Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Speech and Language 18 (2004) 1–22
COMPUTER SPEECH AND LANGUAGE
www.elsevier.com/locate/csl

Semantic processing using the Hidden Vector State model

Yulan He *, Steve Young

Engineering Department, Cambridge University, Trumpington Street, Cambridge CB2 1PZ, UK

Received 29 May 2003; received in revised form 8 December 2003; accepted 4 March 2004

Abstract

This paper discusses semantic processing using the Hidden Vector State (HVS) model. The HVS model extends the basic discrete Markov model by encoding context in each state as a vector. State transitions are then factored into a stack shift operation similar to those of a push-down automaton followed by a push of a new preterminal semantic category label. The key feature of the model is that it can capture hierarchical structure without the use of treebank data for training.

Experiments have been conducted in the travel domain using the relatively simple ATIS corpus and the more complex DARPA Communicator Task. The results show that the HVS model can be robustly trained from only minimally annotated corpus data. Furthermore, when measured by its ability to extract attribute-value pairs from natural language queries in the travel domain, the HVS model outperforms a conventional finite-state semantic tagger by 4.1% in *F*-measure for ATIS and by 6.6% in *F*-measure for Communicator, suggesting that the benefit of the HVS model's ability to encode context increases as the task becomes more complex.

1. Introduction

Semantic processing is one of the key elements in spoken dialogue systems. It involves extracting meanings from recognized word strings and inferring user's dialogue acts or information goals based on the recognized semantic concepts and preceding dialogue context. Traditionally, semantic parser systems have been built using hand-crafted semantic grammar rules. Word patterns corresponding to semantic tokens are used to fill slots in semantic frames and the frame with

* Corresponding author. Tel.: +44-1223-332662; fax: +44-1223-332754.
E-mail addresses: yh213@cam.ac.uk (Y. He), sjy@eng.cam.ac.uk (S. Young).

0885-2308/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.csl.2004.03.001

the highest score is selected as the best semantic representation. The above process is often called *robust parsing* (Dowding et al., 1994, 1996). Such rule-based approaches are typically domain-specific and often fragile. In contrast, statistical models are able to accommodate the variations found in real data and hence can in principle be more robust. Examples of systems which integrate statistical approaches into rule-based semantic grammars include the hidden understanding model (Minker et al., 1999) used for machine translation where its stochastic component is trained using a corpus annotated by a rule-based parser and the SOUP parser (Gavalda, 2000) which encodes context-free grammars as probabilistic recursive transition networks and serves as a stochastic, chart-based, top-down parser designed for real-time analysis of spoken language. Both systems still require hand-crafted semantic grammar rules to be defined. It is claimed that the SOUP parser supports portability to new domains by allowing separate grammar modules for each domain and by allowing grammar of rules to be shared across domains. Nevertheless, the definition of a full semantic grammar for a new domain still requires significant human effort.

An early example of a purely statistical approach to semantic parsing is the finite state semantic tagger used in AT&T's CHRONUS system (Pieraccini et al., 1992, 1995). In this system, utterance generation is modelled by an HMM-like process in which the hidden states correspond to semantic concepts and the state outputs correspond to the individual words. This model is simple and robust to estimate, however, it fails to capture long distance dependencies because it is unable to represent nested structural information. This model is sometimes referred to as the *flat-concept* model to emphasize its inability to represent hierarchical structure.

The limitations of the flat-concept model can be overcome by allowing the finite state networks to become recursive. In this case, each state can generate either a word or a subnetwork. This is formally equivalent to using stochastic phrase structure rules and it essentially extends the class of supported languages from *regular* to *context-free*. Examples of such *Probabilistic Context-Free Grammar* (PCFG) models are BBN's Hidden Understanding Model (HUM) (Miller et al., 1995, 1997), and hierarchical HMMs (Fine et al., 1998). More recently, Charniak (Charniak, 2001) has proposed a lexicalized statistical parsing model. For each constituent c of a parse structure, the probability that the model assigns to it is decomposed into three products: the preterminal tag of c conditioned on the parent nonterminal and headword, the lexical head of c given its preterminal tag and the relevant history of c, and finally, the expansion of c into further constituents $e(c)$. Charniak's model encodes preferences for right-branching structure implicitly through the use of higher probabilities on right-branching grammar rules. In similar work, Chelba et al. have extended a structured language model (SLM) (Chelba and Jelinek, 2000) to extract essential information from a sentence to fill slots in a semantic frame (Chelba and Mahajan, 2001). This work differs from Charniak's in that non-terminal labels are extended to include both lexical items (headwords) and semantic categories. A sentence is parsed as a two-level semantic parse tree: the root node is tagged with a semantic frame label and spans the entire sentence; the leaf nodes are tagged with slot labels and span the strings of words relevant to the corresponding slot. Inspired by Chelba's SLM, Erdogan has developed a semantic structured language model (Erdogan et al., 2002) which performs semantic parsing in two steps: neighbouring words that constitute a certain meaning or concept are first grouped together; then decision trees are used to derive more complex interactions between semantic constituents.

All of the above hierarchically structured models require fully annotated treebank data for robust parameter estimation. However, the provision of such treebank training data is unrealistic

for most practical applications. In a sense, the *flat-concept* model and the hierarchical model represent opposite ends of the spectrum (Young, 2002b). The former is simple and robust to train but it cannot represent hierarchical structure. On the other hand, the latter can successfully capture the nested structures of semantic concepts, but only when provided with large treebanks for training.

The work in this paper attempts to address this problem by seeking a compromise between the flat-concept model and the fully recursive model. It is motivated by the hypothesis that a suitably constrained hierarchical model may be trainable without treebank data whilst simultaneously retaining sufficient ability to capture the hierarchical structure need to robustly extract task domain semantics. One possible constraint is to restrict the underlying generation process to be strictly right-branching (Young, 2002a). Such a constrained hierarchical model can be conveniently implemented using a *Hidden Vector State* (HVS) model which extends the *flat-concept* HMM model by expanding each state to encode the stack of a push-down automaton. This allows the model to efficiently encode hierarchical context, but because stack operations are highly constrained it avoids the tractability issues associated with full context-free stochastic models such as the hierarchical HMM. As will be shown later, such a model is indeed trainable using only lightly annotated data and furthermore, it offers considerable performance gains compared to the flat concept model.

The remainder of this paper is organized as follows. Section 2 briefly discusses some general issues in the design of statistical models for semantic processing and describes some reasonable assumptions relating to the training of statistical models for practical applications. Section 3 then presents a general finite-state tagger (FST) to serve as a baseline for evaluation purposes. Section 4 discusses the HVS model and presents the mathematical framework. It also explains how the HVS model can be trained from minimally annotated data. An experimental evaluation using both the ATIS and the DARPA Communicator Travel tasks is then presented in Sections 5 and 6. Finally, Section 7 concludes the paper.

2. Semantic processing

2.1. General requirements

In the simple task oriented spoken dialogue systems which form the focus of our current work, each user utterance is converted to a dialogue act containing a set of attribute-value pairs such as

```
REQUEST = flights, FROMLOC = Boston, etc
```

The goal of the semantic processing component is to annotate each natural language input so as to allow these attribute-value pairs to be extracted by a simple deterministic task-independent process.

