Incremental Generation by Incremental Parsing

Masayuki Otsuka
Department of Philosophy
King's College London
Strand, London WC2R 2LS, UK
masayuki.otsuka@kcl.ac.uk

Matthew Purver
Department of Computer Science
King's College London
Strand, London WC2R 2LS, UK
matthew.purver@kcl.ac.uk

Abstract

The paper shows how an incremental generator can be constructed based on the incremental parsing framework described in Dynamic Syntax (DS)(Kempson et al., 2001), without adding a generator-specific vocabulary or intermediate levels of representation. The resulting generator is defined purely in terms of the parsing process, together with a notion of tree subsumption. This is shown to have various advantages including easy self-monitoring and psycholinguistic plausibility. A simple Prolog implementation is described, together with various possible improvements in efficiency.

1 Introduction

In this paper we give a description of a prototype generator based on the DS approach. DS defines a formalism that allows the articulation of natural-language grammars that reflect left-to-right processing: natural-language strings are paired with decorated semantic trees by a process of monotonic growth over sequences of partial trees associated with the processing of each word. As such it incorporates a prototype parser as its central subpart. The generation process described here performs the reverse operation, producing possible output strings from a defined semantic tree. The generation method is defined entirely in terms of parsing, providing a tight parsing-generation correspondence, and a simple reflection of natural language use in dialogue.

We also describe a Prolog implementation of a DS system which incorporates both parser and generator, and discuss various possible improvements in efficiency.

Firstly we give some background on the DS formalism in section 2. We then describe our approach to generation in section 3 and the implementation in section 4. We then explain the psycholinguistic implications in section 5, and draw conclusions and outline further work in section 6.

2 Dynamic Syntax

DS is a parsing-directed grammar formalism which claims that parsing is the central mechanism of language processing. DS defines parsing as a process of establishing mappings from an initial tree to complete trees using general *computational rules* (roughly corresponding to syntactic rules) and specific *lexical actions* projected by lexical items in the input string.

The tree structures used in DS represent semantic interpretations for a given string, and are described by a modal tree logic LOFT (Blackburn and Meyer-Viol, 1994). Trees can be partial in two senses: (a) node relations can be partially specified by underspecified modalities; and (b) node decorations can be underspecified by using meta-variables. Such partial specifications are introduced as imposed requirements, which jointly constitute a set of constraints on possible outcomes. Trees are complete iff they have no outstanding requirements, a constraint which is essential to wellformedness (see below), which means that all partiality and underspecification must be resolved.

Requirements, then, are central to the goal-directedness of processing since they are the driving force for updating information. For example, when a node relation is only partially specified, it bears the requirement that it has to be updated to a fully specified relation (as in "left-dislocated" expressions, for which an initial constituent node is introduced into the tree related to the root node using an underspecified modality), and this requirement is met by merging of the initially "unfixed" node with some local node which is in a fully specified relation to the root node. Similarly, if underspecified decorations are projected, they must be updated to "proper" values in the subsequent states (e.g. a pro-

noun projects a meta-variable which has to be substituted with a proper term).

Generally, the initial tree has only one node bearing the requirement to establish a formula of type t, represented as $?Ty(t)$. The input string is scanned from left to right, with each lexical item projecting a lexical action to update the trees. Computational rules are transition functions on trees, which are conditional in the sense that if the input tree satisfies the condition of some computational rule, it yields the output tree updating the input tree. The parse is successful when the parser produces at least one complete tree having the root node decorated with a formula of type t, after using computational rules and all the lexical actions from the input string. In this case, the input string is called grammatical and the complete trees represent interpretations for the string. In other words, in DS, grammaticality of string is defined in terms of parseability.

Technically, a set of parse paths is a partially ordered set of possible trees where the initial tree is the lowest bound, actions are projected by the lexical items in the input string and computational rules, and the goal trees are the highest bounds. The partial ordering $\leq_{ACT}$ indicates a monotonic extension relation on trees bearing a label of the name of the computational/lexical action performed in the transition.

