Systemic Classification and its Efficiency

Chris Brew*
Department of Experimental Psychology
and
Human Communication Research
Centre

This paper examines the problem of classifying linguistic objects on the basis of information encoded in the system network formalism developed by Halliday. It is shown that this problem is NP-hard, and a restriction to the formalism, which renders the classification problem soluble in polynomial time, is suggested. An algorithm for the unrestricted classification problem, which separates a potentially expensive second stage from a more tractable first stage, is then presented.

1. Introduction

In this paper we describe algorithms that enable the system networks of Systemic Grammar (Halliday 1976, 1975) to be exploited in applications, such as natural language understanding, that require incremental description refinement. We are attracted to the system network formalism on three main counts:

Potential Reversibility. System networks fit well with a model of linguistic behavior as goal-directed action, since they encode grammatical information in the form of sets of interconnected choices that a language user needs to make in order to produce apposite and communicative utterances. It seems natural to exploit this information for comprehension as well as generation.

Computational Convenience. System networks look simple, yet are expressive enough to allow the development of substantial grammars. If this apparent simplicity is reflected in the mathematical properties of the system network formalism then it may be possible to design cheap description refinement algorithms that use the networks.

Wider Applications. If the networks turn out to be easy to process, they may have wider applications in tasks requiring representation of nonlinguistic knowledge. We think that the tasks for which such networks are most likely to be appropriate are those for which conventional taxonomic representations are nearly, but not quite, sufficient. Our algorithms (presented in the final part of this paper) operate by converting as much as possible of the information contained in the networks to taxonomic form.

* University of Sussex, U.K., and 2 Buccleuch Place, Edinburgh, U.K., currently at Sharp Laboratories of Europe Ltd., Oxford, U.K. Supported by a studentship provided by the Science and Engineering Research Council's Information Technology Initiative. I am grateful to several colleagues for constructive discussions of this work, and particularly to Stephen Isard and Robert Dale, who were kind enough to provide detailed commentary on earlier drafts. I am also indebted to two anonymous referees for suggestions that substantially improved the paper. None of those mentioned are responsible for any remaining errors or infelicities.

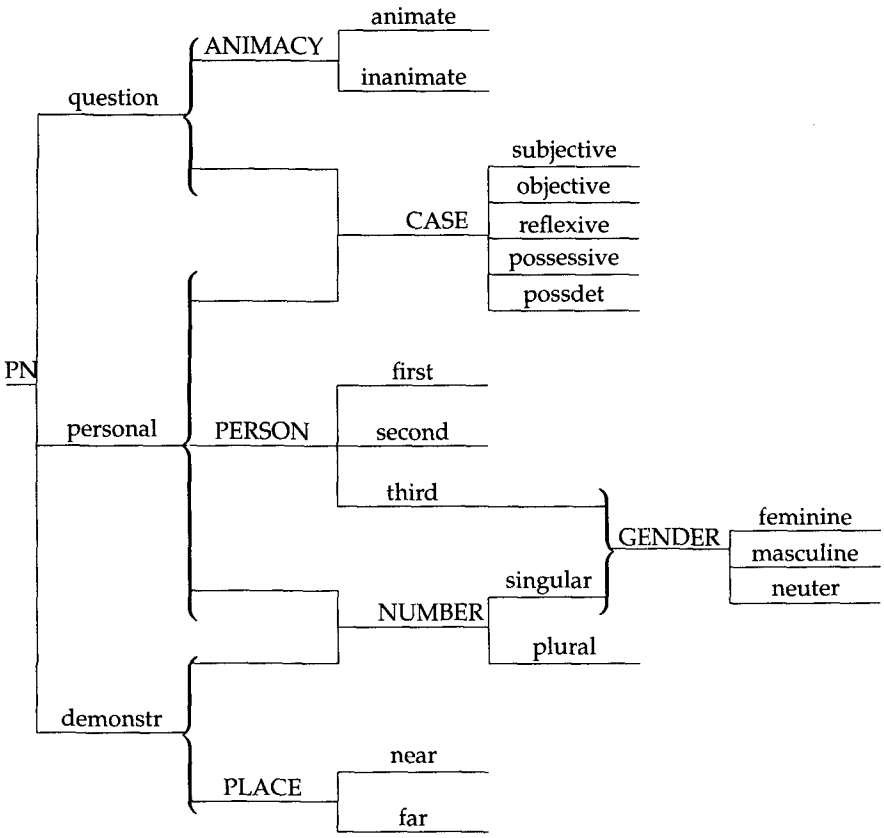

Figure 1
Winograd's pronoun network.

In this paper we shall be presenting formal arguments about the capabilities of the system network formalism, working at a level of detail that would make the presentation of large scale linguistic examples prohibitively tedious. Our main example will therefore be the network shown in Figure 1. This network provides information about the choices a speaker needs to make in order to correctly produce English pronouns. Similar networks can describe the choices that have to be made in order to generate other components of an utterance, such as sentences, verb phrases, or intonation contours, but for present purposes pronouns will prove sufficient illustration. We have applied the ideas described in this paper to a parsing system that accepts the language generated by the grammar of a pre-existing (Houghton 1986; Houghton and Isard 1987) dialog generation system.

Our algorithms are presented in a somewhat stylized form, since the primary intention is to expose the nature of the problems involved rather than to provide detailed information about the implementation of our parsers. For instance, we describe data structures that are closely related to the tapes of a two-tape Turing machine, but in the corresponding implementation the behavior of these data structures is mimicked by a program that manipulates nested Prolog terms. Were we to present the details of the real implementation, it would tend to obscure the most significant properties of the algorithm.