A particularly simple form of annotation results from mapping each word w in an utterance W into a single discrete concept c. For example, the utterance "Show me flights from Boston to New York" might be encoded as

```
DUMMY(show me) FLIGHT(flights) FROMLOC(from) CITY(Boston) TOLOC(to)
CITY(New York)
```

where FLIGHT, FROMLOC, and TOLOC are semantic concepts(tags) and irrelevant input words such as "show me" are mapped into the DUMMY tag and subsequently discarded. This is the flat concept model mentioned in the introduction. Although it works surprisingly well on simple tasks, one obvious drawback of the model is that it cannot directly represent hierarchical dominance relationships and hence in the example above, the role of Boston can only be inferred from its proximity to FROMLOC. The mapping of the annotation to the attribute-value pair FROMLOC = Boston requires heuristics which quickly break in the face of more complex nested sentence structures.

A more complex encoding will preserve the hierarchical structure of the sentence as in

```
DUMMY(show me)
FLIGHT(flights FROMLOC(from CITY(Boston)) TOLOC(to CITY(New York)))
```

In this case, the unambiguous extraction of the attribute-value pairs is now relatively straightforward but at the cost of considerable additional complexity in the parser.

2.2. Design of statistical semantic parsing models

Semantic parsing can be viewed as a pattern recognition problem and statistical decoding can be used to find the most likely semantic representation given a model and an observed word sequence $W = (w_1 \cdots w_T)$. If we assume that the hidden data take the form of a semantic parse tree C then the model should be a push-down automata which can generate the pair $< W, C >$ through some canonical sequence of moves $D = (d_1 \cdots d_T)$. That is,

$$P(W, C) = \prod_{t=1}^{T} P(d_t | d_{t-1} \cdots d_1). \tag{1}$$

For the general case of an unconstrained hierarchical model, D will consist of three types of probabilistic move:

1. popping semantic category labels off the stack;
2. pushing a non-terminal semantic category label onto the stack;
3. generating the next word.

For the special constrained case of the flat concept model, the stack is effectively depth one and $< W, C >$ is built by repeatedly popping one label off the stack, pushing one new label onto the stack and then generating the next word. Given this framework, it is straightforward to postulate alternative forms of constraint. In particular, this paper considers the constrained form of automata where the stack is finite depth and $< W, C >$ is built by repeatedly popping 0 to n labels off the stack, pushing exactly one new label onto the stack and then generating the next word. This constrained form of automata implements a right-branching parser.

In practice, conditional independence can be used to reduce the number of parameters needed to manageable proportions. As in conventional statistical language modelling, this involves defining an equivalence function Φ which groups move sequences into equivalent classes. Thus, the final generic parsing model is:

$$P(W, C) = \prod_{t=1}^{T} P(d_t | \Phi(d_{t-1} \cdots d_1)). \tag{2}$$

A statistical model requires a large training corpus in order to reliably estimate model parameters. If the training corpus is fully annotated, then model parameters can simply be estimated through event counting and smoothing. In practice, however, the provision of fully annotated data is not realistic. Training from unannotated or partially annotated corpora is therefore crucial for practical applications. The next section discusses what forms of training data annotation it is reasonable to assume will be, or can be made, available for spoken dialogue systems.

2.3. Training assumptions

Before training a statistical semantic parsing model from unannotated data, it is natural to ask what kinds of *a priori* knowledge can be easily provided by the dialogue designer. There are two obvious possibilities:

- *A set of domain specific lexical classes*. For example, in an air travel domain, it is possible to group all airline names into one single class `AIRLINE_NAME`. Such domain specific classes can normally be extracted automatically from the application domain database schema.
- *Abstract semantic annotation for each utterance*. Such an annotation need only list a set of valid semantic concepts and the dominance relationships between them without considering the actual realized concept sequence or attempting to identify explicit word/concept pairs.

The provision of abstract annotations implies that the dialogue designer must define the semantics that are encoded in each training utterance but need not provide an utterance level parse. It effectively defines the required input-output mapping whilst avoiding the need for expensive tree-bank style annotations. For example, in an air travel domain, a dialogue system designer may define the following hierarchical semantic relationships:

- `FLIGHT(FROMLOC(CITY) TOLOC(CITY))`
- `AIRLINE(FROMLOC(CITY) TOLOC(CITY) DEPART(DAY_NAME))`
- `GROUND_SERVICE(CITY_NAME)`
- `RETURN(TOLOC(CITY) ON(DATE))`
- ...

Having defined such a set of hierarchical semantic relationships, annotation is simply a method of associating the appropriate semantics with each training utterance and does not require any linguistic skills. For example, when building a system from scratch, a dialogue designer can take each possible abstract schema in turn and give examples of corresponding natural language forms, as in

`RETURN(TOLOC(CITY(X)) ON(DATE(D)))` →

1. I would like to return to X on D.
2. I wanna return on D to X.
3. I want to return to X on D.

Alternatively, if a corpus of representative user utterances are already available, perhaps obtained via Wizard-of-Oz style data collection, then each utterance can easily be tagged with the appropriate abstract annotation as in

"I want to return on Thursday to Dallas"

← RETURN(TOLOC(CITY(Dallas)) ON(DATE(Thursday)))

In the training data used for the evaluations reported later in this paper corpora of real user utterances were used. Annotation was done by extract abstract semantics from the accompanied SQL reference queries or the available parse results by a rule-based semantic parser which is essentially similar to the annotation method described in the latter case.

Note finally that no assumptions are made regarding access to any syntactic information, neither explicitly via grammar rules nor implicitly via syntactic treebank data.

3. The finite-state semantic model

This section briefly describes a general finite-state tagger (FST) of the type used in CHRONUS (Levin and Pieraccini, 1995) mentioned in Section 1. This model is used as a baseline for evaluating the HVS model described in the subsequent sections.

3.1. Overview of the FST model

An FST model treats the generation of a spoken sentence as an HMM-like process whose hidden states correspond to semantic concepts and outputs correspond to individual words in the utterance. Fig. 1 shows an example of a parse C output using an FST model. Each word is tagged with a single discrete semantic concept label and the utterance is enclosed by sentence start and end markers, `sent_start` and `sent_end`, to enable the prediction of the first and last word in the utterance. The corresponding semantic tags for them are `SS` and `SE` respectively.

Note that if required, the output of the FST can be trivially converted to a well-formed parse tree simply by adding a sentence tag dominating all FST state tags.

3.2. Definition of the FST model

Given a vocabulary $\mathscr{V}$ and a word sequence $W = (w_1 \cdots w_T)$ where $w_t \in \mathscr{V}$, assume that each word w_t is tagged with one semantic concept label c_t, to give the sequence $C = (c_1 \cdots c_T)$ where $c_t \in \mathscr{T}$ and $\mathscr{T}$ denotes the semantic concept space. The FST model can then compute the required joint distribution $P(W, C)$ as follows where each move d_t in the generic form of Eq. (1) is realised by first generating a concept label c_t given the concept history and then generating the corresponding word w_t given c_t and the word history:

$$P(W|C)P(C) = \prod_{t=1}^{T} P(w_t|w_{t-1} \cdots w_1, c_t) \prod_{t=1}^{T} P(c_t|c_{t-1} \cdots c_1). \tag{3}$$

In practice, following Eq. (2), the history is truncated to be of finite length:

Fig. 1. An example of finite-state tagger parse result.

$$P(W|C)P(C) \approx \prod_{t=1}^{T} P(w_t|w_{t-1}\cdots w_{t-n+1}, c_t) \prod_{t=1}^{T} P(c_t|c_{t-1}\cdots c_{t-m+1}). \quad (4)$$

If $n = 1$ and $m = 2$, this becomes a conventional first order Markov model with words modelled as a unigram conditioned on the semantic concept c_t and state transitions given by concept bigram probabilities.

