Incremental Semantics Driven Natural Language Generation With Self-Repairing Capability

Julian Hough
Interaction, Media and Communication Research Group,
School of Electronic Engineering and Computer Science,
Queen Mary University of London
julian.hough@eecs.qmul.ac.uk

Abstract

This paper presents the on-going development of a model of incremental semantics driven natural language generation (NLG) for incremental dialogue systems. The approach is novel in its tight integration of incremental goal-driven semantics and syntactic construction, utilizing Type Theory with Records (TTR) record types for goal concepts as its input and the grammar formalism Dynamic Syntax (DS) for a word-by-word tactical generation procedure. The characterization of generation in terms of semantic input and word output graphs allows an integration into the incremental dialogue system Jindigo and facilitates the generation of human-like self-repairs in a semantically and syntactically motivated way.

1 Introduction

Recently, the arrival of incremental frameworks such as that described by Schlangen and Skantze (2011) has reignited the challenges for dialogue systems. Their implementation has recently shown success in micro-domains (Skantze and Schlangen, 2009) and has included some incremental natural language generation (NLG) capabilities which have been shown to be favored by users over non-incremental counterparts (Skantze and Hjalmarsson, 2010). However, this new brand of system has not taken account of incremental semantic processing on a word-by-word level in generation, which is the nature of the model for NLG described here.

The consequences of taking a fine-grained incremental semantics approach for NLG include the possibility of closer integration with parsing, and the incorporation of *self-repair* in a natural and context-sensitive way. Integrating self-repair into generation in the way described here should be beneficial for incremental dialogue systems with fragmentary and changing inputs to generation, and could also give some insights for modeling speech production.

1.1 Related Work

Traditionally, incremental generation has been motivated by developing autonomous processing models of human speech production. In particular, Kempen and Hoenkamp (1987) and Levelt (1989)'s functional decomposition of distinct *conceptualization*, *formulation* and *articulation* phases provided a psycholinguistic model which continues to have an influence on NLG. Motivated by modeling memory limitation, the principle of incrementality was generally taken that the syntactic formulator was able to begin its processing without complete input from the conceptualizer- in grammatical terms, tree formation could be both lexically and conceptually guided- see e.g. De Smedt (1990).

Guhe (2007) modeled an incremental conceptualizer which generated pre-verbal messages in a piece-meal fashion. While formulation was not the focus, the benefit of incremental semantic construction was clear: the conceptualizer's incremental modification of pre-verbal messages could influence downstream tactical generation decisions, particularly with 'correction' increments causing self-repairs.

Albeit less psychologically motivated, Skantze and Hjalmarsson (2010) provide a similar approach to Guhe in implementing incremental speech generation in a dialogue system. Generation input is defined as canned-text *speech plans* sent from the dialogue manager divided up into

Proceedings of the Student Research Workshop associated with RANLP 2011, pages 79–84,
Hissar, Bulgaria, 13 September 2011.

word-length *speech units*. Incremental generation is invoked to allow speech plans to change dynamically during interaction with a user: a simple string-based comparison of the incoming plan with the current one being vocalized allows both *covert* and *overt* self-repairs to be generated depending on the number of units in the plan realized at the point of difference detection.

This paper describes a new model for incremental generation that incorporates word-by-word semantic construction and self-repairing capability, going beyond string-based plan corrections and strict delineation of conceptualization and surface realization. The model is also portable into incremental dialogue systems.

2 Background

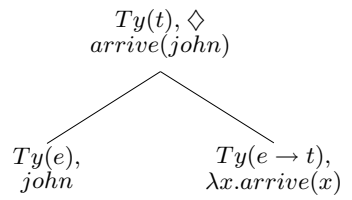
2.1 Dynamic Syntax (DS)

Dynamic Syntax (DS, Kempson et al., 2001) is an action-based and semantically oriented incremental grammar that defines grammaticality as parsability. The DS lexicon consists of *lexical actions* keyed to words, and also a set of globally applicable *computational actions*, both of which constitute packages of monotonic update operations on semantic trees and take the form of IF-THEN action-like structures such as (1).

