Functor-Driven Natural Language Generation with Categorial-Unification Grammars

Dale Gerdemann
Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign
405 N. Mathews
Urbana, IL 61801
USA

Erhard W. Hinrichs
Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign
405 N. Mathews
Urbana, IL 61801
USA

1. Introduction

In this paper we develop a functor-driven approach to natural language generation which pairs logical forms, expressed in first-order predicate logic, with syntactically well-formed English sentences. Grammatical knowledge is expressed in the framework of *categorial unification-grammars* developed by Karttunen (1986), Wittenburg (1986), Uszkoreit (1986), and Zeevat et. al. (1987). The semantic component of the grammar makes crucial use of the principle of *minimal type assignment* whose importance has been independently motivated in recent work in natural language semantics (see Partee and Rooth 1983). The principle of *type-raising as necessary* which follows from minimal type assignment has been implemented using Wittenburg's (1987,1989) idea of supercombinators. This use of supercombinators to achieve semantic compatibility of types generalizes Wittenburg's strictly syntactic use of such combinators.

The use of categorial unification grammars makes it possible to develop an efficient top-down control regime for natural language generation. Rather than generating the syntactic output string in a left-to-right fashion, our algorithm always generates that part of the output string first that belongs to the functor category in a given phrase, before it generates any of the arguments of the functor category. This functor-driven strategy is similar to the head-driven approach to natural language generation developed by Shieber et. al. (1989). However, unlike the head-driven approach, which uses a mixed regime of top-down and bottom-up processing, our algorithm always has sufficient top-down information to guide the generation process. Moreover, due to the principle of minimal type assignment in the semantics, our approach avoids problems of efficiency that arise for the head-driven approach for those classes of grammars that do not satisfy this principle. The work reported here is implemented in the natural language system UNICORN, which can be used for natural language parsing (see Gerdemann and Hinrichs 1989) and natural language generation.

2. The Grammar Formalism: Categorial Unification Grammar

The grammatical formalism that we adopt for categorial unification grammar is similar to that proposed in Uszkoreit (1986). Following the schema for syntactic rules developed for PATR-style grammars, we formulate the categorial grammar rule of functional application by the rule schema in fig. 1. The

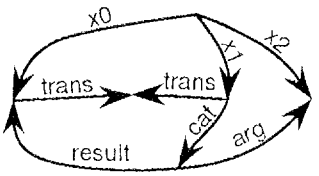


Figure 1: Function Application

$x1$ node (i.e. the node at the end of the path $\langle x1 \rangle$) represents a functor category that combines with an argument at $x2$ to yield as a result the category at $x0$. The rule also specifies that the semantic translation (*trans*) of the result category $x0$ is inherited from the functor $x1$. As is characteristic of categorial grammars, our syntactic rules are highly schematic, with most of the grammatical information encoded in the categorial lexicon. For example, constraints on word order are encoded in lexical representations of functor categories, rather than in the syntactic rules themselves. To this end we adopt an attribute *phon* (for: *phonology*) which is used to encode linear order for syntactic strings. The values for *phon* are structured as difference lists. The use of this data structure, inherited from PROLOG, allows us to concatenate functor categories with their arguments either to the left or to the right. It also allows us to state syntactic rules without having to make reference to constituent order.[1] The graphs in fig. 2 display partial lexical entries for the intransitive verb *smiles*,

[1] In this respect, our representation is more compact than other categorial-unification grammar formalisms which state order constraints in the categorial lexicon **and** in each syntactic rule. In particular, we don't need to distinguish between forward application and backward application