3.3. Training the FST model

If fully labelled training data is available, then the parameters of an FST can be estimated by simple frequency counting. However, in this paper we assume that only word class information and utterance level abstract semantics of the form described in Section 2.3 are available for training. In this case, the training procedure for an FST model consists of three steps: *preprocessing*, *parameter initialization*, and *parameter re-estimation*.

The *preprocessing* step first replaces class members found in the training utterances with their corresponding class names. The abstract annotation associated with each utterance is then expanded to form a sequence of concept labels. Note that the abstract annotations for the FST model do not provide any hierarchical dominance relationships, they simply list out valid semantic concepts that can appear in the parse results. Furthermore, as in the example below, the order of the concept labels is not necessarily monotonic relative to the actual word order.

For example, the abstract annotation

`RETURN(TOLOC(CITY(Dallas)) ON(DATE(Thursday)))`

mentioned in Section 2.3 corresponding to the utterance "I want to return on Thursday to Dallas" would be expanded to the concept sequence. [1]

`RETURN TOLOC TOLOC.CITY ON ON.DATE`

The parameters of the FST model include output vectors for each state and a state transition matrix. The size of the semantic concept space determines the total number of distinct states whilst the vocabulary size determines the output vector size. The *parameter initialization* step simply initializes all of the parameters to be identical.

Finally, the *parameter re-estimation* step uses the forward-backward algorithm to estimate parameters. Note that this is different from CHRONUS where maximum likelihood estimation based on relative frequency counts on a fully annotated ATIS training set was used. There are no explicit word level annotations in our training corpora, hence parameter estimation based on event counts cannot be used and forward-backward estimation must be applied instead. The same situation applies to parameter estimation on the hidden vector state model which will be discussed in the next section. Two constraints are applied during the estimation process. First, for each utterance, a state transition is only allowed if both incoming and outgoing states can be found in the corresponding semantic annotation. Second, if the observed word is a class name (such as `CITY`), then only semantic concepts (states) which contain this class name can be associated with

[1] Note that in practice compound semantic concepts such as TOLOC.CITY are used instead of the atomic semantic tags TOLOC and CITY assumed by the pure model illustrated in Fig. 1. These compounds are limited to those which can be derived directly from the database schema and their use is a heuristic needed to make the FST model work acceptably well in practical systems. The HVS model needs no such heuristic.

the word (eg TOLOC.CITY, FROMLOC.CITY but not FROMLOC). In addition, in order to cater for irrelevant words, the DUMMY tag is allowed everywhere. That is, state transitions from or to the DUMMY state are always allowed.

4. The Hidden Vector State model

This section describes the HVS model. First the overall structure of the HVS model is outlined, then the model parameters are defined. Finally, procedures by which the model can be trained using minimally annotated training data are presented.

4.1. Overview of the HVS model

Consider the semantic parse tree shown in Fig. 2, the semantic information relating to each word is completely described by the sequence of semantic concept labels extending from the preterminal node to the root node. If these semantic concept labels are stored as a single vector, then the parse tree can be transformed into a sequence of vector states as shown in the lower portion of Fig. 2. For example, the word Dallas is described by the semantic vector [CITY, TOLOC, RETURN, SS]. Viewing each vector state as a hidden variable, the whole parse tree can be converted into a first order vector state Markov model, this is the HVS model.

Each vector state is in fact equivalent to a snapshot of the stack in a push-down automaton. Indeed, given some maximum depth of the parse tree, any PCFG formalism can be converted to a first-order vector state Markov model. If we view each vector state as a stack, then state transitions may be factored into a stack shift by n positions followed by a push of one or more new preterminal semantic concepts relating to the next input word. If such operations are unrestricted, then the state space will grow exponentially and the same computational tractability issues of hierarchical HMMs are incurred. However, by imposing constraints on the stack operations, the state space can be reduced to a manageable size. Possible constraints to introduce are limiting the

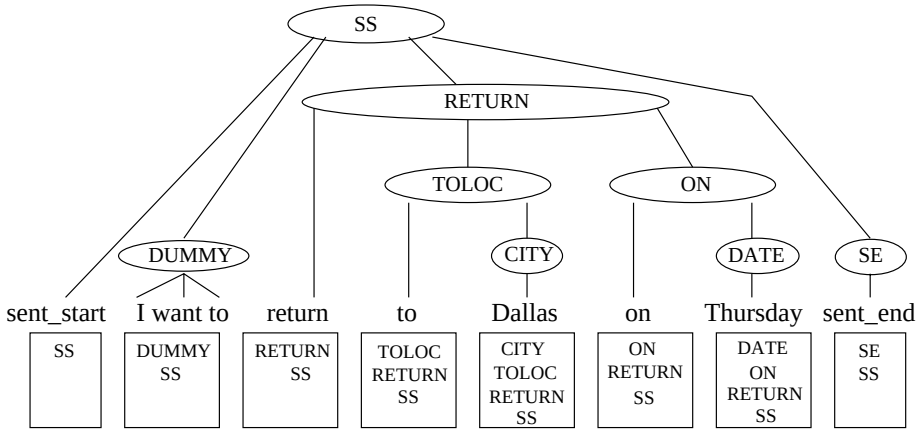

Fig. 2. Example of a parse tree and its vector state equivalent.

maximum stack depth and only allowing one new preterminal semantic concept to be pushed onto the stack for each new input word. These constraints ensure that the size of the underlying probability tables are linear in stack depth, number of concept labels, and vocabulary size. As mentioned earlier in Section 2.2, such constraints effectively limit the class of supported languages to be right-branching. Although left-branching structures do exist in English, the majority of sentences can be represented as right-branching structures. In addition, right-branching structures are generally preferred because they reduce the working memory needed to represent a sentence (Phillips, 1995).

Note also that although any parse tree can be converted into a sequence of vector states, it is not a one-to-one mapping and ambiguities may arise. However for the semantic parsing task defined in this paper, we are not interested in the exact parse tree to be recovered, but only the concept/value pairs to be extracted. Even if there are ambiguities, the extracted concept/value pair will be the same. For example, assuming A and B are semantic concept labels and x and y are words, the partial parse trees B(A(x) A(y)) and B(A(x y)) would share a common HVS representation and hence would yield the same concept/values B.A = x and B.A = y. If the entities x and y were actually distinct, then the preterminal labels would have to be made unique.

4.2. Definition of the HVS model

The joint probability $P(N, \mathbf{C}, W)$ of a series of stack shift operations N, a concept vector sequence $\mathbf{C}$, and a word sequence W can be decomposed as follows:

$$P(N, \mathbf{C}, W) = \prod_{t=1}^{T} P(n_t | W_1^{t-1}, \mathbf{C}_1^{t-1}) P(c_t[1] | W_1^{t-1}, \mathbf{C}_1^{t-1}, n_t) P(w_t | W_1^{t-1}, \mathbf{C}_1^{t}), \tag{5}$$

where

- $\mathbf{C}_1^t$ denotes a sequence of vector states $\mathbf{c}_1..\mathbf{c}_t$. $\mathbf{c}_t$ at word position t is a vector of D_t semantic concept labels (tags), i.e. $\mathbf{c}_t = [c_t[1], c_t[2], \ldots, c_t[D_t]]$ where $c_t[1]$ is the preterminal concept immediately dominating the word w_t and $c_t[D_t]$ is the root concept (SS in Fig. 2),
- $W_1^{t-1}\mathbf{C}_1^{t-1}$ denotes the previous semantic parse up to position $t-1$,
- n_t is the vector stack shift operation and takes values in the range $0, \ldots, D_{t-1}$,
- $c_t[1] = c_{w_t}$ is the new preterminal semantic tag assigned to word w_t at word position t.