Realization Rules		
question animate subjective	→	*who*
question animate objective	→	*whom*
question animate possessive	→	*whose*
question inanimate	→	*what*
demonstr singular near	→	*this*
demonstr singular far	→	*that*
demonstr plural near	→	*these*
demonstr plural far	→	*those*
personal first singular subjective	→	*I*
personal first singular objective	→	*me*
personal first singular reflexive	→	*myself*
personal first singular possessive	→	*mine*
personal first singular possdet	→	*my*
personal second singular subjective	→	*you*
personal second singular objective	→	*you*
personal second singular reflexive	→	*yourself*
personal second singular possessive	→	*yours*
personal second singular possdet	→	*your*
personal first plural subjective	→	*we*
personal first plural objective	→	*us*
personal first plural reflexive	→	*ourselves*
personal first plural possessive	→	*ours*
personal first plural possdet	→	*our*
personal second plural subjective	→	*you*
personal second plural objective	→	*you*
personal second plural reflexive	→	*yourselves*
personal second plural possessive	→	*yours*
personal second plural possdet	→	*your*
personal third singular subjective feminine	→	*she*
personal third singular subjective masculine	→	*he*
personal third singular subjective neuter	→	*it*
personal third singular objective feminine	→	*her*
personal third singular objective masculine	→	*him*
personal third singular objective neuter	→	*it*
personal third singular reflexive feminine	→	*herself*
personal third singular reflexive masculine	→	*himself*
personal third singular reflexive neuter	→	*itself*
personal third singular possessive feminine	→	*hers*
personal third singular possessive masculine	→	*his*
personal third singular possessive neuter	→	*its*
personal third singular possdet feminine	→	*her*
personal third singular possdet masculine	→	*his*
personal third singular possdet neuter	→	*its*
personal third plural subjective	→	*they*
personal third plural objective	→	*them*
personal third plural reflexive	→	*themselves*
personal third plural possessive	→	*theirs*
personal third plural possdet	→	*their*

Figure 2
Realization rules for Winograd's pronoun network.

2. System Networks in NLP

2.1 Systemic Language Production

In many language-generation systems, such as those described by Davey (1978), Winograd (1972), Houghton (1986), Houghton and Isard (1987), and Mann and Matthiessen (1985), system networks are used to structure the decisions the system needs to make in the course of producing a syntactic constituent. Each network lays out a set of interconnected options, but does not specify how the system chooses between the options in a particular situation.

In systemic grammar, language production involves the task of traversing a network, gradually making choices that incrementally specify various aspects of the form of an utterance. As choices are made the system collects features that are eventually realized in an appropriate form. A brief example of how this process might be applied to Figure 1 is given below.

2.1.1 Houghton's Dialog System. The system on which our work is based (Houghton 1986; Houghton and Isard 1987) is an unusual synthesis of ideas from systemic grammar with a unification-based phrase structure grammar. It is at its most systemic in its approach to what (Halliday 1975, p. 5) refers to as functions in structure, but contains nothing directly corresponding to the macro-functions by which Halliday characterizes the functions of language. In other words, Houghton's system retains the role of systemic networks in codifying choice, but the architecture does not align with tripartite systemic distinction between textual, ideational, and interpersonal macro-functions (Winograd 1983, p. 288).

An interaction between two simulated agents is largely controlled by a representational level of interaction frames, which lays out a space of possible moves that allow an agent to initiate conversations, request information from an interlocutor, signal assent, convey information, and so on. This work is in the tradition of Davey (1978) and Power (1979), emphasizing the gamelike structure of multi-agent dialog in preference to the textual structure of extended monolog. In this context there seems to be little need for an explicit representation of the systemic macro-functions: Houghton's interaction frames suffice.

The interaction frames specify a repertoire of actions available to a simple planning mechanism that the agents use to carry out tasks specified by the system designer. The agents make no distinction between different types of action, and linguistic behavior only arises when it becomes necessary for an agent to enlist help to achieve its top-level goal. It is just as valid to find something out by looking as by asking, although interesting linguistic behavior will only occur in the latter case.

Houghton (1986) defines the interaction of *Making Something Known* as follows:

```
Participants    --- initiator <has_type AGENT>
                    addressee  <has_type AGENT>
                    prop       <has_type FACT>
End Goal        --- know(addressee,prop)
Effect          --- know(addressee,
                        know(initiator,prop))
Precondition    --- know(initiator,
                        not(know(addressee,prop)))
Response        --- addressee -> update worldview
                    initiator -> check acceptance
Reply           --- accept or reject information
```

and comments that

> The goal of this interaction is to get the addressee to know something. The immediate effect here is that they will at least know that you know it, and the precondition is that the initiator believes the addressee does not already know it. The response of an addressee is to attempt to integrate that information into its current beliefs. The initiator waits for uptake of the information, which is indicated by the addressee's reply. (Houghton 1986, p. 90).

In order to achieve the goal of informing someone that a door is locked, the speaker would instantiate `initiator` to point to herself, `addressee` to indicate her addressee and `prop` to stand for something like `locked(door_a)`, setting up the environment within which linguistic decisions are to be made.

