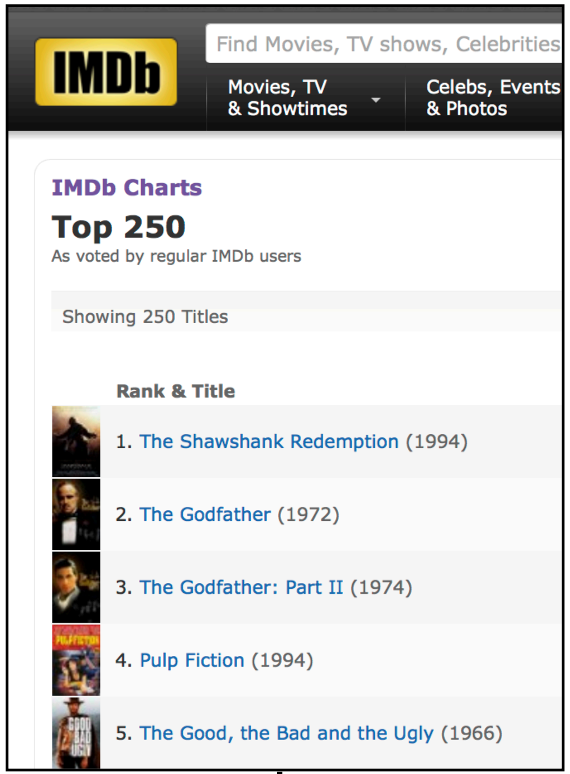
VI.3 Rule-Based Information Extraction

- Goal: Identify and extract **unary, binary, or *n*-ary relations as facts** embedded in **regularly structured text**, to generate entries in a schematized database

- **Rule-driven regular expression matching**
  - Interpret documents from source (e.g., Web site to be wrapped) as **regular language**, and specify/infer rules for matching specific types of facts



Title	Year
The Shawshank Redemption	1994
The Godfather	1972
The Godfather - Part II	1974
Pulp Fiction	1994
The Good, the Bad, and the Ugly	1966

LR Rules

- **L token** (left neighbor) fact token **R token** (right neighbor)
  pre-filler pattern *filler pattern* *post-filler pattern*

- Example:
  L = `<B>`, **R** = `</B>`
  → *Country*
  L = `<I>`, **R** = `</I>`
  → *Code*
  produces relation with
  tuples `<Congo, 242>, <Egypt, 20>, <France, 30>`

```
<HTML>
    <TITLE>Some Country Codes</TITLE>
<BODY>
    <B>Congo</B><I>242</I><BR>
    <B>Egypt</B><I>20</I><BR>
    <B>France</B><I>30</I><BR>
</BODY>
</HTML>
```

- Rules are often very specific and therefore combined/generalized

- Full details: RAPIER [Califf and Mooney ’03]

Advanced Rules: HLRT, OCLR, NHLRT, etc.

- Idea: Limit application of LR rules to proper context (e.g., to skip over HTML table header)

```
<TABLE>
    <TR><TH><B>Country</B></TH><TH><I>Code</I></TH></TR>
    <TR><TD><B>Congo</B></TD><TD><I>242</I></TD></TR>
    <TR><TD><B>Egypt</B></TD><TD><I>20</I></TD></TR>
    <TR><TD><B>France</B></TD><TD><I>30</I></TD></TR>
</TABLE>
```

- **HLRT rules** (head left token right tail)
  apply LR rule only if inside HT (e.g., H = `<TD>` T = `</TD>`)

- **OCLR rules** (open (left token right)* close):
  O and C identify tuple, LR repeated for individual elements

- **NHLRT** (nested HLRT):
  apply rule at current nesting level,
  open additional levels, or return to higher level

Learning Regular Expressions

- Input: Hand-tagged examples of a regular language
- Learn: (Restricted) regular expression for the language of a finite-state transducer that reads sentences of the language and outputs token of interest
- Example:

  This apartment has ***3*** *bedrooms.
 The monthly rent is $* ***995***.
  This apartment has ***4*** *bedrooms.
 The monthly rent is $* ***980***.
  The number of bedrooms is ***2***. *
 The rent is $* ***650*** *per month.*

  yields * **<digit>** * "
" * "$" **<digit>+** * as learned pattern

- Problem: Grammar inference for full-fledged regular languages is hard. Focus therefore often on restricted class of regular languages.