In terms of the generic model given by Eq. (1), each move has now been factored into three terms: choice of how many elements to pop from the stack, the new concept label to push onto the stack and the word to generate given the new vector state.

The stack transition from $t-1$ to t given preterminal semantic concept tag c_{w_t} for word w_t is

$$c_t[1] = c_{w_t}, \tag{6}$$

$$c_t[2..D_t] = c_{t-1}[(n_t + 1)..D_{t-1}], \tag{7}$$

$$D_t = D_{t-1} + 1 - n_t. \tag{8}$$

Thus n_t defines the number of semantic tags which will be popped off the stack before pushing on c_{w_t}. The case $n_t = 0$ corresponds to growing the stack by one element i.e. entering a new

semantic tag. The case $n_t = 1$ corresponds to simply replacing the preterminal at word position $t-1$ by c_{w_t} at word position t, the rest of the stack being unchanged. The case $n_t > 1$ corresponds to shifting the stack i.e. popping off one or more semantic tags.

In the version of the HVS model discussed in this paper, Eq. (5) is approximated by

$$P(n_t|W_1^{t-1}, \mathbf{C}_1^{t-1}) \approx P(n_t|\mathbf{c}_{t-1}), \tag{9}$$

$$P(c_t[1]|W_1^{t-1}, \mathbf{C}_1^{t-1}, n_t) \approx P(c_t[1]|c_t[2..D_t]), \tag{10}$$

$$P(w_t|W_1^{t-1}, \mathbf{C}_1^{t}) \approx P(w_t|\mathbf{c}_t). \tag{11}$$

4.3. Training the HVS model

This section discusses training the HVS model to perform hierarchical semantic parsing. It first describes the preprocessing steps which are needed to add lexical features to the training data and augment the abstract annotations, then it briefly describes model initialization, and parameter re-estimation.

4.3.1. Preprocessing

As mentioned in Section 4.1, two kinds of *a priori* knowledge are assumed to be available or can be provided by dialogue system designers: a set of domain specific lexical classes and abstract annotations for each training utterance.

As in the FST model, the first step needed to train the HVS model is to replace all class members by their corresponding class names. Where there is ambiguity, the class covering the largest span of words is replaced first. Where a word or phrase may occur in more than one class, the first class encountered is chosen arbitrarily. Better solutions to this problem clearly exist, but in practice the problem is quite rare and it does not significantly affect performance.

Recalling again the example illustrated in Section 2.3, in the case of the HVS model, the abstract annotation

```
RETURN(TOLOC(CITY(Dallas)) ON(DATE(Thursday)))
```

corresponding to the utterance "I want to return on Thursday to Dallas" would be expanded to the flattened concept sequence

```
RETURN RETURN+TOLOC RETURN+TOLOC+CITY(Dallas) RETURN+ON
RETURN+ON+DATE(Thursday)
```

In order to cater for irrelevant input words, a `DUMMY` tag is allowed everywhere in preterminal positions. In order to accommodate this, the vector state sequence is finally expanded to

```
RETURN RETURN+DUMMY RETURN+TOLOC RETURN+TOLOC+DUMMY
RETURN+TOLOC+CITY(Dallas) RETURN+TOLOC+CITY(Dallas)+DUMMY
RETURN+ON RETURN+ON+DUMMY RETURN+ON+DATE(Thursday)
RETURN+ON+DATE(Thursday)+DUMMY
```

Note that this final set of vector states only provides the set of valid semantic vector states that can appear in the parse results of the current utterance. As explained further below, this set is used as a constraint in the EM re-estimation algorithm. It does not define the actual vector state transition sequence. Further note that the total number of distinct vector states required to use the HVS model for a particular application can be enumerated directly from this expanded vector state list.

Our system only allows the `DUMMY` tag to appear in preterminal positions, therefore, for consecutive irrelevant word inputs, the model will stay in the same vector state. Only when a relevant word input is observed will the `DUMMY` tag together with zero or more preceding semantic tags be popped off from the previous vector state stack and a new preterminal tag will be pushed into the stack accordingly. For a more complex domain, it may be useful to allow a `DUMMY` tag anywhere in a vector state, for example, `RETURN+DUMMY+TOLOC` may also be considered as valid. This issue will be investigated in future work.

4.3.2. Parameter initialization

Eqs. (9)–(11) define the three main components of the HVS model, namely, the stack shift operation, the push of a new preterminal semantic tag, and the selection of a new word. Thus, each vector state is associated with three sets of probabilities: a vector of stack shift probabilities, a vector of semantic tag probabilities, and a vector of output probabilities. The cardinalities of these three sets of probabilities are determined by the maximum vector state stack depth, the number of preterminal semantic tags, and the vocabulary size, respectively. Hence, the total information required to define an HVS model is:

- number of distinct vector states
- number of preterminal semantic tags
- maximum vector state stack depth
- vocabulary size
- for each vector state
  - state name
  - stack shift operation probability vector
  - tag transition probability vectors (push of a new preterminal tag)
  - output probability vector

Having acquired all of the information required for a particular application domain, the prototype topology of an HVS model can be defined. One example of such a prototype definition is shown in Fig. 5 which uses a format similar to that described for HMM definitions in the HTK Toolkit (Young et al., 2004).

A so-called *flat start* is used whereby all model parameters are initially made identical. These parameters are then iteratively refined using the EM-based re-estimation procedure described in the next section.

4.3.3. Parameter re-estimation

Let the complete set of model parameters be denoted by λ, EM-based parameter estimation aims to maximize the expectation of $L(\lambda) = \log P(N, \mathbf{C}, W|\lambda)$ given the observed data and current estimates. To do this, define the auxiliary Q function as:

$$Q(\lambda|\lambda^k) = E[\log P(N, \mathbf{C}, W|\lambda)|W, \lambda^k] = \sum_{\mathbf{C},N} P(N, \mathbf{C}|W, \lambda) \log P(N, \mathbf{C}, W|\lambda^k). \quad (12)$$

The approximation of $P(N, \mathbf{C}, W)$ given by Eqs. (9)–(11) is

$$P(N, \mathbf{C}, W) = \prod_{t=1}^{T} P(n_t|\mathbf{c}_{t-1}) P(c_t[1]|c_t[2..D_t]) P(w_t|\mathbf{c}_t). \quad (13)$$

Substituting Eq. (13) into (12) and differentiating leads to the following re-estimation formulae

$$\hat{P}(n|\mathbf{c}') = \frac{\sum_t P(n_t = n, \mathbf{c}_{t-1} = \mathbf{c}'|W, \lambda)}{\sum_t P(\mathbf{c}_{t-1} = \mathbf{c}'|W, \lambda)}, \quad (14)$$

$$\hat{P}(c[1]|c[2..D]) = \frac{\sum_t P(\mathbf{c}_t = \mathbf{c}|W, \lambda)}{\sum_t P(c_t[2..D] = c[2..D]|W, \lambda)}, \quad (15)$$

$$\hat{P}(w|\mathbf{c}) = \frac{\sum_t P(\mathbf{c}_t = \mathbf{c}|W, \lambda) \delta(w_t = w)}{\sum_t P(\mathbf{c}_t = \mathbf{c}|W, \lambda)}, \quad (16)$$

where $\delta(w_t = w)$ is one iff the word at time t is w, otherwise it is zero.