In principle an agent is free to draw on any available information in order to reach decisions about the choices offered within Houghton's systemic networks, and the outcome of these choices will affect the eventual form of the utterance. In practice the types of information available to the agent can be brought into a rough and ready correspondence with the types of information carried by Halliday's macro-functions. We present our view of the correspondence solely to provide orientation for readers already familiar with systemic grammar and the tri-partite systemic distinction first mentioned above.

- Information about interpersonal factors is available from inspection of the state of the current dialog game. This allows the agent to keep track of *why* it is trying to speak. At this level we are only interested in the type of dialog game, the speaker's role within that dialog, and the stage that has been reached within the execution of that game.

- Information about ideational factors comes primarily from the propositional content embedded within the instantiated interaction template. Reference to the propositional content ensures that the system knows *what* it is trying to express. The distribution of propositional content between elements of the eventual utterance is mediated by a simple (de)compositional semantics using a lambda calculus–like representation similar to that provided by Pereira and Shieber (1987). The propositional content that is assigned to a particular constituent is extensively used by the chooser functions that determine the choices made in the course of realizing that constituent, but it is not the sole determinant of the eventual form of the utterance.

- Information about the component that Halliday calls textual is almost unnecessary in Houghton's system, since the tightly constrained structure of the dialogs make explicit signalling of topic or textual interconnections largely redundant. Houghton's agents are too simple and too single-minded to indulge in the sort of topic shifts that seem typical of real dialog, so there is no need to make available textual resources capable of signalling such shifts. Textual factors nevertheless play a role in the patterning of pronominalization decisions, and the information necessary for making these decisions comes primarily from

> a record of the preceding dialog maintained by the system. This is part of the task of deciding *how* to say what it is that needs to be said.

Given the simplicity of the domain and the actors, the nature of the dialog, the absence of an explicitly represented version of Halliday's tripartite distinction, and the central role of the planning system, the issue of how the various components interact loses much of the importance it has in standard systemic grammar. While Houghton does provide a mechanism by which making a choice at a particular node of a systemic network may, as a side effect, pre-empt decisions that might otherwise need to be made later, the issue of parallel selection of features does not matter as much as it would in a system where autonomous textual, ideational, and interpersonal components were acting in tandem to constrain the form of a text.

In standard systemic grammars the connection between features and utterances is encoded in the form of realization rules. In Houghton's system the realization relations are specified indirectly by means of an association between sets of features and the production of a context-free phrase structure grammar. Generation begins with a single node that will eventually span the whole sentence, and proceeds by recursively expanding that node until a complete phrase marker is produced. Whenever a node needs to be expanded the system collects a set of features by traversing one or more systemic networks, then uses the features to select an appropriate production from the context-free part of the grammar. The recursive generation of constituents terminates when it reaches a situation where all the current leaf nodes can be realized lexically. The phrase structure component encodes the range of available constructions and mediates the distribution of semantic content to the leaves of the tree. For a more detailed account of the generation process see the descriptions in Houghton (1986) and Houghton and Isard (1987).

2.1.2 The Generation of Pronouns. Having sketched the reasons for which we became interested in the properties of systemic networks and their role in a grammar, we now return to the discussion of system networks as formal objects in their own right, which is the main subject of this paper. Sample realization rules for English pronouns are provided in Figure 2. For pronouns, the realization relationship is particularly simple, since every distinct set of features that can be produced by the network maps onto a single lexical item. Throughout the rest of the paper we shall be using the generation of pronouns as a place holder for the more elaborate generation process outlined above.

Let us consider two situations in which a hypothetical language-generation system might need to generate a pronoun, and trace the choices that need to be made. In both cases the system is attempting to brief its client about a government decision, but in the first situation it has just produced the sentence *The government has decided to raise taxes*, but in the second the corresponding sentence was *The government* **have** *decided to raise taxes*. These sentences are both grammatically acceptable in British English, but in one case the government are seen as a collection of people while in the other it is being treated as an entity in its own right.[1] In either case the system now wants to tell the user that the government is/are unlikely to carry out its/their decision before the next election.[2]

1 The sentence containing this footnote is deliberately infelicitous, illustrating the mistake that we wouldn't want the system to make.

2 Stephen Isard points out that the parallel sentences *The government has fallen* and *The government have fallen* describe very different situations; evidently it is an oversimplification to suppose that a generation system has an entirely free choice between the alternative ways of referring to governments.

The system's understanding of pronoun use is based on the network shown in Figure 1, which it will use in choosing an appropriate pronoun for each of these situations.

The traversal of the network starts at the left-hand side. The first choice that has to be made is the one between `question`, `personal` and `demonstr`. In both cases we need a `personal` pronoun rather than a `question` pronoun or a `demonstr` one. We are now faced with the need for decisions on `CASE`, `PERSON` and `NUMBER`. Let us stipulate that the government, however we describe it, is to be the subject of the sentence; the pronoun therefore receives the feature `subjective` in both cases. The pronoun also has to be `third` person, since neither the system nor its client is the subject of the planned sentence. However the choice of `NUMBER` depends on the choice that was made earlier in the discourse, so in the first case the system will need to choose `singular` and in the second it will need `plural`. If it chooses `plural` it has reached the edge of the network, and can use the collected feature set to access the appropriate pronoun, which is *they*.

However, if it chooses `singular` it still needs to decide whether the government requires a `feminine`, `masculine` or `neuter` pronoun, since otherwise it cannot decide between *she*, *he*, and *it*. The system network formalizes the availability or otherwise of the `GENDER` choice by placing the corresponding system in a position that can only be reached if the pronoun being generated is both `third` and `singular`. Of course, a more elaborate process is involved in the generation of complex constituents, but the principle remains the same: networks are traversed, produce sets of features, and the features are used to guide the construction or selection of sub-constituents.