- Full details: WHISK [Soderland '99]

Properties and Limitations of Rule-Based IE

- Powerful for wrapping **regularly structured web pages** (e.g., template-based from same deep web site)

- Many **complications** with real-life HTML (e.g., misuse of tables for layout)

- Flat view of input limits the same annotation
  - Consider **hierarchical document structure** (e.g., DOM tree, XHTML)
  - Learn extraction patterns for restricted regular languages (e.g., combinations of XPath and first-order logic)

- **Regularities with exceptions** are difficult to capture
  - Learn positive and negative cases (and use statistical models)

Additional Literature for VI.3

- **M. E. Califf and R. J. Mooney**: *Bottom-Up Relational Learning of Pattern Matching Rules for Information Extraction*, JMLR 4:177-210, 2003

- **S. Soderland**: *Learning Information Extraction Rules for Semi-Structured and Free Text*, Machine Learning 34(1-3):233-272, 1999

VI.4 Learning-Based Information Extraction

- For heterogeneous sources and for natural-language text
  - **NLP techniques** (PoS tagging, parsing) for tokenization
  - **Identify patterns** (regular expressions) as features
  - **Train statistical learners** for segmentation and labeling (e.g., HMM, CRF, SVM, etc.) augmented with lexicons
  - Use learned model to **automatically tag** new input sequences
- Training data:

The ***WWW conference*** *takes place in* ***Banff*** *in* ***Canada***
Today's keynote speaker is ***Dr. Berners-Lee*** *from* ***W3C***
The panel in ***Edinburgh*** *chaired by* ***Ron Brachman*** *from* ***Yahoo!***

with **event**, **location**, **person**, and **organization** annotations

IE as Boundary Classification

- Idea: Learn classifiers to **recognize start token** and **end token** for the facts under consideration. Combine multiple classifiers (ensemble learning) for more robust output.

- Example:
  There will be a talk by Alan Turing at the University at 4 PM.

  Prof. Dr. James Watson will speak on DNA at MPI at 6 PM.

  The lecture by Francis Crick will be in the IIF at 3 PM.

  person
  place
  time

- Classifiers test each token (with PoS tag, LR neighbor tokens, etc. as features) for two classes: begin-fact, end-fact

Text Segmentation and Labeling

- Idea: Observed text is concatenation of structured record with limited reordering and some missing fields

- Example: Addresses and bibliographic records

House number	Building	Road	City	State	Zip
4089	Whispering Pines	Nobel Drive	San Diego	CA	92122

Author Year Title Journal Volume Page

P.P.Wangikar, T.P. Graycar, D.A. Estell, D.S. Clark, J.S. Dordick (1993) Protein and Solvent Engineering of Subtilising BPN' in Nearly Anhydrous Organic Media J.Amer. Chem. Soc. 115, 12231-12237.

- Source: [Sarawagi ’08]

Hidden Markov Models (HMMs)

- Assume that the observed text is generated by a **regular grammar** with some probabilistic variation (i.e., stochastic FSA = **Markov Model**)

- Each **state corresponds to a category** (e.g., noun, phone number, person) that we seek to label in the observed text

- Each **state** has a known **probability distribution over words** that can be output by this state

- The objective is to **identify the state sequence** (from a start to an end state) with **maximum probability of generating the observed text**

- Output (i.e., observed text) is known, but the state sequence cannot be observed, hence the name ***Hidden*** **Markov Model**

Hidden Markov Models

- Hidden Markov Model (HMM) is a discrete-time, finite-state Markov model consisting of
  - **state space** $S = \{s_1, \ldots, s_n\}$ and the state in step t is denoted as $X(t)$
  - **initial state probabilities** p_i ($i = 1, \ldots, n$)
  - **transition probabilities** $p_{ij} : S \times S \rightarrow [0,1]$, denoted $p(s_i \rightarrow s_j)$
  - **output alphabet** $\Sigma = \{w_1, \ldots, w_m\}$
  - **state-specific output probabilities** $q_{ik} : \mathrm{S} \times \Sigma \rightarrow [0,1]$, denoted $q(\mathrm{s}_i \uparrow w_k)$

