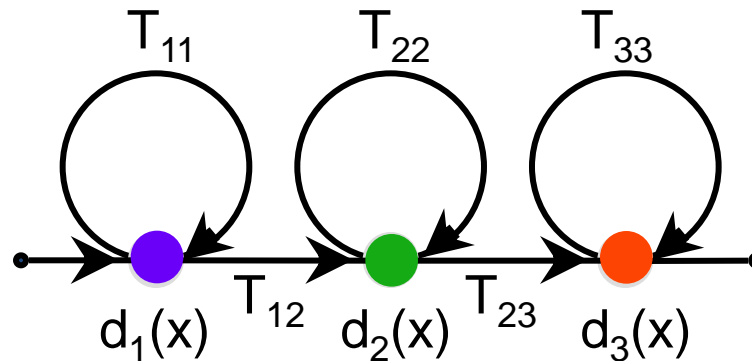


Design and Implementation of Speech Recognition Systems

Spring 2011

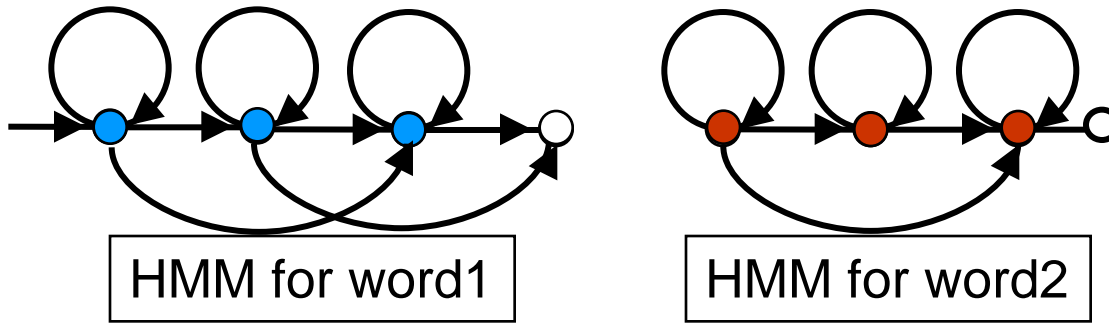
Class 14: Grammars
14 Mar 2011

Recap: HMMs are Generalized Templates



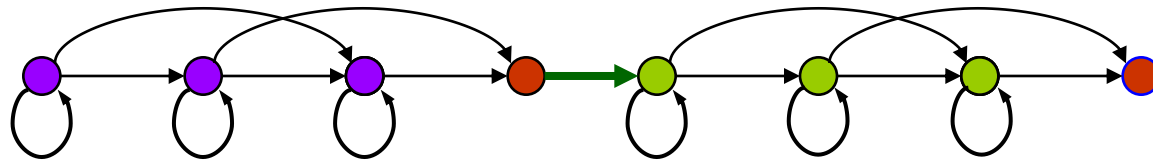
- A set of “states”
 - A distance function associated with each state
- A set of transitions
 - Transition-specific penalties

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

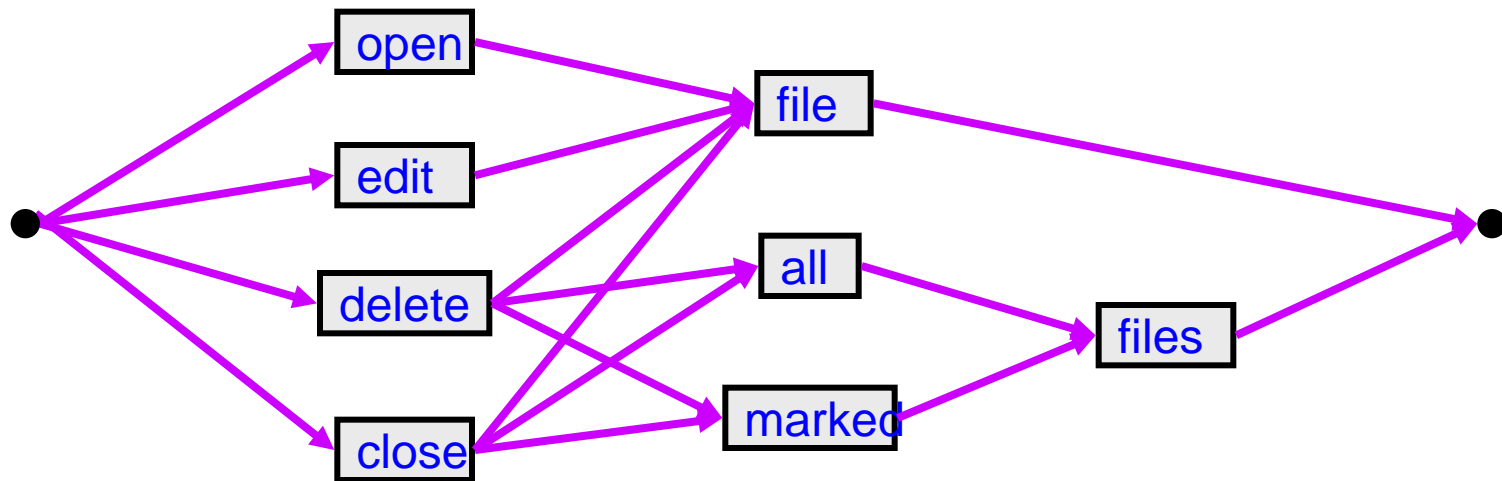
Recap: Recognizing word sequences



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

Motivation

- Prior to this, we have looked at speech recognition without worrying about *language structure*
 - i.e. we've treated all word sequences as being equally likely
 - 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

- Deprived of context, humans flounder just as badly, or worse

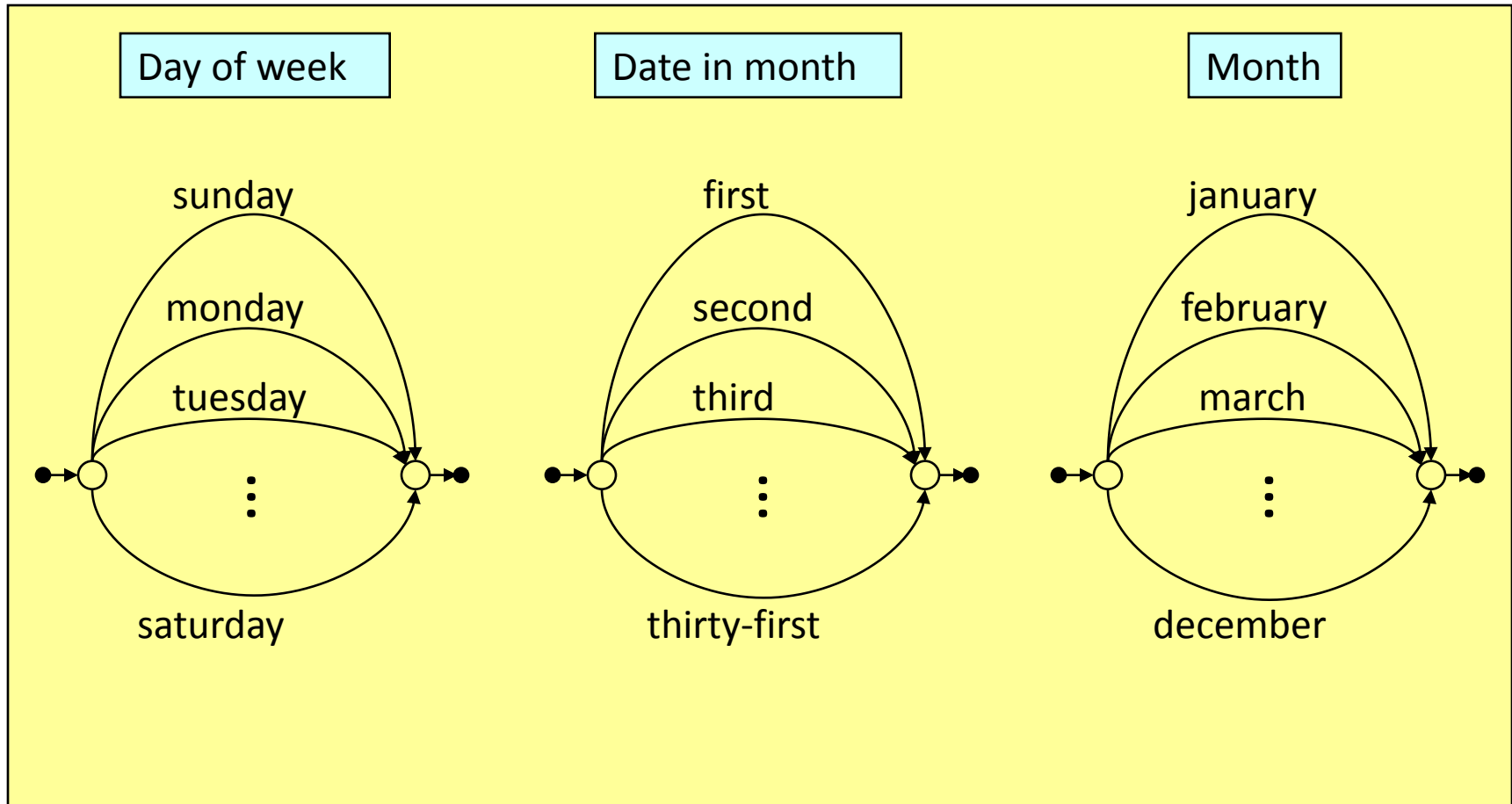
- Still, we will focus only on the syntactic level

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):

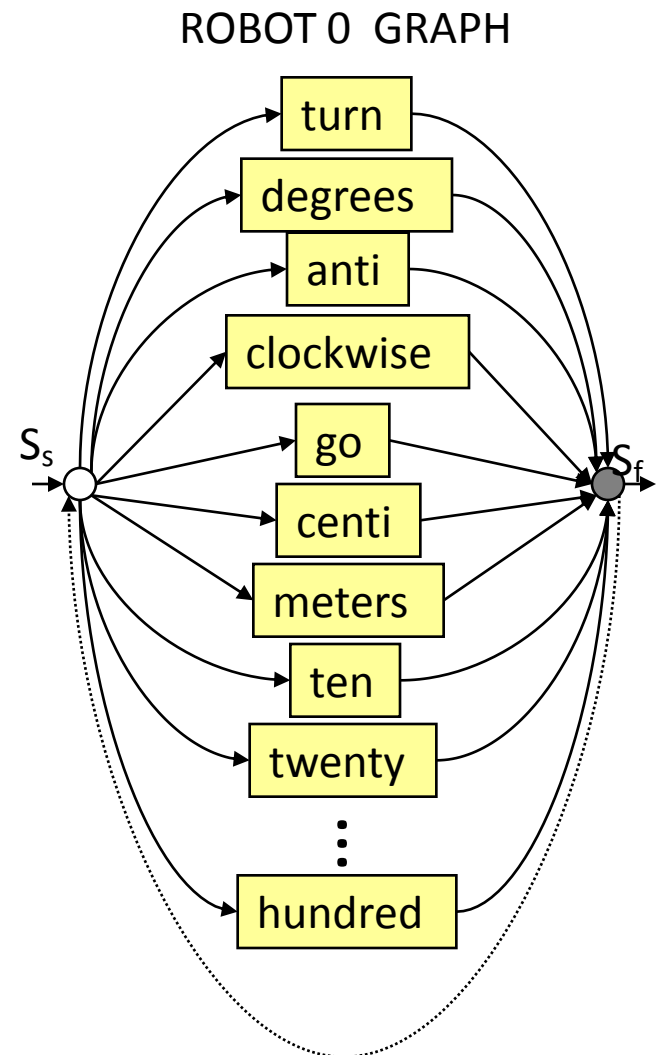


A More Complex Example

- 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?