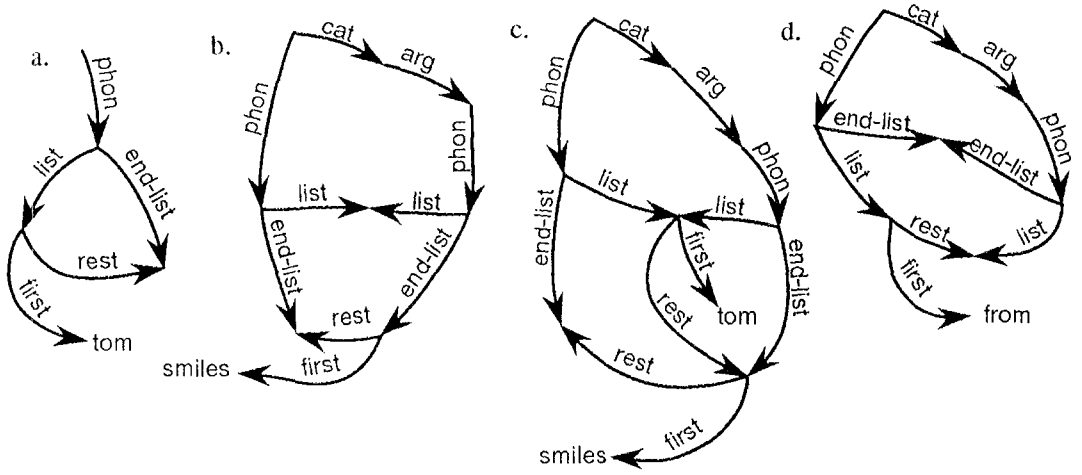


Figure 2: Phonology Rules

for the proper name *Tom* and for the sentence *Tom smiles*. The *phon* attribute for argument categories such as proper names is encoded as a singleton list which contains the argument string in question, e.g. *Tom*. The *phon* attribute for functor categories is designed to combine the string for the functor category with the *phon* feature structure of its argument categories. In the case of the intransitive verb *smiles*, the morpheme *smiles* appears as the first element in a list that is appended to the difference list for its subject argument. When the phonology attributes for *Tom* and *smiles* are combined by function application, the resulting sentence exhibits the correct word order, as fig. 2c shows. For the sake of completeness, we also include the representation of the preposition *from* as an example of a forward functor in fig. 2d.

For the remainder of this paper we will concentrate on the interplay between syntax and semantics for the purposes of language generation. We will assume that information about word order propagates from the lexicon in the manner we just outlined by example.

3. Natural Language Generation with Categorial-Unification Grammars

In this section we describe our functor-driven approach to natural language generation which pairs logical forms (represented in first-order predicate logic) with syntactically well-formed expressions of English. For example, given a first-order formula such as

(1) $\forall x[\text{person}'(x) \rightarrow \text{smile}'(x)]$

we want to generate a sentence such as *Everyone smiles*.

In order to produce the appropriate sentence, the generator is supplied with a start Dag as in fig. 3.

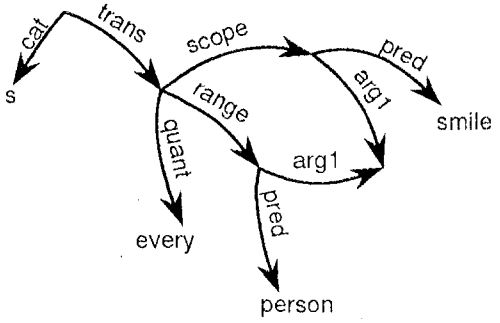


Figure 3: Start Dag for *Everyone smiles*

The first order formula (1) is represented in fig. 3 under the attribute *trans* (for: logical form translation). The value for the attribute *cat* specifies that the translation corresponds to a syntactic expression of category *s* (for: sentence). Unlike functional categories which take other syntactic categories as arguments, *s* is a basic category, i.e. a category which does not take an argument.

The task of the generator is to further instantiate start Dags such as that in fig. 3 so that appropriate syntactic expressions are generated in the most efficient manner possible.

3.1 A Functor-Driven Generation Algorithm

One advantage of the use of categorial grammars is that efficient generation can be effected by a completely general principle: at each step in the derivation of a syntactic expression, constituents that correspond to functor categories are to be generated before the generation of constituents that correspond to the functor's argument categories. The strategy underlying this principle is that in any grammatical construction, functor categories always provide more syntactic and semantic information than any of the argument categories. By generating the functor cat-