(1)

john:	
IF	$?Ty(e)$
THEN	put($Ty(e)$)
	put($fo(john')$)
ELSE	abort

(2)



In DS parsing, if the pointer object ($\Diamond$) currently satisfies the precondition of an action, (e.g. is at a node of type $?Ty(e)$), then simple monotonic tree update operations of the tree logic LOFT (Blackburn and Meyer-Viol, 1994) are licensed. The trees represent terms in the typed lambda calculus, with mother-daughter node relations corresponding to semantic predicate-argument structure with no independent layer of syntax- see (2). Parsing begins with an axiom tree with a single node of requirement type $?Ty(t)$, and intersperses testing and application of lexical actions triggered by input words and execution of permissible (Kleene* iterated) sequences of computational actions.

Successful parses are sequences of applications of actions that lead to a tree which is complete (i.e. has no type requirements $?Ty(..)$ on any node, and has type $Ty(t)$ at its root node as in (2)) with a compiled formula. Incomplete *partial* structures are also maintained in the parse state as words are scanned in the input.

2.2 DS Generation as Parsing

As Purver and Kempson (2004) demonstrate, a tactical model of DS generation can be neatly defined in terms of the DS parsing process and a *subsumption check* against a *goal tree*. Goal trees are complete and fully specified DS trees such as (2), and generation consists of attempting to parse each word in the lexicon given the trees under construction, followed by a check to remove trees which do not subsume the goal tree from the parse state. Due to the stage-by-stage iteration through the lexicon, the DS generation process effectively combines lexical selection and linearization into a single action. Also, while no formal model of self-repair has hitherto been proposed in DS, self-monitoring is inherently part of the generation process, as each word generated is parsed.

2.3 Jindigo and the DyLan Interpreter

Jindigo (Skantze and Hjalmarsson, 2010) is a Java-implemented dialogue system following the abstract framework described by Schlangen and Skantze (2011). Its system is a network of modules, each consisting of a *left buffer* for input increments, a *processor* and a *right buffer* for the output increments. It is the *adding*, *commitment* and *revoking* of *incremental units* (IUs) in a module's right buffer and the effect of doing so on another module's left buffer that determines system behaviour, and the IUs can have *groundedIn* relations between one another if it is desirable that they should be in some way inter-dependent. The buffers are defined graphically, with vertices and edges representing IUs, allowing for multiple hypotheses in speech recognition and, as mentioned, revision of canned-text speech plans in generation (Skantze and Hjalmarsson, 2010).

There is an implementation of a DS parser in Jindigo in the `DyLan` interpreter module (Purver et al., 2011), where a parse state is characterized as a Directed Acyclic Graph (DAG), following Sato (2011), with DS *actions* for edges and *trees* for nodes. The characterization allows an exploitation of Jindigo's graph-based buffers, particularly

the interface with word graphs sent from a voice recognition (ASR) module: DyLan incrementally attempts to parse word hypothesis edges as they become available, and parse paths in the DAG are *groundedIn* the corresponding word edges of the ASR graph- see figure 1.

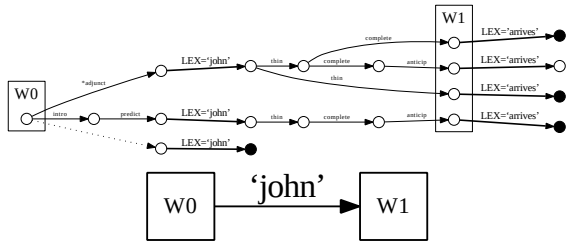


Figure 1: DS parsing process as a DAG, *groundedIn* corresponding word graph hypothesis edge 'john' spanning vertices W0 and W1

The application of a computational or lexical action can be seen as a labeled left-right transition edge between the trees under construction, which are represented by circular white nodes in the graph. Parts of the parse DAG are *groundedIn* the edge labelled with the word whose parse process they represent.

The other addition to DS in DyLan is the incorporation of Type Theory with Records (TTR) (Cooper, 2005), which can seen in (3). TTR *record types* decorate the nodes of the tree as opposed to simple atomic formulae, with each *field* in the record type containing a variable name, a value (after the =), which can be null for *unmanifest* fields, and a type (after the colon) which represents the node type of the DS tree at which its potential formula value is situated- basic types e and t are used here for clarity.

