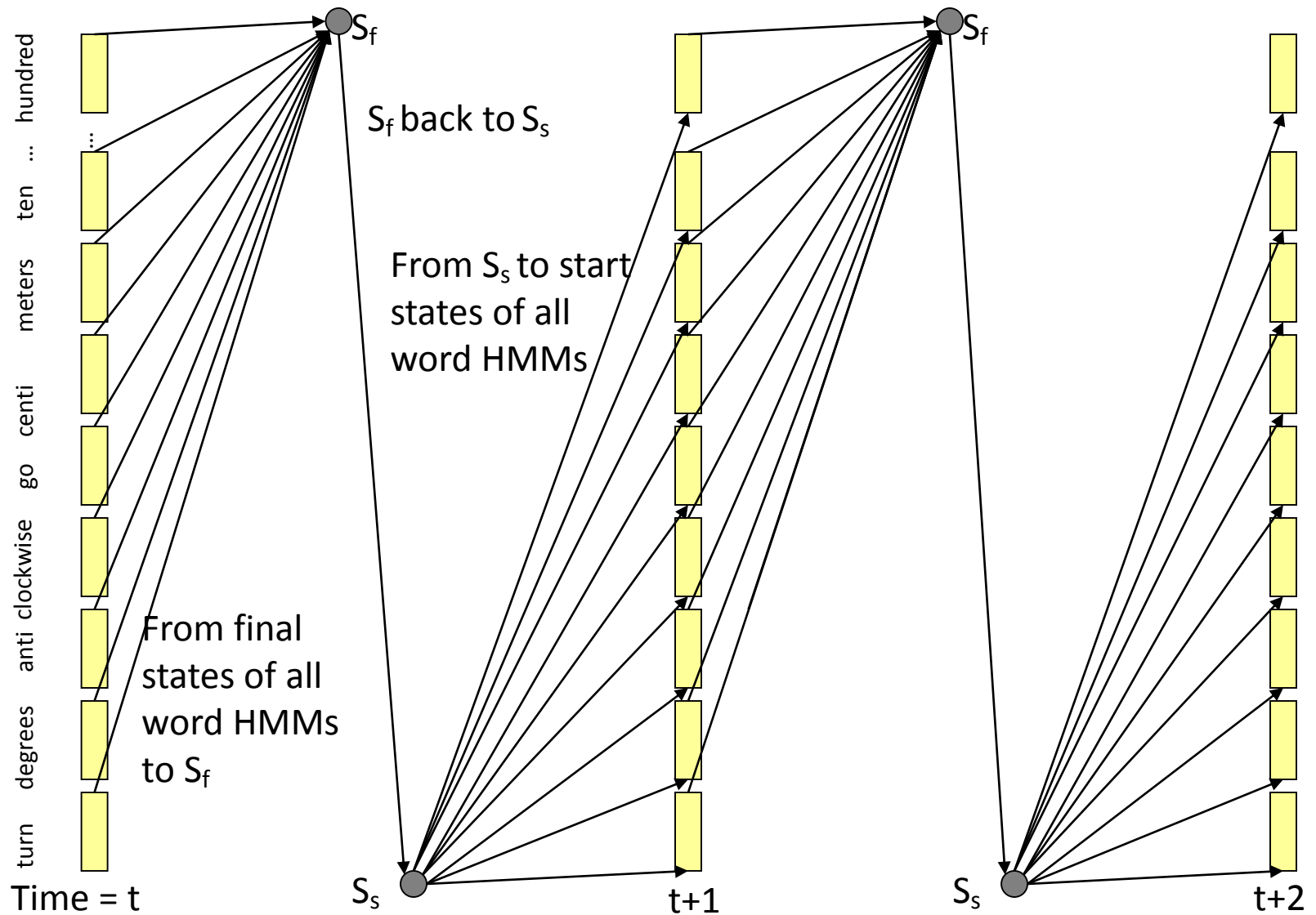


# Trellis Construction for *ROBOT0*

- How many rows does a trellis constructed from *ROBOT0* language HMM have?
  - Assume 3 emitting states + 1 non-emitting start state + 1 non-emitting final state, for each word HMM

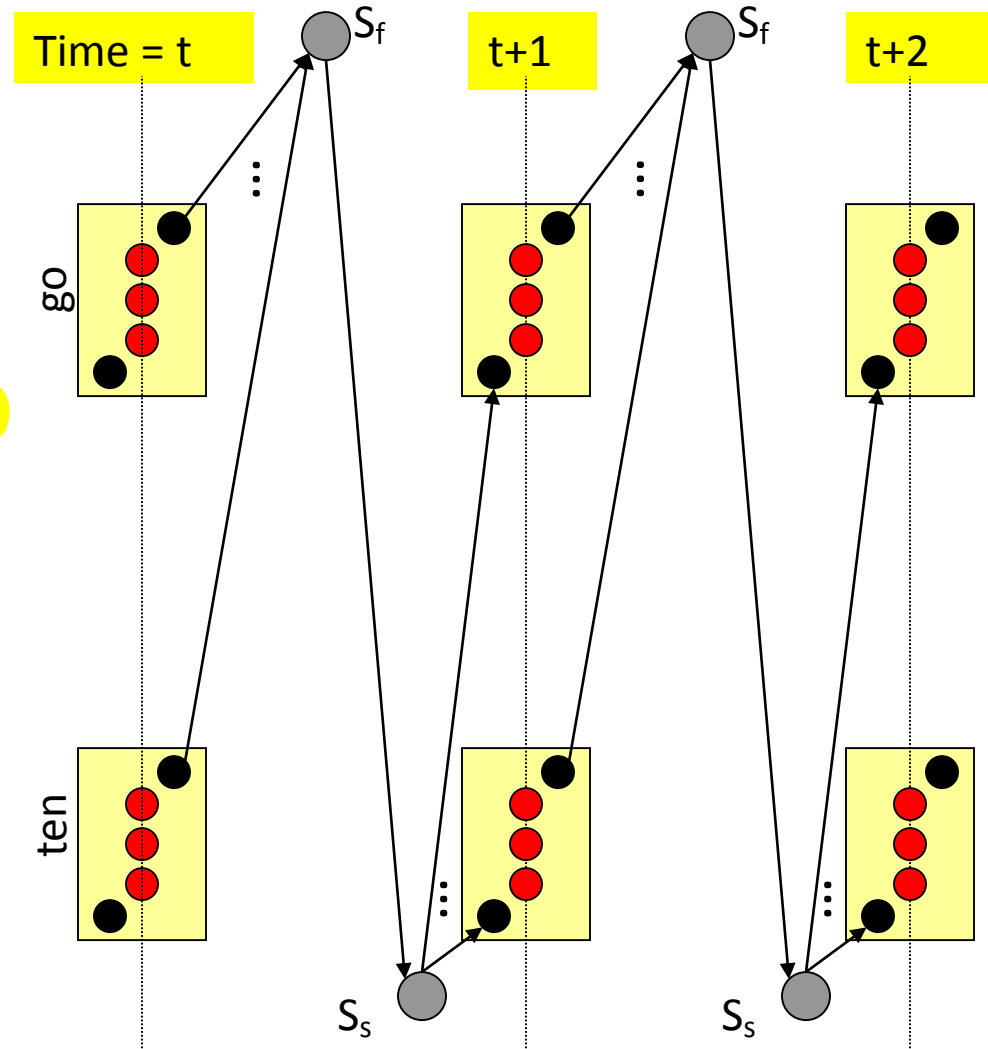


# ROBOT0 Cross Word Transitions



# ROBOT0 Cross Word Transitions

- A portion of trellis shown between time  $t$  and  $t+2$
- Similar Transitions happen from final states of all 17 words to start states of all 17 words
- Non-emitting states shown “between frames”
  - Order them as follows:
    - Find all null state sequences
    - Make sure there are no cycles
    - Order them by dependency
- Other trellis details not shown