The search for a parse path can be viewed as the search for a successful composition of actions which yields complete trees. The set of sequences of permissible actions is obtained by interweaving the linearly ordered set of lexical actions projectable by the input string into a finite number of partially ordered Kleene*-iterated computational actions, e.g. $ACT(\text{John sneezed}) = \langle C^* < john < C^* < sneezed < C* \rangle$.

DS specifies the theoretical parser, but, being a grammar formalism, it has no specification as to how a parser performs, i.e. no algorithm is defined giving the concrete parser. A parse state can be thought of as a pair of a string and a set of possible trees (semantic interpretations for the string). Without any parsing strategy, we may define the parser to generate all the possible trees using the computational rules (C-possible trees) for the input parser state, then scan the input string to apply the lexical action to all of the trees to update the parser state. This routine is repeated until the parser finishes scanning the input string.

3 Generation

As DS identifies grammaticality with parseability, rather than using concepts such as syntactic constituents or heads, standard approaches to generation such as head-driven methods (Shieber et al., 1990) cannot be applied. Instead, generation must be defined in terms of parsing.

3.1 Connecting Parsing and Generation

Despite informal observations made in psycholinguistics that production and comprehension systems have much in common (Garrett, 1982; Frazier, 1982), parsing and generation have generally been treated as quite separate research enterprises, both in the psycholinguistic and computational linguistic communities.

Nevertheless, following Shieber (1988), attempts have increasingly been made to define the two in terms of a broadly common architecture and using shared reversible grammars (see e.g. (Erbach, 1991; Neumann, 1994; Gardent and Thater, 2001)).

Although the contribution made by the present system is modest in only addressing the level of tactical generation, it nevertheless contributes to this co-articulation of parsing and generation by defining a generation system which purports to reflect the process of left-to-right incremental production, by using incremental parsing as the basic building block.

3.2 Generation as Parsing

Generation can be achieved using very little beyond the standard DS notion of parsability. We assume a fully specified goal tree as input[1]: given this, the generator incrementally produces a set of corresponding strings by following standard parsing routines and using the goal tree as a check. In other words, for a naive generator, all that is required is a notion of tree subsumption: this allows any candidate partial tree produced in the generation process to be checked against the goal tree to determine whether it is a sensible candidate (by checking for tree mergeability without inconsistency in node relations and decorations).

[1] In assuming a source tree, we are currently ignoring some issues that might be required in a dialogue system, such as concept generation, and translation from some flat meaning representation (e.g. a FOL formula) into a decorated source tree. We are therefore treating a generator as a module which supports that part of the generation process called sugaring (Ranta, 1994) or linguistic realisation (Reiter and Dale, 1997) – translation from an unambiguously structured object in a meaning representation language to a natural-language string.