(3) *"John arrived"* $\longmapsto$

$$\Diamond, Ty(t), \left[\begin{array}{ll} x =_{john} & : e \\ p =_{arrive(x)} & : t \end{array}\right]$$

$$Ty(e), \quad [\ x =_{john} : e\ ]$$

$$Ty(e \rightarrow t), \quad \lambda r : [\ x1 : e\ ] \quad \left[\begin{array}{ll} x_{=r.x1} & : e \\ p_{=arrive(x)} & : t \end{array}\right]$$

The TTR adaptation is made to provide representations that can interface with domain conceptual structures in the rest of the dialogue system. DyLan automatically compiles a record type at the root node of a complete tree and checks this against system domain concepts.

3 Incremental Semantics Driven NLG

3.1 Goal Concepts and Incremental TTR Construction

To achieve thorough-going incrementality in terms of semantic content, the model proposed here modifies the DS generation procedure described in (Purver and Kempson, 2004) in two principal ways. Firstly, a TTR record type is compiled after each word candidate is parsed, giving maximal TTR representations for *partial trees* in addition to complete ones. Implementationally, this is achieved by a simple two-stage algorithm of firstly decorating nodes lacking formulae with record types containing the appropriate types- e.g. $[p =_{U(x)}: t]$ for a $Ty(e \rightarrow t)$ node, $[p =_{U(x,y)}: t]$ for a $Ty(e \rightarrow (e \rightarrow t))$ node etc.[1] Secondly, the functional application from the record types of the functor nodes to the record types of their sister argument nodes is carried out, compiling a β-reduced record type at their mother node. The ordering of the applications is achieved through an iterative search for functor nodes with uncompiled mother nodes, halting upon compilation of a formula at the root node.

The second principal modification is the replacement of a goal tree with a *goal concept* represented by a TTR record type. Consequently, tree subsumption checking is replaced by a *semantic pruning* stage, whereby parse paths that do not compile a valid *supertype* of the goal TTR record type are abandoned. The supertype check involves a recursive mapping of the fields of the candidate record type under inspection to the goal subtype, with testing for type consistency, arity of predicates, and position of arguments[2].

An example of a successful generation path is shown in figure 2, where the incremental generation of "john arrives" succeeds as successful lexical action applications are interspersed with applicable computational action sequences (e.g. transitions $\boxed{0} \mapsto \boxed{1}$ and $\boxed{2} \mapsto \boxed{3}$), at each stage passing the supertype relation check against the goal, until arriving at a tree that *type matches* in $\boxed{4}$.

[1] Technically, these functor node record types should be functions from record type to record type, as can be seen on the $Ty(e \rightarrow t)$ node in figure 3, for simplicity they are not fully represented in the discussion from here on but as simple record types with metavariable arguments.

[2] The fields with the underspecified value U are mapped successfully if they pass this type-checking stage, as they have underspecified semantics.