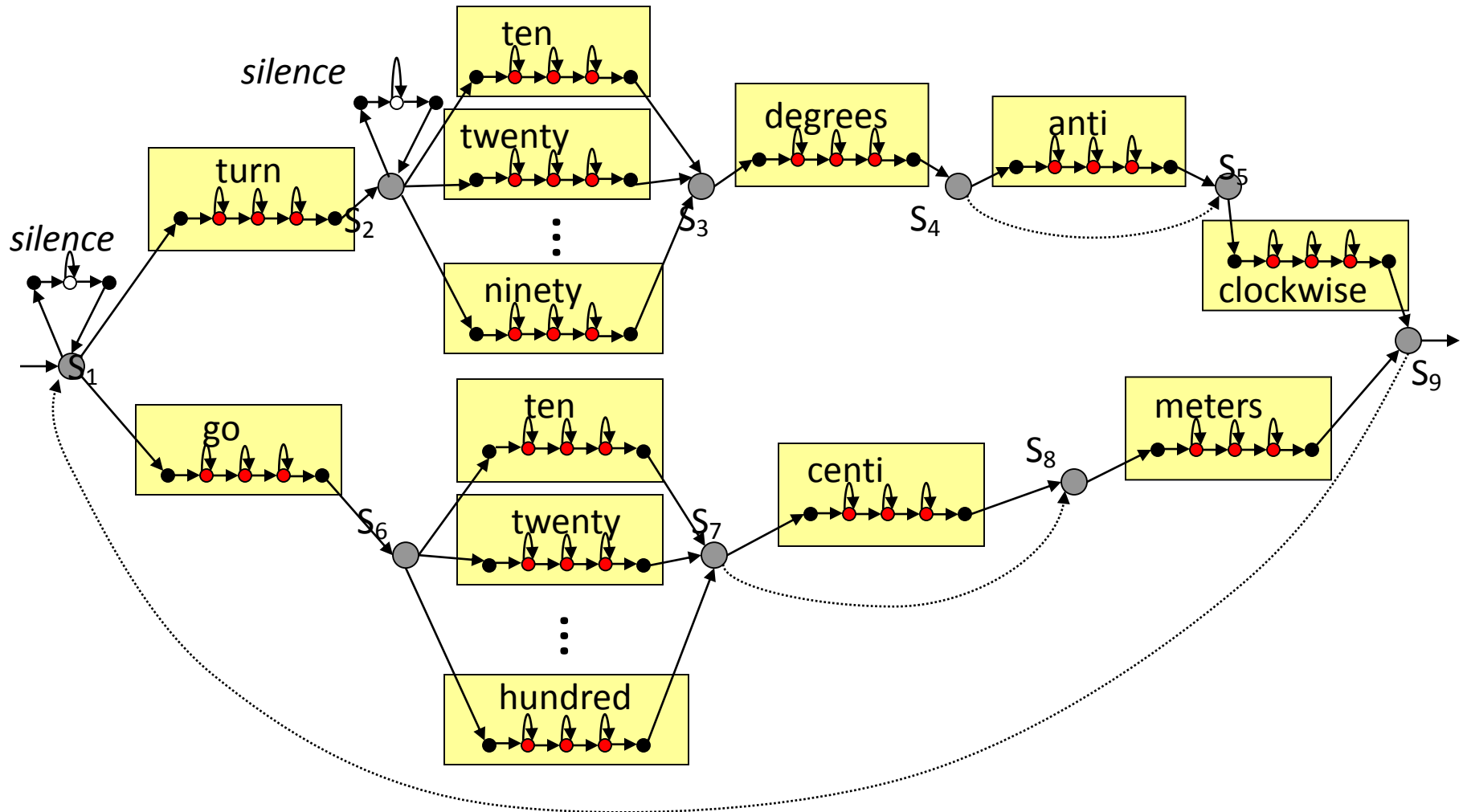


# Trellis Construction for *ROBOT1*

---

- How many rows does a trellis constructed from *ROBOT1* language HMM have?
  - Assume 3 emitting states + 1 non-emitting start state + 1 non-emitting final state, for each word HMM, as before

# ROBOT1 Language HMM



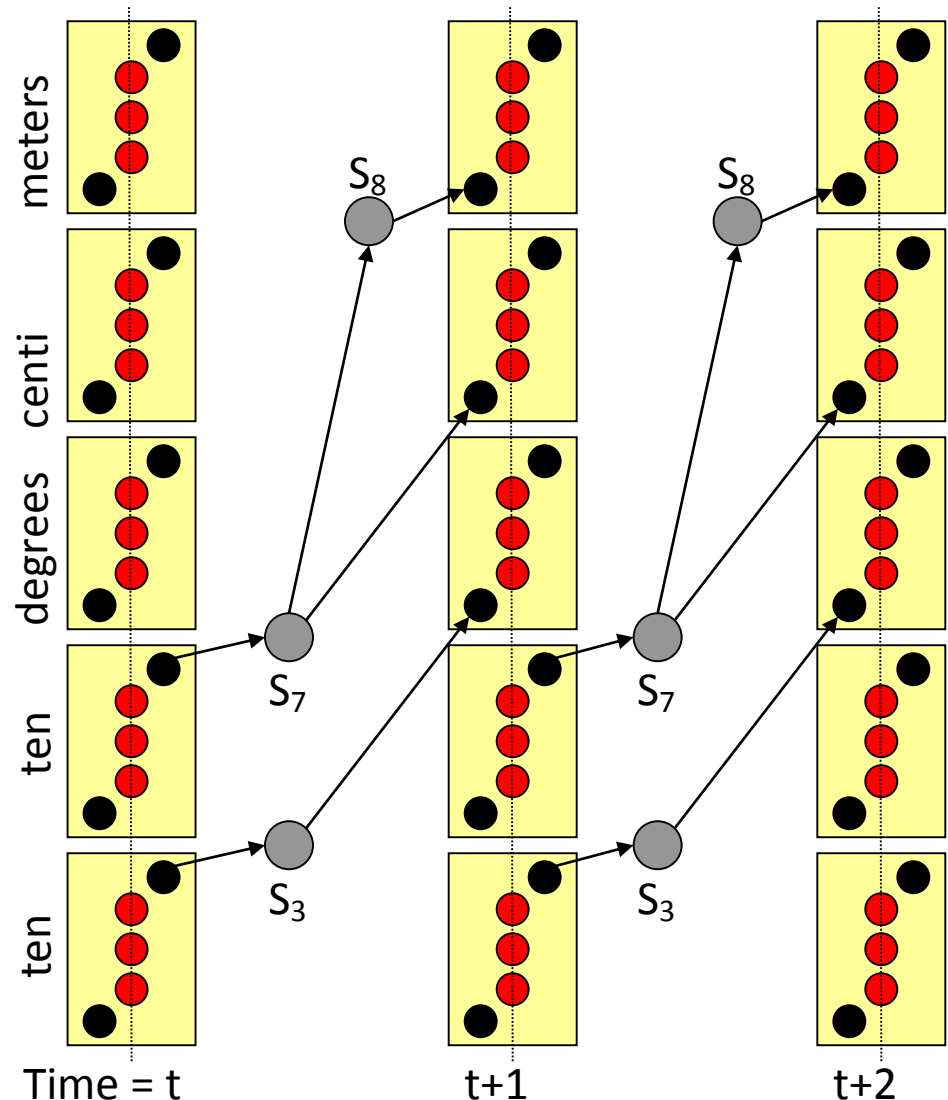
# Trellis Construction for *ROBOT1*

---

- No. of trellis rows = No. of states in Language HMM
  - 26 x 5 word HMM states
    - Note: words “ten” through “ninety” have two copies since they occur between different FSG states! (More on this later)
  - The 9 FSG states become sentence HMM non-emitting states
  - 9 x 3 silence HMM states, one at each FSG state
  - =  $130 + 9 + 27 = 166$  states or 166 rows
- Often it is possible to reduce the state set, but we won't worry about that now
- What about word exit and entry transitions?