- Probability of emitting output sequence $o_1, \ldots, o_T \in \Sigma^T$

$$\sum_{x_1,\ldots,x_T \in S} \prod_{i=1}^{T} p(x_{i-1} \rightarrow x_i)\, q(x_i \uparrow o_i) \text{ with } p(x_0 \rightarrow x_i) = p(x_i)$$

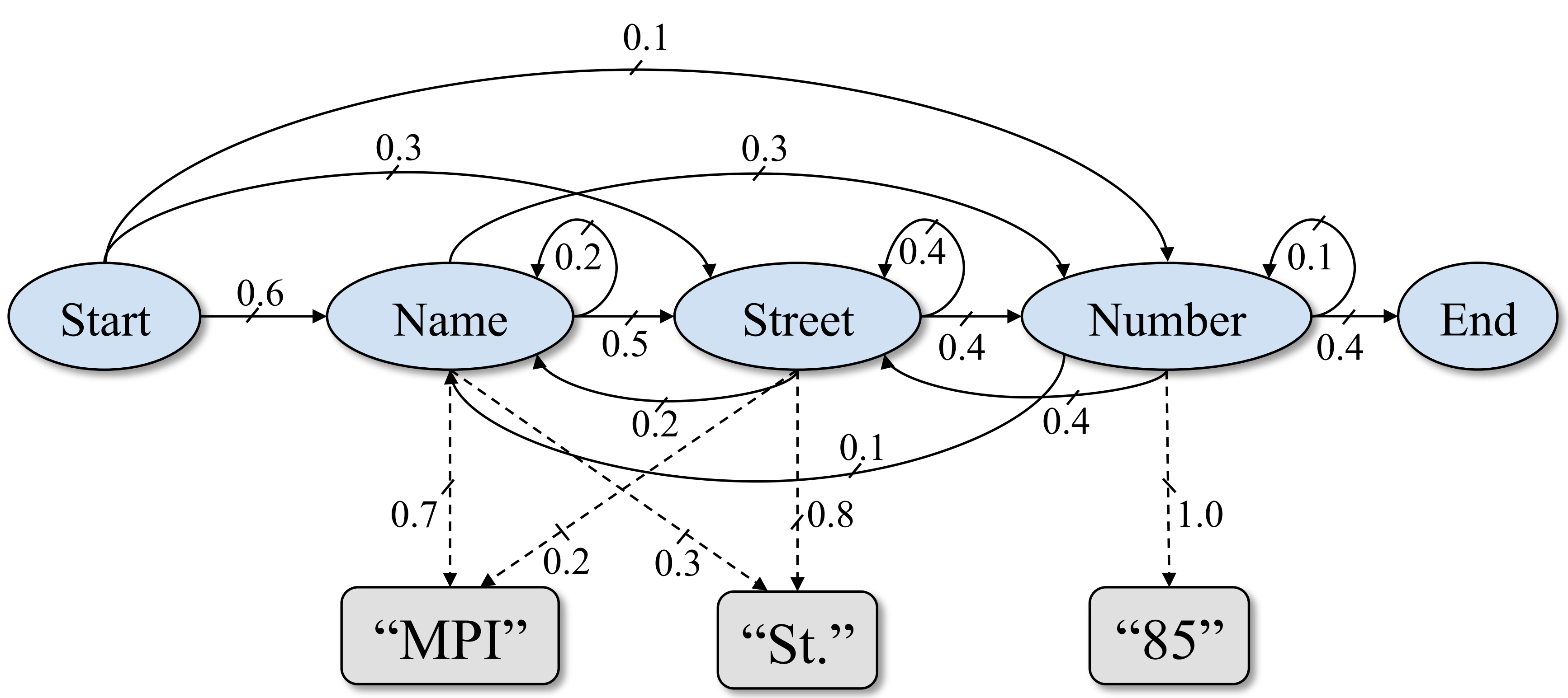
HMM Example

- <u>Goal</u>: Label the tokens in the sequence

  Max-Planck-Institute, Stuhlsatzenhausweg 85

  with the labels **Name**, **Street**, **Number**

$\mathbf{\Sigma}$ = {"MPI", "St.", "85"} // output alphabet
S = {Name, Street, Number} // (hidden) states
$\boldsymbol{p_i}$ = {0.6, 0.3, 0.1} // initial state probabilities