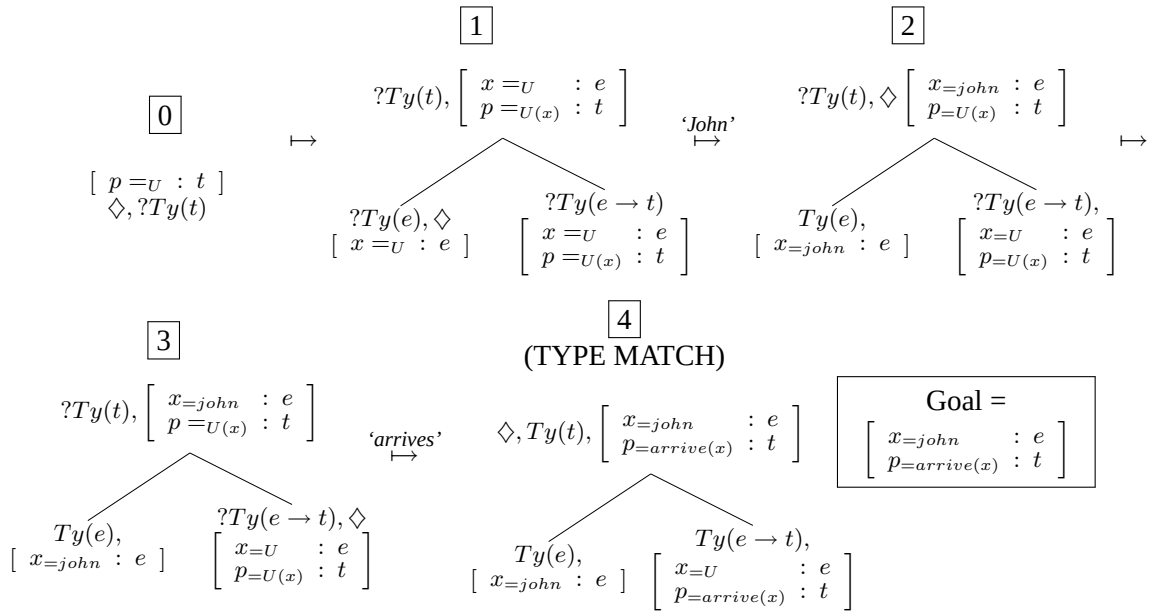


Figure 2: Successful generation path in DS-TTR

3.2 Implementation In Jindigo

The proposed DS-TTR generator has been implemented in Jindigo as a prototype module, not only facilitating tight integration with the DS parsing module `DyLan`, but also allowing dynamically changing inputs to generation and revisions of word selection.

In our module, incremental units (IUs) in the left buffer are defined as goal concept TTR record types (as in the Goal in figure 2), encoded in a simple XML attribute-value structure and posted by the dialogue manager. Our module's other input IUs are DS *parse state edges* from the `DyLan` parsing module which are used to update the module's current parse state during generation. Output IUs in the right buffer are *word edges* in a word graph made available to the speech synthesizer module, and *parse state edges* available to `DyLan's` left buffer[3]. Word edges in the word graph are *groundedIn* their corresponding parse state edge IUs, the same way as shown for parsing in figure 1.

Schematically, the procedure for word-by-word generation is as follows: At a state vertex S_n in the parse state edge graph, given latest committed parse state edge SE_{n-1} (or null edge if S_n is initial), current goal concept G and latest word graph vertex W_n:

1. **Syntactic Parse**: For each path-final tree in SE_{n-1}, attempt to apply all lexical actions Σl_k keyed by words Σw_k in the lexicon[4]. Apply all possible sequences of computational actions to the new trees. For each successful parse i `add` a hypothesis parse state edge SE_n^i to the parse state graph and `add` corresponding word edge WE_n^i to the word graph, making it *groundedIn* SE_n^i.
2. **Semantic Prune**: For each edge SE_n^i calculate maximal TTR representation T_n^i, and `revoke` SE_n^i from output buffer if T_n^i is not a valid supertype of G.
3. **Repair** IF all edges ΣSE_n are revoked, `repair`[5]: return to vertex S_{n-1} and repeat step (1) for *committed* edge SE_{n-2} that SE_{n-1} is *groundedIn* ELSE Continue.
4. **Update output buffer**: `commit` all remaining edges ΣSE_n. For each word edge candidate WE_n^i in the output buffer word graph, `commit` if *groundedIn* committed parse state edge SE_n^i.
5. **Halt**: If for some i, T_n^i and G are *type matched*.

4 Self-Repairing Capability

Due to Jindigo's constantly updating system threads, a goal concept may be revised shortly

[3]While not being fully addressed here, it is worth noting that our generation module and `DyLan` both maintain the same parse state edge graph in their output buffers, effecting an interleaving of the two modules.

[4]Work towards addressing the computational cost of testing each lexical action in the lexicon is currently being worked on.

[5]Optional commitment of interregnum filler such as "uhh", dependent on generation time taken.