The key components of the above re-estimation formulae can be efficiently calculated using the forward-backward algorithm. A full derivation of the re-estimation formulae is given in Appendix B.

During training, two constraints similar to those described for the FST model are applied to limit the search space explored in the E-step. Firstly, state transitions are only allowed if both incoming and outgoing states are listed in the semantic annotation defined for the utterance. Secondly, if there is a lexical item attached to a preterminal tag of a vector state, that vector state must appear bound to that lexical item in the training annotation.

The following example illustrates how these two constraints are applied. Consider again the annotation for the utterance "I want to return on Thursday to Dallas" which was in abstract form

```
RETURN RETURN+DUMMY RETURN+TOLOC RETURN+TOLOC+DUMMY
RETURN+TOLOC+CITY(X) RETURN+TOLOC+CITY(X)+DUMMY RETURN+ON
RETURN+ON+DUMMY RETURN+ON+DATE(D) RETURN+ON+DATE(D)+DUMMY
```

The transition from `RETURN+TOLOC+CITY(X)` to `RETURN` is allowed since both states can be found in the semantic annotation. However, the transition from `RETURN` to `FLIGHT` is not allowed as `FLIGHT` is not listed in the semantic annotation. Also, for the lexical item `X` in the training utterance, the only valid vector state is `RETURN+TOLOC+CITY(X)` since `X` has to be bound with the preterminal tag `CITY`.

5. Experimental evaluation

Experiments have been conducted on a relatively simple corpus – ATIS (Dahl et al., 1994) which contains air travel information data, and a more complex corpus – DARPA Communicator

Table 1
Statistics of the ATIS and DARPA Communicator Data

Dataset	Training set	Test set
ATIS	4978	448 (NOV93) 445 (DEC94)
DARPA Communicator Data	12,702	1178

Travel Data (CUData, 2004) which contains not only air travel related data but also information on hotel reservation, car rental, etc. The ATIS training set consists of 4978 utterances selected from the Class A (context independent) training data in the ATIS-2 and ATIS-3 corpora whilst the ATIS test set contains both the ATIS-3 NOV93 and DEC94 datasets. The DARPA Communicator Travel Data are available to the public as open source download. The data contain utterance transcriptions and the semantic parse results from the Phoenix parser (CUPheonix, 2003). These parse results were hand corrected and then used to generate both the abstract training annotations and the test set reference results for our experiments. Table 1 gives the overall statistics of both ATIS and DARPA data.

In order to evaluate the performance of the model, a reference frame structure was derived for every test set utterance consisting of slot/value pairs. An example of such a reference frame is:

```
Show me flights from Boston to New York.
Frame: FLIGHT
Slots: FROMLOC.CITY = Boston
       TOLOC.CITY = NewYork
```

Performance was then measured in terms of F-measure on slot/value pairs (Goel and Byrne, 1999), which combines the precision (P) and recall (R) values with equal weight and is defined as

$$F = \frac{2PR}{P+R}. \tag{17}$$

Various smoothing techniques have been tested and it was found that linear discounting for transition probabilities and Good-Turing for output probabilities yielded the best result for the FST model, whereas Witten-Bell for vector state stack operation probabilities, push of a new preterminal tag, and output probabilities achieve the highest F-measure score for the HVS model (Ney et al., 1994, 1991).

5.1. ATIS

A set of 30 domain-specific lexical classes were extracted from the ATIS-3 database. [2] The training data was then preprocessed to replace class members by their corresponding class names as described in Sections 3.3 and 4.3. Abstract semantics for each training utterance were derived semi-automatically from the SQL queries provided in ATIS-3.

[2] Refer to Table 7 for the complete set of lexical classes extracted from both ATIS and DARPA Communicator corpora.

After the parse results have been generated for the test sets, post-processing is required to extract relevant slot/value pairs and convert them into a format compatible with the reference frames. This post-processing depends on a pre-defined list of semantic tags to be ignored, examples of which include the sentence start/end markers, the `DUMMY` tag, the tags about departure and arrival indicators such as `FROMLOC`, `TOLOC`, `DEPART`, `ARRIVE`, etc. The extraction of slot/value pairs from parse results then follows the rules below:

- ignore the vector state if its preterminal tag is in the tag ignore list;
- if a preterminal tag is about location or date/time, then search the indicators of departure or arrival in its corresponding vector state in a bottom-up manner, extract the first encountered indicator tag;
- for all other vector states, only extract preterminal tags, e.g., for the word `flight` which is tagged as `SS+FLIGHT`, the extracted slot/value pair is `FLIGHT = flight`.

Note that in the slot/value pairs extraction rules described above, essentially only the semantic tag ignore list needs to be built manually, which is relatively straightforward as the number of preterminal semantic tags for any particular task is normally limited. For example, in the experimental work reported here, there are only 85 preterminal tags for the ATIS corpus and 65 preterminal tags for the DARPA Communicator data. Hence, it is easy to identify those tags which should be ignored.

Taking the HVS parse result shown in Fig. 3 as an example, the vector state associated with the word `Denver` is `SS+FLIGHT+ARRIVE+FROMLOC+CITY`, the preterminal tag `CITY` indicates that `Denver` is a location and the vector state has two location indicators `ARRIVE` and `FROMLOC`. However, only `FROMLOC` will be extracted as it is the first indicator found by the system when the vector state is scanned from pre-terminal tag back up towards the root tag. Therefore, the slot/value pair extracted will be `FROMLOC.CITY = Denver`. The full list of slots extracted in this example would be:

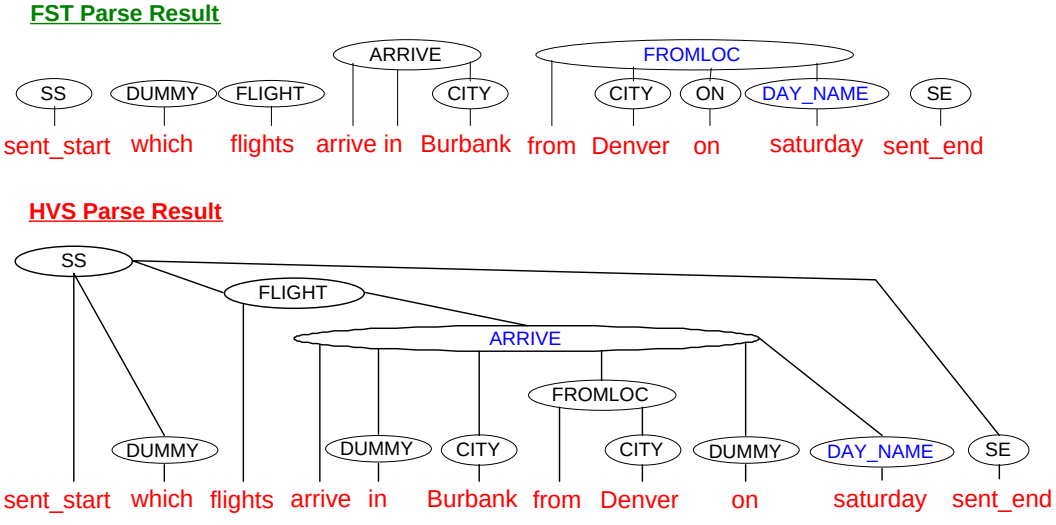

Fig. 3. Comparison of parses generated by the FST and HVS models.

```
FLIGHT = flight
ARRIVE.CITY = Burbank
FROMLOC.CITY = Denver
ARRIVE.DAY_NAME = saturday
```

