

# Probabilistic induction for an incremental semantic grammar\*

Arash Eshghi                      Matthew Purver

Julian Hough

Cognitive Science Research Group

School of Electronic Engineering and Computer Science

Queen Mary University of London

{arash,mpurver,jhough}@eeecs.qmul.ac.uk

## Abstract

We describe a method for learning an incremental semantic grammar from a corpus in which sentences are paired with logical forms as predicate-argument structure trees. Working in the framework of Dynamic Syntax, and assuming a set of generally available compositional mechanisms, we show how lexical entries can be learned as probabilistic procedures for the incremental projection of semantic structure, providing a grammar suitable for use in an incremental probabilistic parser. By inducing these from a corpus generated using an existing grammar, we demonstrate that this results in both good coverage and compatibility with the original entries, without requiring annotation at the word level. We show that this semantic approach to grammar induction has the novel ability to learn the syntactic and semantic constraints on pronouns.

## 1 Introduction

Dynamic Syntax (DS) is an inherently incremental semantic grammar formalism (Kempson et al., 2001; Cann et al., 2005) in which semantic representations are projected on a word-by-word basis. It recognises no intermediate layer of syntax (see below), but instead reflects grammatical constraints via constraints on the incremental construction of partial logical forms (LFs). Given this, and its definition of parsing and generation in terms of the same incremental processes, it is in principle capable of modelling and providing semantic interpretations for phenomena such as unfinished utterances, co-constructions and interruptions, beyond the remit of standard grammar formalisms but important for dialogue systems.

However, its definition in terms of semantics (rather than the more familiar syntactic phrase structure) makes it hard to define or extend broad-coverage grammars: expert linguists are required. Here, we present a method for automatically inducing DS grammars, by learning lexical entries from sentences paired with complete, compositionally structured, propositional LFs. By assuming only the availability of a small set of general compositional semantic operations, reflecting the properties of the lambda calculus and semantic conjunction, we ensure that the lexical entries learnt include the grammatical constraints and corresponding compositional semantic structure of the language; by additionally assuming a general semantic copying operation, we can also learn the syntactic and semantic properties of pronouns.

## 2 Previous work on grammar induction

Existing grammar induction methods can be divided into two major categories: supervised and unsupervised. Fully supervised methods use a parsed corpus as the training data, pairing sentences with syntactic trees and words with their syntactic categories, and generalise over the phrase structure rules to learn a grammar which can be applied to a new set of data. By estimating probabilities for production rules that

---

\*We would like to thank Ruth Kempson and Yo Sato for helpful comments and discussion. This work was supported by the EPSRC, RISER project (Ref: EP/J010383/1), and in part by the EU, FP7 project, SpaceBook (Grant agreement no: 270019).

share the same LHS category, this produces a grammar suitable for probabilistic parsing and disambiguation (e.g. PCFGs, Charniak, 1996). Such methods have shown great success, but presuppose detailed prior linguistic information (and are thus not adequate as human grammar learning models). Unsupervised methods, on the other hand, proceed from unannotated raw data; they are thus closer to the human language acquisition setting, but have seen less success. In its pure form—positive data only, without bias—unsupervised learning has been demonstrated to be computationally too complex (‘unlearnable’) in the worst case (Gold, 1967). Successful approaches involve some prior learning or bias, e.g. a fixed set of known lexical categories, a probability distribution bias (Klein and Manning, 2005) or a hybrid, semi-supervised method with shallower (e.g. POS-tagging) annotation (Pereira and Schabes, 1992).

More recently, another interesting line of work has emerged: *lightly* supervised learning guided by *semantic* rather than syntactic annotation, using sentence-level propositional logical form rather than detailed word-level annotation (more justifiably arguable to be ‘available’ to a human learner in a real-world situation, with some idea of what a string in an unknown language could mean). This has been successfully applied in Combinatorial Categorical Grammar (Steedman, 2000), as it tightly couples compositional semantics with syntax (Zettlemoyer and Collins, 2007; Kwiatkowski et al., 2010, 2012); as CCG is a lexicalist framework, grammar learning involves inducing a lexicon assigning to each word its syntactic and semantic contribution. Moreover, the grammar is learnt ground-up in an ‘incremental’ fashion, in the sense that the learner collects data over time and does the learning sentence by sentence.

Here we follow this spirit, inducing grammar from a propositional meaning representation and building a lexicon which specifies what each word contributes to the target semantics. However, taking advantage of the DS formalism, we make two novel contributions: first, we bring an added dimension of incrementality: not only is learning sentence-by-sentence incremental, but the grammar learned is word-by-word incremental, commensurate with psycholinguistic results showing incrementality to be a fundamental feature of human parsing and production Lombardo and Sturt (1997); Ferreira and Swets (2002). While incremental parsing algorithms for standard grammar formalisms have seen much research (Hale, 2001; Collins and Roark, 2004; Clark and Curran, 2007), to the best of our knowledge, a learning system for an explicitly incremental grammar is yet to be presented. Second, by using a grammar in which syntax and parsing context are defined in terms of the growth of semantic structures, we can learn lexical entries for items such as pronouns the constraints on which depend on semantic context.