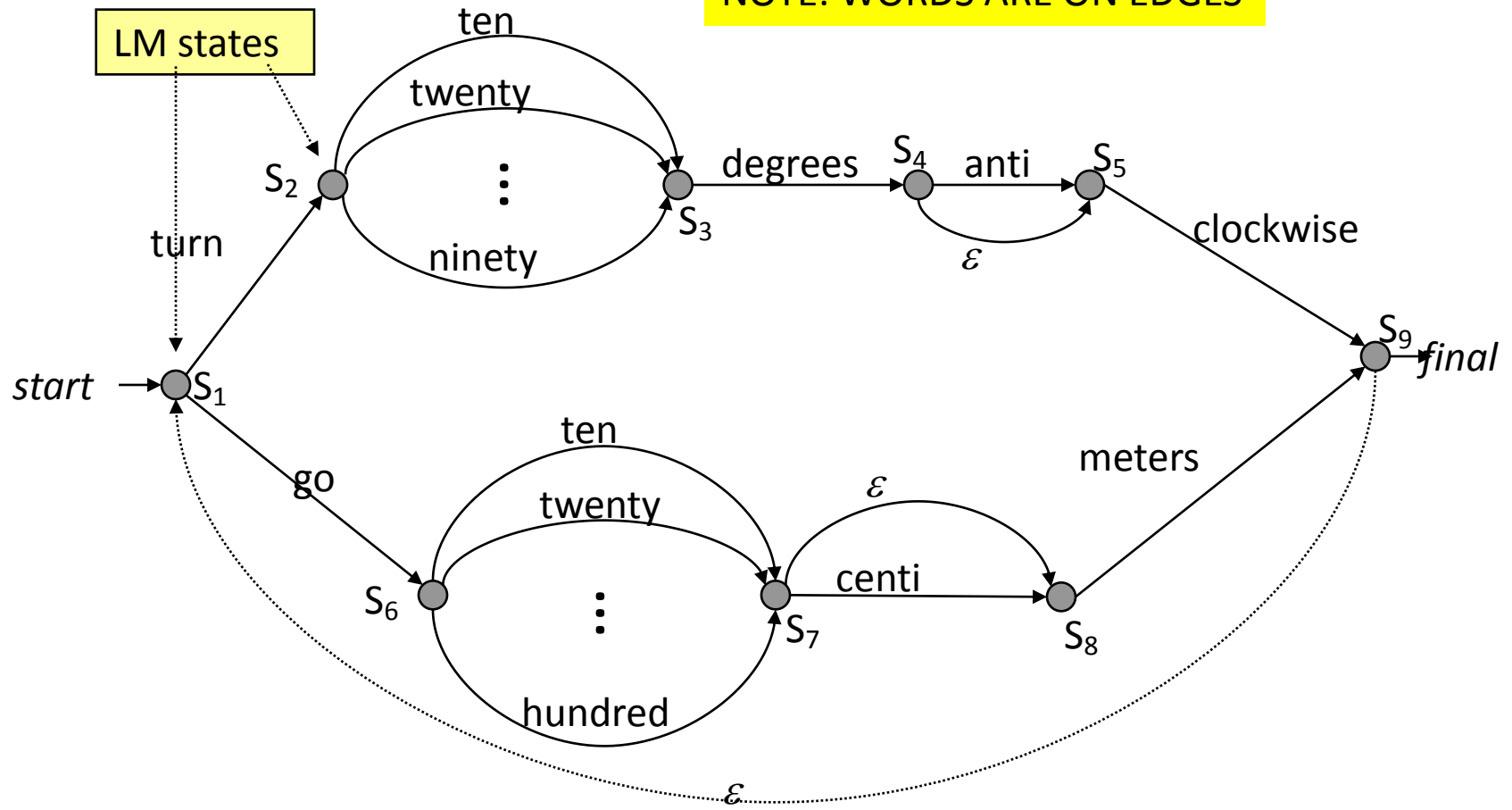
A More Complex Example

- 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 ϵ transitions
- 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*

The All-Word Model

- Is the “any word can follow any word” model also an FSG?
(ROBOT0)

Decoding with Finite State Grammars

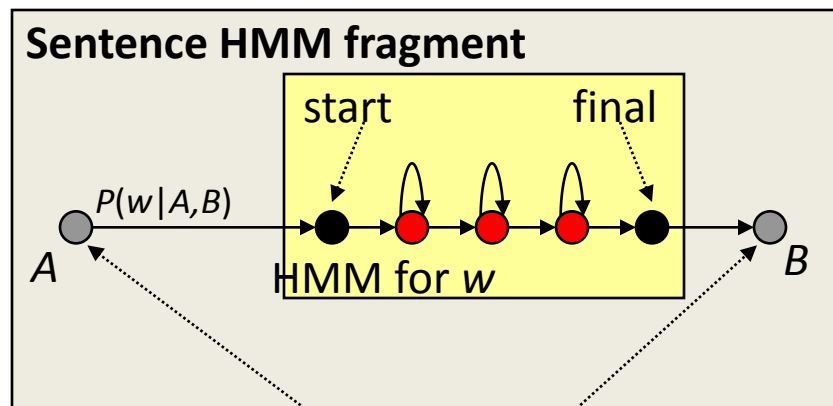
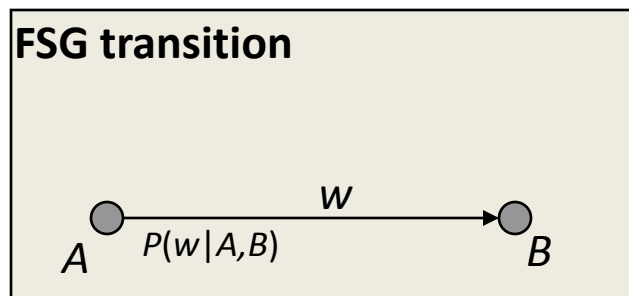
- How can we incorporate our *ROBOT1* FSG into the Viterbi decoding scheme?

Decoding with Finite State Grammars

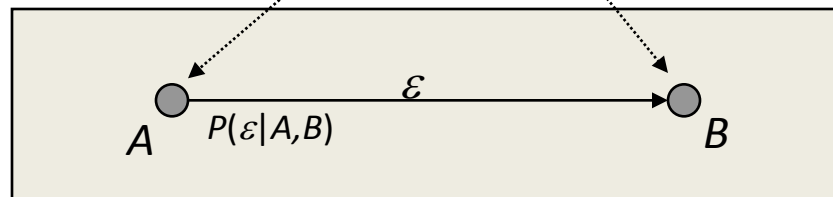
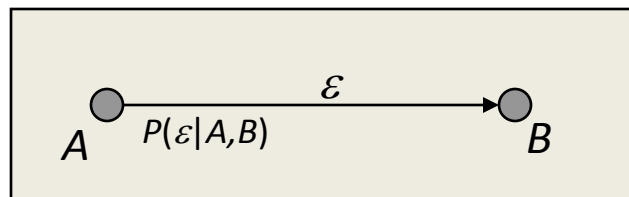
- Construct a *sentence HMM* from the given FSG
 - Replace edges in the FGS with the HMMs for words
 - We are now in familiar territory
 - Apply the standard time synchronous Viterbi search
 - Only modification needed: need to distinguish between LM states (see later)
- First, how do we construct the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

Sentence HMMs from FSGs

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



Non-emitting states created to represent FSG states

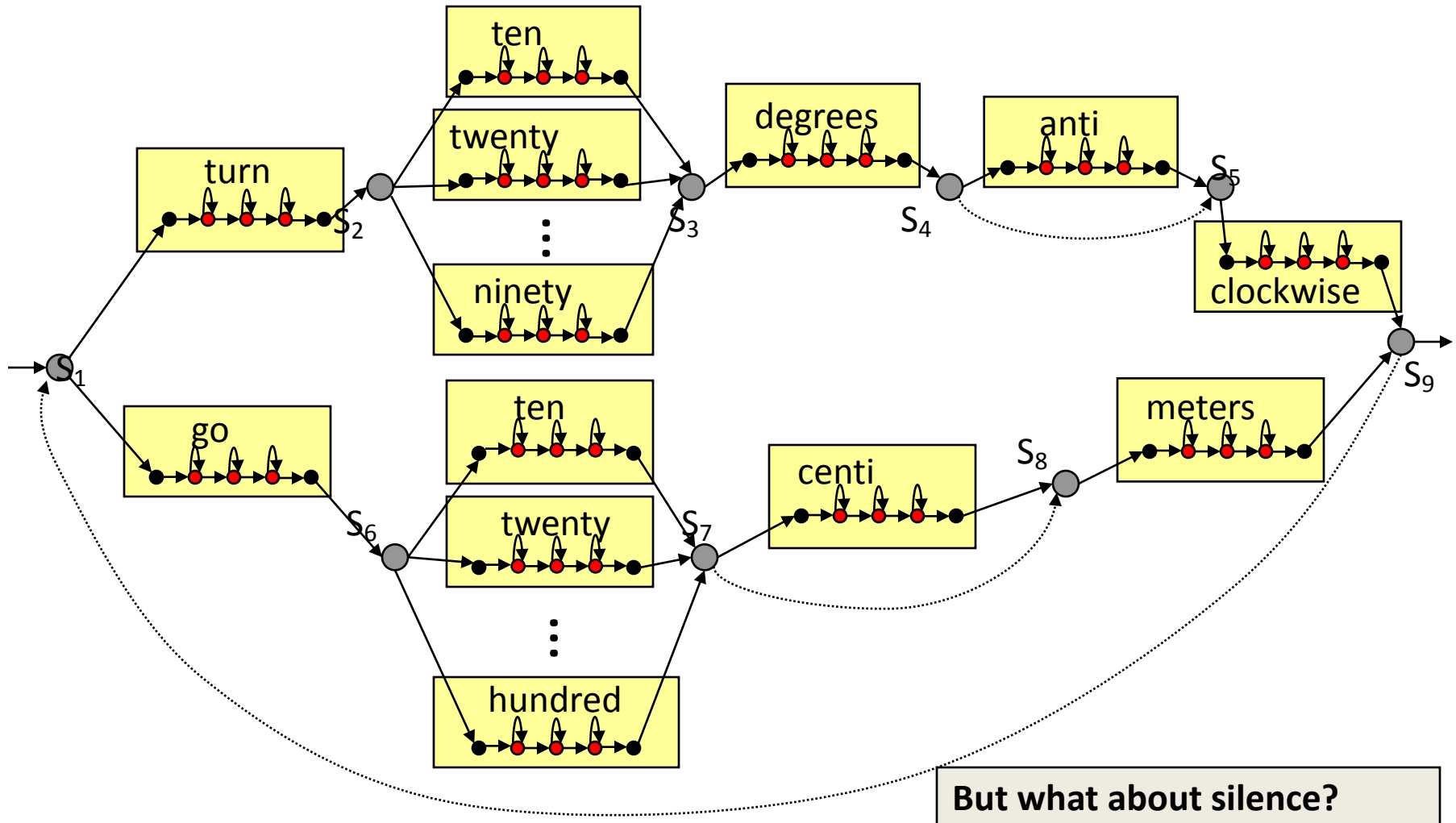


Sentence HMMs from FSGs (contd.)