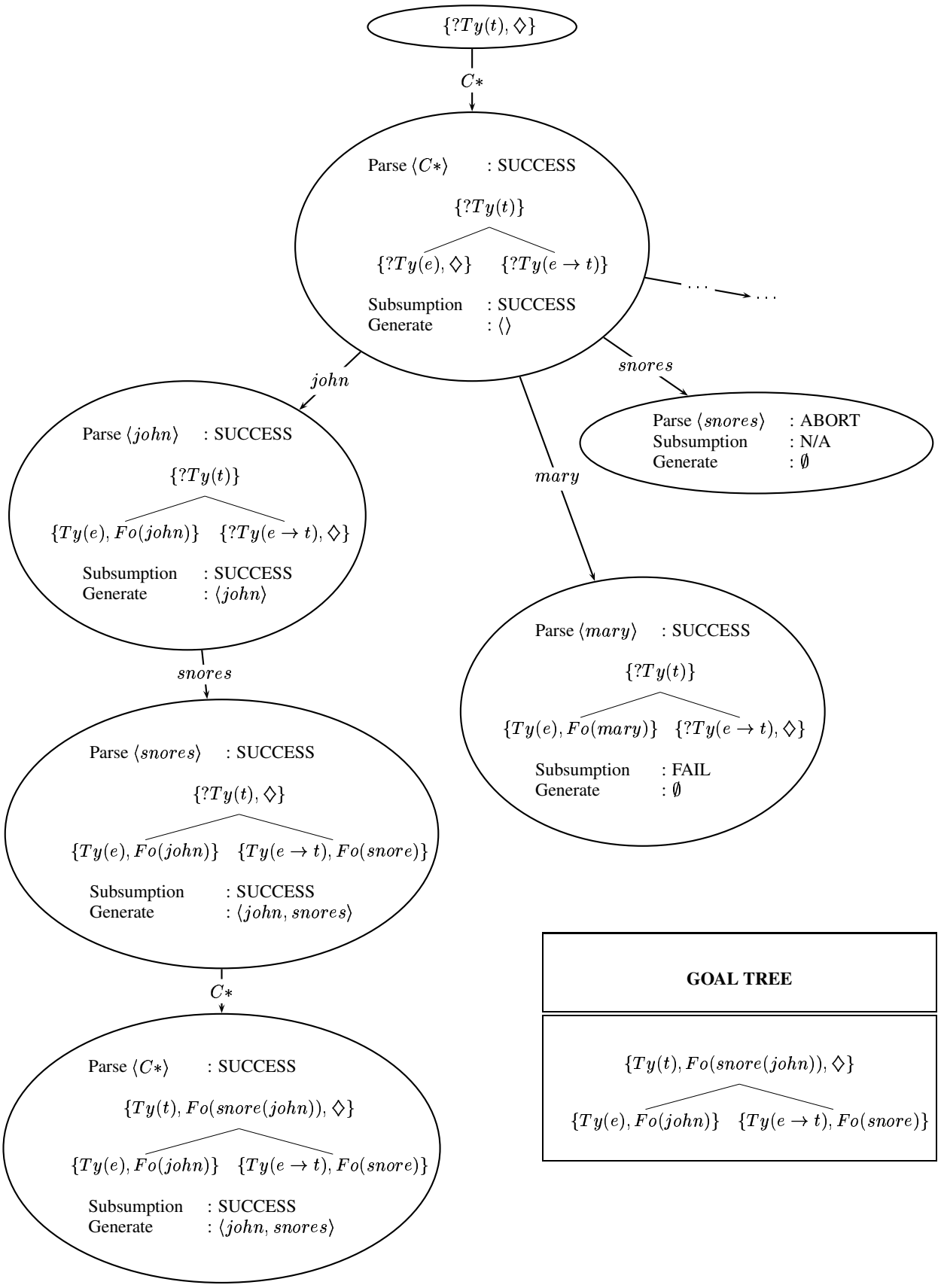

Figure 1: Generating *"john snores"*

At any point in the parsing process, a DS parser state can be seen as a pair of a partial string (the input so far) and a set of associated partial trees. With generation, we view the generator state as a set of these parser states (a set of pairs of 'possibly acceptable' partial strings and their associated sets of partial trees).

At each stage of generation, each pair is extended by tentatively extending the partial string by adding any word from the lexicon. The associated set of possible (partial) trees is produced using the standard parser – it may of course be empty if the word under consideration cannot be grammatically added to the partial string – and only those which are subsumed by the goal tree are kept. Any strings associated with empty tree sets are then rejected.

Generation is complete when the process can be continued no further (any string extension results in an empty parser state), and the set of output strings is taken as those for which the associated tree set contains a member identical to the goal tree – see figure 1.

Lexical selection is implicit: any word in the lexicon which is not associated with a node (or formula decorating a node) in the goal tree will produce empty tree sets (as the trees produced will not be subsumed by the goal tree), and therefore cannot produce acceptable extended partial strings.[2]

3.3 Self-Monitoring

Self-monitoring (the step-by-step monitoring of the generation process by an associated parse routine) has long been taken to be essential by psycholinguists (de Smedt and Kempen, 1990, for example). As discussed by Neumann and van Noord (1994), it is also useful in a computational dialogue system, as it allows generation to be controlled from the point of view of the expected hearer. As well as checking parsability, it can be used e.g. to prevent ambiguous utterances (if a self-monitoring process can spot that a string under consideration during generation is ambiguous, that string can be excluded on the basis that it will be less clear for the hearer than other unambiguous alternatives).