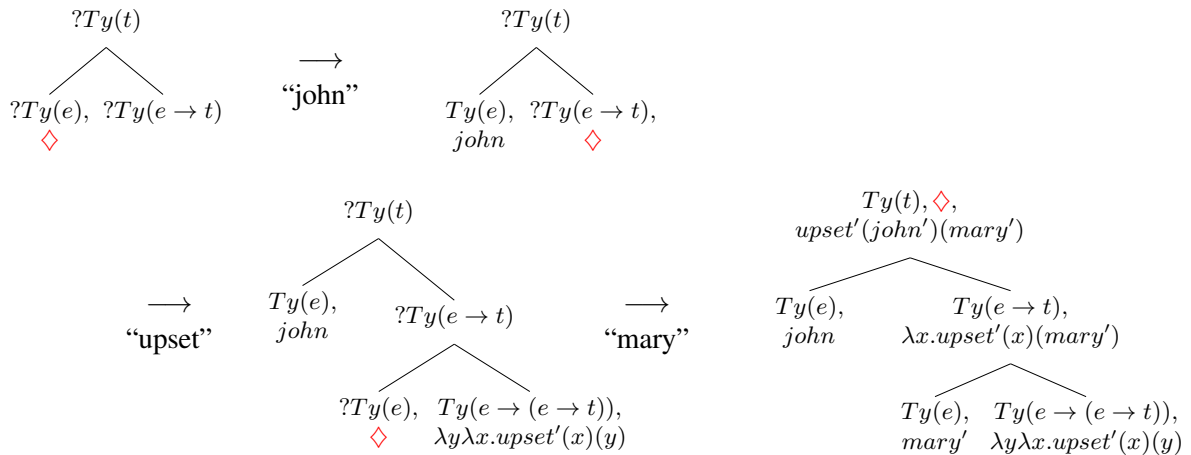


Figure 1: Incremental parsing in DS producing semantic trees: “John upset Mary”

### 3 Dynamic Syntax

Dynamic Syntax is a parsing-directed grammar formalism, which models the word-by-word incremental processing of linguistic input. Unlike many other formalisms, DS models the incremental building up of *interpretations* without presupposing or indeed recognising an independent level of syntactic processing. Thus, the output for any given string of words is a purely *semantic* tree representing its predicate-argument structure; tree nodes correspond to terms in the lambda calculus, decorated with la-

bels expressing their semantic type (e.g.  $Ty(e)$ ) and formula, with beta-reduction determining the type and formula at a mother node from those at its daughters (Figure 1).

These trees can be *partial*, containing unsatisfied requirements for node labels (e.g.  $?Ty(e)$  is a requirement for future development to  $Ty(e)$ ), and contain a *pointer*  $\diamond$  labelling the node currently under development. Grammaticality is defined as parsability: the successful incremental construction of a tree with no outstanding requirements (a *complete* tree) using all information given by the words in a sentence. The input to our induction task here is therefore sentences paired with such complete, *semantic* trees, and what we learn are constrained lexical procedures for the incremental construction of such trees. Note that in these trees, leaf nodes do not necessarily correspond to words, and may not be in linear sentence order (see Figure 1); and syntactic structure is not explicitly represented, only the structure of semantic predicate-argument combination.

### 3.1 Actions in DS

The parsing process is defined in terms of conditional *actions*: procedural specifications for monotonic tree growth. These take the form both of general structure-building principles (*computational actions*), putatively independent of any particular natural language, and of language-specific actions induced by parsing particular lexical items (*lexical actions*). The latter are what we here try to learn from data.

**Computational actions** These form a small, fixed set, and we assume them as given here. Some merely encode the properties of the lambda calculus and the logical tree formalism itself (LoFT Blackburn and Meyer-Viol, 1994) – these we term *inferential* actions. Examples include THINNING (removal of satisfied requirements) and ELIMINATION (beta-reduction of daughter nodes at the mother). These actions are entirely language-general, cause no ambiguity, and add no new information to the tree; as such, they apply non-optionally whenever their preconditions are met.

Other computational actions reflect the fundamental predictivity and dynamics of the DS framework. For example, \*ADJUNCTION introduces a single unfixed node with underspecified tree position (replacing feature-passing concepts for e.g. long-distance dependency); and LINK-ADJUNCTION builds a paired (“linked”) tree corresponding to semantic conjunction (licensing relative clauses, apposition and more). These actions represent possible parsing strategies and can apply optionally at any stage of a parse if their preconditions are met. While largely language-independent, some are specific to language type (e.g. INTRODUCTION-PREDICTION in the form used here applies only to SVO languages).