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

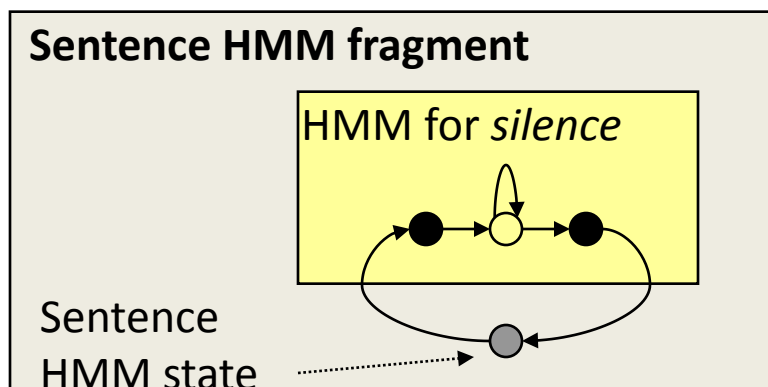
Robot Control (*ROBOT1*) Sentence HMM

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

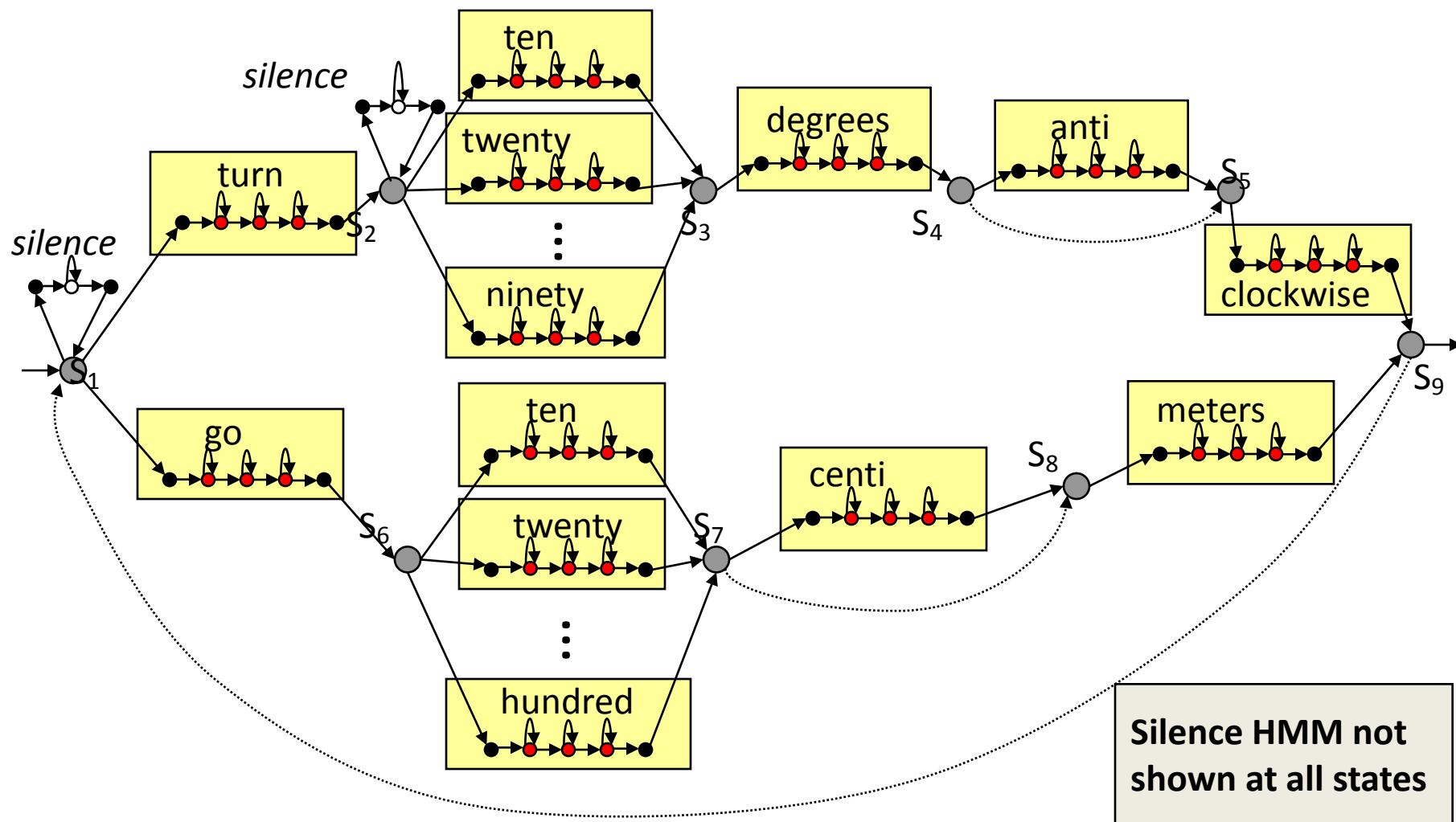


Sentence HMMs from FSGs (contd.)