With standard generation approaches, self-monitoring must be carried out in parallel by a parsing module which communicates with the generation module in some suitable manner. With our approach, however, self-monitoring comes built-in, as parsing is the building block for generation: all information that would be produced by a separate parsing process is already available and there is no need for a separate module.

The basic requirement for parsability by a hearer is of course guaranteed, as strings are produced by incremental parsing. Further control can be added easily by including the desired check at the subsumption-checking stage: rejection of ambiguous strings can be added by checking for partial trees that are *not* subsumed by the goal tree, and rejecting any parser paths which produce such trees (rather than just removing such trees from the path).

3.4 Incrementality

The sense in which our generation process (and the DS parsing process) is *incremental* differs from that standardly used in both generation and psycholinguistic literature. In the generation literature (see e.g. (Erbach, 1991; Stone and Doran, 1997)), incrementality is taken to refer to the ability to produce substrings corresponding to (incomplete) sub-parts of the semantic representation, but without any requirement that this reflect left-to-right processing. In the psycholinguistic literature, "strong" incrementality (Sturt and Crocker, 1996; de Smedt and Kempen, 1990) is taken to require the projection of fixed tree relations as early and as close to word-by-word processing as possible[3] – and this progressive update enforces revision and backtracking.

Our use of the term is closer to the psycholinguistic concept in reflecting left-to-right parsing, with all relevant processing being complete after the addition of each word. However, it differs from the concept of strong incrementality, by allowing the articulation of partial trees, in particular trees in which not all relations are uniquely determined. This notion of incrementality is central to the DS approach: trees are extended monotonically as the string is consumed in parsing.

It is this monotonicity that allows us to select lexical items based just on a subsumption check, and allows selection to be economical compared to lexicalist approaches such as (Gardent and Thater, 2001).[4] In their framework, syntactic variation is

[2] This may not be the most efficient way of rejecting unsuitable words – see section 4.2.3 below.

[3] Sturt and Crocker reflect on how different aspects of language processing are more or less incremental, with syntactic processing involving word-by-word update of the syntactic tree, semantic processing, eg pronoun resolution, allowing some delay.

[4] We are not claiming the overall approach is more economical.

encoded in the lexicon, and lexical selection only checks if a subformula of a candidate subsumes the input semantics, allowing selection of lexical items of different syntactic structures for the same input semantics: for example, verbs might be given one lexical entry for canonical SVO order, and one for left-dislocation. This means that the number of generation search paths is multiplied by the number of distinct structures licensed by the lexical item. In contrast, in conforming to word-by-word incrementality, DS is able to project a single set of actions, relying on the update intrinsic to the tree growth process to determine the diversity of generation possibilities.

However, it is worth noting that our approach reflects the computational concept of incrementality as well. This concept is useful for dialogue system architecture, in that processing bottlenecks in the semantic production module can be worked around by passing any complete sub-parts to an incremental generation module as they become available. Our approach can also be considered incremental in this sense: sub-strings can be generated from suitable goal sub-trees with no change to the algorithm (the root node of the generation tree is merely given a node name which is not fully specified, and requirements which correspond to the root node of the goal sub-tree).[5]

4 Implementation

The formal definition of DS species constraints of possible forms and extensions of tree structure and node decorations in terms of axioms (in LOFT) and algebraic definitions, and update actions (input/output relations of grammar rules) are defined in a way that they respect the constraints. This leaves room for implementation as to how the parser performs. This section describes the parser algorithm and possible strategies to improve efficiency.

4.1 Overview

A prototype DS system has been implemented in Prolog[6] The number of lexical entry types is currently small (i.e. the "grammar" is only small), and limited to English, but some relatively complex constructions such as left-dislocation and relative clauses can be processed.