2.2 Systemic Language Analysis

By specifying the range of options available to the language user a system network defines a set of possible outcomes to the generation process, and hence also a space of possible structures corresponding to the various feature sets that can be obtained by traversing the network. It may be that the networks admit structures that are never in fact used by the system as a whole, but without knowledge of the details of the mechanism by which choices are made we cannot exploit any additional constraints imposed by the process of realization. In this paper we focus on the use of the networks as a knowledge source in their own right. Systemic networks form a terminological representation system in the sense of the term used in work on KL-ONE and its successors (Brachman and Schmolze 1985; Nebel 1990). We would like to understand the nature of the knowledge they express well enough to be able to use system networks as the basis of a general, and perhaps computationally convenient, terminological language for use in linguistic applications. Our approach is very close to that of Mellish (1988), as well as to a multitude of other formalisms involving partial descriptions, such as Shieber (1986).

By way of example, consider the word *your*, as it is described by the network and rules in Figures 1 and 2. Without further information we do not know whether it is `singular` or `plural`, although we can be sure that it is `second`, `personal` and `possdet`. This information will sometimes be provided by context. In the sentence *You should not take my word for it, but bolster your own intuitions by inspecting the realization rules for yourselves*, we eventually discover that the writer is addressing the sentence to an audience of more than one person. In generation the system would presumably have made a decision about which mode of address was more appropriate, but when using the same networks and rules for interpretation the partial nature of the information at our disposal complicates the task of using the knowledge encoded by the networks. Note that the clinching *yourselves* appears several words after the instance of *your* in question, and that it could equally well have been *yourself*.

2.2.1 Systemic Classification. Following Mellish, we refer to the task of using system networks in comprehension as systemic classification. We conceive of this as an activity similar to the taxonomic classification carried out by biologists. By making use of the observable features of a specimen we attempt to pin down the point at which that specimen ought to be accommodated within the space of structural descriptions we have at our disposal. On the assumption that the speaker's linguistic options are correctly described by a particular collection of networks, a listener is entitled to make inferences from this assumption, and to use knowledge of the form of the system networks in the search for a plausible global hypothesis about the properties of the utterance that the speaker seems to have produced. Because the choices made by a speaker are interdependent, a plausible hypothesis about part of the description of an utterance may allow us to reconstruct part or all of the missing portion of the description. The job of a scheme for systemic classification is to provide efficient and correct algorithms for the construction of consistent descriptions and the rejection of inconsistent hypotheses. Nebel (1990) discusses a slightly more general notion of classification, applicable to a knowledge representation language like NIKL (Kaczmarek, Bates, and Robins 1986), BACK (Von Luck et al. 1986) or LOOM (MacGregor and Bates 1987). The notions of subsumption that operate in such rich formalisms are more sophisticated, but the essential idea of classification, namely the organization of terms into a pre-computed subsumption hierarchy, remains the same. As in Mellish's work, the main goal is to gain efficiency by pre-computing some or all of the useful subsumption relationships that hold between descriptions. For the purposes of this paper the descriptions are feature-based systemic descriptions and the role of the pre-computed subsumption graphs described by Nebel is played by the specially designed encodings that ensure that subsumption information is readily available at run-time. There is of course a price to pay for this run-time efficiency, since we have to pre-process the networks in order to establish the encoding relations that will be of use at run-time.

For the purposes of analyzing an utterance, the most important property of the feature system induced by a system network is the following

> Any system network defines a set of atomic labels that can be combined to form descriptions of linguistic objects. It also provides a complex of constraining principles that conspire to declare inconsistent certain combinations of features.

For example, the network in Figure 1 expresses the constraint that no pronoun can be specified as both `feminine` and `masculine`, and also the constraint that only third person singular pronouns can bear either of these features.

2.3 Prior Work on Systemic Language Analysis

Winograd (1983) and McCord (1977) have both provided techniques that re-express system networks and their associated realization rules as hand-crafted recognition programs.

In both cases this amounts to a proceduralization of the grammar, and runs the risk of making it even harder than necessary to debug a malfunctioning grammar. We prefer to adopt a more declarative approach, in which the grammar is represented in a form as close as possible to that with which linguists actually work.

Patten and Ritchie (1986) have produced a formal account of what a systemic grammar is. This covers the whole of Systemic Grammar, including the realization rules and the details of the way in which choices are made as well as the networks that are the present concern. Patten and Ritchie are primarily concerned with language-generation

rather than comprehension, but since they aim for a declarative specification of the generation relationship between systemic grammars and the utterances they license, it might be possible to use their grammars in either direction. For Patten and Ritchie a systemic grammar is encoded as a set of rules for a production system, and the generation process involves the application of these rules.

Kasper (1987a, 1987b) has designed algorithms that involve the encoding of both system networks and the associated realization rules as constraints expressed within a feature logic involving disjunction. He then implements the key operation of unification by applying a general technique for unification of disjunctive feature descriptions (Kasper 1988). These techniques decompose the problem of disjunctive unification into three stages, only the last of which is exponentially hard. We are not attempting to match his general algorithms for disjunctive unification, but rather to design special purpose algorithms that are particularly suitable for use with system networks, and particularly for those networks that play a role in Houghton's system.