# ROBOT1 Cross Word Transitions

- A portion of trellis shown between time  $t$  and  $t+2$
- Note the FSG-constrained cross word transitions; no longer fully connected
- Note there are two instances of “ten”!
  - From different portions of the graph





# Words and Word Instances

---

- Key points from illustration:
  - FSG states (LM states in general) are distinct, and need to be preserved during decoding
  - If the same word is emitted by multiple different transitions (*i.e.* either the source or destination states are different), there are actually **multiple** copies of the word HMM in the sentence HMM

# Creation of Application FSGs

---

- While FSGs can be trained from training data, they can be easily handcrafted from prior knowledge of expected inputs
  - Suitable for situations where little or no training data available
  - Small to medium vocabulary applications with well structured dialog
- Example applications:
  - Command and control (*e.g.* robot control or GUI control)
  - Form filling (*e.g.* making a train reservation)
- Constraints imposed by an FSG lead to very efficient search implementation
  - FSGs rules out many improbable or illegal word sequences outright
  - Parts of the full NxT search trellis are *a priori* ruled out

# Example Application: A Reading Tutor

- Project LISTEN: A reading tutor for children learning to read



# Example Application: Reading Tutor

- A child reads a story aloud, one sentence at a time
- The automated tutor “listens” to the child and tries to help if it has any difficulty
  - Pausing too long at a word
  - Misreading a word
  - Skipping a word

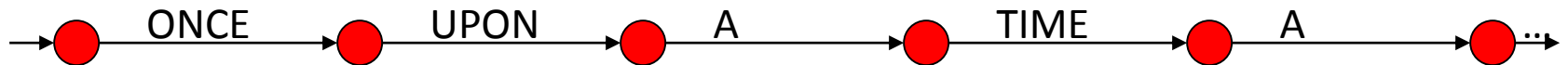


- The child should be allowed to have “normal” reading behavior
  - Repeat a word or phrase, or the entire sentence
  - Partially pronounce a word one or more times before reading it correctly
- Hence, the tutor should account for both normal and incorrect reading
- We do this by building an FSG for the current sentence, as follows

# Example Application: Reading Tutor

---

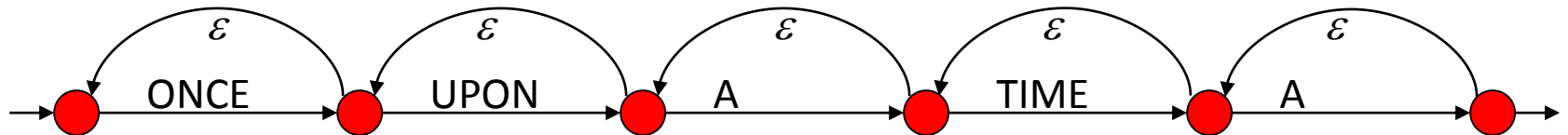
- For each sentence, the tutor builds a new FSG
- An typical sentence:
  - ONCE UPON A TIME A BEAUTIFUL PRINCESS ...
- First we have the “backbone” of the FSG:
  - The backbone models straight, correct reading
    - (Only part of the FSG backbone is shown)
  - FSG states mark positions in text



# Example Application: Reading Tutor

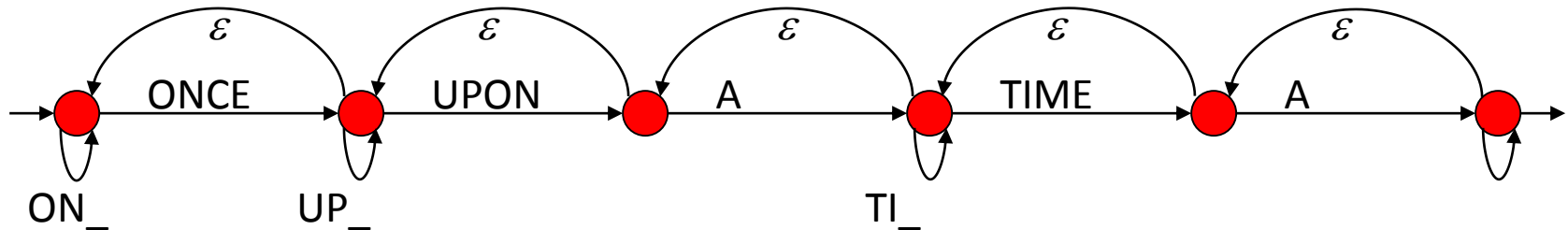
---

- We add backward null transitions to allow repetitions
  - Models jumps back to anywhere in the text
  - It is not necessary to add long backward transitions!



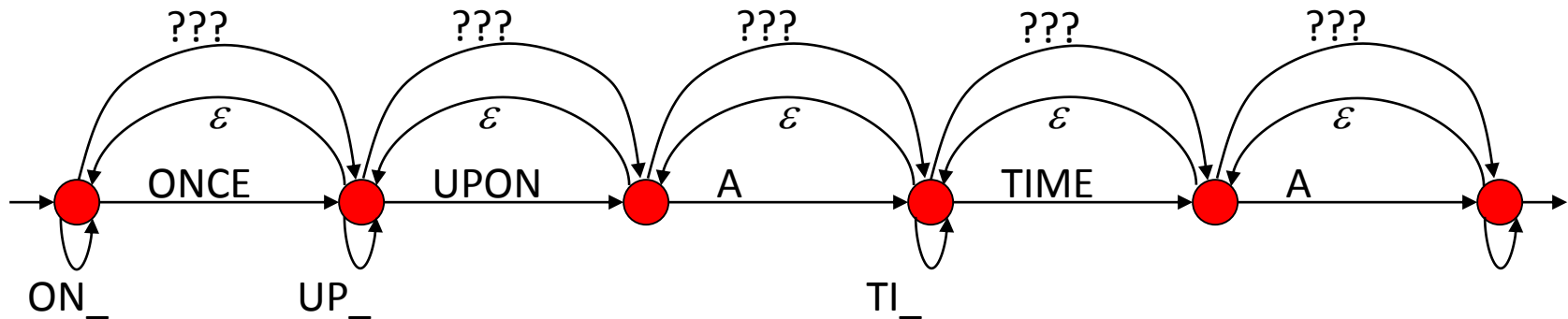
# Example Application: Reading Tutor

- We add truncated word models to allow partial reading of a word (shown with an \_; *e.g.* ON\_)
  - There may be more than one truncated form; only one is shown
  - Partial reading is assumed to mean the child is going to attempt reading the word again, so we do not change state
  - Short words do not have truncated models



# Example Application: Reading Tutor

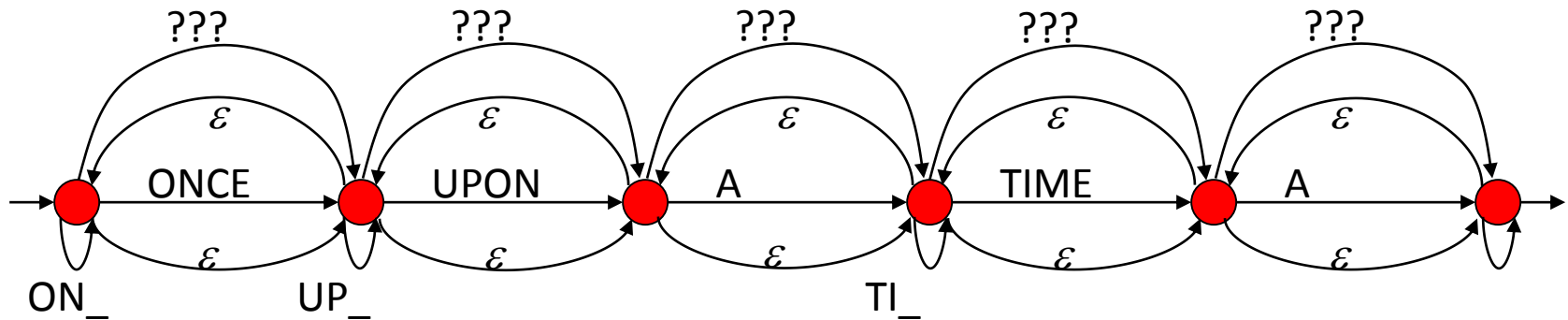
- We add transitions parallel to each correct word, to model misreading, labeled with a *garbage model* (shown as ???)
  - How we obtain the garbage model is not important right now
  - It essentially models any *unexpected* speech; *e.g.*
    - Misreading, other than the truncated forms
    - Talking to someone else





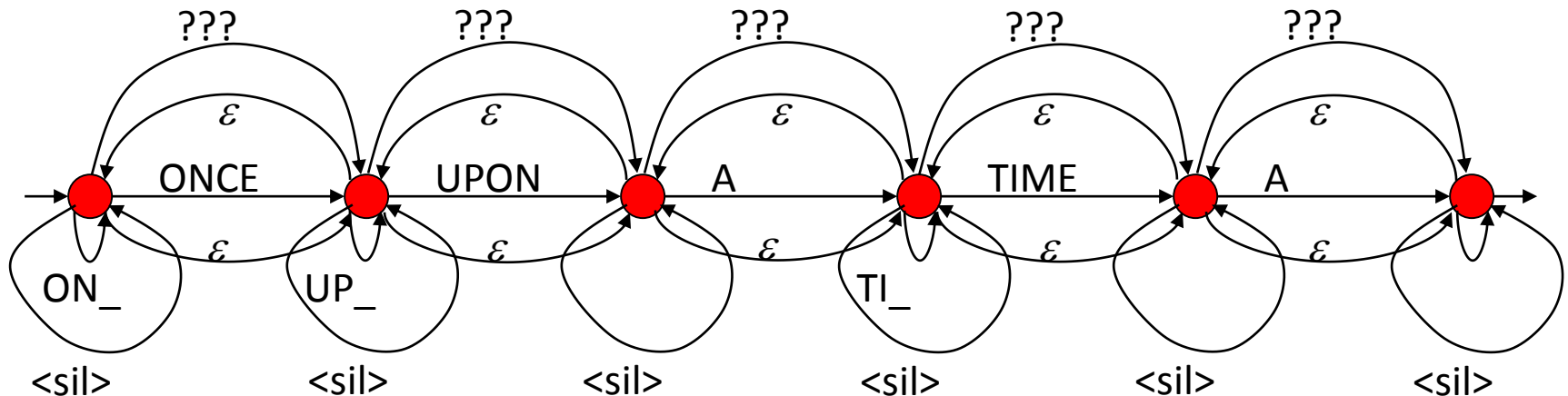
# Example Application: Reading Tutor

- We add forward null transitions to model one or more words being skipped
  - It is not necessary to add long forward transitions!