5.2. DARPA Communicator Travel Data

The DARPA Communicator Travel Data were collected in 461 days and consist of 2211 dialogues or 38,408 utterances in total. From these, 46 days were randomly selected for use as test set data and the remainder were used for training. As the data provided contain only utterances from the user's side of the conversation, there exist some seemingly meaningless utterances such as "No, thank you", "Yes, please" etc., which were classified as "Respond" class in the reference annotations generated by the Phoenix parser. These utterances plus short backchannel utterances for which "No Parse" could be found have no defined semantics outside of an interactive dialogue context and hence have been excluded. The statistics of the DARPA Communicator Travel Data are shown in Table 2. After cleaning up the data, the training set consists of 12,702 utterances while the test set contains 1178 utterances.

A set of 18 domain-specific lexical classes were extracted from the DARPA Communicator database. As mentioned earlier, the abstract annotations used for training and the reference annotations needed for testing were derived by hand correcting the Phoenix parse results. It should be emphasised here that the use of the Phoenix parser for generating the abstract training annotations was purely a matter of convenience. If the parser had not been available, it would have been straightforward to manually generate the abstract annotations using a simple annotation tool. It would certainly have taken far less time to do this, than it would have taken to provide detailed treebank annotations.

6. Experimental results

The performance of the HVS model was evaluated using both the ATIS and the DARPA Communicator corpora. The performance of the FST model on the same data was also evaluated to provide a baseline for comparison. The following sections present the evaluation results in detail.

Table 2
Statistics of the DARPA Communicator Travel Data

Criteria	Training set	Test set
Collection dates	415	46
Total utterance number	35,531	2877
Utt. number after removing "No Parse"	29,947	2442
Utt. number after further removing "Respond"	13,060	1192
Utt. number after further removing "no slots extracted"	12,702	1178

Table 3
Performance comparison of FST model on various language modelling techniques on ATIS and DARPA Communicator data

Lang. Model		ATIS			DARPA Communicator		
States	Outputs	Recall	Precision	*F*-measure	Recall	Precision	*F*-measure
Bigram	Unigram	86.71%	84.84%	85.77%	82.94%	82.34%	82.64%
Bigram	Bigram	74.72%	67.55%	70.95%	68.52%	77.39%	72.68%
Trigram	Unigram	62.15%	64.33%	62.59%	71.70%	59.90%	65.27%
Trigram	Bigram	61.32%	56.51%	58.81%	49.68%	46.69%	48.14%

Table 4
Performance comparison of FST and HVS models on ATIS and DARPA data

Measurement	ATIS		DARPA Communicator	
	FST model	HVS model	FST model	HVS model
Recall	86.71%	89.82%	82.94%	87.31%
Precision	84.84%	88.75%	82.34%	88.84%
F-measure	85.77%	89.28%	82.64%	88.07%

6.1. Evaluation of the FST baseline system

The FST model was implemented according to Eq. (3) and experiments were conducted to find the best values for n and m where $n \in (1,2)$ and $m \in (2,3)$. The results listed in Table 3 indicate that the first order FST model ($n = 1, m = 2$) gives the best performance results on both the ATIS and DARPA Communicator Travel Task corpora, this model was therefore chosen as the baseline system for subsequent experiments.

6.2. Comparison of the HVS model with the FST model

Experiments have been conducted using the FST model and the HVS model. The results are listed in Table 4. For a simple corpus like ATIS, the HVS model outperforms the FST model by 4.1%, while for the more complex DARPA Communicator Travel corpus, it improves on the FST model by 6.6%.

The DARPA Communicator data contains a large number of short utterances due to short responses to system queries. For example, if the system asks *"What time do you want to leave Denver"*, the user may simply answer *"In the morning"*. Among 1178 test utterances, 847 utterances have length less than five words, while 8516 out of 12,702 training utterances contain only

Table 5
Performance comparison of FST and HVS models on long and short utterances in DARPA Communicator corpus

Measurement	Utterance length $\leqslant 4$ (847)		Utterance length > 4 (331)	
	FST Model	HVS Model	FST Model	HVS Model
Recall	88.42%	92.01%	74.16%	79.74%
Precision	91.57%	96.78%	69.31%	77.40%
F-measure	89.97%	94.33%	71.65%	78.56%

four or less words. Table 5 gives results partitioned by utterance length. These results show that the performance of the FST and HVS model do degrade with utterance length. The HVS model outperforms the FST model by 4.9% for short utterances and 9.6% for long utterances. From the above, we can conclude that the HVS model always outperforms the FST model and the improvement increases with more complex data and longer sentences.

6.3. Comparative parse examples generated by the FST and HVS models

The ability of the HVS model to represent hierarchical structure is illustrated in the parse examples given in Fig. 3 which have been extracted from the ATIS test results. It is clear that `saturday` has the strongest correlation with the phrase `arrives in` and should be interpreted as `ARRIVE_DATE`. However, such long distance dependency requires at least 5-grams to capture, which is beyond the range of the FST. As a consequence, the FST model erroneously tagged `saturday` as `DEPARTURE_DATE` since it follows `FROMLOC.ON`. In contrast, the HVS model was able to reproduce the hierarchical structure of the utterance. It carried forward the `ARRIVE` context in the stack and thereby properly interpreted `saturday` as an `ARRIVE_DATE`.

6.4. Optimal vector stack depth

The stack depth in the HVS model determines the number of previous semantic tags which can be encoded as historical context. Fig. 4 shows the variation of system performance as a function of

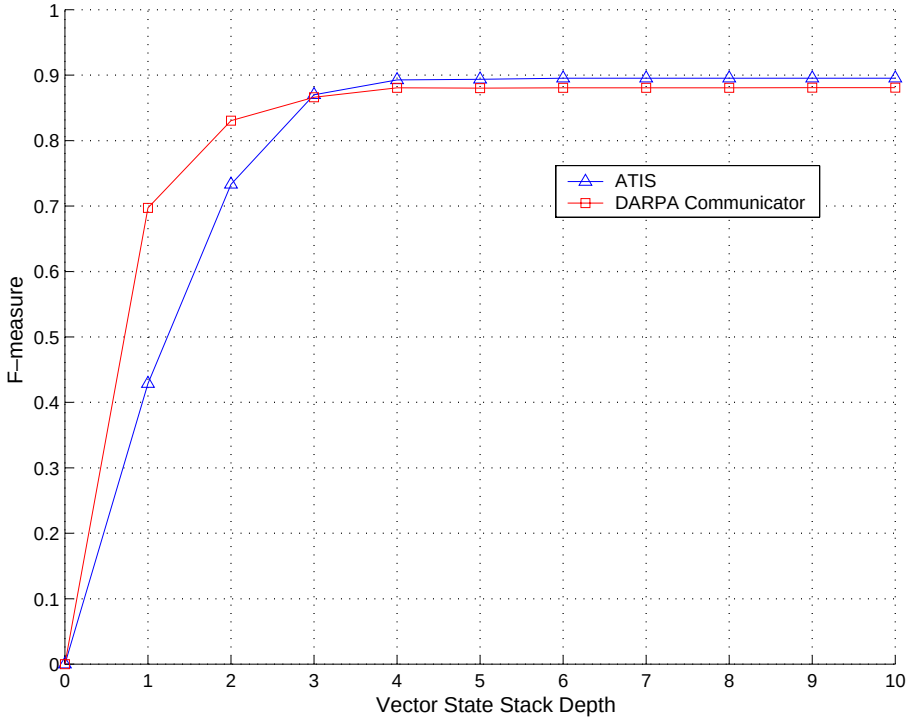

Fig. 4. Slot *F*-measure vs. stack depth.