Three Major Issues with HMMs

- Compute **probability of output sequence** for known parameters
  - **Forward/Backward computation**
- Compute **most likely state sequence** for given output and known parameters (decoding)
  - **Viterbi algorithm** (using dynamic programming)
- **Estimate parameters** (transition probabilities and output probabilities) from training data (output sequences only)
  - **Baum-Welch algorithm** (specific form of Expectation Maximization)
- Full details: [Rabiner '90]

Forward Computation

- Probability of emitting output sequence $o_1, \ldots, o_T \in \Sigma^T$ is

$$\sum_{x_1,\ldots,x_T \in S} \prod_{i=1}^{T} p(x_{i-1} \rightarrow x_i)\, q(x_i \uparrow o_i) \text{ with } p(x_0 \rightarrow x_i) = p(x_i)$$

- **Naïve computation** would require $O(n^T)$ operations!

- **Iterative forward computation** with clever caching and reuse of intermediate results ("memoization") requires $O(n^2\, T)$ operations
  - Let $\alpha_i(t) = P[o_1, \ldots, o_{t-1}, X(t) = i]$ denote the probability of being in state i at time t and having already emitted the prefix output $o_1, \ldots, o_{t-1}$
  - <u>Begin</u>: $\alpha_i(1) = p_i$
  - <u>Induction</u>: $\alpha_j(t+1) = \sum_{i=1}^{n} \alpha_i(t)\, p(s_i \rightarrow s_j)\, p(s_i \uparrow o_t)$

Backward Computation

- Probability of emitting output sequence $o_1, \ldots, o_T \in \Sigma^T$ is

$$\sum_{x_1,\ldots,x_T \in S} \prod_{i=1}^{T} p(x_{i-1} \rightarrow x_i)\, q(x_i \uparrow o_i) \text{ with } p(x_0 \rightarrow x_i) = p(x_i)$$

- **Naïve computation** would require $O(n^T)$ operations!

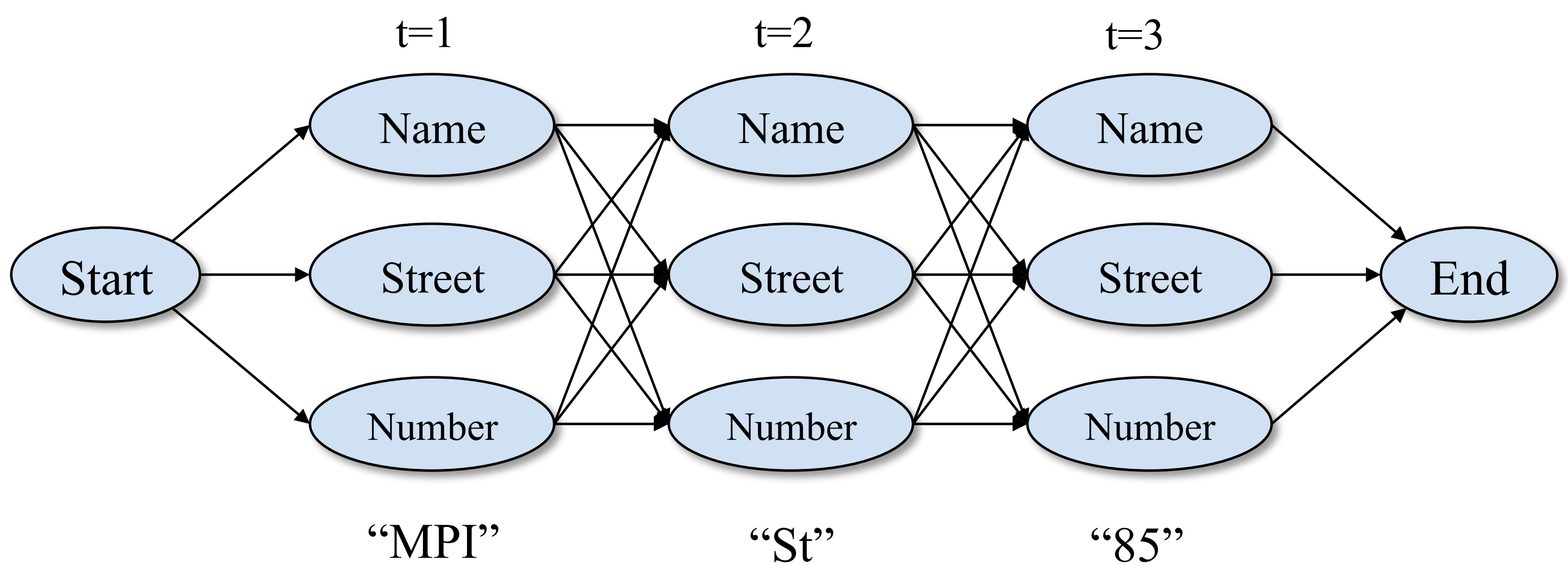
- **Iterative backward computation** with clever caching and reuse of intermediate results ("memoization")