**Lexical actions** The lexicon associates words with lexical actions; like computational actions, these are sequences of tree-update actions in an IF..THEN..ELSE format, and composed of explicitly procedural *atomic* tree-building actions such as *make*, *go*, *put*. *make* creates a new daughter node, *go* moves the pointer, and *put* decorates the pointed node with a label. Figure 2 shows an example for a proper noun, *John*. The action checks whether the pointed node (marked as  $\diamond$ ) has a requirement for type  $e$ ; if so, it decorates it with type  $e$  (thus satisfying the requirement), formula  $John'$  and the bottom restriction  $\langle \downarrow \rangle \perp$  (meaning that the node cannot have any daughters). Otherwise (if no requirement  $?Ty(e)$ ), the action aborts, meaning that the word ‘*John*’ cannot be parsed in the context of the current tree.

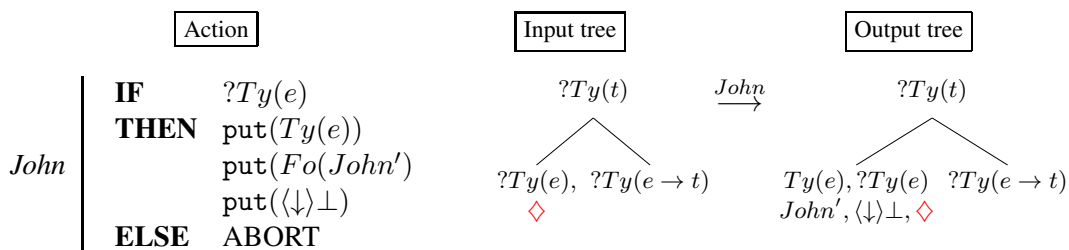
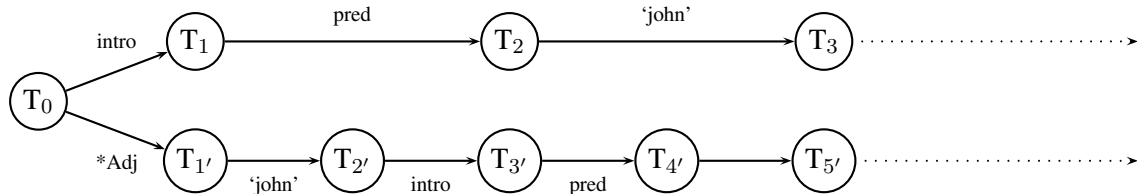


Figure 2: Lexical action for the word ‘John’

### 3.2 Graph Representation of DS Parsing

These actions define the parsing process. Given a sequence of words  $(w_1, w_2, \dots, w_n)$ , the parser starts from the *axiom* tree  $T_0$  (a requirement  $?Ty(t)$  to construct a complete tree of propositional type), and applies the corresponding lexical actions  $(a_1, a_2, \dots, a_n)$ , optionally interspersing computational actions – see Figure 1. Sato (2011) shows how this parsing process can be modelled on a *Directed Acyclic Graph* (DAG), rooted at  $T_0$ , with partial trees as nodes, and computational and lexical actions as edges (i.e. transitions between trees):



In this DAG, *intro*, *pred* and *\*Adj* correspond to the computational actions INTRODUCTION, PREDICTION and \*-ADJUNCTION respectively; and ‘john’ is a lexical action. Different paths through the DAG represent different parsing strategies, which may succeed or fail depending on how the utterance is continued. Here, the path  $T_0 - T_3$  will succeed if ‘John’ is the subject of an upcoming verb (“John upset Mary”);  $T_0 - T_4$  will succeed if ‘John’ turns out to be a left-dislocated object (“John, Mary upset”).

This DAG makes up the *parse state* at any point, and contains all information available to the parser. This includes semantic tree and tree-transition information taken to make up the *linguistic context* for ellipsis and pronominal construal (Purver et al., 2011). It also provides us with a basis for probabilistic parsing (see Sato, 2011): given a conditional probability distribution  $P(a|w, T)$  over possible actions  $a$  given a word  $w$  and (some set of features of) the current partial tree  $T$ , the DAG can then be incrementally constructed and traversed in a best-first, breadth-first or beam parsing manner.

## 4 Learning lexical actions

### 4.1 Problem Statement

Our task here is data-driven, probabilistic learning of lexical actions for all the words occurring in the corpus. Throughout, we will assume that the (language-independent) *computational actions* are known. We also assume that the supervision information is structured: i.e. our dataset pairs sentences with complete DS trees encoding their predicate-argument structures, rather than just a flat logical form (LF) as in e.g. Zettlemoyer and Collins (2007). DS trees provide more information than LFs in that they disambiguate between different possible predicate-argument decompositions of the corresponding LF; note however that this provides *no* extra information on the mapping from words to meaning. The input to the induction procedure is now as follows:

- the set of computational actions in Dynamic Syntax,  $G$ .
- a set of training examples of the form  $\langle S_i, T_i \rangle$ , where  $S_i = \langle w_1 \dots w_n \rangle$  is a sentence of the language and  $T_i$  – henceforth referred to as the *target tree* – is the complete semantic tree representing the compositional structure of the meaning of  $S_i$ .