# Example Application: Reading Tutor

- Not to forget! We add optional silences between words
  - Silence transitions (labeled <sil>) from a state to itself
    - If the child pauses between words, we should not change state
- Finally, we add transition probabilities estimated from actual data recorded with children using the reading tutor



# Example Application: Reading Tutor

---

- The FSG is crafted from an “expert’s” mental model of how a child might read through text
- The FSG does *not* model the student getting stuck (too long a silence)
  - There is no good way to model durations with HMMs or FSGs
  - Instead, the application specifically uses word segmentation information to determine if too long a silence has elapsed
- The application creates a new FSG for each new sentence, and destroys old ones
- Finally, the FSG module even allows dynamic fine-tuning of transition probabilities and modifying the FSG start state
  - To allow the child to continue from the middle of a sentence
  - To adapt to a child’s changing reading behavior

# FSG Representation

---

- A graphical representation is perfect for human visualization of the system
- However, difficult to communicate to a speech recognizer!
  - Need a textual representation
  - Two possibilities: tabular, or rule-based
    - Commonly used by most real ASR packages that support FSGs

# Tabular FSG Representation Example

- Example FSG from Sphinx-2 / Sphinx-3

```
FSG_BEGIN
NUM_STATES 5
START_STATE 0
FINAL_STATE 4
TRANSITION 0 1 0.9 ONCE
TRANSITION 0 0 0.01 ONCE
TRANSITION 1 2 0.9 UPON
TRANSITION 1 1 0.01 UPON
TRANSITION 2 3 0.9 A
TRANSITION 2 2 0.01 A
TRANSITION 3 4 0.9 TIME
TRANSITION 3 3 0.01 TIME
TRANSITION 0 1 0.01
TRANSITION 1 2 0.01
TRANSITION 2 3 0.01
TRANSITION 3 4 0.01
TRANSITION 1 0 0.017
TRANSITION 2 0 0.017
TRANSITION 3 0 0.017
TRANSITION 4 0 0.017
TRANSITION 2 1 0.01
TRANSITION 3 2 0.01
TRANSITION 4 3 0.01
FSG_END
```

Table of transitions

# Tabular FSG Representation

---

- Straightforward conversion from graphical to tabular form:
  - List of states (*e.g.* states may be named or numbered)
    - *E.g.* Sphinx-2 uses state numbers
  - List of transitions, of the form:  
Origin-state, destination state, emitted word, transition probability
    - Emitted word is optional; if omitted, implies a null transition
    - Transition probability is optional
      - All unspecified transition probabilities from a given state are equally likely
  - Set of start states
  - Set of final states

# Rule-Based FSG Representation

---

- Before we talk about this, let us consider something else first

# Recursive Transition Networks

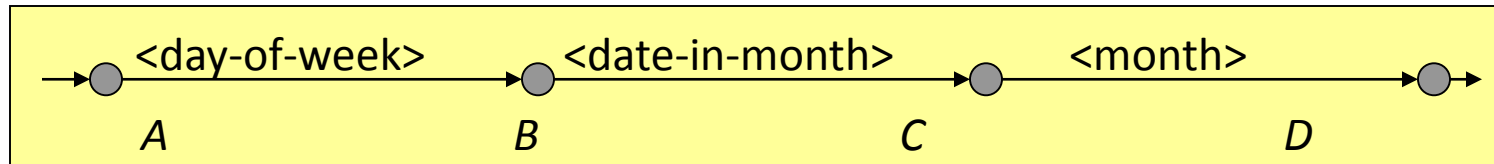
---

- What happens if we try to “compose” an FSG using other FSGs as its components?
  - *Key idea*: A transition in an FSG-like model can be labeled with an entire FSG, instead of a single word
    - When the transition is taken, it can emit any one of the *sentences* in the *language* of the label FSG
- Such networks of nested grammars are called *recursive transition networks* (RTNs)
  - Grammar definitions can be *recursive*
- But first, let us consider such composition *without* any recursion
  - Arbitrary networks composed in this way, that include recursion, turn out not to be FSGs at all



# Nested FSGs

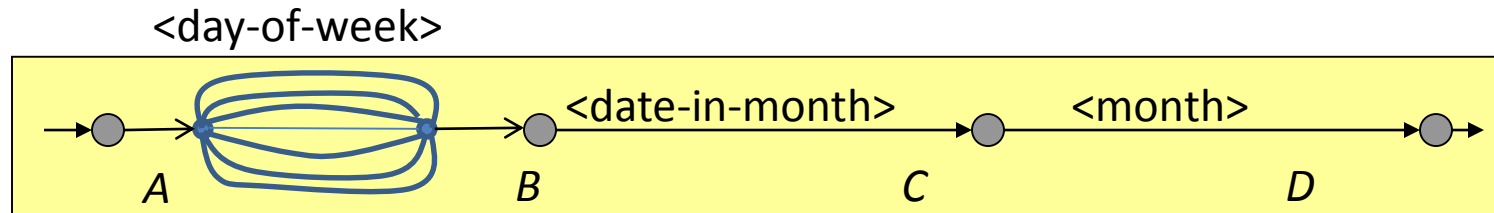
- *E.g.* here is a *<date>* FSG:



- Where, *<day-of-week>*, *<date-in-month>* and *<month>* are the FSGs defined earlier
- *Exercise:* Include *<year>* into this specification, and allow reordering the components

# Nested FSGs

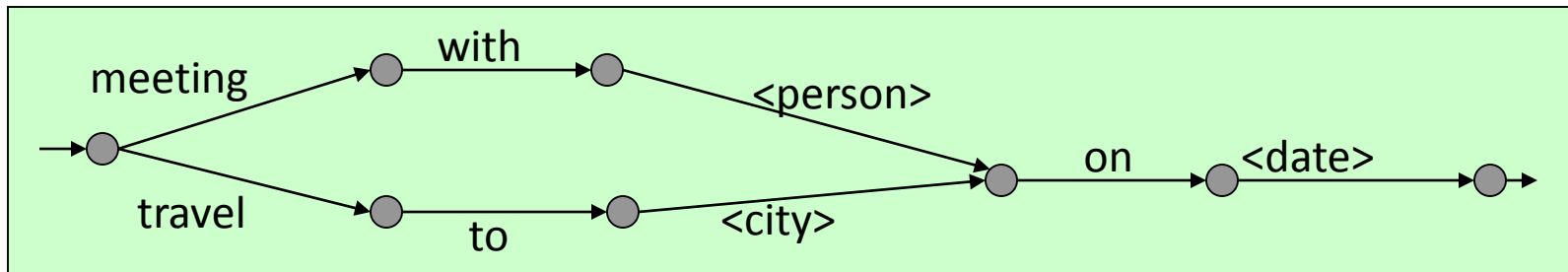
- *E.g.* here is a *<date>* FSG:



- Each edge in day of week sub-graph represents the HMM for one of “Sunday”, “Monday”, ... “Saturday”
- Note: Insertion of language HMM for day of week similar to insertion of word HMMs

# More Nested FSGs

- *Example:* Scheduling task
  - (Transition labels with <> actually refer to other FSGs)



- The <date> FSG above is further defined in terms of other FSGs
  - Thus, FSG references can be *nested* arbitrarily deeply
- As usual, we have not shown transition probabilities, but they are nevertheless there, at least implicitly
  - *E.g.* meetings are much more frequent than travels (for most office-workers!)