[5]Though this approach remains to be fully worked out, some first steps in this direction have been taken in considering head-final languages such as Japanese.

[6]The system can be accessed at `http://st228.dcs.kcl.ac.uk:8080/ds`.

The implementation remains close to the logical DS formalism. A tree node is represented as a pair of a node name and a set[7] of labels; a tree is represented as a pair of a set of nodes and a pointer (a node name). Labels can be requirements `?REQ`, directed requirements `?([DIR],REQ)` or features `+FEAT`. An example of a simple DS tree and its equivalent Prolog representation are shown in figure 2 and listing 1.

There are some representational differences: mother-daughter relations are not expressed directly as labels but are implicit in the node naming scheme (e.g. node 0 has daughters 00, 01); LINKs are represented as 2-daughters; and unfixed nodes are *-daughters.

Computational actions can now be defined in terms of list manipulation. Prolog backtracking is used to allow any number of actions to be applied to any parser state (where possible).

Lexical actions are defined similarly and given as templates for individual parts-of-speech and verb subcategorisation frames. The templates can then be interfaced to a standard computational lexicon (we are currently using one derived from the OALD (Hornby, 1974)).

Parsing is now easily defined declaratively: the initial parser state is one in which the only possible tree is a single root node with a $?Ty(t)$ requirement, and none of the input string has been consumed; a final parser state is one in which a complete (all requirements discharged) tree is available and the entire input string has been consumed; and possible intermediate states are produced from other states by any number of computational actions, plus lexical actions defined by the consumption of the next word from the input string.

4.2 Efficiency

4.2.1 Parsing

As described above, the current parser is highly unconstrained (it is defined declaratively and liberal use is made of Prolog backtracking). As generation is performed by parsing, any efficiency in parsing will be reflected in generation. Parsing efficiency can be improved by considering certain computational actions[8] as required whenever they can be applied (thus reducing the number of possible partial trees and the need for backtracking).

[7]We represent sets as Prolog lists: no use is made of list order.

[8]Currently *thinning*, *elimination*, and *star-adjunction*.

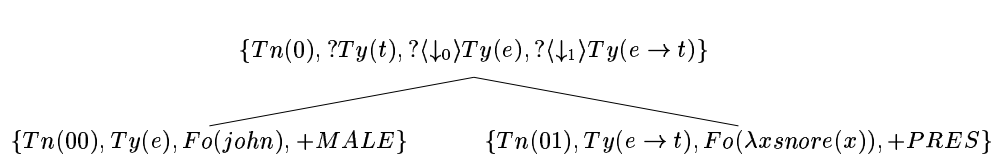

Figure 2: Example DS Tree

```
tree( [node( '0', [?ty(t), ?([\/0],ty(e)), ?([\/1],ty(e>t))] ),
       node( '00', [ty(e), fo(john), +male] ),
       node( '01', [ty(e>t), fo(X^snore(X)), +pres] )] ).
```

Listing 1: Example DS Tree (Prolog)

Further improvements in parsing may be possible by using e.g. probabilistic methods to constrain search, but this is outside the scope of the current work.

4.2.2 Generation

The naive generation process described in section 3 is not as inefficient as it might initially appear. Statically speaking, the basic concept is to generate all possible strings and check whether parsing them gives a sensible tree – therefore one might expect N^W possible paths given a lexicon of N words and typical string length W. However, the incremental nature means that this is not the case: unsuitable search paths are eliminated on a stage-by-stage basis (by the parser for ungrammatical paths, and by the subsumption check for paths incompatible with the goal tree).[9] The implementation of this naive version generates simple sentences in a few seconds with a small test lexicon.

However, we briefly explore efficiency improvements by lexical selection and search method.

4.2.3 Lexical Selection

Lexical selection uses the decorations of the input goal tree to produce a limited set of lexical items which can be used by the generator, consisting of a set L of words which correspond to logical formula decorations (e.g. N, V, Adj, Det) and a set F of functional words (e.g. relativiser, complementiser). This reduces the search space considerably, with the number of paths at most $(L + F)^W$ and in fact significantly less.