The output is a grammar specifying the possible lexical actions for each word in the corpus. Given our data-driven approach, we take a probabilistic view: we take this grammar as associating each word  $w$  with a probability distribution  $\theta_w$  over lexical actions. In principle, for use in parsing, this distribution should specify the posterior probability  $p(a|w, T)$  of using a particular action  $a$  to parse a word  $w$  in the context of a particular partial tree  $T$ . However, here we make the simplifying assumption that actions are conditioned solely on one feature of a tree, the semantic type  $Ty$  of the currently pointed node; and that actions apply exclusively to one such type (i.e. ambiguity of type leads to multiple actions). This effectively simplifies our problem to specifying the probability  $p(a|w)$ .

In traditional DS terms, this is equivalent to assuming that all lexical actions have a simple IF clause of the form  $\text{IF } ?Ty(X)$ ; this is true of most lexical actions in existing DS grammars (see examples above), but not all. This assumption will lead to some over-generation – inducing actions which can parse some ungrammatical strings – we must rely on the probabilities learned to make such parses unlikely, and evaluate this in Section 5. Given this, the focus of what we learn here is effectively the THEN clause of lexical actions: a sequence of DS atomic actions such as *go*, *make*, and *put* (see Fig. 2), but now with an attendant posterior probability. We will henceforth refer to these sequences as *lexical hypotheses*. We first describe our method for constructing lexical hypotheses with a single training example (a sentence-tree pair). We then discuss how to generalise over these outputs, while updating the corresponding probability distributions incrementally as we process more training examples.

## 4.2 Hypothesis Construction

DS is *strictly monotonic*: actions can only *extend* the tree under construction, deleting nothing except satisfied requirements. Thus, hypothesising lexical actions consists in an incremental search through the space of all monotonic, and well-formed extensions of the current tree,  $T_{cur}$ , that subsume (i.e. can be extended to) the target tree  $T_t$ . This gives a bounded space which can be described by a DAG equivalent to the parsing DAG of section 3.2: nodes are trees, starting with  $T_{cur}$  and ending with  $T_t$ , and edges are possible extensions. These extensions may be either DS’s basic computational actions (already known) or new *lexical hypotheses*.

This space is further constrained by the fact that not all possible trees and tree extensions are well-formed (meaningful) in DS, due to the properties of the lambda-calculus and those of the modal tree logic LoFT. Mother nodes must be compatible with the semantic type and formula of their daughters, as would be derived by beta-reduction; formula decorations cannot apply without type decorations; and so on. We also prevent arbitrary type-raising by restricting the types allowed, taking the standard DS assumption that noun phrases have semantic type  $e$  (rather than a higher type as in Generalized Quantifier theory) and common nouns their own type  $cn$  (see Cann et al., 2005, chapter 3 for details).

We implement these constraints by packaging together permitted sequences of tree updates as macros (sequences of DS atomic actions such as *make*, *go*, and *put*), and hypothesising possible DAG paths based on these macros. We can divide these into two classes of lexical hypothesis macros: (1) *tree-building* hypotheses, independent of the target tree, and in charge of building appropriately typed daughters for the current node; and (2) *content decoration* hypotheses in charge of the semantic decoration of the leaves of the current tree ( $T_{cur}$ ), with formulae taken from the leaves of the target tree ( $T_t$ ).

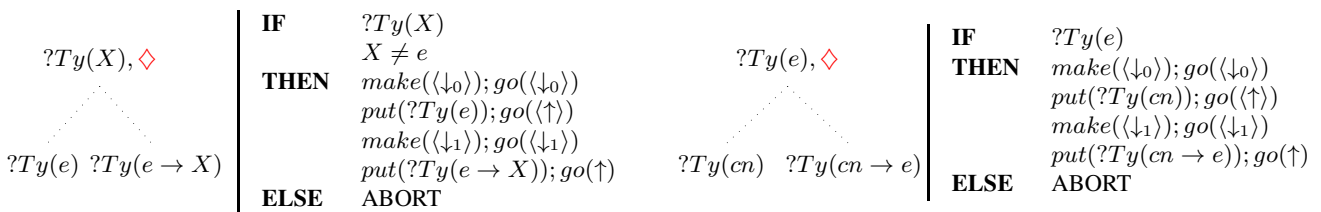


Figure 3: Target-independent tree-building hypotheses

Figure 3 shows example *tree-building* hypotheses which extend a mother node with a type requirement to have two daughter nodes which would (once themselves developed) combine to satisfy that requirement. On the left, an general rule in which a currently pointed node of some type  $X$  can be hypothesised to be formed of types  $e$  and  $e \rightarrow X$  (e.g. if  $X = e \rightarrow t$ , the daughters will have types  $e$  and  $e \rightarrow (e \rightarrow t)$ ). This reflects only the fact that DS trees correspond to lambda calculus terms, with  $e$  being a possible type. The other is more specific, suitable only for a type  $e$  node, allowing it to be composed of nodes of type  $cn$  and  $cn \rightarrow e$  (where  $cn \rightarrow e$  turns out to be the type of determiners), but again reflects only general semantic properties which would apply in any language.

*Content decoration* hypotheses on the other hand depend on the target tree: they posit possible addition of semantic content, via sequences of *put* operations (e.g. `content-dec: put (Ty (e)) ; put (Fo (john))`) which develop the pointed node on  $T_{cur}$  towards the corresponding leaf node on  $T_t$ .