# Flattening Composite FSGs for Decoding

---

- In the case of the above scheduling task FSG, it is possible to flatten it into a regular FSG (*i.e.* without references to other FSGs) simply by embedding the target FSG in place of an FSG transition
  - Very similar to generation of sentence HMMs from FSGs
- At this point, the flattened FSG can be directly converted into the equivalent sentence HMM for decoding

# Flattening Composite FSGs for Decoding

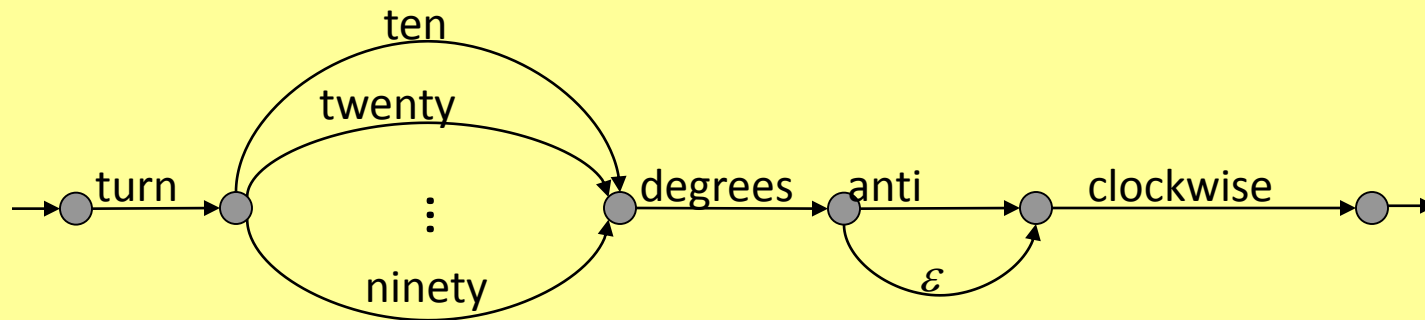
---

- However, not all composite “FSGs” can be flattened in this manner, if we allow recursion!
  - As mentioned, these are really RTNs, and not FSGs
- The grammars represented by them are called *context free grammars* (CFGs)
- Let us consider this recursion in some detail

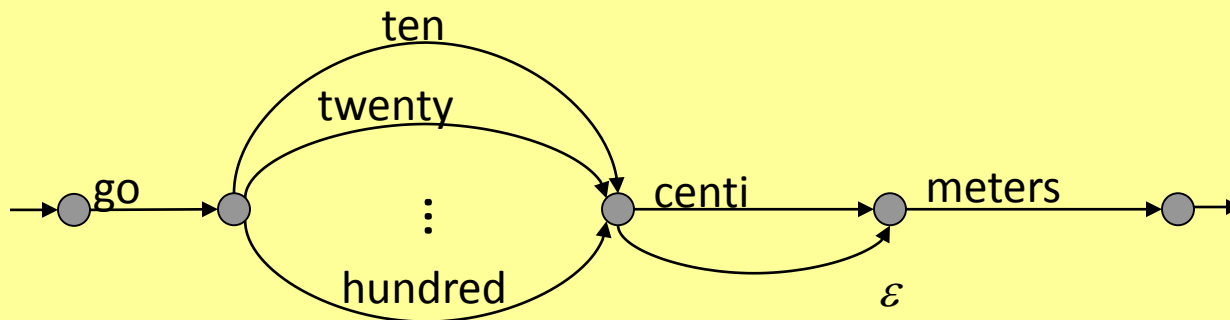
# Recursion in Grammar Networks

- It is possible for a grammar definition to refer to *itself*
- Let us consider the following two basic FSGs for robot control:

— <Turn-command> FSG:



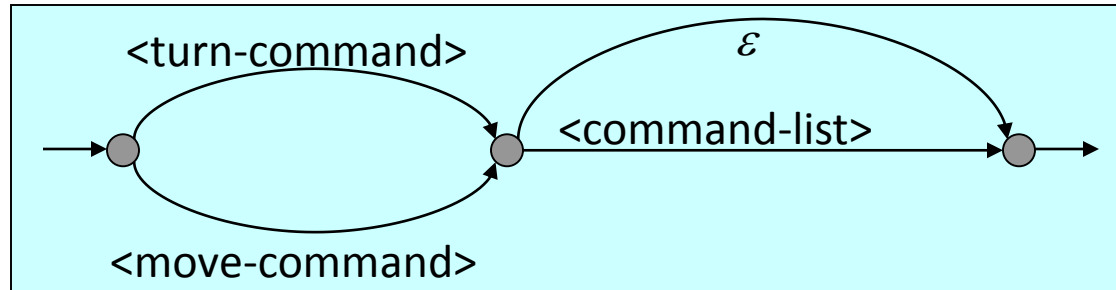
— <Move-command> FSG:



# Recursion in Grammar Networks (contd.)

- We can rewrite the original robot control grammar using the following *recursive* definition:

– <command-list> grammar:



– <command-list> grammar is defined in terms of itself

# A Little Digression: Grammar Libraries

---

- It is very useful to have a *library* of reusable grammar components
  - New applications can be designed rapidly by composing together already existing grammars
- A few examples of common, reusable grammars:
  - Date, month, day-of-week, etc.
  - Person names and place name (cities, countries, states, roads)
  - Book, music or movie titles
  - Essentially, almost any *list* is a potentially reusable FSG



# RTNs and CFGs

---

- Clearly, RTNs are a powerful tool for defining structured grammars
- As mentioned, the class of grammars represented by such networks is called the class of *context free grammars* (CFGs)
  - Let us look at some characteristics of CFGs

# Context Free Grammars

---

- Compared to FSGs, CFGs are a more powerful mechanism for defining *languages* (sets of acceptable sentences)
  - “Powerful” in the sense of imposing more structure on sentences
  - CFGs are a superset of FSGs
    - Every language accepted by an FSG is also accepted by some CFG
    - But not every CFG has an equivalent FSG
- Human languages are actually fairly close to CFGs, at least syntactically
  - Many applications use them in structured dialogs

# Context Free Grammars

---

- What is a CFG?
  - Graphically, CFGs are exactly what we have been discussing:
    - The class of grammars that can have *concepts* defined in terms of other grammars (possibly themselves, recursively)
  - They are context free, because the definition of a *concept* is the same, regardless of the *context* in which it occurs
    - *i.e.* independent of where it is embedded in another grammar
- However, unlike FSGs, may not have graphical representations
- In textual form, CFGs are defined by means of *production rules*

# Context Free Grammars (contd.)

---

- Formally, a CFG is defined by the following:
  - A finite set of *terminal* symbols (*i.e.* words in the vocabulary)
  - A finite set of *non-terminal* symbols (the *concepts*, such as <date>, <person>, <move-command>, <command-list> etc.
  - A special non-terminal, usually *S*, representing the CFG
  - A finite set of *production rules*
    - Each rule defines a non-terminal as a *possibly empty sequence* of other symbols, each of which may be a terminal or a non-terminal
      - There may be multiple such definitions for the same non-terminal
    - The empty rule is usually denoted: <non-terminal> ::= *e*
- The language generated by a CFG is the set of all sentences of terminal symbols that can be derived by expanding its special non-terminal symbol *S*, using the production rules

# Why Are CFGs Useful?

---

- The syntax of large parts of human languages can be defined using CFGs

- *e.g.* a simplistic example:

- <sentence> ::= <noun-phrase> <verb-phrase>
    - <noun-phrase> ::= <name> | <article> <noun>
    - <verb-phrase> ::= <verb> <noun-phrase>
    - <name> ::= HE | SHE | JOHN | ALICE...
    - <article> ::= A | AN | THE
    - <noun> ::= BALL | BAT | FRUIT | BOOK ...
    - <verb> ::= EAT | RUN | HIT | READ ...