In practice the set of constraints that Kasper develops from his systemic grammar is too large to solve conveniently with the constraint-solving technology available to him, so he augments the constraint grammar with a small hand-crafted phrase structure component. The main motivation for this move is the search for efficiency rather than any particular wish to use a phrase structure grammar. Kasper's basic strategy remains that of building up a large set of constraints that can then be solved by general theorem-proving techniques.

Like Kasper, we use a chart parser, but in our case the phrase structure grammar is an essential component of the knowledge used by the system rather than a small addition required for the sake of efficiency. This makes no difference from a formal point of view, but reflects our strategy of using appropriate special purpose mechanisms in preference to potentially costly general techniques. Although system networks, systemic realization rules, and context-free grammars can all be translated into formulae of Kasper's extended FUG, our approach has been to investigate ways in which the knowledge contained in these components can be exploited using simpler methods.

While Kasper's approach involves the translation of system networks into a more general constraint formalism, Mellish (1988) displays encodings that reduce the key operations of systemic classification to straightforward manipulations of PROLOG-like terms. Unfortunately, it turns out that system networks sometimes need a rather opaque encoding, which would require costly re-translation if users needed to inspect the workings of an NLP system that used it. The general encoding is in principle capable of handling any system of constraints expressed in the propositional calculus, and makes no use of the particular structure of systemic descriptions.

Mellish's approach to the implementation of systemic classification is to construct structure preserving mappings from the space of systemic descriptions to an isomorphic space in which classes of entities are represented by terms taken from the GAF lattice.[3] These terms act as partial descriptions of linguistic objects, and are combined using the operation of term unification. Unfortunately the only generally applicable mapping that Mellish is able to offer has unpleasant properties, producing large terms whose internal structure has little in common with the structure of relationships between the descriptive feature labels used in the network.[4] Although the terms produced by the brute force mapping are spectacularly large, our main objection to

3 For most purposes this lattice is the same as the recursively defined space of terms used by PROLOG, but for a technical introduction see (Reynolds 1970).

4 The terms produced represent classes of objects by a technique little different from exhaustive enumeration.

this approach is the loss of transparency associated with the mapping. It doesn't seem likely that a linguist would feel comfortable with the output of the mapping, so a grammar development system would presumably have to include facilities for translating the machine representation back into more readable form. Although the simpler mappings that Mellish has developed for more restricted forms of network do offer more perspicuity, we feel that even in those cases our representations are at least as accessible to the human user as Mellish's, which do not in any case generalize to the full system network formalism.

Our view of the nature of system networks is close to those of Mellish and Kasper, since we try to separate questions about the underlying meaning of the networks from decisions about the processes by which they can be exploited. We treat system networks as alternative notations for sets of logical axioms constraining the co-occurrence of property symbols in descriptions. Nothing more elaborate than the propositional calculus is required. In principle one could apply standard theorem-proving techniques to these axioms, but in practice it is better to employ special purpose techniques that make more use of the structure inherent in the networks. The translation of system networks into logical axioms is very similar to the one described in Mellish (1988), but by adopting a slightly different labeling scheme we obtain a little extra clarity, which helps when we come to design algorithms that make use of the axioms produced.

2.3.1 Classification and Consistency Checking. In order to carry out classification we need to answer the following questions

1. When is an object an instance of a given description?
2. When may two given descriptions describe the same object?
3. Given two descriptions of the same object, how may the descriptions be combined to produce a more fully specified description?

and in order to answer the questions we need to understand the behavior of the key operations of subsumption checking and unification as they apply to the description space within which we are operating. Both these operations can effectively be treated as special cases of consistency checking.

This is the operation of checking a conjunction of positive and negative boolean attributes for consistency with the axioms derived from a network. This should succeed if the conjunctive description picks out a nonempty class of objects from those capable of being derived from the network, but to fail if there is no object which can meet the constraints imposed.

Unification. For descriptions consisting of atomic symbols generated from a system network the operation of unification between two descriptions D_1 and D_2 consists of two stages.

1. The formation of the conjunction D such that D contains the union of all the feature specifications given in D_1 or D_2.
2. The checking of this new description D for consistency with the constraints arising from the system network.

We shall see that it is the consistency checking that makes this operation potentially costly.

Subsumption. One description D_1 subsumes another D_2 if all possible objects correctly described by D_2 are also correctly described by D_1. If A subsumes B then $B \supset A$. If that is so then $B \wedge \neg A$ is unsatisfiable. If A is an atomic symbol then $B \wedge \neg A$ can be checked directly for consistency; otherwise $\neg A$ has to be expanded out, yielding

$$(B \wedge \neg a_1) \vee (B \wedge \neg a_2) \vee \ldots (B \wedge \neg a_n)$$

In order to prove that $B \supset A$ we have to show that none of the alternatives given above can ever be satisfied. The subsumption check fails the moment any of the disjunctive alternatives is found to be a description of a legal object. While the individual branches of this checking process can be carried out as efficiently (or inefficiently) as unification itself, the preliminary expansion to disjunctive normal form is potentially costly. This is doubly unfortunate since the expansion depends on the input expressions, and therefore cannot be carried out at compile time. This problem evaporates if we restrict ourselves to networks lacking *disjunctive* systems,[5] since Mellish's term-encoding techniques allow both subsumption and unification to be implemented using standard operations familiar in logic programming.