They are constrained to apply only to *leaf* nodes (i.e. nodes in  $T_{cur}$  whose counterparts on  $T_t$  are leaf nodes), other nodes being assumed to receive their content via beta-reduction of their daughters.

### 4.3 Hypothesis Splitting

Hypothesis construction therefore produces, for each training sentence  $\langle w_1 \dots w_n \rangle$ , all possible sequences of actions that lead from the axiom tree  $T_0$  to the target tree  $T_t$  (henceforth, the *complete* sequences); where these sequences contain both lexical hypotheses and general computational actions. To form discrete lexical entries, we must split each such sequence into  $n$  sub-sequences,  $\langle cs_1 \dots cs_n \rangle$ , with each *candidate subsequence*  $cs_i$ , corresponding to a word  $w_i$ , by hypothesising a set of word boundaries.

This splitting process is subject to two constraints. Firstly, each candidate sequence  $cs_i$  must contain exactly one *content decoration* lexical hypothesis (see above); this ensures both that every word has some contribution to the sentence’s semantic content, and that no word decorates the leaves of the tree with semantic content more than once. Secondly, candidate subsequences  $cs_i$  are *computationally maximal* on the left:  $cs_i$  may begin with (possibly multiple) computational actions, but must end with a lexical hypothesis. This reduces the splitting hypothesis space, and aids lexical generalisation (see below).

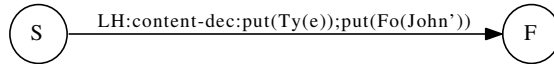
Each such possible set of boundaries corresponds to a *candidate sequence tuple*  $\langle cs_1 \dots cs_n \rangle$ . Importantly, this means that these  $cs_i$  are not independent, e.g. when processing “John arrives”, a hypothesis for ‘John’ is only compatible with certain hypotheses for ‘arrives’. This is reflected below in how probabilities are assigned to the word hypotheses.

### 4.4 Hypothesis Generalisation

DS’s general computational actions can apply at any point before the application of a lexical action, thus providing strategies for adjusting the syntactic context in which a word is parsed. Removing computational actions on the left of a candidate sequence will leave a more general albeit equivalent hypothesis: one which will apply successfully in more syntactic contexts. However, if a computational subsequence seems to occur *whenever* a word is observed, we would like to lexicalise it, including it within the lexical entry for a more efficient and constrained grammar. We therefore want to generalise over our candidate sequence tuples to partition them into portions which seem to be achieved lexically, and portions which are better achieved by computational actions alone.

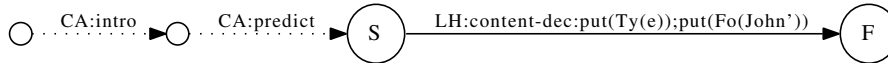
First Training Example: ‘john’ in fixed object position;

Sequence intersected:  $\langle LH : content-dec : put(Ty(e)); put(Fo(John')) \rangle$ :



Second Training Example: ‘john’ in subject position;

Sequence intersected:  $\langle CA : intro, CA : predict, LH : content-dec : put(Ty(e)); put(Fo(John')) \rangle$



Third Training Example: ‘john’ on unfixed node, i.e. left-dislocated object;

Sequence intersected:  $\langle CA : star-adj, LH : content-dec : put(Ty(e)); put(Fo(John')) \rangle$

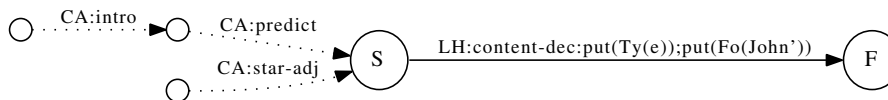


Figure 4: Incremental intersection of candidate sequences; CA=Computational Action, LH=Lexical Hypothesis

We therefore group the candidate sequence tuples produced by splitting, storing them as members of equivalence classes which form our final *word hypotheses*. Two tuples belong to the same equivalence class if they can be made identical by removing *only* computational actions from the beginning of either one. We implement this via a single packed data-structure which is again a DAG, as shown in Fig. 4; this represents the full set of candidate sequences by their intersection (the solid central common path)

and differences (the dotted diverging paths at beginning). Nodes here therefore no longer represent single trees, but sets of trees. Figure 4 shows this process over three training examples containing the unknown word ‘John’ in different syntactic positions. The ‘S’ and ‘F’ nodes mark the start and finish of the intersection – initially the entire sequence. As new candidate sequences arrive, the intersection – the maximal common path – is reduced as appropriate. Word hypotheses thus remain as general as possible.

In our probabilistic framework, these DAGs themselves are our lexical entries, with associated probabilities (see below). If desired, we can form traditional DS lexical actions: the DAG intersection corresponds to the THEN clause, with the IF clauses being a type requirement obtained from the pointed node on all partial trees in the initial ‘S’ node. As lexical hypotheses within the intersection are identical, and were constrained when formed to add type information before formula information (see Section 4.2), any type information must be common across these partial trees. In Figure 4 for ‘john’, this is  $?Ty(e)$ , i.e. a requirement for type  $e$ , common to all three training examples.