Members of L are chosen on the basis of goal tree node search followed by lexical lookup: only those words corresponding to $Fo(X)$ semantic formula decorations are admitted. For pronouns, node features can also be used (to admit only relevant gender & number), and similarly for verbs to control tense. This modification has been implemented and results in significant speed increase (although no formal evaluation has yet been performed, even more complex sentences including relative clauses can now be generated in a few seconds with a large 40,000 word lexicon to search through).

Heuristics could be used to govern lexical lookup, based on e.g. recency of words in the current dialogue context (ensuring more recently used words are used in preference), and prior probabilities (ensuring more common words are used in preference to rarer ones).

4.2.4 Search Method

The incremental nature of the generation process allows us to apply various search methods, and stop the process once any suitable string has been found (rather than continuing until all search paths are terminated). Using a depth-first approach with this method will therefore cause only one string to be produced; whereas using a breadth-first approach will cause only the set of all strings of the shortest possible length to be produced. Both approaches have been successfully implemented.

This provides scope for probabilistic (or other heuristic) methods to increase efficiency in the future by constraining the search using techniques well known in areas such as speech recognition (e.g. beam search). However, a prerequisite for this will be a theory of heuristically constrained DS parsing.

5 Psycholinguistic Observations

A further advantage of the DS system is that it promises to meet the challenge set out by Garrod

[9] In fact, in the best case there will be $N \times W$ paths, although this only applies for an unambiguous lexicon and grammar that gives a strict one-to-one correspondence between strings and trees.

and Pickering (2001) that linguistic systems be evaluated by their success in reflecting phenomena characteristic of dialogue. This includes alignment between dialogue participants at many levels (lexical, syntactic and semantic), and the ability of participants to collaborate on and complete each other's sentences.

We believe that the view of generation as incremental parsing, as well as conforming to the general principle that production and comprehension are tightly coupled, allows us to explain these phenomena (where standard models of generation cannot).

5.1 Alignment

Standard generation models have little trouble explaining how lexical choice can be co-ordinated (word preference can be adjusted based on recency) – but syntactic mirroring as observed by e.g. Branigan et al. (2000) is more problematic (grammar rule preferences must be adjusted, which is non-trivial).

In DS, alignment of syntactic and semantic structure comes by definition, given the reduction of syntax to the progressive projection of semantically transparent structure. Lexical alignment can be explained as a preference for repeated recent lexical items, bypassing the general lexicon search. Apparent alignment of syntactic structure over and above semantic alignment, as in repetition of double object constructions as opposed to a switch to the prepositional phrase equivalent, can be explained by exactly the same mechanism: since syntactic and semantic preferences are encoded in the lexical actions associated directly with words (rather than in any general grammar rules), so adjustment of syntactic and semantic constructions reduces directly to lexical choice.

5.2 Collaboration / Completion

The phenomenon of shared utterances, in which participants are able to switch roles and complete one another's utterances at any point in a sentence, is problematic for standard models of generation, as Garrod and Pickering (2001) observe.[10]

The treatment of generation as *incremental* parsing allows a simple explanation. Speaker and hearer are building the same partial trees in parallel, using the same parsing process and directly corresponding (if not perhaps identical) lexical entries: the only significant difference is that the speaker has knowledge of the goal tree that is being generated. If the hearer can guess or deduce the goal tree,[11] (s)he is in just as good a position to complete the utterance as the original speaker.

[10] In particular, any shift prior to the sentence head is problematic for head-driven and LTAG approaches.

6 Conclusions and Further Work

6.1 Conclusions

The paper sets out a formal definition of a generation process for DS, in terms of the parsing process, and describes a naive computational implementation together with some possible improvements. In closing, we note that this (a) parsing-oriented, and (b) incremental view of generation promises to provide a basis for an explanation of certain psycholinguistic observations about dialogue such as co-ordination and collaboration between speakers.