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

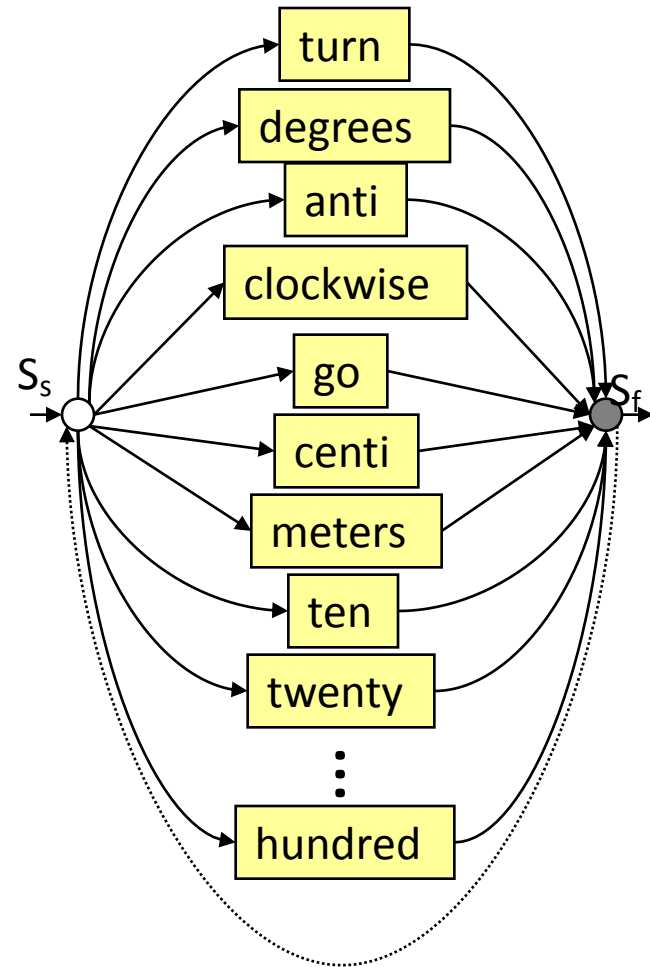


ROBOT1 with Optional Silences



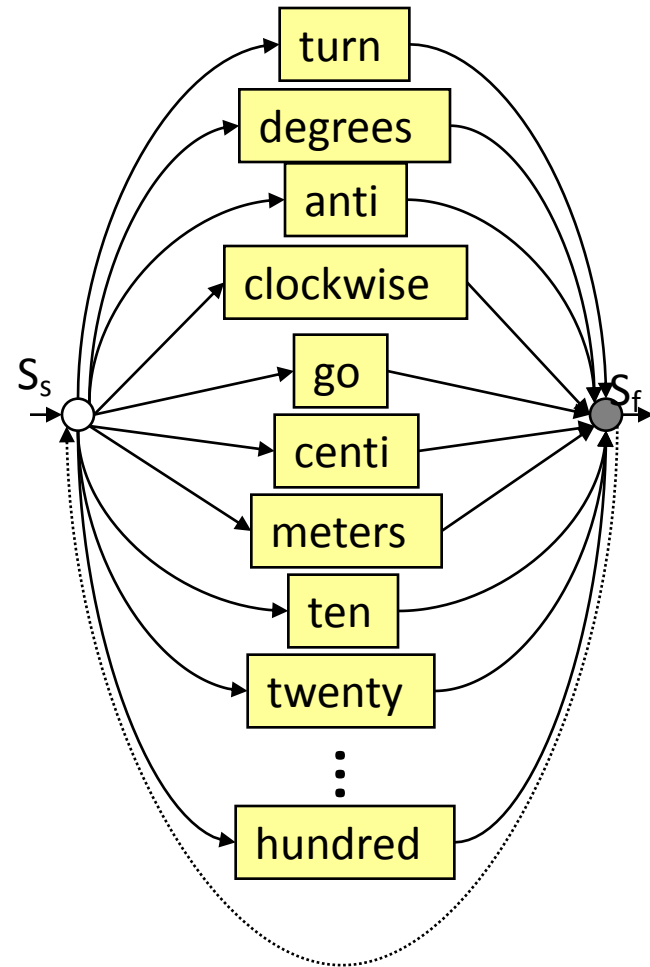
Trellis Construction for *ROBOT0*

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

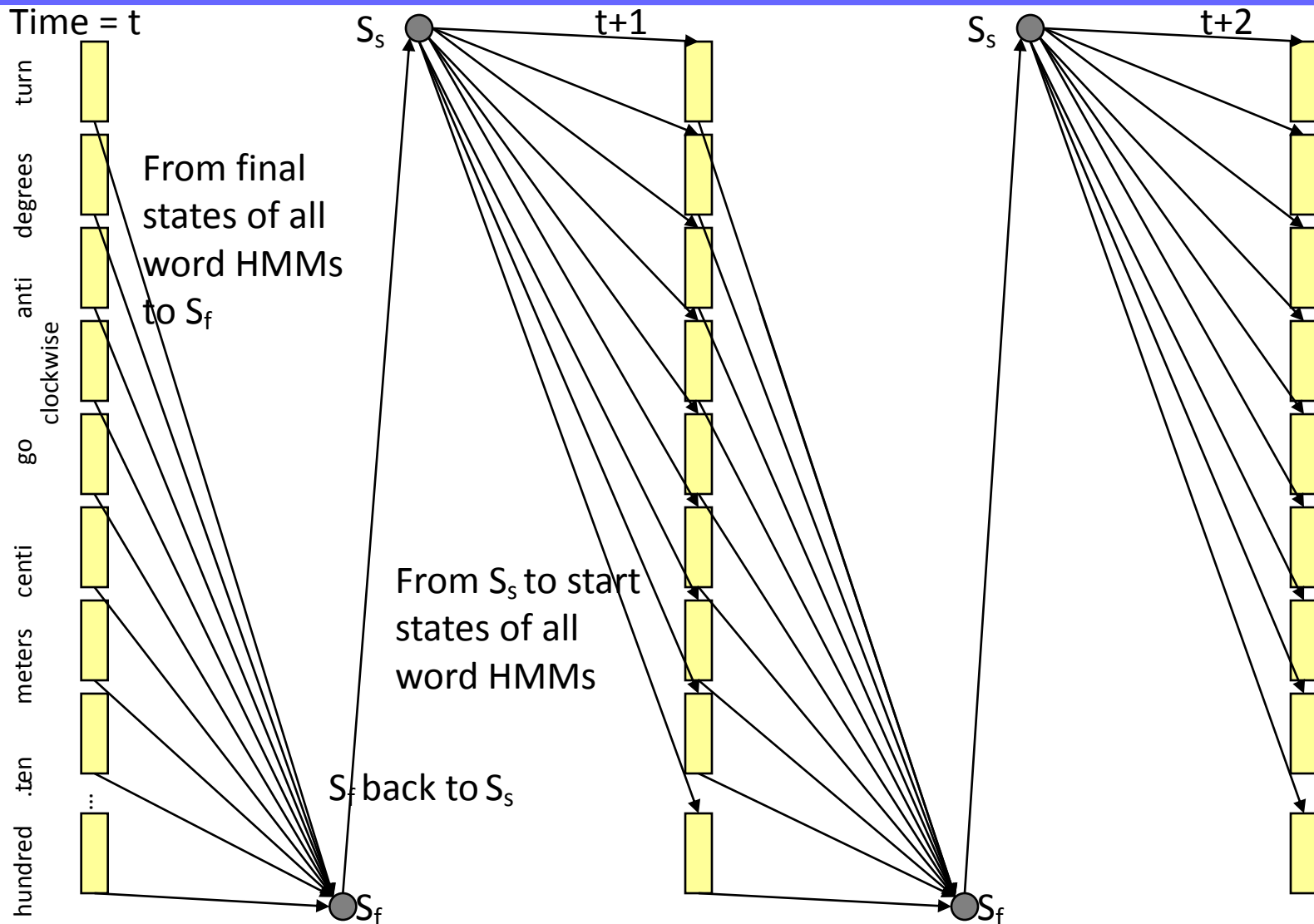


Trellis Construction for *ROBOT0*

- What are the cross-word transitions in the trellis?
 - (More accurately, word-exit and word-entry transitions)

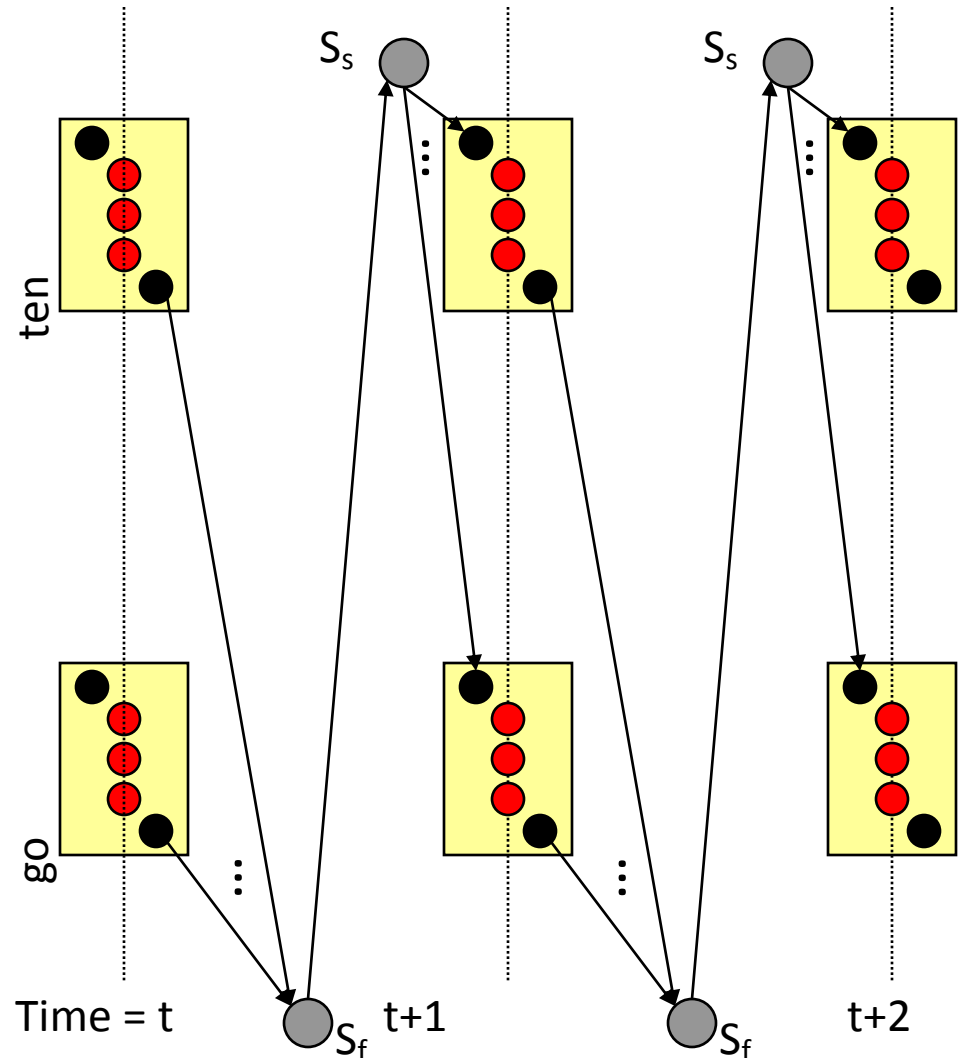


ROBOT0 Cross Word Transitions



ROBOT0 Cross Word Transitions

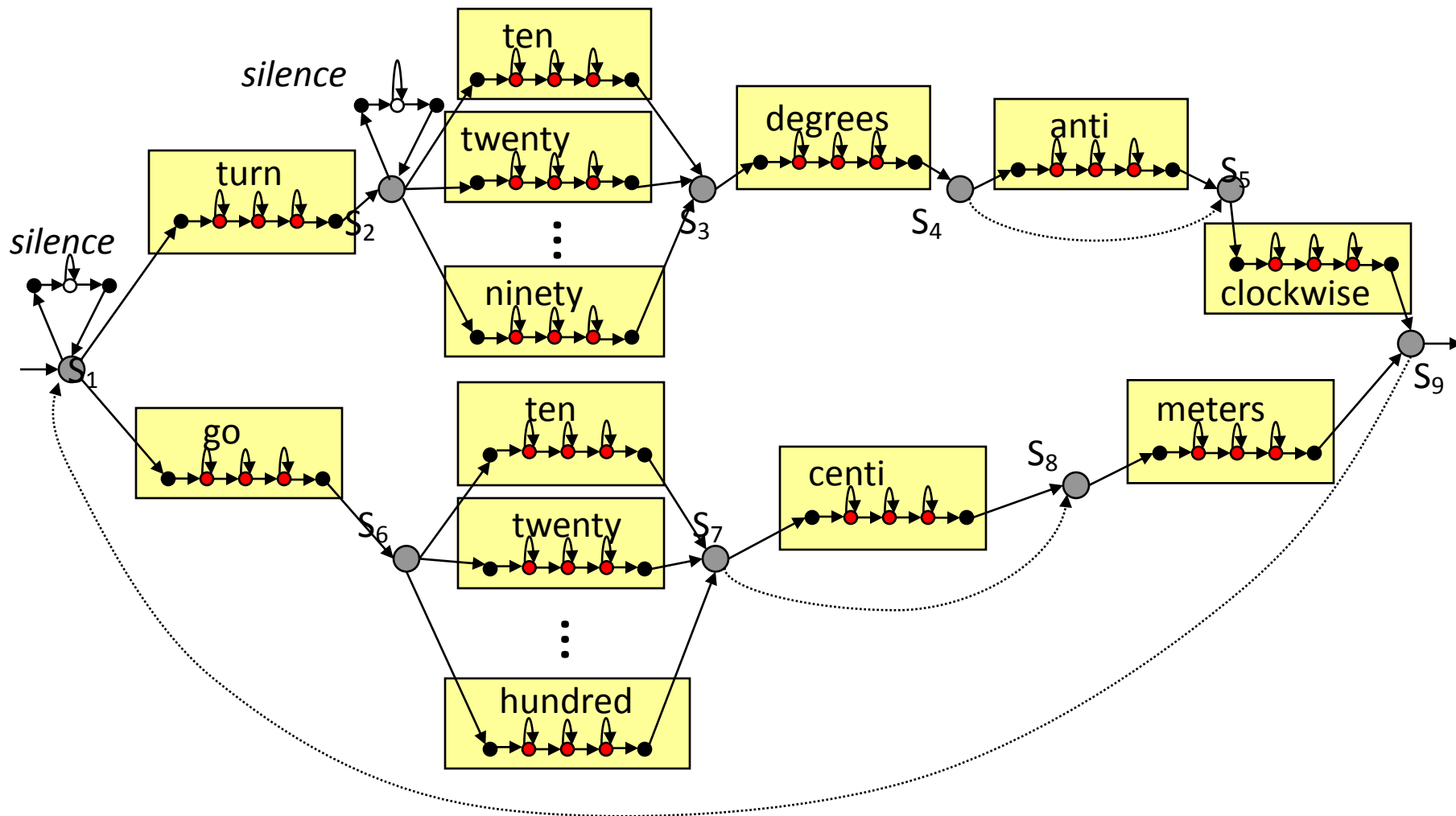
- A portion of trellis shown between time t and $t+2$
- Similar Transitions happen from final states of all 17 words to start states of all 17 words
- Non-emitting states shown “between frames”
 - Order them as follows:
 - Find all null state sequences
 - Make sure there are no cycles
 - Order them by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