#### 4.5 Probability Estimation

The set of possible word hypotheses induced as above can of course span a very large space: we must therefore infer a probability distribution over this space to produce a useful grammar. This can be estimated from the observed distribution of hypotheses, as these are constrained to be compatible with the target tree for each sentence; and the estimates can be incrementally updated as we process each training example. For this process of probability estimation, the input is the output of the splitting and generalisation procedure above, i.e. for the current training sentence  $S = \langle w_1 \dots w_n \rangle$  a set  $HT$  of *Hypothesis Tuples* (sequences of word hypotheses), each of the form  $HT_j = \langle h_1^j \dots h_n^j \rangle$ , where  $h_i^j$  is the word hypothesis for  $w_i$  in  $HT_j$ . The desired output is a probability distribution  $\theta_w$  over hypotheses for each word  $w$ , where  $\theta_w(h)$  is the posterior probability  $p(h|w)$  of a given word hypothesis  $h$  being used to parse  $w$ .

**Re-estimation** Given some prior estimate of  $\theta'_w$ , we can use a new training example to produce an updated estimate  $\theta''_w$  directly. We assign each hypothesis tuple  $HT_j$  a probability based on  $\theta'_w$ ; the probability of a sequence  $\langle h_1^j \dots h_n^j \rangle$  is the product of the probabilities of the  $h_i^j$ ’s within it (by the Bayes chain rule):

$$p(HT_j|S) = \prod_{i=1}^n p(h_i^j|w_i) = \prod_{i=1}^n \theta'_{w_i}(h_i^j) \quad (1)$$

Now, for any word  $w$  and possible hypothesis  $h$ , we can re-estimate the probability  $p(h|w)$  as the normalised sum of the probabilities of all observed tuples  $HT_j$  which contain  $h$ , that is the set of tuples,  $HT^h = \{HT_j|h \in HT_j\}$ :

$$\theta''_w(h) = p(h|w) = \frac{1}{Z} \sum_{HT_j \in HT^h} p(HT_j|S) = \frac{1}{Z} \sum_{HT_j \in HT^h} \prod_{i=1}^n \theta'_{w_i}(h_i^j) \quad (2)$$

where  $Z$ , the normalising constant, is the sum of the probabilities of all the  $HT_j$ ’s:

$$Z = \sum_{HT_j \in HT} \prod_{i=1}^n \theta'_{w_i}(h_i^j)$$

**Incremental update** Our procedure is now to update our overall estimate  $\theta_w$  incrementally: after the  $N$ th example, our new estimate  $\theta_w^N$  is a weighted average of the previous estimate  $\theta_w^{N-1}$  and the new value from the current example  $\theta''_w$  from equation (2), with weights reflecting the amount of evidence on which these estimates are based:

$$\theta_w^N(h) = \frac{N-1}{N} \theta_w^{N-1}(h) + \frac{1}{N} \theta''_w(h) \quad (3)$$

Note that for training example 1, the first term’s numerator is zero, so  $\theta_w^{N-1}$  is not required and the new estimates are equal to  $\theta''_w$ . However, to produce  $\theta''_w$  we need some prior estimate  $\theta'_w$ ; in the absence of

any information, we simply assume uniform distributions  $\theta'_w = \theta_w^0$  over the lexical hypotheses observed in the first training example.

In subsequent training examples, there will arise new hypotheses  $h$  not seen in previous examples, and for which the prior estimate  $\theta'_w$  gives no information. We incorporate these hypotheses into  $\theta'_w$  by discounting the probabilities assigned to known hypotheses, reserving some probability mass which we then assume to be evenly distributed over the new unseen hypotheses. For this we use the same weight as in equation (3):

$$\theta'_w(h) = \begin{cases} \frac{N-1}{N}\theta_w^{N-1}(h) & \text{if } h \text{ in } \theta_w^{N-1} \\ \frac{1}{N_u} \sum_{h \in \theta_w^{N-1}} \frac{1}{N}\theta_w^{N-1}(h) & \text{otherwise} \end{cases} \quad (4)$$

where  $N_u$  here is number of new unseen hypotheses in example  $N$ . Given (4), we can now more accurately specify the update procedure in (3) to be:

$$\theta_w^N(h) = \theta'_w(h) + \frac{1}{N}\theta''_w(h) \quad (5)$$

**Non-incremental estimation** Using this incremental procedure, we use the estimates from previous sentences to assign prior probabilities to each hypothesis tuple (i.e. each possible path through the hypothesised parse DAG), and then derive updated posterior estimates given the observed distributions. Such a procedure could similarly be applied non-incrementally at each point, by repeatedly re-estimating and using the new estimates to re-calculate tuple probabilities in a version of the Expectation-Maximisation algorithm (Dempster et al., 1977). However, this would require us to keep all *HT* sets from every training example; this would be not only computationally demanding but seems psycholinguistically implausible (requiring memory for all lexical and syntactic dependencies for each sentence). Instead, we restrict ourselves here to assuming that this detailed information is only kept in memory for one sentence; intermediate versions would be possible.