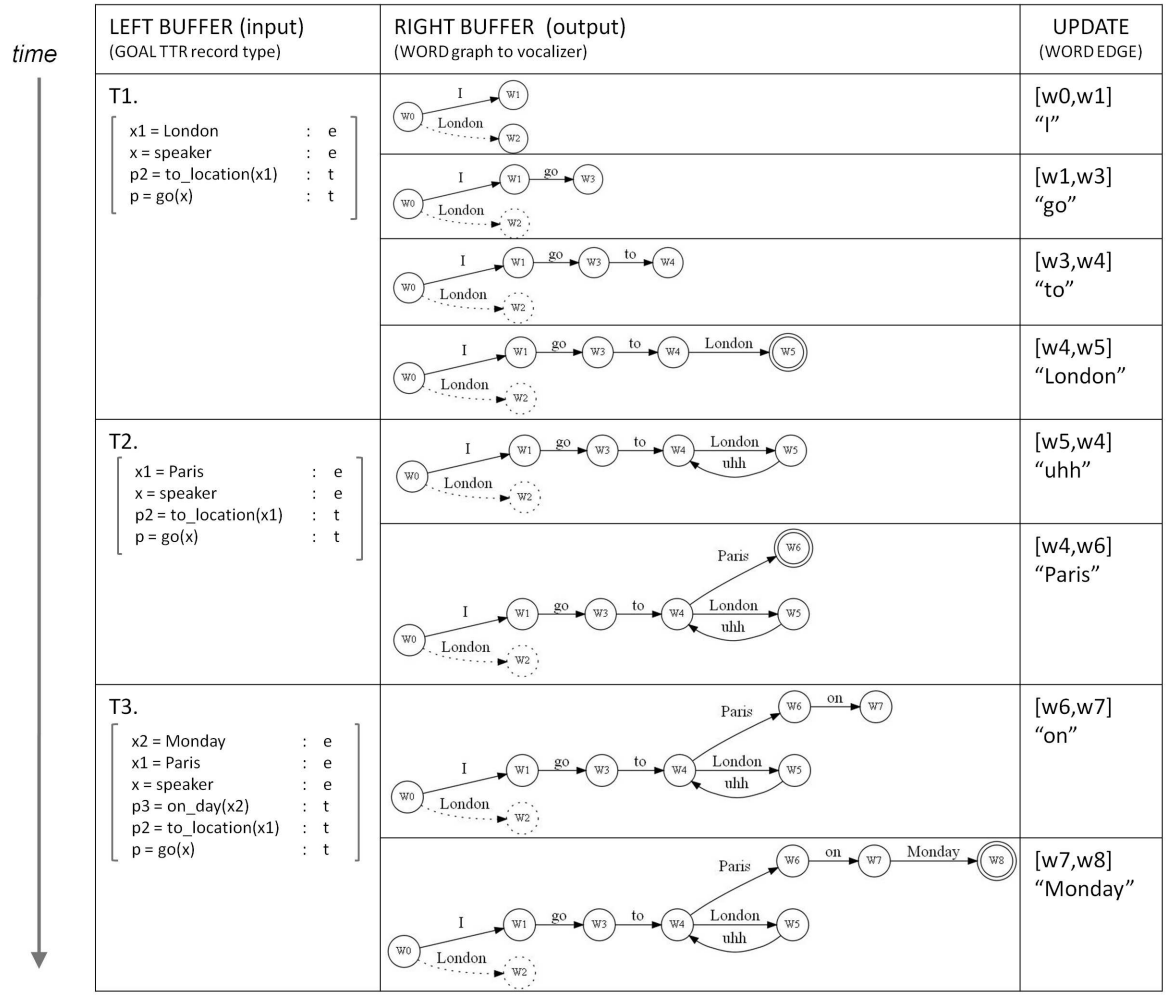


Figure 3: Incremental DS-TTR generation of a repair at time point T2 and extension at T3. Type-matched paths are double-circled nodes and uncommitted nodes and edges are dotted

after, or even during, the generation process, so trouble in generation may be encountered. Our `repair` function explained above operates if there is an empty state, or no possible DAG extension, after the semantic pruning stage of generation (resulting in no candidate succeeding word edge) by restarting the generation procedure from the last committed parse state edge. It continues backtracking by one vertex at a time in an attempt to extend the DS DAG until successful.

Our protocol is consistent with Shriberg and Stolcke (1998)'s empirical observation that the probability of retracing N words back in an utterance is more likely than retracing from N+1 words back, making the repair as local as possible. Utterances such as "I want to go, uhh, leave from Paris" are processed on a semantic level, as the repair is integrated with the semantics of the part of the utterance before the repair point to maximize re-use of existing semantic structure.

A subset of self-repairs, *extensions*, where the repair effects an "after-thought", usually in a transition place in a dialogue turn, are dealt with straight-forwardly in our system. The DS parser treats these as monotonic growth of the matrix tree through LINK adjunction (Kempson et al., 2001), resulting in subtype extension of the root TTR record type. Thus, a change in goal concept during generation will not always put demands on the system to backtrack, such as in the case of generating the fragment after the pause in "I go to Paris . . . from London". It is only a semantics/syntax mismatch, where the revised goal TTR record type does not correspond to a permissible extension of a DS tree in the DAG, where overt repair will occur (for a comparison see figure 3).