  - Let $\beta_i(t) = P[o_{t+1}, \ldots, o_T, X(t) = i]$ denote the probability of being in state i at time t and having already emitted the suffix output $o_{t+1}, \ldots, o_T$

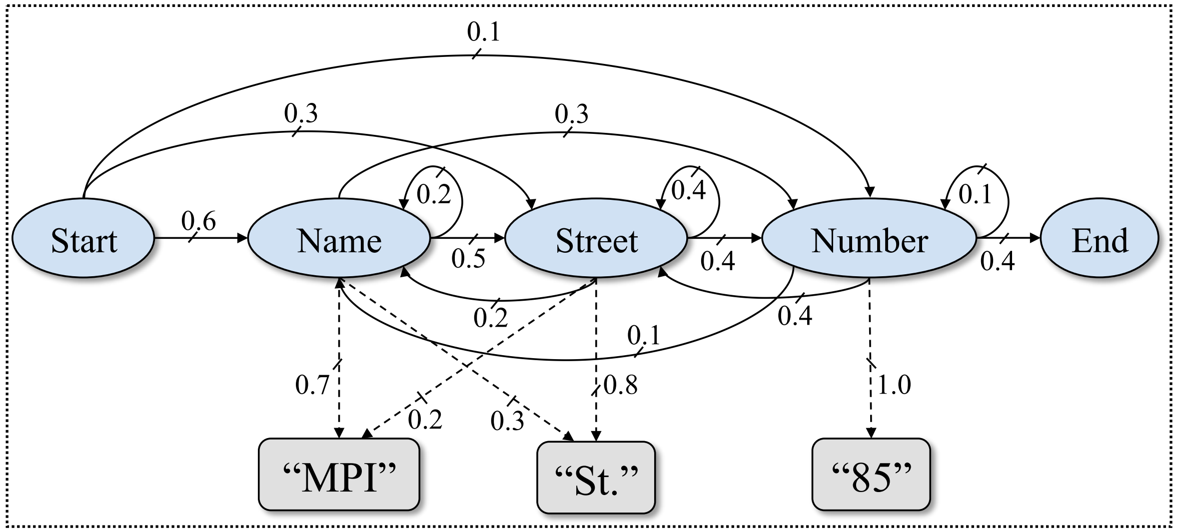
  - Begin: $\beta_i(T) = 1$

  - Induction: $\beta_j(t-1) = \sum_{i=1}^{n} \beta_i(t)\, p(s_j \rightarrow s_i)\, p(s_i \uparrow o_t)$

Trellis Diagramm for HMM Example



- Forward probabilities:



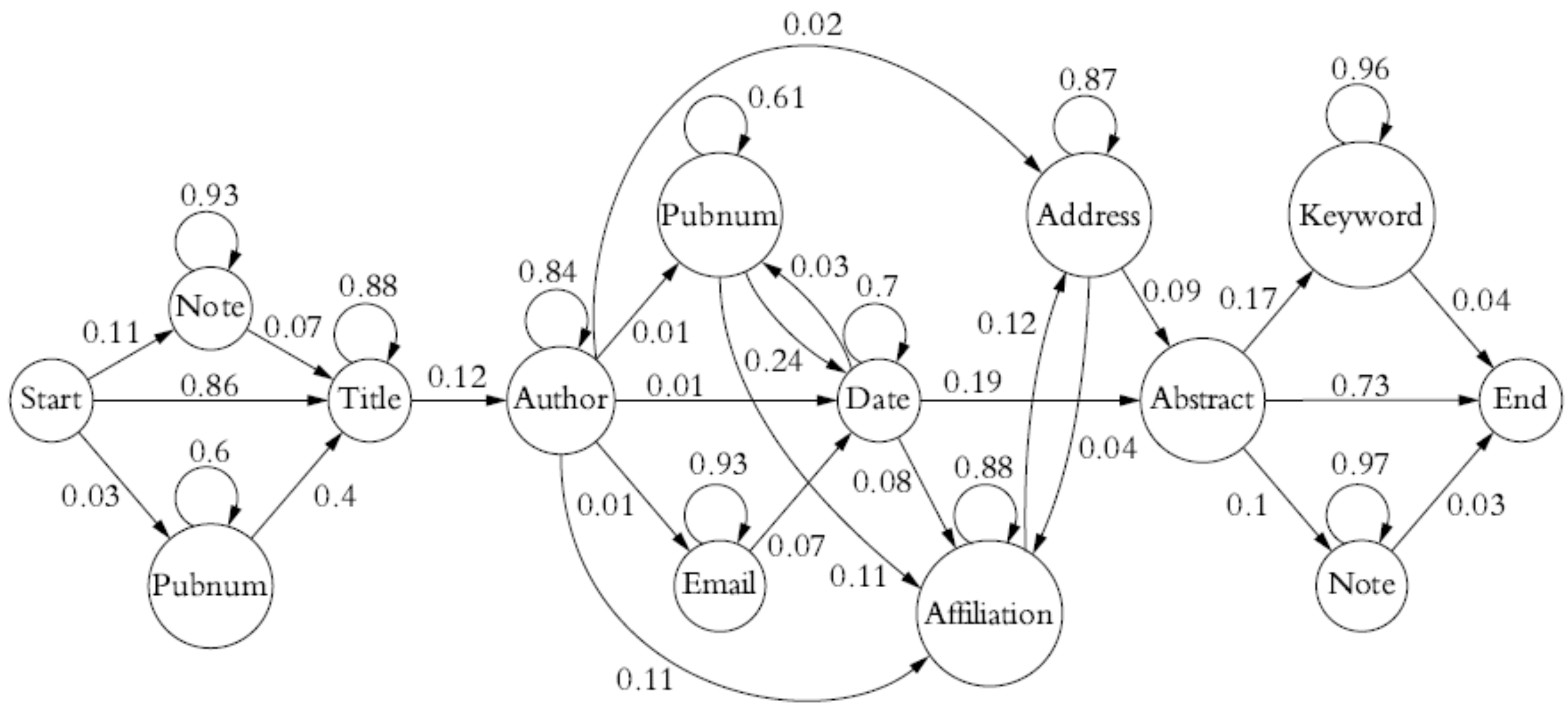
$\alpha_{Name}(1) = 0.6$

$\alpha_{Street}(1) = 0.3$

$\alpha_{Number}(1) = 0.1$

$\alpha_{Name}(2) = 0.6 * 0.7 * 0.2 + 0.3 * 0.2 * 0.2 + 0.1 * 0.0 * 0.1$
$= 0.096$

Larger HMM for Bibliographic Records



- Source: [Chakrabarti '09]

Viterbi Algorithm

- Goal: Identify state sequence $x_1, \ldots, x_T$ **most likely of having generated the observed output** $o_1, \ldots, o_T$

- **Viterbi algorithm** (dynamic programming)

$$\delta_i(1) = p_i \qquad \text{// highest probability of being in state } i \text{ at step 1}$$
$$\psi_i(1) = 0 \qquad \text{// highest-probability predecessor of state } i$$

for $t = 1, \ldots, T$

$$\delta_j(t+1) = \max_{i=1,\ldots,n} \delta_i(t)\, p(x_i \rightarrow x_j)\, q(x_i \uparrow o_t) \quad \text{// probability}$$

$$\psi_j(t+1) = \arg\max_{i=1,\ldots,n} \delta_i(t)\, p(x_i \rightarrow x_j)\, q(x_i \uparrow o_t) \quad \text{// state}$$