Realization Rules and Disjunction. Although the features mentioned in the system networks may in fact correspond to large disjunctions of different low-level features when the networks and the realization rules are compiled together into one large constraint system, we treat the realization rules by a different mechanism and can therefore continue to regard the features as atomic. Our primary concern is to ensure that the basic feature-matching operation carried out by our chart parser is efficient enough to be carried out many times in the course of analyzing utterances. The implicit disjunction between the different productions of Houghton's phrase structure grammar is handled by the usual techniques of chart parsing, which are described in, for example, section 3.6 of Winograd's textbook (Winograd 1983). Since these techniques are acceptably efficient, there is no pressing need to adopt Kasper's approach, within which the realization rules are conjunctive constraints, and the effect of including realizations is to change the representation of choice systems from disjunctions of atomic formulae into disjunctions of complex formulae. As before, our approach is to replace the general mechanism with a less powerful technique that is well attuned to the task in hand.

2.4 Labeling System Networks

This section describes the process of translating system networks into logical axioms. As an intermediate stage we build a specially labeled version of the network with which we are working. In order to do this we need a more formal characterization of system networks than we have yet presented.

A system network is made up of possibly labeled lines tied together by systems. In the system networks described by Mellish some, but not necessarily all, of the lines are labeled with items chosen from a set of distinct atomic symbols. The example in Figure 1 shows such a labeling. In this scheme there is only one sort of label, no distinction being made between words that are usually thought of as feature names (such as `NUMBER`) and those that would more naturally be feature values (such as `singular` or `plural`). We would in any case like a labeling scheme that makes a clean distinction between feature names and feature values, and it turns out that

5 These are defined in the next section.

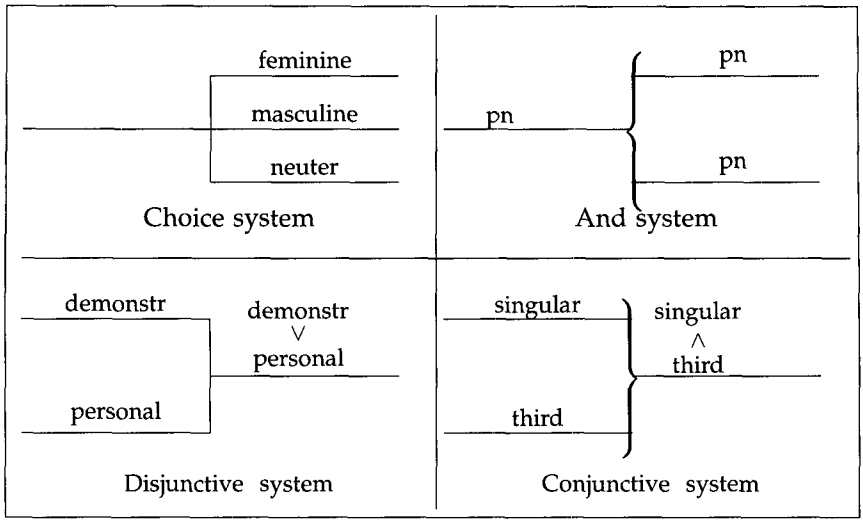

Figure 3
Types of system.

suitable slight modifications in Mellish's scheme greatly simplify the task of translating networks into equivalent sets of axioms. We start by introducing the new labeling scheme, and showing where it differs from Mellish's version.

The differences between the labeling schemes are small. Networks labeled in our style continue to represent the same information as would be contained in the same networks labeled by Mellish or Winograd. All that is gained is a measure of uniformity and clarity, which helps simplify the task of translating networks into logic.

2.4.1 Types of System. There are four distinct types of system, each of which has one collection of lines as its right-hand side and another collection as its left-hand side. For a given system s these are notated as $RHS(s)$ and $LHS(s)$, respectively.

The choice system has one line on its left-hand side and a number of lines greater than one as its right-hand side. Informally it represents a choice of exactly one attribute from those given by the labels attached to the lines in the right-hand side.

The system at the top left of Figure 3 is a *choice* system in which the alternation between `feminine`, `masculine`, and `neuter` is expressed.

The system at the top right of the figure represents a form of conjunction. Where the `choice` system contains a vertical line standing for disjunction, the `and` system represents conjunction with a left curly bracket (i.e. {). Again exactly one line must appear on the left-hand side of the system, and there should be a larger number of lines on the right-hand side. `And` systems mean that all the right-hand lines of the system will be tried whenever the left-hand line is reached. Since no choice is made, no feature is generated by traversing an *and* system.

The next two types of system both have compound entry conditions, with one line on their right-hand side, but several feeding in to their left-hand side. Since they represent forms of disjunction and conjunction they will be referred to as disjunctive systems and conjunctive systems respectively. We only make progress through the right-hand line of a *conjunctive* system if all the left-hand lines of the system are

traversed. For *disjunctive* systems we can get through to the right-hand line if one or more of the lines on the left-hand side are traversed. The system at the bottom left of the figure is a *disjunctive* system, and the one at the bottom right is a *conjunctive* system.

2.4.2 Well-Formedness Requirements for System Networks. This section sets out our definition of what it means when we say that a system network is well formed. Given a system network, the *precedes* relation between lines is the smallest relation such that

1. n_1 precedes n_2 if

$$n_1 \in LHS(s), n_2 \in RHS(s)$$

   for some system s in the network.

2. n_1 precedes n_2 if for any line n_3 in the network n_1 precedes n_3 and n_3 precedes n_2.