Trellis Construction for *ROBOT1*

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

ROBOT1 Sentence HMM

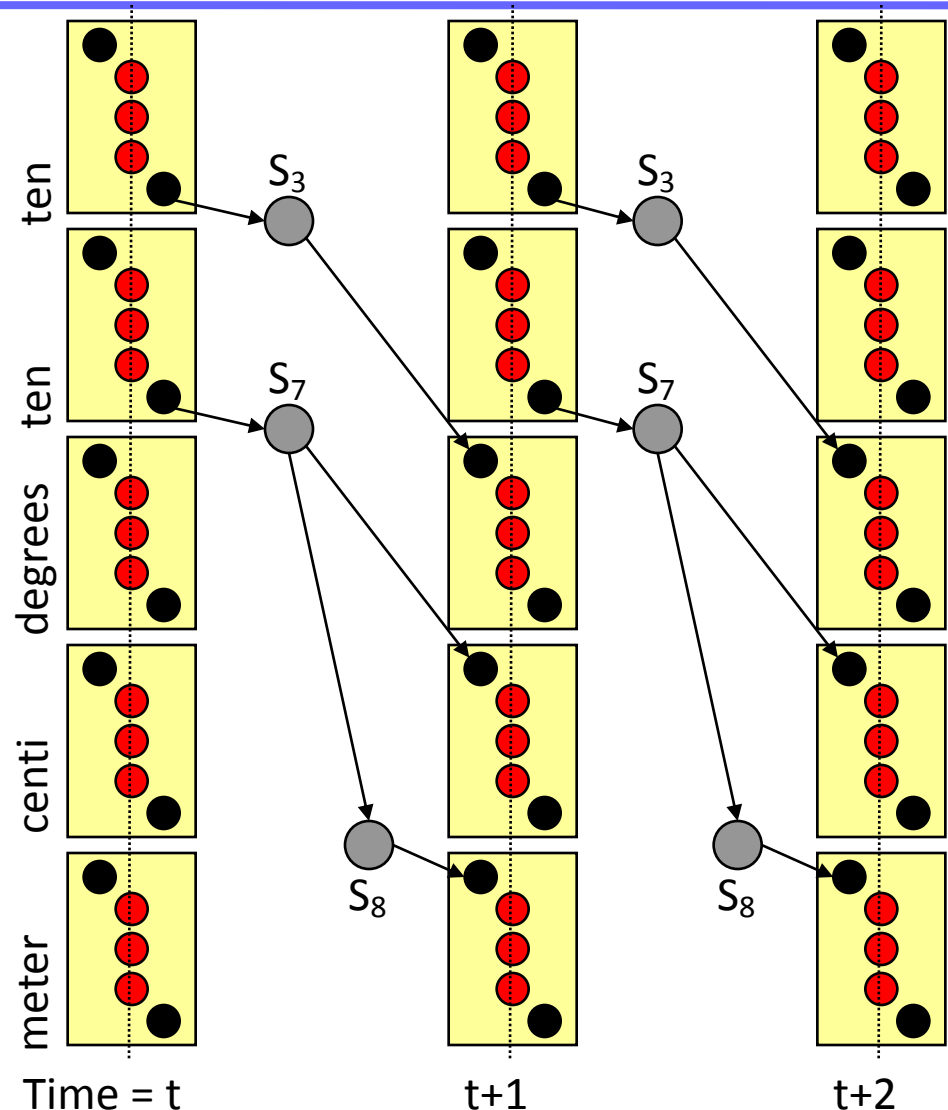


Trellis Construction for *ROBOT1*

- No. of trellis rows = No. of states in sentence 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

- The previous example brings forth an important point
 - FSG states (LM states in general) are distinct, and need to be preserved during decoding
 - If the same word is emitted by two different transitions (*i.e.* either the source or destination states are different), there are actually two 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

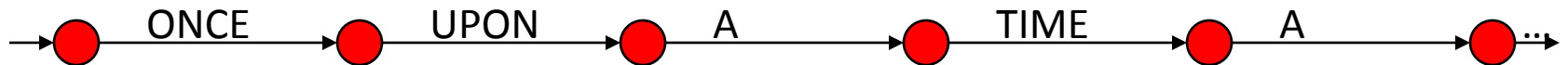
- A child reads a story aloud, one sentence at a time
- The automated tutor “listens” to the child and tries to help if it has any difficulty
 - Pausing too long at a word
 - Misreading a word
 - Skipping a word



- 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

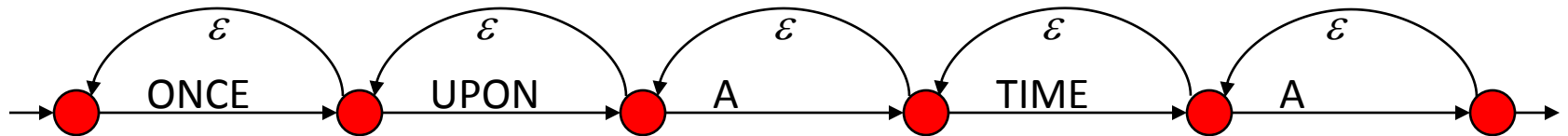
Example Application: Reading Tutor

- For each sentence, the tutor builds a new FSG
- Let's say the current sentence is:
 - 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



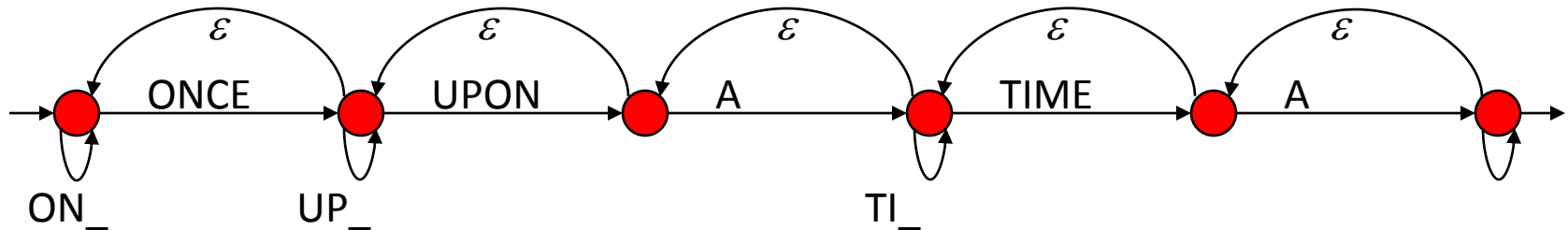
Example Application: Reading Tutor

- 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!