- Clearly, the language allows nonsensical sentences:

- JOHN EAT A BOOK

- But it is syntactically “correct”
    - The grammar defines the syntax, not the semantics

# Robot Control CFG

---

- Example rules for robot control
- <command-list> is the CFG being defined (= S):

```
<command-list> ::= <turn-command> | <move-command>
<command-list> ::= <turn-command><command-list>
<command-list> ::= <move-command><command-list>

<turn-command> ::= TURN <degrees> DEGREES <direction>
<direction>      ::= clockwise | anti clockwise
<move-command> ::= GO <distance> <distance-units>
<distance-units> ::= meters | centi meters
<degrees>        ::= TEN | TWENTY | THIRTY | FORTY | ... | NINETY
<distance>       ::= TEN | TWENTY | THIRTY | ... | HUNDRED
```

# Probabilistic Context Free Grammars

---

- CFGs can be made probabilistic by attaching a probability to each production rule
  - These probabilities are used in computing the overall likelihood of a recognition hypothesis (sequence of words) matching the input speech
- Whenever a rule is used, the rule probability is applied

# Context Free Grammars: Another View

---

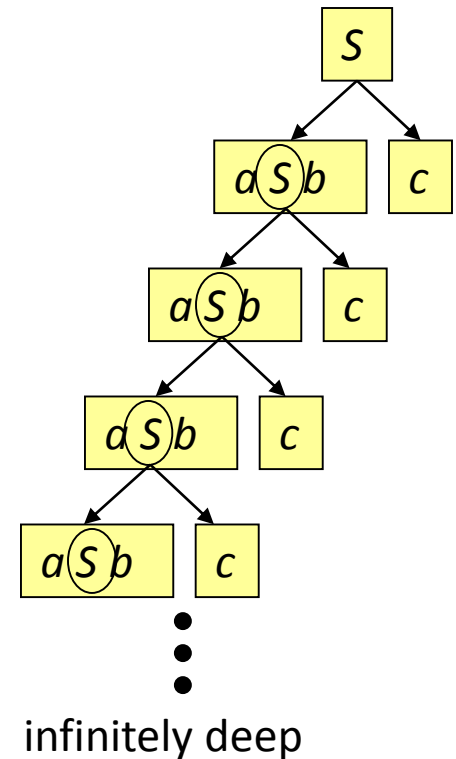
- Non-terminals can be seen as *functions* in programming languages
  - Each production rule defines the function body; as a sequence of statements
  - Terminals in the rule are like ordinary assignment statements
  - A non-terminal within the rule is a *call to a function*
- Thus, the entire CFG is like a program made up of many functions
  - Obviously, program execution can take many paths!
  - Each program execution produces a complete sentence



# CFG Based Decoding

- Consider the following simple CFG:
  - $S$  is like an *overloaded* function
    - It is also the entire “program”
  - The “call tree” on the right shows all possible “program execution paths”
- CFG based decoding is equivalent to finding out which rules were used in what sequence, to produce the spoken sentence
  - A general algorithm to determine this is too complex to describe here
  - Instead, we can try to approximate CFGs by FSGs

$S ::= aSb \mid c$



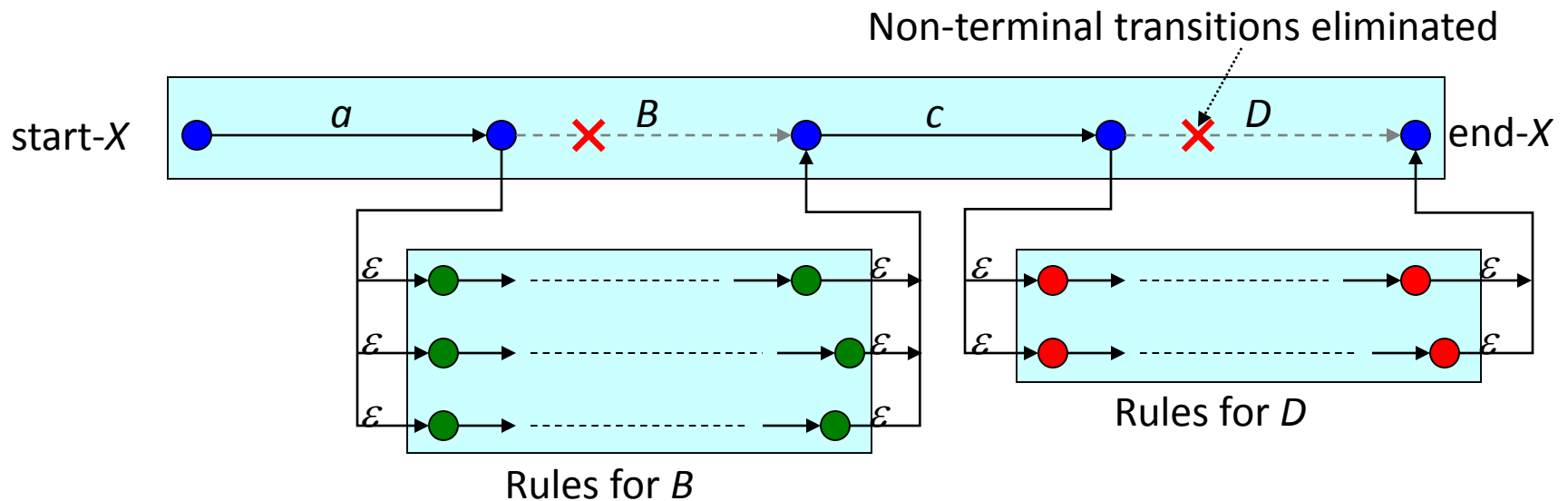
# Approximating a CFG by an FSG

---

- Advantage: back in familiar, efficient decoding territory
- Disadvantage: depends on the approximation method
  - In some, the FSG will allow illegal sentences to become legal
  - In others, the FSG will disallow some legal sentences
- For practical applications, the approximations can be made to work nicely
  - Many applications need only FSGs to begin with
  - The errors committed by the approximate FSG can be made extremely rare

# FSG Approximation to CFGs:

- Consider a rule:  $X ::= aBcD$ , where  $a$  and  $c$  are terminal symbols (words), and  $B$  and  $D$  are non-terminals
- We can create the following FSG for the rule:



- It should be clear that when the above construction is applied to all the rules of the CFG, we end up with an FSG

# FSG Approximation to CFGs:

---

- Example

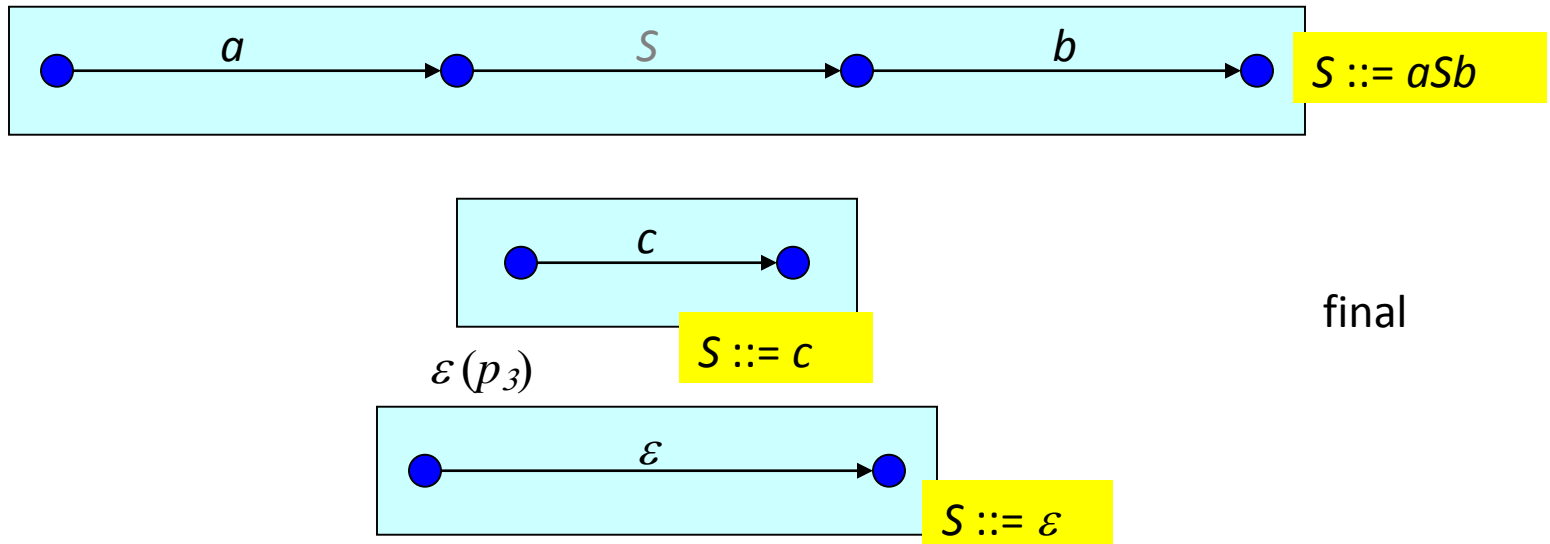
$$S ::= aSb \mid c \mid \varepsilon$$

# FSG Approximation to CFGs:

- We can construct an FSG from a CFG as follows:

$S ::= aSb \mid c \mid \varepsilon$

- Take each production rule in the CFG as a *sequence* of state transitions, one transition per symbol in the rule
  - The first state is the start state of the rule, and the last the final state of the rule

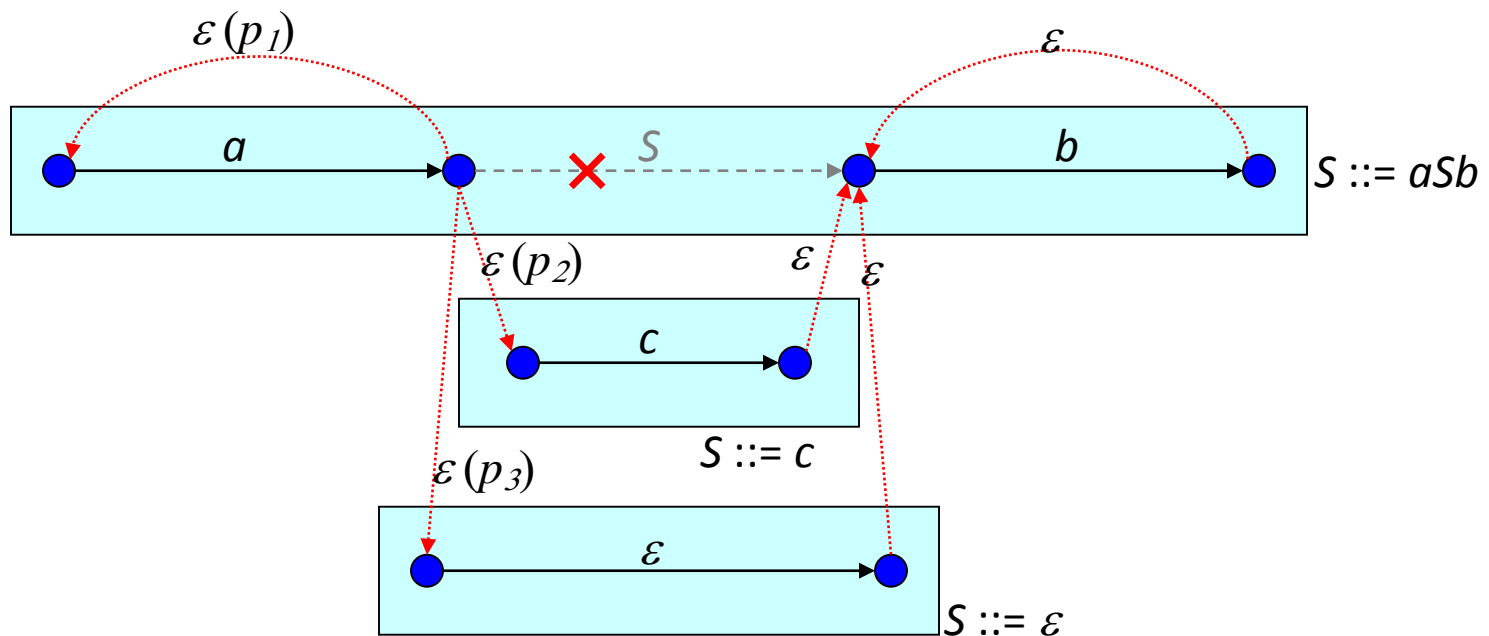


# FSG Approximation to CFGs:

- We can construct an FSG from a CFG as follows:

$S ::= aSb \mid c \mid \varepsilon$

- Take each production rule in the CFG as a *sequence* of state transitions, one transition per symbol in the rule
  - The first state is the start state of the rule, and the last the final state of the rule
- Replace each non-terminal in the sequence with null transitions *to* the start, and *from* the end of each rule for that non-terminal



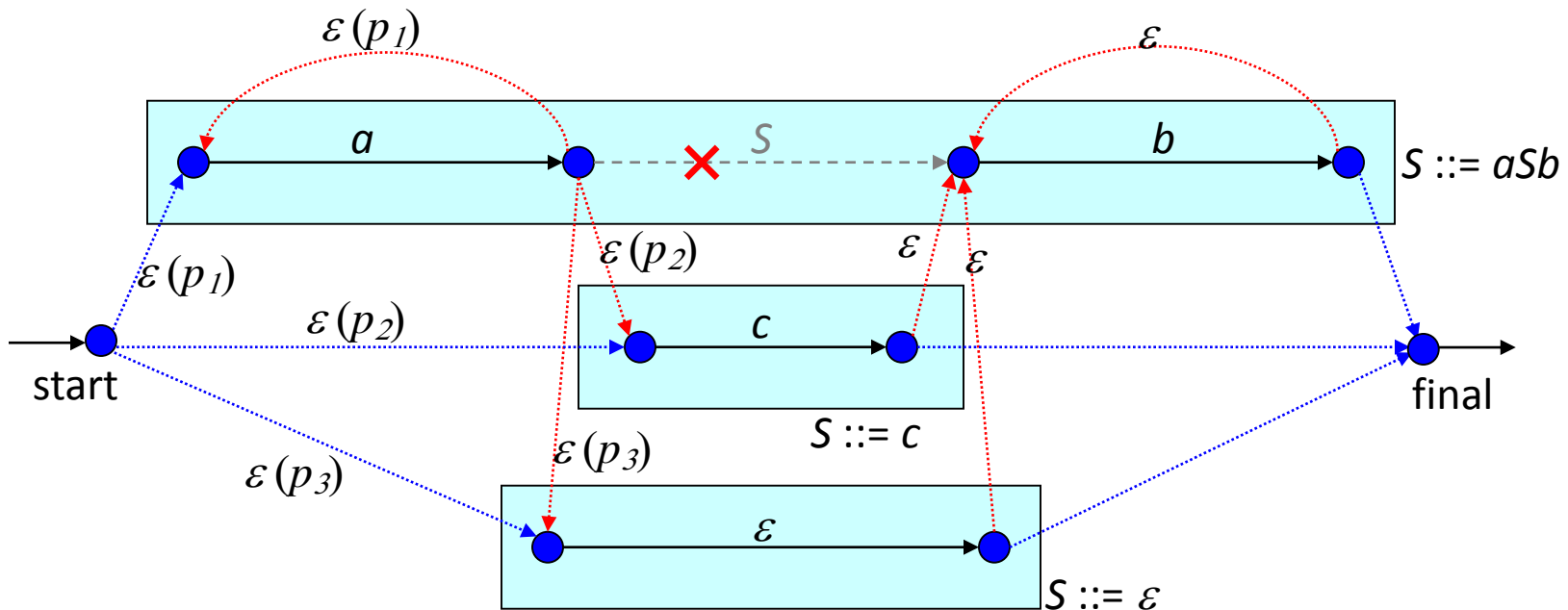
# FSG Approximation to CFGs:

---

- We can construct an FSG from a CFG as follows:
  - Take each production rule in the CFG as a *sequence* of state transitions, one transition per symbol in the rule
    - The first state is the start state of the rule, and the last the final state of the rule
  - Replace each non-terminal in the sequence with null transitions *to* the start, and *from* the end of each rule for that non-terminal
    - (The empty string  $\epsilon$  is considered to be a terminal symbol)
  - Make the start states of all the rules for the distinguished CFG symbol  $S$  to be the start states of the FSG
  - Similarly, make the final states of the rules for  $S$  to be the final states of the FSG
    - Or, add new start and final states with null transitions to and from the above
- Since the CFG has a *finite* set of rules of *finite* length, and we remove all non-terminals, we end up with a plain FSG

# CFG to FSG Example

- Let's convert the following CFG to FSG:
  - Assume the rules have probabilities  $p_1$ ,  $p_2$  and  $p_3$  ( $p_1 + p_2 + p_3 = 1$ )
- We get the FSG below:



The over-generating approximation

$S ::= aSb \mid c \mid \epsilon$



# Why is this FSG an Approximation?

---

- Consider  $S ::= aSb \mid c \mid \varepsilon$
- Any string *must* have as many “a”s as “b”s
- The approximate FSG does not guarantee that the number of “a”s and “b”s are the same
  - The FSG behavior is governed entirely by its *current* state, and not how it got there
  - To implement the above requirement, the FSG would have to remember that it took a particular transition a long time ago
- The constructed FSG allows all sentences of the CFG, since the original paths are all preserved
- Unfortunately, it also *allows illegal paths to become legal*

# Another FSG Approximation to CFGs

---

- Another possibility is to eliminate the infinite recursion
  - In most practical applications, one rarely sees recursion depths beyond some small number
- So, we can arbitrarily declare that recursion cannot proceed beyond a certain depth
- We only need to explore a *finite* sized tree
- A finite sized search problem can be turned into an FSG!
- This FSG will never accept an illegal sentence, but it may reject legal ones (those that exceed the recursion depth limit)
  - The deeper the limit, the less the chance of false rejection

# FSG Optimization

---

- In the first version, the FSG created had a large number of null transitions!
- We can see from manual examination that many are redundant
- Blindly using this FSG to create a search trellis would be highly inefficient
- We can use FSG optimization algorithms to reduce its complexity
  - It is possible to eliminate unnecessary (duplicate) states
  - To eliminate unnecessary transitions, usually null-transitions
- Topic of discussion for another day!