## 4.6 Pronouns

Standard approaches to grammar induction treat pronouns simply as entries of a particular syntactic category. Here, as we learn from semantic annotations, we can learn not only their anaphoric nature, but syntactic and semantic constraints on their resolution. To achieve this, we assume one further general strategy for lexical hypothesis formation: a copying operation from context whereby the semantic content (formula and type decorations) can be copied from any existing type-compatible and complete node on  $T_{cur}$  (possibly more than one) accessible from the current pointed node via some finite tree modality. This assumption therefore provides the general concept of anaphoricity, but nothing more: it can be used in hypothesis formation for any word, and we rely on observed probabilities of its providing a successful parse to rule it out for words other than pronouns. By requiring access via some tree modality ( $\uparrow_0, \downarrow_*$  etc), we restrict it to intrasentential anaphora here, but the method could be applied to intersentential cases where suitable LFs are available.

This modal relation describes the relative position of the antecedent; by storing this as part of the hypothesis DAG, and subjecting it to a generalisation procedure similar to that used for computational actions in Section 4.4, the system learns constraints on these modal relations. The lexical entries resulting can therefore express constraints on the possible antecedents, and grammatical constraints on their presence, akin to Principles A and B of Government and Binding theory (see Cann et al. (2005), chapter 2); in this paper, we evaluate the case of relative pronouns only (see below).

## 5 Evaluation

### 5.1 Parse coverage

This induction method has been implemented and tested over a 200-sentence artificial corpus. The corpus was generated using a manually defined DS grammar, with words randomly chosen to follow the



Word class	Type	Token	Type%	Token%
noun	119	362	76.3%	48.7%
verb	29	263	18.6%	35.4%
determiner	3	56	1.9%	7.5%
pronoun	5	62	3.2%	8.4%
Total	156	743	100.00%	100.00%
Total of 200 sentences				
Min, Max and Mean sentence lengths : 2, 6, 3.7 words				
Mean tokens per word = 4.01				

Table 1: Training and test corpus distributions and means

	Parsing Coverage	Same Formula
Top one	26%	77%
Top two	77%	79%
Top three	100%	80%

Table 2: Test parse results: showing percentage parsability, and percentage of parses deriving the correct semantic content for the whole sentence

distributions of the relevant POS types and tokens in the CHILDES maternal speech data (MacWhinney, 2000) - see Table 1. 90% of the sentences were used as training data to induce a grammar, and the remaining 10% used to test it. We evaluate the results in terms of both parse coverage and semantic accuracy, via comparison with the logical forms derived using the original, hand-crafted grammar.

The induced hypotheses for each word were ranked according to their probability; three separate grammars were formed using the top one, top two and top three hypotheses and were then used independently to parse the test set. Table 2 shows the results, discounting sentences containing words not encountered in training at all (for which no parse is possible). We give the percentage of test sentences for which a complete parse was obtained; and the percentage of those for which one of the top 3 parses resulted in a logical form identical to the correct one.

As Table 2 shows, when the top three hypotheses are retained for each word, we obtain 80% formula derivation accuracy. Manual inspection of the individual actions learned revealed that the words which have incorrect lexical entries at rank one were those which were sparse in the corpus - we did not control for the exact frequency of occurrence of each word. The required frequency of occurrence varied across different categories; while transitive verbs require about four occurrences, intransitive verbs require just one. Count nouns were particularly sparse (see type/token ratios in Table 1).

As we have not yet evaluated our method on a real corpus, the results obtained are difficult to compare directly with other baselines such as that of Kwiatkowski et al. (2012) who achieve state-of-the-art results; cross-validation of this method on the CHILDES corpus is work in progress, which will allow direct comparison with Kwiatkowski et al. (2012).

## 5.2 Lexical Ambiguity

We introduced lexically ambiguous words into the corpus to test the ability of the system to learn and distinguish between their different senses; 10% of word types were ambiguous between 2 or 3 different senses with different syntactic category. Inspection of the induced actions for these words shows that, given appropriately balanced frequencies of occurrence of each separate word sense in the corpus, the system is able to learn and distinguish between them. 57% of the ambiguous words had lexical entries with both senses among the top three hypotheses, although in only one case were the two senses ranked one and two. This was the verb ‘tramped’ with transitive and intransitive readings, with 4 and 21 occurrences in the corpus respectively.

### 5.3 Pronouns

For pronouns, we wish to learn both their anaphoric nature (resolution from context) and appropriate syntactic constraints. Here, we tested on relative pronouns such as ‘who’ in “John likes Mary, who runs”: the most general lexical action hypothesis learned for these is identical to hand-crafted versions of the action (see Cann et al. (2005), chapter 3):

who	IF	?Ty(e) $\langle \uparrow_* \uparrow_L \rangle Fo(X)$
	THEN	put(Ty(e)) put(Fo(X)) put( $\langle \downarrow \rangle \perp$ )
	ELSE	ABORT

This action instructs the parser to copy a semantic type and formula from a type  $Ty(e)$  node at the modality  $\langle \uparrow_* \uparrow_L \rangle$ , relative to the pointed node. The system has therefore learnt that pronouns involve resolution from context (note that many other hypotheses are possible, as pronouns are paired with different LFs in different sentences). It also expresses a syntactic constraint on relative pronouns, that is, the relative position of their antecedents  $\langle \uparrow_* \uparrow_L \rangle$  (the first node above a dominating LINK tree relation – i.e. the head of the containing NP).