If n_1 precedes n_2 then n_1 is a predecessor of n_2. In a well-formed system network no line may precede itself. Strictly speaking, this is a deviation from Systemic Grammar as presented in Hudson (1971), which explicitly permits cyclic systems. For our purposes cyclic networks are unsatisfactory, because useful cyclic paths in the network have to be traversed more than once. If network traversal is to terminate, the choices made on the last traversal of such a path have to differ from those made on earlier traversals. The sets of features that could be generated by a network would then depend not just upon the structure of the network but also on the way it was used by a particular generation system. Our techniques would have nothing to say about this situation, since we are examining the networks in their own right, and do not have access to the criteria by which the choices are made. We are therefore outlawing cyclic networks for the moment.

Assuming that the graph formed by the system network is acyclic implies that there must be at least one line that is not preceded by any other. If there is more than one such line, then the network has multiple entry points and no obvious intuitive meaning. For the sake of definiteness we decree that such networks will not be translated as such, but will instead be prefixed with a single *and* connective, whose left-hand side contains a single new line, and whose right-hand side comprises the set of lines that would otherwise have had no predecessor. There is now exactly one line with no predecessor, which can be referred to as the root. The root of Figure 4 is the line labeled with *e*. It is possible to draw networks in which certain regions of the network can never be reached. The simplest case of this involves distinct branches of a **choice** system feeding into a *conjunctive* system, as shown in Figure 4. Here the choice between b_1 and b_2 will never be reached. At best this is a waste of space. We say that networks containing unreachable portions are unparsimonious, and declare them to be ill formed. While it may not always be easy to see whether a complicated network is parsimonious, the necessary work can be carried out when the network is defined, and need not carry a run-time cost.

2.4.3 Basic Labelings. In this section we introduce the idea of a basic labeling, which is intended as a formalization of what we take to be the uncontroversial part of the process of labeling a system network. A basic labeling is defined to be a partial function

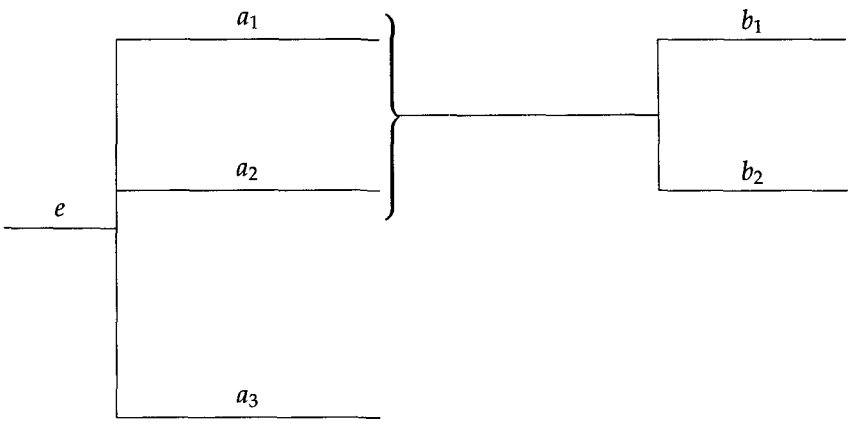

Figure 4
An unparsimonious system.

from lines to names such that

- a line receives a name if and only if there is a *choice* system to whose right-hand side it is directly attached.
- no two lines carry the same name.

In practice, system networks are labeled in ways that are very nearly basic labelings. The major difference is that some nodes attached to the left but not to the right of *choice* systems get given labels that correspond approximately to feature names. Since these labels add nothing to the meaning of the network they can safely be ignored.

The restriction to networks without duplicated labels simplifies the discussion in the rest of the paper, but it is not hard to transform networks with duplicated labels into ones in which all labels are distinct. The details of this transformation are not discussed here.

2.4.4 Exhaustive Labelings.. Every exhaustive labeling builds on a basic labeling, and is a labeling in which every line in the network is labeled with some value. In contrast to the conventional systemic labelings, in which labels are constrained to be atomic names, exhaustive labelings assign boolean conjunctions and disjunctions of atomic names as labels for some lines. The full definition of an exhaustive labeling follows.

Let $\mathcal{F}$ be a 1-1 function from lines to names for these lines. $\mathcal{F}$ is an exhaustive labeling of a network if and only if the following conditions hold:

1. That part of $\mathcal{F}$ that provides names for lines attached directly to the right of *choice* systems must be a basic labeling.
2. If $\mathcal{F}$ assigns a line name l_{lhs} to the line directly attached to the left-hand side of an *and* system, then it must also assign that name to the lines directly attached to the right-hand side of that system.