- Most likely state sequence can be obtained by means of **backtracking** through the memoized values $\delta_i(t)$ and $\psi_i(t)$

Training of HMMs

- Simple case: If **fully tagged training sequences** are available, we can use MLE to estimate the parameters

$$p(s_i \rightarrow s_j) = \frac{\#\ \text{transitions}\ s_i \rightarrow s_j}{\sum_{s_k} \#\ \text{transitions}\ s_i \rightarrow s_k}$$

$$q(s_i \rightarrow w_k) = \frac{\#\ \text{outputs}\ s_i \rightarrow w_k}{\sum_{w_l} \#\ \text{outputs}\ s_i \rightarrow w_l}$$

- Standard case: Training with **unlabeled sequences** (i.e., observed output only, state sequence is unknown)
  - **Baum-Welch algorithm** (variant of Expectation Maximization)

- Note: There exists some work on learning the structured of HMMs (# states, connections, etc.), but this remains very difficult and computationally expensive

Problems and Extensions of HMMs

- Individual output letters/words may not show learnable patterns
  - output words can be entire **lexical classes** (e.g., numbers, zip code, etc.)
- Geared for flat sequences, not for structured text documents
  - use **nested HMMs** where each state can hold another HMM
- Cannot capture **long-range dependencies** (Markov property) (e.g., in addresses: with first word being "Mr." or "Mrs.", the probability of seeing a P.O. box later decreases substantially)
- Cannot easily incorporate multiple **complex word features** (e.g., isYear(w), isDigit(w), allCaps(w), etc.)
  - **Conditional Random Fields** (CRFs) address these limitations

Additional Literature for VI.4

- **S. Chakrabarti:** *Extracting, Searching, and Mining Annotations on the Web*, Tutorial, WWW 2009 (http://www2009.org/pdf/T10-F%20Extracting.pdf)
- **L. R. Rabiner**: *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, Readings in Speech Recognition, 1990

VI.5 Named Entity Reconciliation

- <u>Problem 1</u>: Same entity appears in
  - Different **spellings** (incl. misspellings, abbreviations, etc.) (e.g., *brittnee spears* vs. *britney spears*)
  - Different **levels of completeness** (e.g., *joe hellerstein* vs . *prof. joseph m. hellerstein*)
- <u>Problem 2</u>: Different entities happen to look the same (e.g., *george w. bush* vs. *george h. w. bush*)

- Problems even occur in **structured databases** and require **data cleaning** when integrating multiple databases
- Integrating heterogeneous databases or Deep Web sources also requires **schema matching** (aka. data integration)

Entity Reconciliation Techniques

- **Edit distance** measures (both strings and records)
- Exploit **context information** for higher-confidence matchings (e.g., publications/co-authors of Dave Dewitt and D. J. DeWitt)
- Exploit **reference dictionaries** as ground truth (e.g., for address cleaning)
- Propagate matching confidence values in link-/reference-based **graph structure**
- **Statistical learning** in (probabilistic) **graphical models** (also: **joint disambiguation** of multiple mentions onto most compact/consistent set of entities)

Entity Reconciliation by Matching Functions

- **Fellegi-Sunter Model** as framework for entity reconciliation
  Input: Two sets A and B of strings or records
  each with features (e.g., n-grams, attributes, etc.)

- Method:

  - Define family γ_i : A × B → {0,1} (i = 1, …, k)
    of **attribute comparisons** or **similarity tests** (aka. **matching functions**)

  - Identify **matching pairs** M ⊆ A × B and **non-matching pairs** U ⊆ A × B
    as training data and compute $m_i = P[\gamma_i(a,b) = 1 \mid (a,b) \in M]$ and
    $u_i = P[\gamma_i(a,b) = 1 \mid (a,b) \in U]$

  - For pairs (a,b) ∈ A × B \ (M ∪ U), consider a and b equivalent if

$$\sum_{i=1}^{k} \frac{m_i}{u_i} \cdot \gamma_i(a,b) \geq \tau \quad \text{for user-defined threshold } \tau$$

- Full details: [Fellegi and Sunter '69]

Additional Literature for VI.5

- **I. Fellegi and A. Sunter**: *A Theory for Record Linkage*, Journal of the American Statistical Association 64(328):1183-1210, 1969