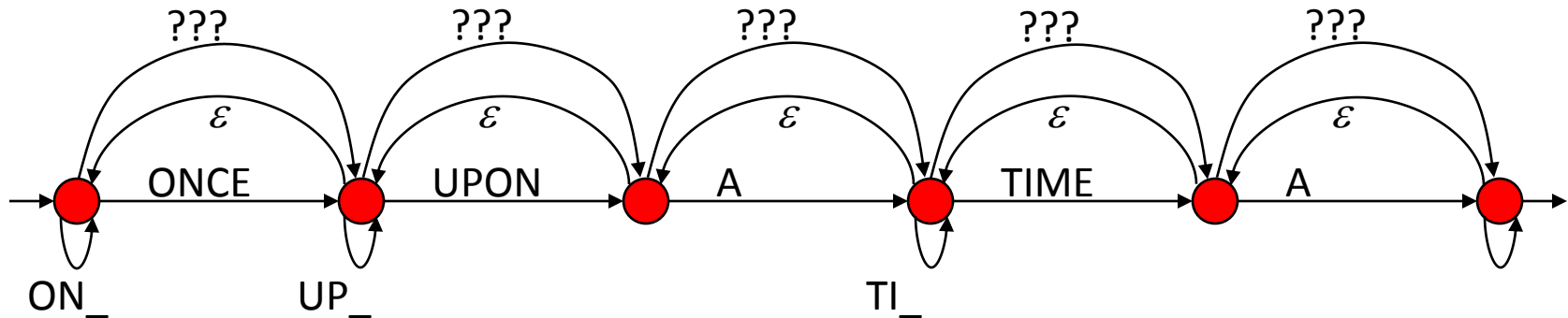
Example Application: Reading Tutor

- 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



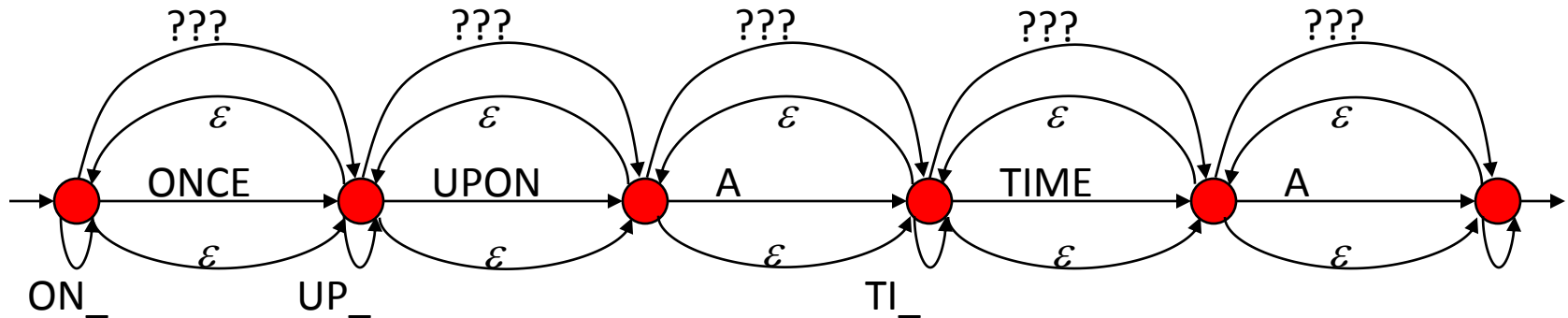
Example Application: Reading Tutor

- 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



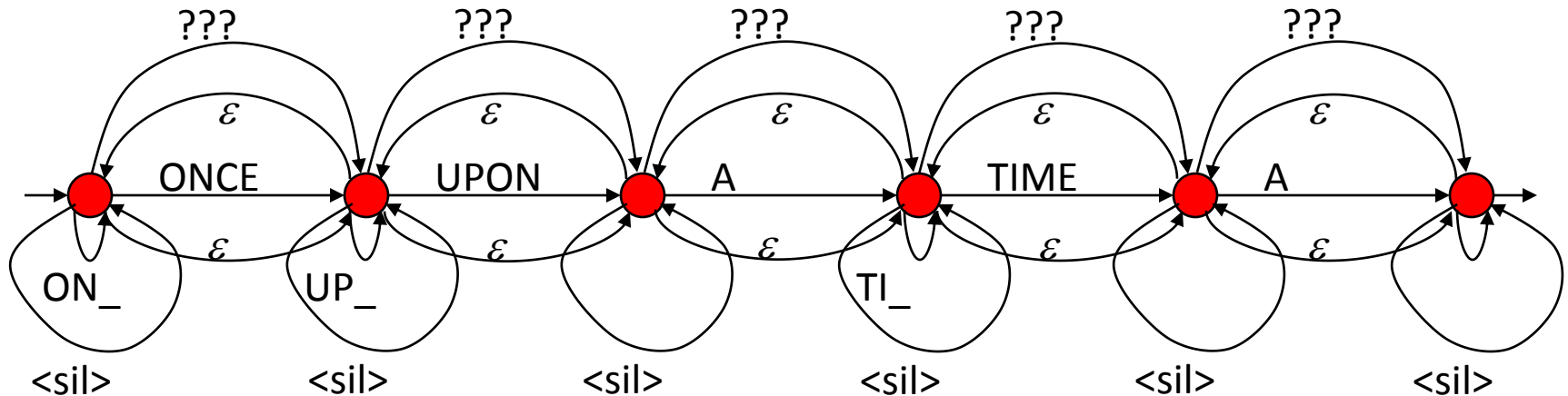
Example Application: Reading Tutor

- 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 continuing from the middle of the sentence after being helped
 - 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

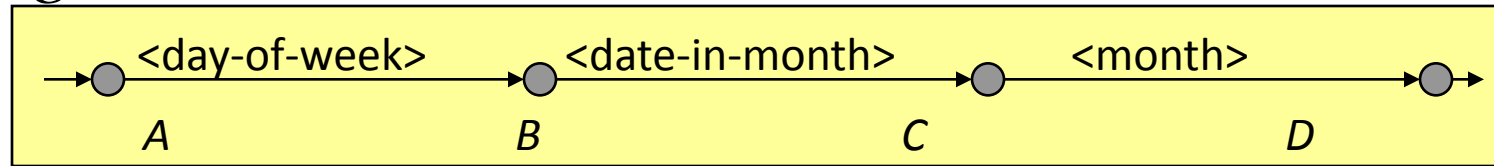
- Before we talk about this, let us consider something else first

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

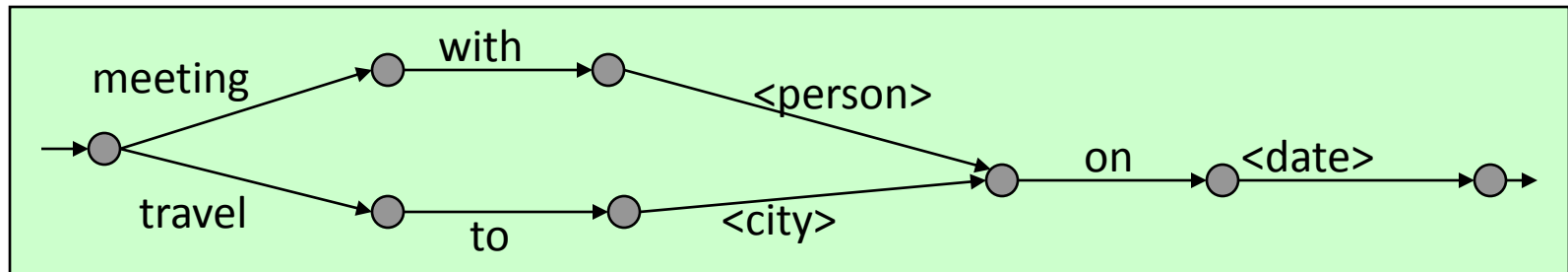
- *E.g.* here is a *<date>* FSG:



- 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

More Nested FSGs

- *Example: Scheduling task*
 - (Transition labels with $\langle \rangle$ actually refer to other FSGs)



- The $\langle \text{date} \rangle$ 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

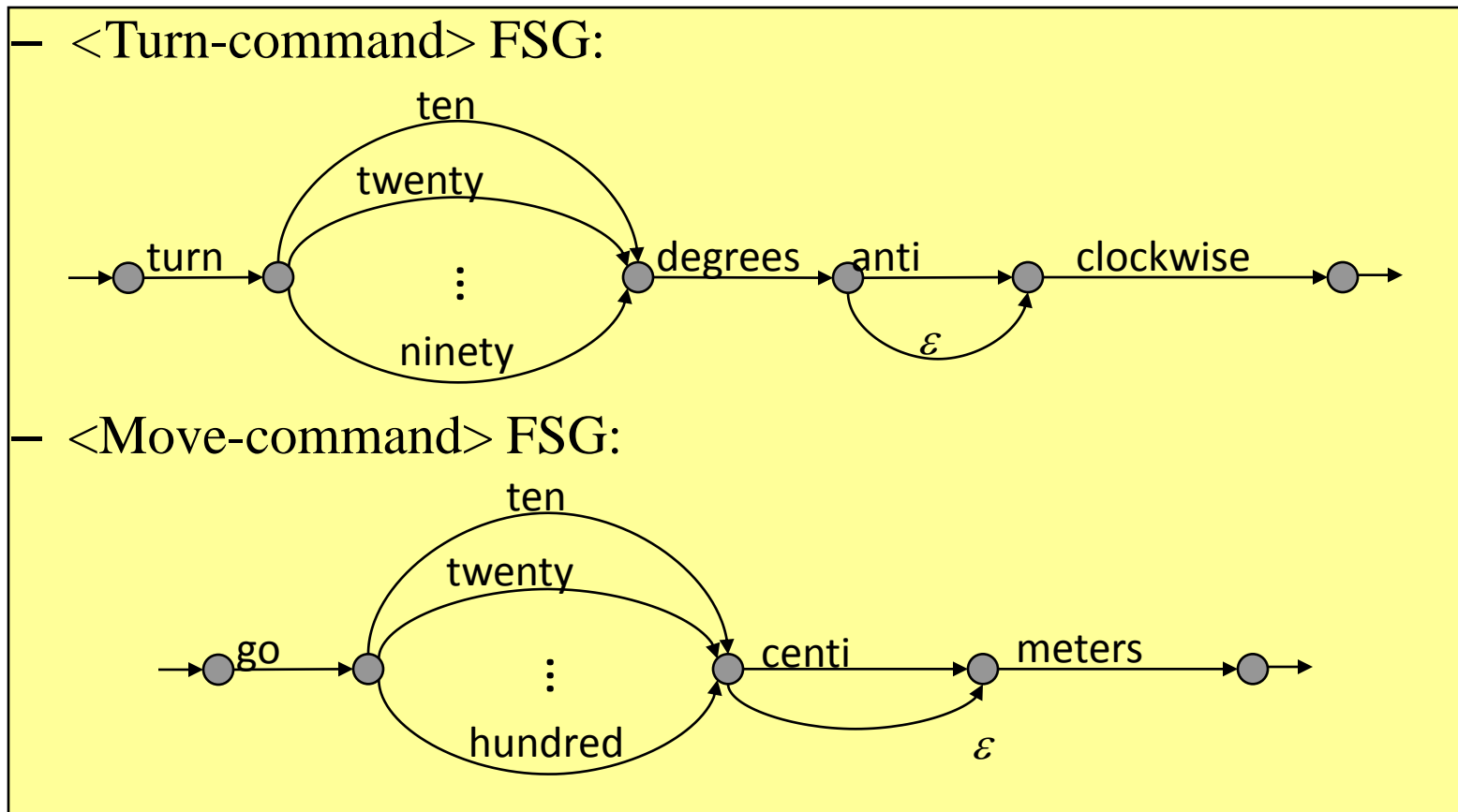
- In the case of the above scheduling task FSG, it is possible to flatten it into a regular FSG (*i.e.* without references to other FSGs) simply by embedding the target FSG in place of an FSG transition
 - Very similar to the generation of sentence HMMs from FSGs
- At this point, the flattened FSG can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

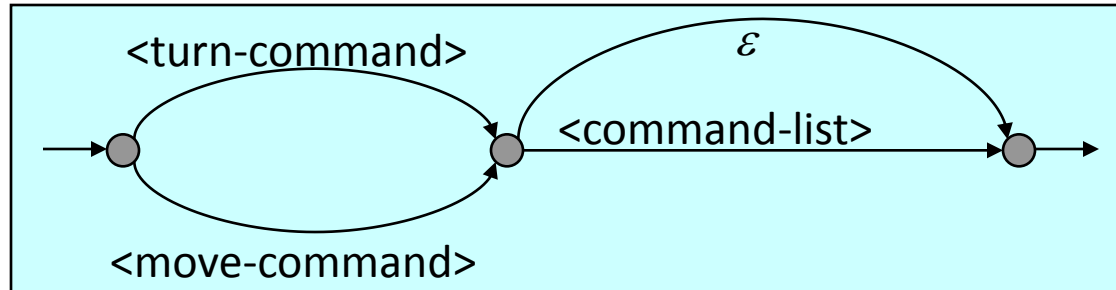
- It is possible for a grammar definition to refer to *itself*
- Let us consider the following two basic FSGs for robot control:



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

Recursion in Grammar Networks (contd.)

- Recursion can be *direct*, or *indirect*
 - <command-list> grammar is defined directly in terms of itself
 - Indirect recursion occurs when we can find a sequences of grammars, $F_1, F_2, F_3, \dots, F_k$, such that:
 - F_1 refers to F_2 , F_2 refers to F_3 , etc., and F_k refers back to F_1
- Problem with recursion:
 - It is not always possible to simply blindly expand a grammar by plugging in the component grammars in place of transitions
 - Leads to infinite expansion

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: $\langle \text{non-terminal} \rangle ::= \varepsilon$
- 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 | RAJ ...
 - <article> ::= A | AN | THE
 - <noun> ::= BALL | BAT | FRUIT | BOOK ...
 - <verb> ::= EAT | RUN | HIT | READ ...