6.2 Further Work

The current system is a prototype which we intend to extend in two main directions. Firstly, efficiency can be improved by employing computational tactics such as probabilistic/heuristic parsing and generation, and possibly more goal-directed methods. Secondly, we intend to incorporate the parsing/generation model into a full dialogue system, to test out in detail the extent to which the apparent potential for modelling dialogue phenomena such as alignment, collaboration etc. can actually be fulfilled. This will require a detailed model of semantic tree creation and manipulation.

[11] We have nothing to say here about how this deduction is performed: of course it will depend on world and domain knowledge, the participants' shared situation and experience, and so on.

References

Patrick Blackburn and Wilfried Meyer-Viol. 1994. Linguistics, logic and finite trees. *Bulletin of the IGPL*, 2:3–31.

Holly Branigan, Martin Pickering, and Alexandra Cleland. 2000. Syntactic co-ordination in dialogue. *Cognition*, 75:13–25.

Koenrad de Smedt and Gerard Kempen. 1990. Incremental sentence production, self-correction and coordination. In G. Kempen, editor, *Natural Language Generation*, pages 365–376. Martinus Nijhoff, Dordrecht.

Gregor Erbach. 1991. A bottom-up algorithm for parsing and generation. CLAUS report, Universität des Saarlandes, Saarbrücken, February.

Lyn Frazier. 1982. Shared components of production and perception. In M. Arbib et al., editor, *Neural Models of Language Processes*, chapter 11, pages 225–236. Academic Press, New York.

Claire Gardent and Stefan Thater. 2001. Generating with a grammar based on tree descriptions: a constraint-based approach. In *Proceedings of the 39th Annual Meeting of the ACL*. Association for Computational Linguistics.

Merrill Garrett. 1982. Remarks on the relation between language production and language comprehension systems. In M. Arbib et al., editor, *Neural Models of Language Processes*, chapter 10, pages 209–224. Academic Press, New York.

Simon Garrod and Martin Pickering. 2001. Toward a mechanistic psychology of dialogue: The interactive alignment model. In P. Kühnlein, H. Rieser, and H. Zeevat, editors, *Proceedings of the Fifth Workshop on Formal Semantics and Pragmatics of Dialogue*. BI-DIALOG.

Albert S. Hornby. 1974. *Oxford Advanced Learner's Dictionary of Current English*. Oxford University Press, third edition. With the assistance of Anthony P. Cowie and J. Windsor Lewis.

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

Günter Neumann and Gertjan van Noord. 1994. Reversibility and self-monitoring in natural language generation. In T. Strzalkowski, editor, *Reversible Grammars in Natural Language Processing*. Kluwer Academic Publishers.

Günter Neumann. 1994. *A Uniform Computational Model for Natural Language Parsing and Generation*. Ph.D. thesis, Universität des Saarlandes, Saarbrücken.

Aarne Ranta. 1994. *Type-Theoretical Grammar*. Oxford University Press.

Ehud Reiter and Robert Dale. 1997. Building applied natural language generation systems. In K. van Deemter and M. Stone, editors, *Formal Issues in Natural Language Generation*. CORSO C05.

Stuart Shieber, Gertjan van Noord, Robert Moore, and Fernando Pereira. 1990. A semantic head-driven generation algorithm for unification-based formalisms. *Computational Linguistics*, 16(1).

Stuart Shieber. 1988. A uniform architecture for parsing and generation. In *Proceedings of the 12th International Conference on Computational Linguistics*, pages 614–619. COLING.

Matthew Stone and Christine Doran. 1997. Sentence planning as description using tree-adjoining grammar. In P. Cohen and W. Wahlster, editors, *Proceedings of the 35th Annual Meeting of the ACL*, pages 198–205. Association for Computational Linguistics.

Patrick Sturt and Matthew Crocker. 1996. Monotonic syntactic processing: a cross-linguistic study of attachment and reanalysis. *Language and Cognitive Processes*, 11:448–494.