# The Language Weight

---

- According to the basic speech recognition equation, we wish to maximize:  $P(X|W) P(W)$  over all word sequences  $W$
- In practice, it has been found that left in this form, the language model (*i.e.*  $P(W)$ ) has little effect on accuracy
- Empirically, it has been found necessary to maximize:  $P(X|W)P(W)^k$ , for some  $k>1$ 
  - $k$  is known as the *language weight*
  - Typical values of  $k$  are around 10, though they range rather widely
  - When using log-likelihoods, the LM log-likelihoods get multiplied by  $k$

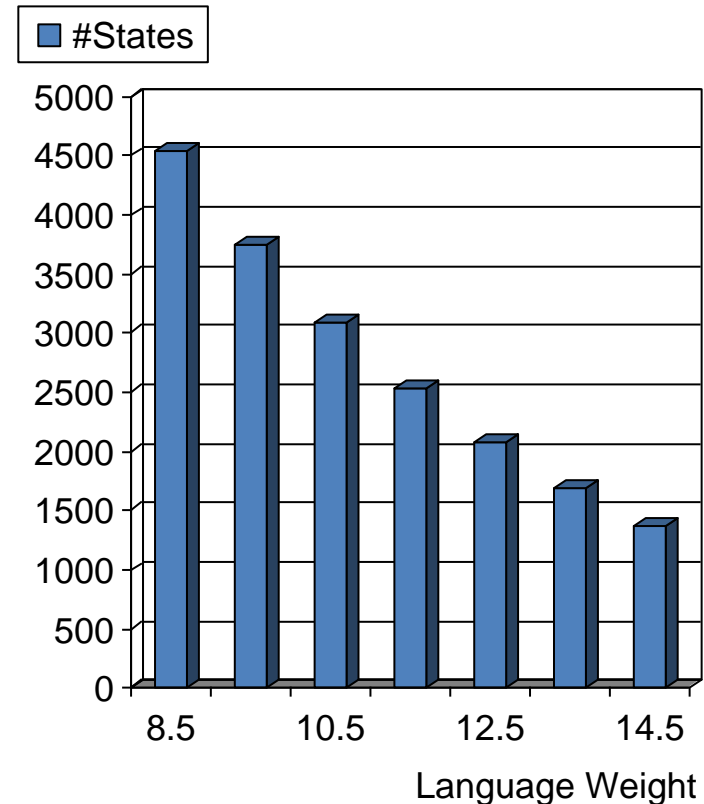
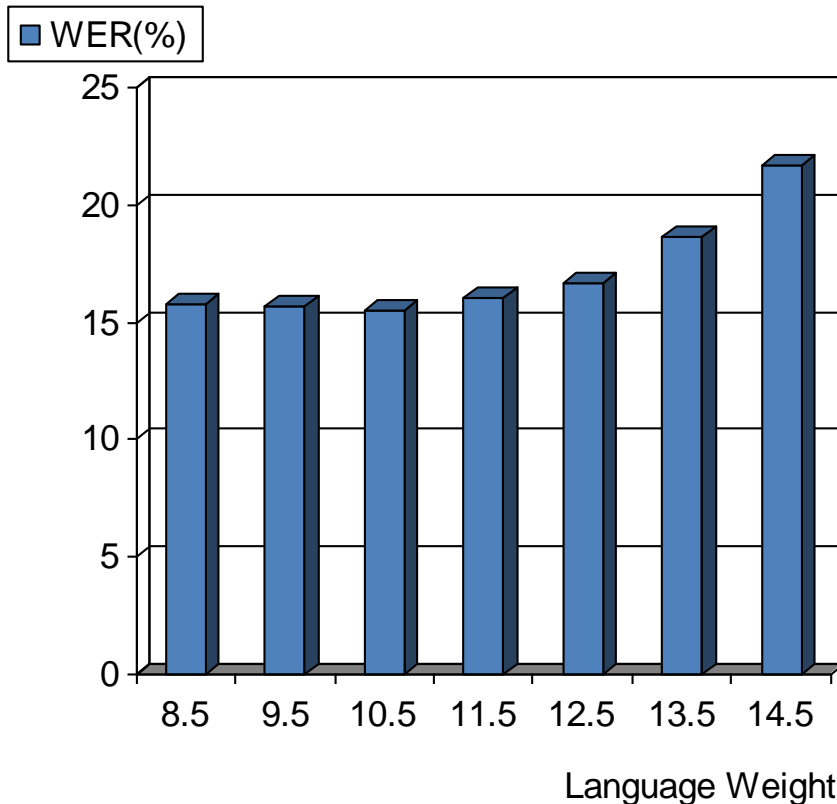
# Optimizing Language Weight

---

- The optimum setting for the language weight is determined empirically, by trying a range of values on some test data
  - This process is referred to as *tuning* the language weight
- When attempting such tuning, one should keep in mind that changing the language weight changes the range of total path likelihoods
- As a result, beam pruning behavior gets affected
  - As language weight is increased, the LM component of the path scores decreases more quickly ( $p^k$ , where  $p < 1$  and  $k > 1$ )
  - If the beam pruning threshold is kept constant, more paths fall under the pruning threshold and get pruned
- Thus, it is necessary to adjust the beam pruning thresholds while changing language weight
  - Makes the tuning process a little more “interesting”

# Optimizing Language Weight: Example

- No. of active states, and word error rate variation with language weight (20k word task)



- Relaxing pruning improves WER at LW=14.5 to 14.8%

# Rationale for the Language Weight

---

- Basically an *ad hoc* technique, but there are arguments for it:
- HMM state output probabilities are usually density values, which can range very widely (*i.e.*, not restricted to the range 0..1)
- LM probabilities, on the other hand, are true probabilities ( $< 1.0$ )
- Second, acoustic likelihoods are computed on a frame-by-frame basis as though the frames were completely independent of each other
  - Thus, the acoustic likelihoods tend to be either widely under or over estimated
- In combination, the effect is that the *dynamic range* of acoustic likelihoods far exceeds that of the LM
- The language weight is needed to counter this imbalance between the range of the two scores

# CFG Support in ASR Systems and SRGS

---

- Most commercial systems provide some support for CFG grammars
- May be specified in many formats
  - BNF (Backus Naur format)
  - ABNF (Augmented BNF)
- Many standards
  - SRGS (Speech Recognition Grammar Specification) is a proposed W3C standard
    - Specifies the format in which CFG grammars may be input to a speech recognizer
    - For details: <http://www.w3.org/TR/speech-grammar/>
  - JSGF (Java speech grammar format)
  - Others
- Need tools to read the CFG in the appropriate format and convert it to the desired internal representation
  - Several open source tools
  - Write your own: YACC / Bison / ..



# Summary

---

- Language models are essential for recognition accuracy
- LMs can be incorporated into the decoding framework through the graph structure
  - Appropriate probabilities may be introduced if required
- The simplest forms of language models are structured grammars
- Structured grammars are of two common types: finite state (FSG) and context free (CFG)
- FSGs can be easily converted into sentence HMM for decoding
- CFGs obtain their power and appeal from their ability to function as building blocks
- CFGs are much harder to decode exactly
- However, CFGs can be approximated by FSGs
- *For free form natural language we require N-gram language models*

# Looking Forward

---

- It is hard to construct structured grammars for large vocabulary applications
- Our next focus will be large vocabulary and its implications for all aspects of modeling and decoding strategies