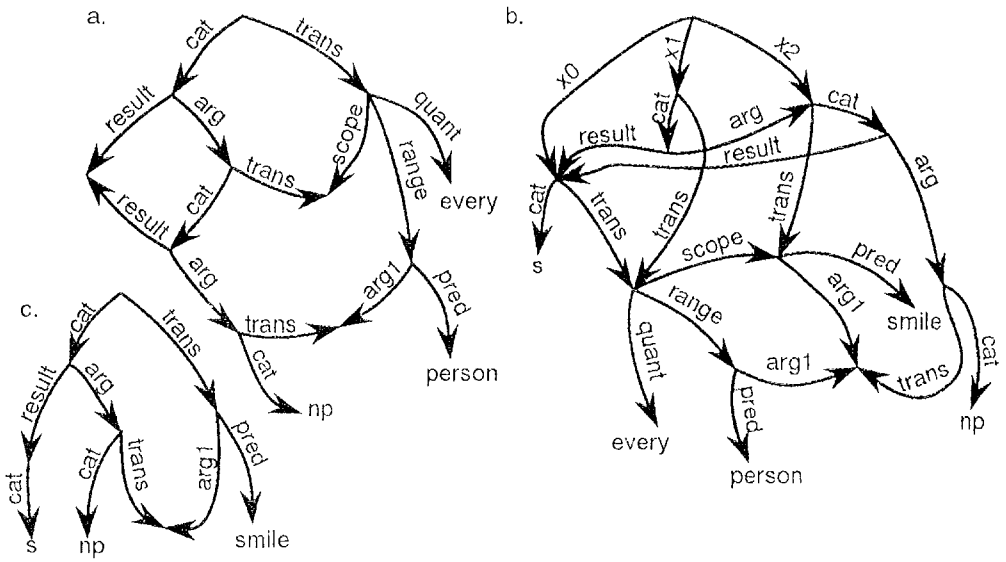


Figure 5: Generating *Everyone smiles*

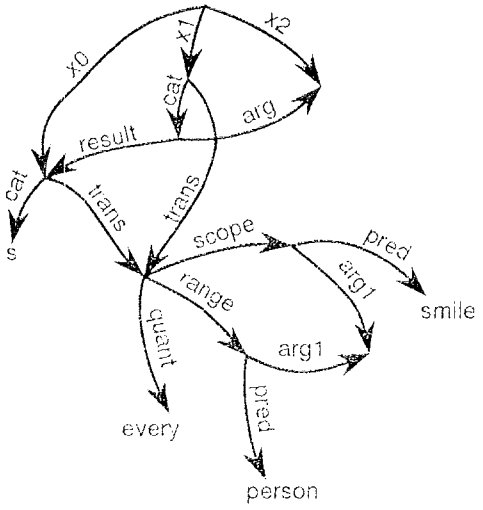


Figure 4: Start Dag unifies with function application rule

egory first, the choice of argument categories will be severely constrained, which significantly prunes the search space in which the algorithm has to operate.

We will illustrate our approach by discussing the functor-driven order of processing for the generation of the sentence *Everyone smiles*. First the generator will make a top-down prediction by unifying the start Dag in fig. 3 with the $x0$ node of the functional application rule shown in fig. 1. The resulting Dag is shown in fig. 4.

The predicted Dag in fig. 4 then becomes subject to the principle of generating functor categories first. Identification of a functor category in a rule of categorial-unification grammar is straightforward: the functor category is represented by the subdag whose value for the attribute *cat* is a Dag with attributes *arg* and *result* and whose *result* arc is reentrant with the value of the subdag rooted in $x0$. Thus, in the case of fig. 4, the functor category is $x1$.[2] At this point there is enough information on the $x1$ node to uniquely determine the choice of a functor category, whereas the choice of an argument category would be completely unconstrained. When the lexical entry for *everyone* (fig. 5a) unifies with the $x1$ node, the result is the Dag in fig. 5b.[3] Then, at this point, the $x2$ node is fully enough instantiated to uniquely determine the choice of *smiles* (fig. 5c) from the lexicon.

3.2 Non-minimally Type Raised Functors

Now consider what happens when non-quantified NPs like *Tom* are type-raised as in Montague (1974). That is, suppose that the lexical entry for *Tom* is the Dag in fig. 6a rather than the lower type in fig. 6b. It turns out that if the type raised NP is used, it will not be possible to constrain the choice of functor in generation. For example, fig. 7a shows the rule of function application (fig. 1) in which the $x0$ node has been unified with a start Dag appropriate to generate *Tom smiles*. In fig. 7b, the $x1$ node has unified with a type-raised entry for *Harry*, showing that the start Dag has done nothing to constrain the choice of functor. Thus, apart from introducing spurious ambiguity into the grammar (see Wittenburg 1987 for detailed discussion), the operation of type-raising, when used unconstrained, can also lead to considerable inefficiency in generation. In order

[2]Alternatively, one could could simply take $x1$ to always be the functor since, given our use of the *phon* attribute, the order of $x1$ and $x2$ no longer corresponds to linear order.