- Clearly, the language allows non-sensical 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
- $\langle \text{command-list} \rangle$ is the CFG being defined (= S):

```
 $\langle \text{command-list} \rangle ::= \langle \text{turn-command} \rangle \mid \langle \text{move-command} \rangle$   
 $\langle \text{command-list} \rangle ::= \langle \text{turn-command} \rangle \langle \text{command-list} \rangle$   
 $\langle \text{command-list} \rangle ::= \langle \text{move-command} \rangle \langle \text{command-list} \rangle$   
  
 $\langle \text{turn-command} \rangle ::= \text{TURN } \langle \text{degrees} \rangle \text{ DEGREES } \langle \text{direction} \rangle$   
 $\langle \text{direction} \rangle ::= \text{clockwise} \mid \text{anti clockwise}$   
 $\langle \text{move-command} \rangle ::= \text{GO } \langle \text{distance} \rangle \langle \text{distance-units} \rangle$   
 $\langle \text{distance-units} \rangle ::= \text{meters} \mid \text{centi meters}$   
 $\langle \text{degrees} \rangle ::= \text{TEN} \mid \text{TWENTY} \mid \text{THIRTY} \mid \text{FORTY} \mid \dots \mid \text{NINETY}$   
 $\langle \text{distance} \rangle ::= \text{TEN} \mid \text{TWENTY} \mid \text{THIRTY} \mid \dots \mid \text{HUNDRED}$ 
```

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

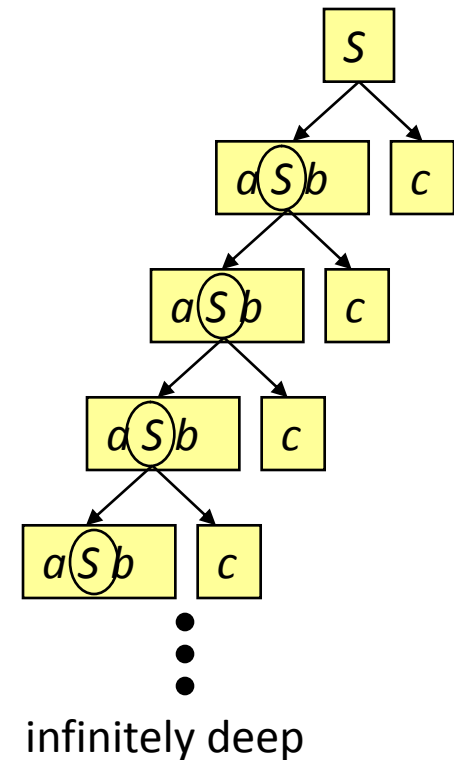
Context Free Grammars: Another View

- 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

$S ::= aSb \mid c$

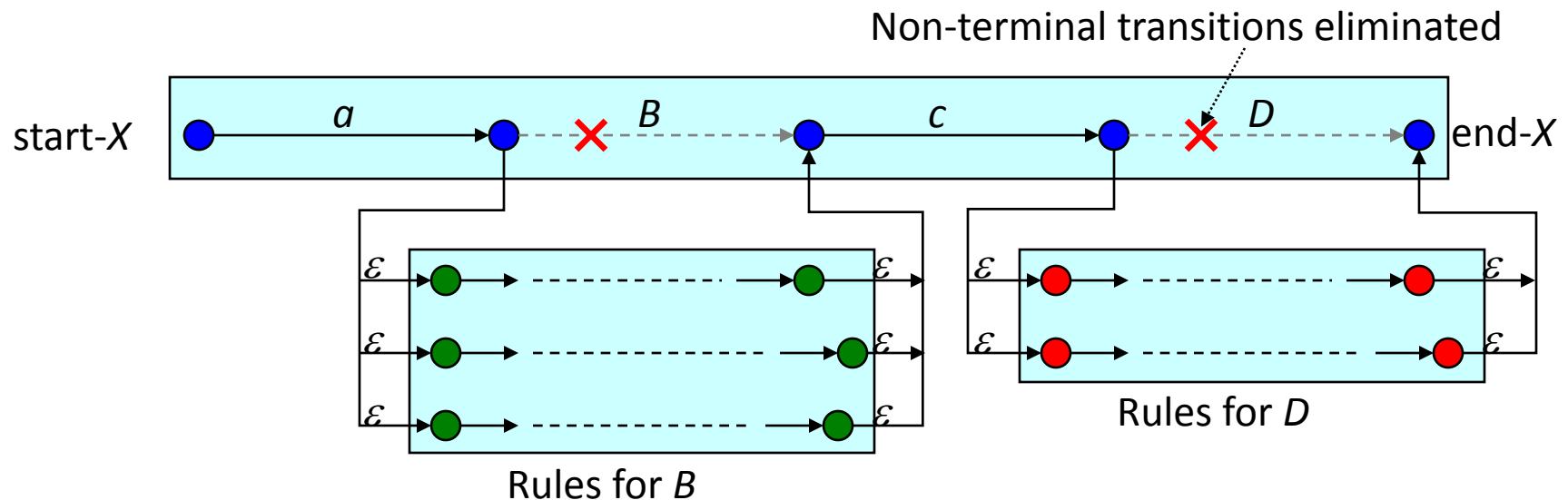


Approximating a CFG by an FSG

- 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:

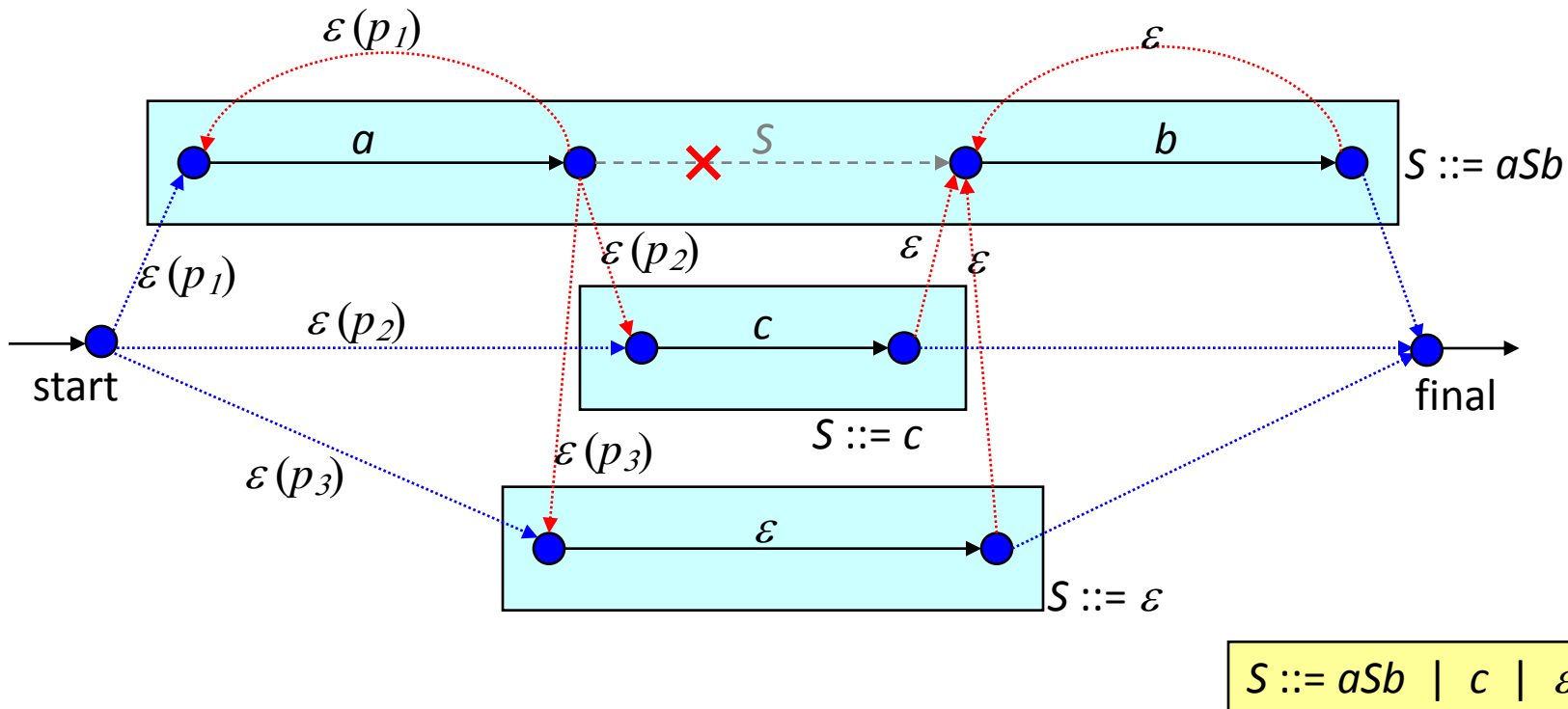
- Consider a rule: $X ::= aBcD$, where a and c are terminal symbols (words), and B and D are non-terminals
- We can create the following FSG for the rule:



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

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:



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 ε 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

Why is this FSG an Approximation?

- To exactly follow the rules of the CFG, when a non-terminal is replaced by null transitions to and from its rules, we would need to ensure that for every transition to a rule, there is a return from the rule
- In the created FSG, there is no way to enforce the above requirement
 - 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
- *Q*: How are the CFG and FSG on the last slide different?

Another FSG Approximation to CFGs

- Another possibility is to eliminate the root of the CFG decoding problem: 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
- Consider the function call analogy and the “call tree example”
- We only need to explore a *finite* sized tree
- A finite sized search problem can be turned into an FSG!
 - Proof? Construction algorithm?
- 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