Of course, relative pronouns are a special case: the modality from which their antecedents are copied is relatively fixed. Equivalent constraints could be learned for other pronouns, given generalisation over several modal relations; e.g. locality of antecedents for reflexives is specified in DS via a constraint  $\langle \uparrow_0 \uparrow_1^* \downarrow_0 \rangle$  requiring the antecedent to be in *some* local argument position. In learning reflexives, this modal relation can come from generalisation over several different modalities obtained from different training examples; this will require larger corpora.

## 6 Conclusions and Future work

In this paper we have outlined a novel method for the probabilistic induction of new lexical entries in an inherently incremental and semantic grammar formalism, Dynamic Syntax, with no independent level of syntactic phrase structure. Our method learns from sentences paired with semantic trees representing the sentences’ predicate-argument structures, assuming only very general compositional mechanisms. While the method still requires evaluation on real data, evaluation on an artificial but statistically representative corpus demonstrates that the method achieves good coverage. A further bonus of using a semantic grammar is that it has the potential to learn both semantic and syntactic constraints on pronouns: our evaluation demonstrates this for relative pronouns, but this can be extended to other pronoun types.

Our research now focusses on evaluating this method on real data (the CHILDES corpus), and on reducing the level of supervision by adapting the method to learn from sentences paired not with trees but with less structured LFs, using Type Theory with Records Cooper (2005) and/or the lambda calculus. Other work planned includes the integration of the actions learned into a probabilistic parser.

## References

- Blackburn, P. and W. Meyer-Viol (1994). Linguistics, logic and finite trees. *Logic Journal of the Interest Group of Pure and Applied Logics* 2(1), 3–29.
- Cann, R., R. Kempson, and L. Marten (2005). *The Dynamics of Language*. Oxford: Elsevier.
- Charniak, E. (1996). *Statistical Language Learning*. MIT Press.
- Clark, S. and J. Curran (2007). Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics* 33(4), 493–552.
- Collins, M. and B. Roark (2004). Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the ACL*, Barcelona, pp. 111–118.

- Cooper, R. (2005). Records and record types in semantic theory. *Journal of Logic and Computation* 15(2), 99–112.
- Dempster, A., N. Laird, and D. B. Rubin (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* 39(1), 1–38.
- Ferreira, F. and B. Swets (2002). How incremental is language production? evidence from the production of utterances requiring the computation of arithmetic sums. *Journal of Memory and Language* 46, 57–84.
- Gold, E. M. (1967). Language identification in the limit. *Information and Control* 10(5), 447–474.
- Hale, J. (2001). A probabilistic Earley parser as a psycholinguistic model. In *Proceedings of the 2nd Conference of the North American Chapter of the Association for Computational Linguistics*, Pittsburgh, PA.
- Kempson, R., W. Meyer-Viol, and D. Gabbay (2001). *Dynamic Syntax: The Flow of Language Understanding*. Blackwell.
- Klein, D. and C. D. Manning (2005). Natural language grammar induction with a generative constituent-context mode. *Pattern Recognition* 38(9), 1407–1419.
- Kwiatkowski, T., S. Goldwater, L. Zettlemoyer, and M. Steedman (2012). A probabilistic model of syntactic and semantic acquisition from child-directed utterances and their meanings. In *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.
- Kwiatkowski, T., L. Zettlemoyer, S. Goldwater, and M. Steedman (2010, October). Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, Cambridge, MA, pp. 1223–1233. Association for Computational Linguistics.
- Lombardo, V. and P. Sturt (1997). Incremental processing and infinite local ambiguity. In *Proceedings of the 1997 Cognitive Science Conference*.
- MacWhinney, B. (2000). *The CHILDES Project: Tools for Analyzing Talk* (Third ed.). Mahwah, New Jersey: Lawrence Erlbaum Associates.
- Pereira, F. and Y. Schabes (1992, June). Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, Newark, Delaware, USA, pp. 128–135. Association for Computational Linguistics.
- Purver, M., A. Eshghi, and J. Hough (2011, January). Incremental semantic construction in a dialogue system. In J. Bos and S. Pulman (Eds.), *Proceedings of the 9th International Conference on Computational Semantics*, Oxford, UK, pp. 365–369.
- Sato, Y. (2011). Local ambiguity, search strategies and parsing in Dynamic Syntax. In E. Gregoromichelaki, R. Kempson, and C. Howes (Eds.), *The Dynamics of Lexical Interfaces*. CSLI Publications.
- Steedman, M. (2000). *The Syntactic Process*. Cambridge, MA: MIT Press.
- Zettlemoyer, L. and M. Collins (2007). Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.