[3]A problem that arises here is that the $x1$ node in fig. 4 will also unify with the lexical entry for *smiles* (fig. 5c) giving a nonsensical translation. Clearly, what needs to be done is to modify the semantic representations so that quantified expressions will not unify with non-quantified expressions. One line that could be investigated would be to have a type system which distinguishes quantified and non-quantified signs as in Pollard and Sag (1987).

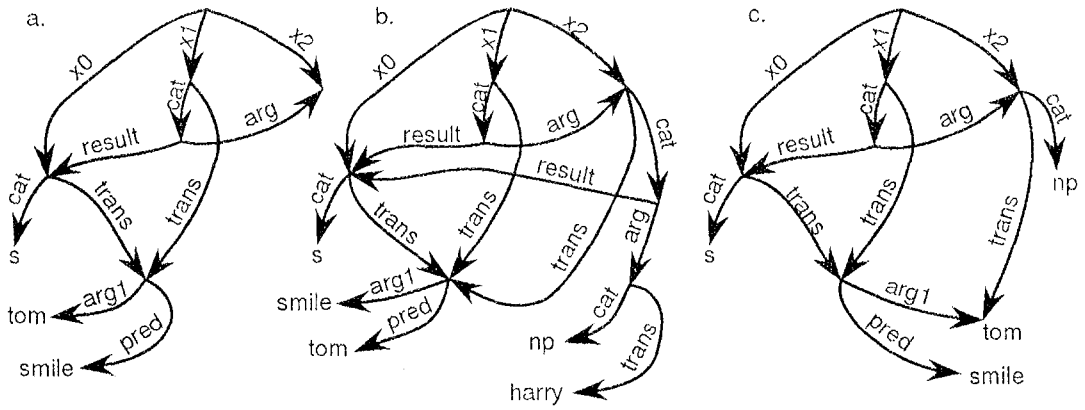


Figure 7: Generating *Tom smiles*

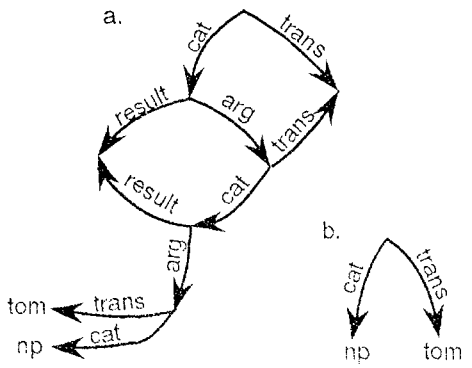


Figure 6: Lexical entries for *Tom*

to constrain the use of type-raising, we adopt the principle of minimal type assignment suggested on independent grounds by Partee and Rooth (1983). Partee and Rooth argued for the principle of minimal type assignment to account for scopal properties of NPs in a variety of coordinate structures. Among the examples they discuss is the contrast between sentences such as (2) and (3).

(2) Every student failed or got a D.

(3) Every student failed or every student got a D.

(2) and (3) have different truth conditions. (2) is true if some students failed and did not get a D, while others got a D and did not fail. (3), however, would be false in that situation. Partee and Rooth point out that appropriate truth conditions for (2) can only obtained if intransitive verbs are given a non-type-raised interpretation and if their conjunction is represented by the λ-abstract in (4). When (4) is combined with the translation for *every student*, the desired reduced formula in (5) is obtained.

(4) $\lambda x[\text{fail'}(x) \vee \text{got\_a\_D'}(x)]$

(5) $\forall x[\text{student'}(x) \rightarrow [\text{failed'}(x) \vee \text{got\_a\_D'}(x)]]$

The use of conjoined type-raised predicates as in (6), however, would incorrectly yield the formula in (7), which is appropriate for (3) but not for (2).

(6) $\lambda\wp.\wp(\lambda x.\text{fail'}(x)) \vee \wp(\lambda x.\text{got\_a\_D'}(x))$

(7) $\forall x[\text{student'}(x) \rightarrow \text{failed'}(x)] \vee$
$\forall x[\text{student'}(x) \rightarrow \text{got\_a\_D'}(x)]$

On the other hand, Partee and Rooth point out that for the interpretation of sentences such as (8), intransitive verbs do have to be type-raised, since (9) is a paraphrase of (8).