Table 6
Statistics of model parameters

Model parameters	ATIS		DARPA Communicator	
	FST Model	HVS Model	FST Model	HVS Model
Number of states	120	2799	111	705
Number of preterminal tags	120	85	111	65
Vocabulary size	611	611	612	612
Total parameters	87720	1726983	80253	435690

the maximum stack depth where solid lines represent the *F*-measure of the ATIS test set and dashed lines represent the *F*-measure for the DARPA test data. [3] The optimal stack depth is 4 and this is consistent with the hierarchical complexity of the abstract semantic concepts defined in the training utterances.

6.5. Model complexity

Another interesting issue is the number of free parameters in each model. This is related to the total number of distinct states, the number of preterminal tags, and the vocabulary size. Table 6 gives the relevant statistics for the FST model and HVS model of stack depth 4. Although the state space for the HVS model is large relative to the FST model, the possible transitions between any two states are constrained by the maximum depth of the vector state stack and the condition that only one new semantic tag can be added per input word. Note also that although the FST has far fewer parameters than the HVS model, it cannot be claimed that it is being unnecessarily handicapped in the comparison with the HVS model since, as shown in Section 6.1, the simple expedient of increasing the history lengths in the FST model to take advantage of more free parameters does not yield any improvement in performance.

7. Conclusions

The major problem of building a statistical model for semantic processing is how to collect large quantities of training data in order to reliably estimate model parameters. In general, fully annotated tree-bank data are not available for most applications. An ideal situation would be where the initial training data could be easily provided by the dialogue designer and where the semantic parsing model could be trained directly from unannotated data in a constrained way whilst at the same time being able to capture the underlying hierarchical semantic structures. This is the motivation underlying our interest in the HVS model.

In this paper, a baseline Finite State Tagger (FST) model and the HVS model have been built and tested on the ATIS and DARPA Communicator Travel Data. The experimental results have shown that both models can be trained directly from the abstract semantics without the use of

[3] Note here the actual number of semantic tags kept in a vector state is $n + 1$ for the stack depth n as the root tag *SS* exists in all vector states.

word-level annotations. However, unlike the FST, the HVS model is able to capture hierarchical structure in the semantics. Furthermore it can do this without incurring the computational tractability issues inherent in fully recursive models such as the hierarchical HMM.

The current version of the HVS model only allows one new semantic concept to be pushed into a vector state stack per input word, and this may not be sufficient for a more demanding domain. We are currently investigating alternative ways to apply stack transition constraints, such as allowing more than one new concept to be added for each input word. Another important topic for future work is to port the HVS model to a real dialogue application and explore the possibilities for on-line learning. For example, if the mapping from an input utterance to semantic interpretation was initially wrong but was then repaired during the subsequent dialogue, it may be possible to use the corrected semantics to update the HVS model online.

Finally, our current HVS-based system uses only the single best hypothesis string from the recognizer. Improved performance may be gained by using the decoder to score and rank hypotheses encoded in a word lattice. Also, since the semantic decoder provides additional knowledge about dialogue acts, such knowledge may be passed back to the speech recognizer in order to constrain the recognition, for example, by conditioning the recognizer's N-gram language model. Indeed, the HVS model may be a viable replacement, for the N-gram language model itself.

Acknowledgements

The authors thank Wayne Ward for providing the Phoenix parser output for the DARPA Communicator Travel Data.

Appendix A. Definition of an HVS model

Fig. 5 gives a sample topology definition of an HVS model. The first line of the definition contains a macro type ~o which specifies global features of the HVS model, such as the total number of distinct vector states, number of preterminal semantic tags, maximum stack depth, vocabulary size, etc. Another macro ~h indicates an HVS definition whose name is specified by the following string "hvsStk4". The HVS definition is bracketed by the symbol <BeginHVS> and <EndHVS>. Parameters associated with each vector state are enumerated within the HVS definition and are listed following the keyword <State>. For example, a stack shift operation probability vector is introduced by the keyword <StackOp>, a tag transition probability vector is introduced by the keyword <TagTrans >, and finally, an output probability vector is introduced by the keyword <Output>.

It should be noted that the actual number of semantic tags kept in a vector state is $n + 1$ for the stack depth n as the root tag SS exists in all vector states. Therefore, though the maximum stack depth is 4 in Fig. 5, the vector size of stack shift probabilities is 5 as can be seen from 0.200*5. Here, to allow efficient coding, successive repeated values are represented as a single value plus a repeat count in the form of an asterisk followed by an integer multiplier, e.g., 0.200*5 represents the probability 0.200 repeated by five times.

```
~o
<NumStates> 2799
<NumTermTags> 85
<MaxStackDepth> 4
<VocabSize> 611
~h "hvsStk4"
<BeginHVS>
    <State> 1
        <StateName> SS
        <StackOp> 0.200*5
        <TagTrans> 0.015
        <Output>
            0.000*255  0.000*225  1.000  0.000*130
    <State> 2
        <StateName> SS+DUMMY
        <StackOp> 0.200*5
        <TagTrans> 0.015
        <Output>
            0.002*255  0.002*255  0.002*101
    <State> 3
        <StateName> SS+FLIGHT
        <StackOp> 0.200*5
        <TagTrans> 0.015
        <Output>
            0.002*255  0.002*255  0.002*101
        ...
<EndHVS>
```

Fig. 5. Definition for an HVS model.