In contrast to Skantze and Hjalmarsson (2010)'s string-based *speech plan* comparison approach, there is no need to regenerate a fully-formed string from a revised goal concept and compare it with

the string generated thus far to characterize self-repair. Instead, repair here is driven by attempting to extend existing parse paths to construct the new target record type, backtracking through the parse state in an attempt to find suitable departure points for restarting generation, *retaining* the semantic representation already built up during the DS-TTR generation process.

5 Conclusion and Future Work

A prototype NLG module has been described that utilizes incremental semantic construction of TTR record types constrained by incremental Dynamic Syntax tree extension and TTR supertype relation checking to generate on a word-by-word basis. Although yet to undergo thorough evaluation, the system is capable of generating test-set self-repairs given manually induced goal TTR record type changes in the input buffer. The coming evaluation will not only involve a computational analysis, but an interactional one, involving human judges and experimental measures along the lines proposed by Schlangen (2009).

In terms of future development, the conceptual and phonological levels of the model could be expanded upon to get even finer granularity and consequently allow more natural system responses. A possible immediate extension could be the incorporation of a TTR subtyping process in the construction of goal concepts during generation by the dialogue manager, so as to incorporate incrementality into the conceptualization process (Guhe, 2007) as well as in surface realization.

Acknowledgments

Thanks go to my supervisor Matthew Purver and the RANLP reviewers for their helpful comments.

References

Patrick Blackburn and Wilfried Meyer-Viol. 1994. Linguistics, logic and finite trees. *Logic Journal of the Interest Group of Pure and Applied Logics*, 2(1):3–29.

Robin Cooper. 2005. Records and record types in semantic theory. *Journal of Logic and Computation*, 15(2):99–112.

Koenraad De Smedt. 1990. IPF: An incremental parallel formulator. In *Current research in natural language generation*, pages 167–192. Academic Press, London.

Markus Guhe. 2007. *Incremental Conceptualization for Language Production*. NJ: Lawrence Erlbaum Associates.

Gerard Kempen and Edward Hoenkamp. 1987. An incremental procedural grammar for sentence formulation. *Cognitive Science*, 11(2):201–258.

Ruth Kempson, Wilfried Meyer-Viol, and Dov Gabbay. 2001. *Dynamic Syntax: The Flow of Language Understanding*. Blackwell.

W.J.M. Levelt. 1989. *Speaking: From intention to articulation*. Mit Pr.

Matthew Purver and Ruth Kempson. 2004. Incremental context-based generation for dialogue. In A. Belz, R. Evans, and P. Piwek, editors, *Proceedings of the 3rd International Conference on Natural Language Generation (INLG04)*, number 3123 in Lecture Notes in Artifical Intelligence, pages 151–160, Brockenhurst, UK, July. Springer.

Matthew Purver, Arash Eshghi, and Julian Hough. 2011. Incremental semantic construction in a dialogue system. In J. Bos and S. Pulman, editors, *Proceedings of the 9th International Conference on Computational Semantics*, pages 365–369, Oxford, UK, January.

Yo Sato. 2011. Local ambiguity, search strategies and parsing in Dynamic Syntax. In E. Gregoromichelaki, R. Kempson, and C. Howes, editors, *The Dynamics of Lexical Interfaces*, pages 205–233. CSLI.

David Schlangen and Gabriel Skantze. 2011. A general, abstract model of incremental dialogue processing. *Dialogue and Discourse*, 2(1):83–111.

David Schlangen. 2009. What we can learn from dialogue systems that dont work. In *Proceedings of DiaHolmia (Semdial 2009*, pages 51–58.

Elizabeth Shriberg and Andreas Stolcke. 1998. How far do speakers back up in repairs? A quantitative model. In *Proceedings of the International Conference on Spoken Language Processing*, pages 2183–2186.

Gabriel Skantze and Anna Hjalmarsson. 2010. Towards incremental speech generation in dialogue systems. In *Proceedings of the SIGDIAL 2010 Conference*, pages 1–8, Tokyo, Japan, September. Association for Computational Linguistics.

Gabriel Skantze and David Schlangen. 2009. Incremental dialogue processing in a micro-domain. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 745–753, Athens, Greece, March. Association for Computational Linguistics.