(8) A tropical storm was expected to form off the coast of Florida and did form there within a few days of the forecast.

(9) A tropical storm was expected to form off the coast of Florida and A tropical storm did form there within a few days of the forecast.

In order to reconcile this conflict, Partee and Rooth propose that extensional intransitive verbs such as *formed* should be assigned to the lowest possible type and be type-raised only when they are conjoined with an intensional verb such as *be expected*.

Given the principle of minimal type assignment, the entry for *smiles* fig. 5c will now be the main functor in generating the sentence *Tom smiles*. It can be seen that *smiles* (and no other non-type-raised category) will unify with the $x1$ node of fig. 7a. The resulting prediction is shown in fig. 7c. At this point the $x2$ node is constrained to unify with the minimal, non-type-raised entry for *Tom* (fig. 6b). Thus, the principle of minimal type assignment turns out to be crucial for constructing efficient generation algorithms for categorial-unification grammars.

3.3 Allowing Type-Raising as Needed

As seen in the previous section, efficient generation requires the use of basic (non-type-raised) NPs, whenever possible. However, this is not to suggest

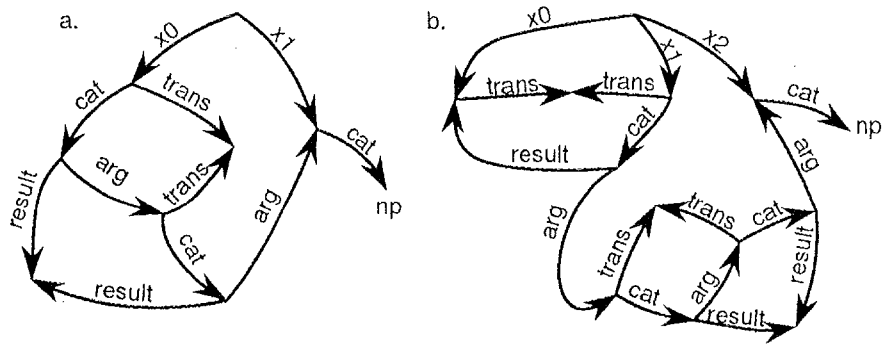


Figure 8: Type-Raising

that the operation of type-raising can be eliminated from the grammar altogether. For example, type-raising needs to apply in the case of conjoined NP's such as *Tom and every boy*. If we assume, as in Wittenburg (1986), that *and* is assigned the category in (10),[4] then to parse or generate a conjoined NP like *Tom and every boy* the category for *Tom* will have to be raised so that its type will match that of *every boy*.

(10) (X|X)|X

What is needed then is some operation that will convert the non-type-raised entry for *Tom* in fig. 6b to its raised counterpart in fig. 6a. One way of incorporating the necessary operation into the grammar would be via the type-raising rule in fig. 8a, in which the non-type-raised entry unifies with the $x1$ node to yield the type-raised result at $x0$[5] However, the problem with the rule in fig. 8a is that it will allow type-raising not just as needed but also anywhere else. So the problem of spurious predictions like that in fig. 7b reemerges.

Clearly, what is needed is some way of allowing type-raising only in those cases where it is needed. Partee and Rooth suggest that type raising should be constrained by some kind of processing strategy,[6] without indicating how such a processing strategy can be implemented. It turns out that the processing strategy that Partee and Rooth suggest can be stated declaratively as part of the grammar, if the operation of type-raising is incorporated into a *supercombinator* (in the sense of Wittenburg 1987,89) that combines type-raising and functional application into a single operation.

Wittenburg himself was interested in constraining type-raising in order to eliminate the spurious ambiguity problem of combinatory categorial grammars. He noted that in some of Steedman's (1985,1988) grammars type-raising was needed just in those cases where an NP needed to compose with an adjacent functor. He, therefore, proposed that the type-raising rule be included into the function composition rule. The use of type-raising in coordinate structures that we have considered in this paper, is quite similar: We want type-raising to be licensed, just in case an NP is adjacent to a functor that is looking for a type-raised argument. We, therefore, incorporate type-raising into the function application rule as seen in fig. 8b. Now, the old type-raising rule in fig. 8a is no longer needed, and spurious type-raising will no longer be a problem.