Appendix B. The HVS model re-estimation formulae

The key components of Eqs. (14)–(16) are the likelihoods $P(n_t = n, \mathbf{c}_{t-1} = \mathbf{c}'|W, \lambda)$, $P(c_t[2..D] = c[2..D]|W, \lambda)$ and $P(\mathbf{c}_t = \mathbf{c}|W, \lambda)$. These can be efficiently calculated using the forward-backward algorithm. First define the forward probability α as

$$\alpha_{\mathbf{c}}(t) = P(w_1, w_2, \ldots, w_t, \mathbf{c}_t = \mathbf{c}|\lambda). \tag{B.1}$$

Similarly, define the backward probability β as

$$\beta_{\mathbf{c}}(t) = P(w_{t+1}, w_{t+2}, \ldots, w_T|\mathbf{c}_t = \mathbf{c}, \lambda). \tag{B.2}$$

Then, omitting the conditioning on W and λ for clarity, the required likelihoods are

$$P(n_t = n, \mathbf{c}_{t-1} = \mathbf{c}') = \alpha_{\mathbf{c}'}(t-1) \cdot P(n_t = n|\mathbf{c}_{t-1} = \mathbf{c}') \cdot \sum_{\{\mathbf{c}|\mathbf{c}' \Rightarrow \mathbf{c}\}} P(c[1]|c[2..D]) P(w_t|\mathbf{c}) \beta_{\mathbf{c}}(t), \tag{B.3}$$

$$P(c_t[2..D] = c[2..D]) = \sum_{\{\mathbf{c}|c_t[2..D] = c[2..D]\}} \alpha_{\mathbf{c}}(t) \beta_{\mathbf{c}}(t), \tag{B.4}$$

$$P(\mathbf{c}_t = \mathbf{c}) = \alpha_{\mathbf{c}}(t)\beta_{\mathbf{c}}(t). \tag{B.5}$$

Finally, the forward and backward probabilities are defined recursively as follows:

$$\alpha_{\mathbf{c}}(t) = \sum_{\{\mathbf{c}'|\mathbf{c}'\Rightarrow\mathbf{c}\}} \alpha_{\mathbf{c}'}(t-1)P(\mathbf{c}|\mathbf{c}')P(w_t|\mathbf{c}), \tag{B.6}$$

$$\beta_{\mathbf{c}}(t) = \sum_{\{\mathbf{c}''|\mathbf{c}\Rightarrow\mathbf{c}''\}} P(\mathbf{c}''|\mathbf{c})P(w_{t+1}|\mathbf{c}'')\beta_{\mathbf{c}'}(t+1), \tag{B.7}$$

where

$$P(\mathbf{c}|\mathbf{c}') = P(n^*|\mathbf{c}')P(c[1]|c[2..D]). \tag{B.8}$$

Here, n^* is the value of stack shift needed to map $\mathbf{c}'$ into $\mathbf{c}$.

The notation $\mathbf{c}' \Rightarrow \mathbf{c}$ denotes all valid or legal derivations from any state vector $\mathbf{c}'$ to the state vector $\mathbf{c}$. This is the place where we apply the constraints on the dominance relationships between semantic concepts as described in Section 4.3.3 of the main text.

In all of the above formulae, a single training utterance is assumed. As in the case of regular HMMs, the extension to multiple utterances is straightforward.

Appendix C. Lexical classes extracted

See Table 7.

Table 7
Lexical classes extracted from ATIS and DARPA Communicator data

Corpus	Lexical classes
ATIS	aircraft_code, airline_code, airline_name, airport_code, airport_name, city_code, city_name, class_type, cost_relative, country_name, day_name, day_number, days_code, fare_basis_code, flight_mod, flight_stop, flight_time, manufacturer, meal_code, meal_description, month_name, period_of_day, restriction_code, round_trip, state_code, state_name, time, today_relative, transport_type, year
DARPA	airline_name, airport_code, airport_name, city_name, continent_name, country_name, day_name, day_number, hotel_name, month_name, period_of_day, rental_company, rental_type, round_trip, state_name, time, today_relative, year

References

Charniak, E., 2001. Immediate-head parsing for language models. In: Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics.

Chelba, C., Jelinek, F., 2000. Structured language modeling. Computer Speech and Language 14 (4), 283–332.

Chelba, C., Mahajan, M., 2001. Information extraction using the structured language model. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing.

CUData, 2004. DARPA Communicator Travel Data. University of Colorado at Boulder. Available from <http://communicator.colorado.edu/phoenix>.

CUPheonix, 2003. CU Pheonix Parser. University of Colorado at Boulder. Available from <http://communicator.colorado.edu/phoenix>.

Dahl, D., Bates, M., Brown, M., Hunicke-Smith, K., Pallett, D., Pao, C., Rudnicky, A., Shriberg, L., Mar. 1994. Expanding the scope of the ATIS task: the ATIS-3 corpus. In: ARPA Human Language Technology Workshop. Princeton, NJ.

Dowding, J., Moore, R., Andry, F., Moran, D., 1994. Interleaving syntax and semantics in an efficient bottom-up parser. In: Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics. Las Cruces, New Mexico, pp. 110–116.

Erdogan, H., Sarikaya, R., Gao, Y., Picheny, M., 2002. Semantic structured language models. In: Proceedings of the International Conference on Spoken Language Processing. Denver, CO.

Fine, S., Singer, Y., Tishby, N., 1998. The hierarchical hidden markov model: analysis and applications. Machine Learning 32, 41–62.

Gavalda, M., 2000. SOUP: a parser for real-world spontaneous speech. In: Proceedings of the 6th International Workshop on Parsing Technologies. Trento, Italy.

Goel, V., Byrne, W., 1999. Task dependent loss functions in speech recognition: application to named entity extraction. In: ESCA ETRW Workshop on Accessing Information from Spoken Audio. Cambridge, UK, pp. 49–53.

Levin, E., Pieraccini, R., 1995. CHRONUS, the next generation. In: Proceedings of the DARPA Speech and Natural Language Workshop. Morgan Kaufman, Austin, TX, pp. 269–271.

Miller, S., Bates, M., Bobrow, R., Ingria, R., Makhoul, J., Schwartz, R., 1995. Recent progress in hidden understanding models. In: Proceedings of the DARPA Speech and Natural Language Workshop. Morgan Kaufman, Austin, TX, pp. 276–280.

Minker, W., Gavalda, M., Waibel, A., 1999. Hidden understanding models for machine translation. In: European Speech Communication Association Tutorial and Research Workshop on Interactive Dialogue in Multi-Modal Systems (ESCA-1999). Kloster Irsee, Germany.

Ney, H., Essen, U., Kneser, R., 1994. On structuring probabilistic dependencies in stochastic language modelling. Computer Speech and Language 8 (1), 1–38.

Phillips, C., 1995. Right association in parsing and grammar. In: Schütze, C., Broihier, K., Ganger, J. (Eds.), Papers on Language Processing and Acquisition. pp. 37–93, MITWPL 26.

Pieraccini, R., Tzoukermann, E., Gorelov, Z., Levin, E., Lee, C., Gauvain, J., 1992. Progress report on the CHRONUS system: ATIS benchmark results. In: Proceedings of the DARPA Speech and Natural Language Workshop. Morgan Kaufman, Harriman, NY, pp. 67–71.

Schwartz, R., Miller, S., Stallard, D., Makhoul, J., 1997. Hidden understanding models for statistical sentence understanding. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing. Munich, pp. 1479–1482.

Ward, W., Issar, S., 1996. Recent improvements in the CMU spoken language understanding system. In: Proceedings of the ARPA Human Language Technology Workshop. Morgan Kaufman, pp. 213–216.

Witten, I., Bell, T., 1991. The zero frequency problem: estimating the probabilities of novel events in adaptive text compression. IEEE Transactions on Information Theory, 1085–1093.

Young, S., 2002a. The statistical approach to the design of spoken dialogue systems. Tech. rep. cued/f-infeng/tr.433, Cambridge, University Engineering Department.

Young, S., 2002b. Talking to machines (statistically speaking). In: Proceedings of the International Conference on Spoken Language Processing. Denver, CO.

Young, S., Evermann, G., Kershaw, D., Moore, G., Odell, J., Ollason, D., Povey, D., Valtchev, V., Woodland, P., 2004. The HTK Book (for HTK Version 3.2). Cambridge University Engineering Department. Available from <http://htk.eng.cam.ac.uk/>.