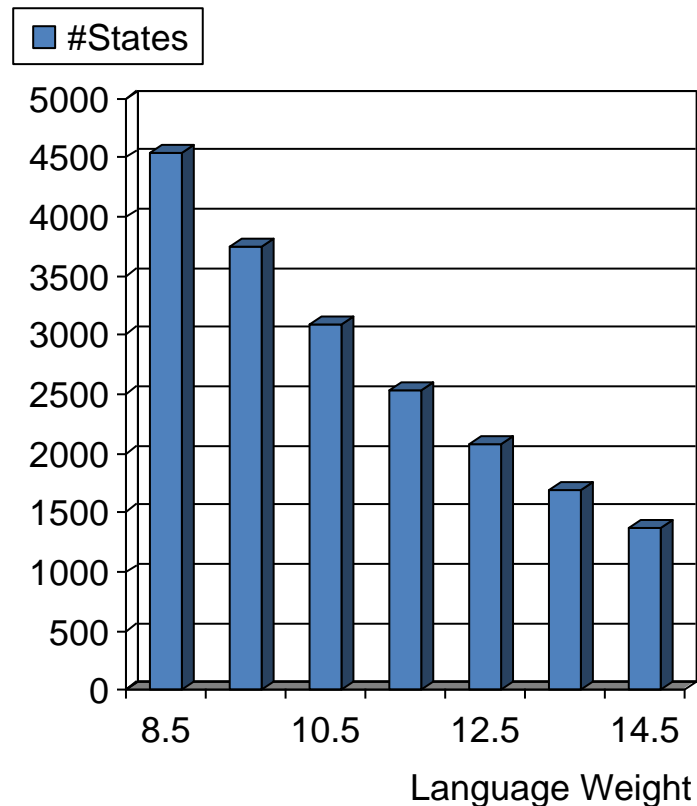
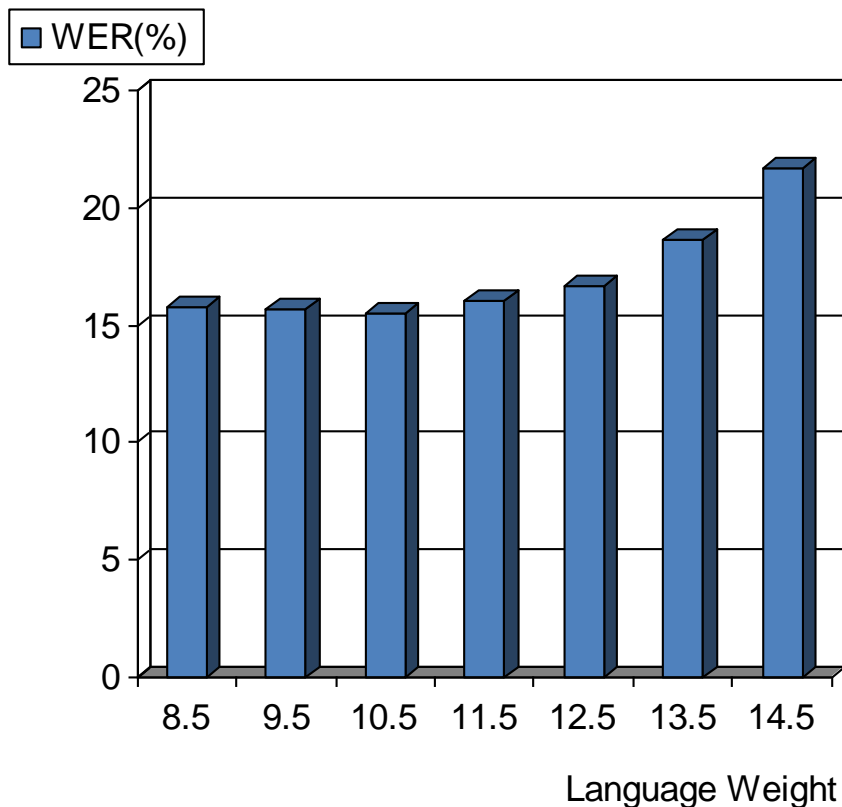
- According to the basic speech recognition equation, we wish to maximize: $P(X|W) P(W)$ over all word sequences W
- In practice, it has been found that left in this form, the language model (*i.e.* $P(W)$) has little effect on accuracy
- Empirically, it has been found necessary to maximize: $P(X|W)P(W)^k$, 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
- SRGS (Speech Recognition Grammar Specification) is a proposed W3C standard
 - Specifies the format in which CFG grammars may be input to a speech recognizer
- In addition to the plain grammar specification, SRGS allows the CFGs to perform a few other functions
- For details: <http://www.w3.org/TR/speech-grammar/>

Summary

- Language models are essential for recognition accuracy
- LMs can be introduced into the decoding framework using the standard speech equation
- The formula for $P(w_1, w_2, w_3, \dots, w_n)$ naturally leads to the notion of N-gram grammars for language models
- However, N-gram grammars have to be trained
- When little or no training data are available, one can fall back on structured grammars based on expert knowledge
- Structured grammars are of two common types: finite state (FSG) and context free (CFG)
- CFGs obtain their power and appeal from their ability to function as building blocks
- FSGs can be easily converted into sentence HMM for decoding
- CFGs are much harder to decode exactly
- However, CFGs can be approximated by FSGs by making some assumptions

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

Backup slides

The Fundamental Speech Recognition Problem

- Fundamental problem of speech recognition:

Given input speech $X = x_1, x_2, x_3, \dots, x_T$, find the most likely word sequence $W = w_1, w_2, w_3, \dots, w_n$

$$\text{i.e. } \operatorname{argmax}_W P(W|X)$$

$$\begin{aligned} P(W|X) &= P(W,X)/P(X) & P(X|W) &= \\ P(X,W)/P(W) & P(W|X)P(X) &= \\ P(X|W)P(W) & & \end{aligned}$$

- By Bayes' rule: $P(W|X) = P(X|W)P(W)/P(X)$
- So, the above expression becomes

$$\operatorname{argmax}_W P(W|X) = \operatorname{argmax}_W (P(X|W) P(W)) / P(X)$$

- For finding W that maximizes $P(W|X)$, X is constant and $P(X)$ can be ignored (remember, X is the given speech input)
- Thus, we are finally left with the fundamental equation:

$$\operatorname{argmax}_W P(W|X) = \operatorname{argmax}_W (P(X|W) P(W))$$

Breaking Down the Fundamental Equation

$$\operatorname{argmax}_W P(W|X) = \operatorname{argmax}_W (P(X|W) P(W))$$

- $P(W|X)$ = *posterior probability* of the word sequence W given the speech signal X
 - We wish to find the W with maximum posterior probability
- $P(X|W)$ = acoustic likelihood of the word sequence W producing the observed speech signal X
 - This is obtained from the *acoustic model* (forward algorithm)
- $P(W)$ = *language model probability* of the word sequence W
 - We now have the desired formalism for using LMs
 - But, what is $P(W)$?
 - *E.g.* what is P (“speech recognition is so much fun”)?

Interpretation of $P(W)$

- We can rewrite $P(w_1, w_2, w_3, \dots, w_n)$ as:

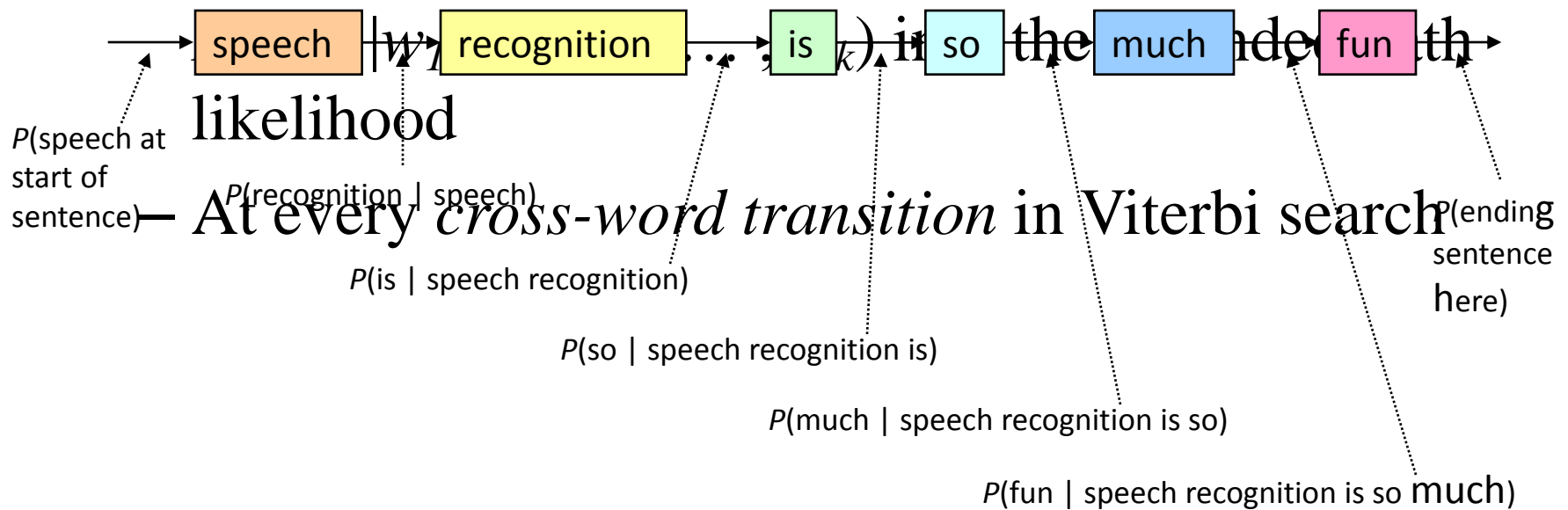
$$= P(w_1) * P(w_1w_2)/P(w_1) * P(w_1w_2w_3)/P(w_1w_2) * P(w_1w_2w_3w_4)/P(w_1w_2w_3) \dots$$

$$= P(w_1) P(w_2/w_1) P(w_3|w_1, w_2) P(w_4|w_1, w_2, w_3) \dots$$

(Use the rule from probability theory: $P(a/b) = P(a,b) / P(b)$)

Interpretation of $P(W)$

- How can we use this in decoding?
 - Whenever we consider extending a partial path $w_1, w_2, w_3, \dots, w_k$ by another word w_{k+1} , we incorporate the probability



Language Model State

- In the expression $P(w_{k+1}|w_1, w_2, w_3, \dots, w_k)$, the sequence $w_1, w_2, w_3, \dots, w_k$ is known as the *LM state*, *history*, or *grammar state*
 - Like HMM states, we have LM states
- Building an LM implies computing probability values for all possible words, all possible LM states!
- We can estimate these distributions from *LM training data*
- However, as k (the history length) grows larger, the number of possible histories grows exponentially
 - Even for a small vocabulary of 10 words, a 10-word history contains 10^{10} possibilities!
 - Hopeless to try to estimate or store $P(w_i|history)$ for 10^{10} histories

Language Model State Approximation

- Hence, histories (LM states) are often approximated by truncating them to a few *most recent* words:

- E.g. only the most recent one-word history:

$$P(w_{k+1}|w_1, w_2, w_3, \dots, w_k) \sim P(w_{k+1}|w_k)$$

e.g.: $P(\text{fun} | \text{speech recognition is so much}) \sim P(\text{fun} | \text{much})$

- These are 2-gram or *bigram* grammars

- Or, the most recent two-word history:

$$P(w_{k+1}|w_1, w_2, w_3, \dots, w_k) \sim P(w_{k+1}|w_{k-1}, w_k)$$

e.g.: $P(\text{fun} | \text{speech recognition is so much}) \sim P(\text{fun} | \text{so much})$

- These are 3-gram or *trigram* grammars

- And there are 4-grams, 5-grams etc.

- No longer an exact Bayesian solution, but efficient!

Building N-gram Grammars

- N-gram LMs are suited for large vocabulary applications
- However, N-gram LMs require training
 - A large training corpus provides estimates of the history conditional probabilities (*i.e.* bigram and trigram probabilities)
- We will study these later in the course

Finite State and Context Free Grammars

- For many applications:
 - Either no training data exists, or
 - The allowed set of sentences is much more *structured* and can be described concisely from *expert knowledge*
- Grammar states can be explicitly specified without being implicitly defined by histories
- Most frequently used grammar types:
 - *Finite state* and *context free* grammars (FSGs and CFGs)