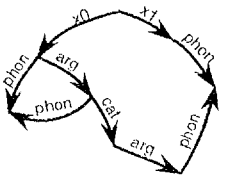
The type-raising supercombinator schema in fig. 8b is, for example, used in the generation of coordinate structures such as *Tom and every boy*. Space will not allow us to fully present an analysis of such an NP here, but the important point is that a non-type-raised lexical entry such as that in fig. 6b will be able to unify with the $x2$ node, and when it does so, the subdag at the end of the path $\langle x1\ cat\ arg \rangle$ will become identical to the type-raised entry for *Tom* in fig. 6a.

4. Conclusion

In this paper we have argued that a functor-driven generation algorithm for categorial unification grammars leads to efficient natural language generation, if the algorithm incorporates Partee and Rooth's (1983) principle of minimal type assignment. In order to have minimal type assignment and still allow type-raising in restricted contexts, we have adopted Wittenburg's (1986) idea of supercombina-

[4] We use a non-directional calculus here, since word order is encoded into lexical items. The domain is to the right of the bar and the range is to the left. The capital *Xs* represent a variable over categories. This is just a schematic representation of a considerably more complicated category.

[5] Note again that, since phonology is encoded into lexical items, we can get by with a single rule of type-raising whereas most formalisms would require two. The phonological counterpart of type-raising would be:



[6] Partee and Rooth were actually more interested in psycholinguistic processing strategies. Still their ideas carry over straightforwardly to computational linguistics.

tors. Type-raising has been incorporated into the function application rule so that type-raising can only apply when some functor is looking for a type-raised argument. This use of supercombinators to achieve semantic compatibility generalizes Wittenburg's strictly syntactic application of these combinators.

References

Gerdemann, D. and Hinrichs, E. 1988. UNICORN: a unification parser for attribute-value grammars. *Studies in the Linguistic Sciences*, 18(2):41–86.

Karttunen, L. 1986. D-patr: a development environment for unification-based grammars. In *COLING-86*.

Montague, R. 1974. The Proper treatment of quantification in ordinary English. In R. Thomason (Ed.), *Formal Philosophy: Selected Papers of Richard Montague*, Yale University Press, New Haven.

Partee, B. and Rooth, M. 1983. Generalized conjunction and type ambiguity. In R. Bauerle, C. Schwarze, and A. von Stechow (Eds.), *Meaning, Use and Interpretation of Language*, 361–383, Walter de Gruyter.

Pollard, C. and Sag, I. 1987. *An Information-Based Approach to Syntax and Semantics: Volume 1 Fundamentals. CSLI Lecture Notes No. 13*, Chicago University Press, Chicago.

Shieber, S. 1988. A uniform architecture for parsing and generation. In *COLING-88*, 614–9.

Shieber, S., van Noord, G., Moore, R. C., and Pereira, F. C. N. 1989. A semantic-head-driven generation algorithm for unification-based formalisms. In *ACL Proceedings, 27th Annual Meeting*, 7–17.

Steedman, M. 1985. Dependency and coordination in the grammar of dutch and english. *Language*, 61:523–568.

Steedman, M. 1988. Combinators and grammar. In R. Oehrle, E. Bach, and D. Wheeler (Eds.), *Categorial Grammar and Natural Language Structures*, 417–442, Reidel, Dordrecht.

Uszkoreit, H. 1986. Categorial unification grammar. In *COLING-86*.

Wall, R. and Wittenburg, K. 1989. Predictive normal forms for composition in categorial grammars. In *Proceedings of International Workshop on Parsing Technologies*, 152–161.

Wittenburg, K. 1986. *Natural Language Parsing with Combinatory Categorial Grammar in a Graph-Unification-Based Formalism.* PhD thesis, The University of Texas at Austin.

Wittenburg, K. 1987. Predictive combinators: a method for efficient parsing of combinatory categorial grammars. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, 73–80.

Zeevat, H, Klein, E, and Calder, J. 1987. Unification categorial grammar. In N. Haddock, E. Klein, and G. Morrill (Eds.), *Edinburgh Working Papers in Cognitive Science*, 195–222, Centre for Cognitive Science, University of Edinburgh.