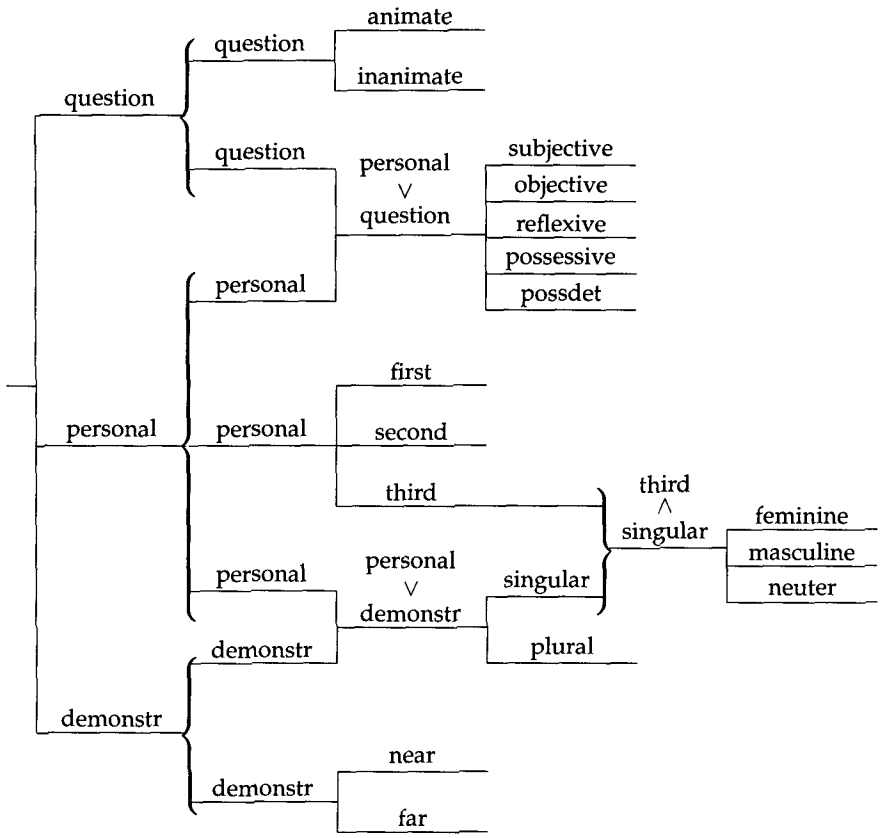

Figure 5
An exhaustively labeled network.

3. If $\mathcal{F}$ assigns line names $l_1, l_2, \ldots l_m$ to the lines entering a *conjunctive* system (where l_1 is the label for the line appearing at the top of the system and l_m that at the bottom), then it must assign the label $l_1 \wedge l_2 \wedge \ldots l_m$ to the line that leaves that system.
4. If $\mathcal{F}$ assigns line names $l_1, l_2, \ldots l_m$ to the lines entering a *disjunctive* system (adopting the same ordering convention as above), then it must assign the label $l_1 \vee l_2 \vee \ldots l_m$ to the line that leaves that system.

The ordering convention we introduced above is a technical convenience rather than an essential part of the nature of an exhaustive labeling; all that matters is that appropriate expressions representing conjunctions and disjunctions are propagated to the right-hand lines of *conjunctive* and *disjunctive* systems.

One exhaustive labeling for the sample network given earlier is shown in Figure 5. Three of the systems in the diagram involve complex entry conditions. This expresses itself in the presence of complex labels having the form of boolean combinations.

2.5 Translating the Labeled Networks

In this section we show how to convert an exhaustively labeled network into a set of axioms expressing the information from the network in the framework of propositional calculus. This work is closely related to a similar translation scheme given in Mellish (1988), but because our scheme works with slightly different labelings the results are a bit easier to understand.

The translation scheme in Mellish's paper can be applied to all four types of systems in an exhaustively labeled network, and the correct results will be produced, but it turns out that many of the systems in an exhaustively labeled network map into uninteresting tautologies.

Mellish proposed that a *choice* system such as the one for GENDER in Figure 1 should be translated as

$$AMO\{feminine, masculine, neuter\} \tag{1}$$

$$GENDER \equiv feminine \vee masculine \vee neuter \tag{2}$$

where *AMO* stands for a complex formula meaning 'at most one of.' (Mellish actually presents these axioms in a predicate calculus notation, but we have suppressed this detail, which makes no difference.) In the example of a *choice* system given above the full expansion of the complex formula would be

$$\neg(feminine \wedge masculine) \wedge$$

$$\neg(feminine \wedge neuter) \wedge \neg(masculine \wedge neuter) \tag{3}$$

In Figure 5 we have removed the GENDER label from the left-hand side of the *choice* system and replaced it with the appropriate complex label. When we employ Melish's scheme the translation becomes:

$$AMO\{feminine, masculine, neuter\} \tag{4}$$

$$third \wedge singular \equiv feminine \vee masculine \vee neuter \tag{5}$$

but the essential principle remains the same. Two intimately linked axioms arise from each *choice* system. Where necessary we shall refer to the axiom involving *AMO* as the exclusivity axiom and the other one as the accessibility axiom. It is easy to construct the exclusivity axiom given the corresponding accessibility axiom, but the converse is not true. We shall therefore be basing our implementation on the use of the accessibility axiom, with the exclusivity axiom understood to be implicitly present.

For *and* systems Mellish proposes a translation that is as if the label from the left of the *and* system had been written onto the lines to the direct right of the *and* system. In our scheme the label actually *has* been written there by the time the translation comes into operation, so there is no need to provide an independent translation for *and* systems. Instead the systems to the right of the *and* system will automatically receive translations reflecting the presence of the *and* system. This happens for the three *and* systems of Figure 5, with question, personal, and demonstr being propagated across the system in the appropriate way.

For *disjunctive* and *conjunctive* systems applying Mellish's translation would again yield a tautology. Where Mellish translates the *disjunctive* system of Figure 3 (which carries the feature name NUMBER on its right-hand branch) as

$$personal \vee demonstr \equiv NUMBER \tag{6}$$

applying Mellish's translation to our labeling scheme yields the tautologous

$$personal \vee demonstr \equiv personal \vee demonstr \tag{7}$$

Intuitively it looks as if the real work that goes on in a system network happens in *choice* systems, and the other lines exist only in order to link together *choice* systems in the appropriate ways. From any point of view except that of a mathematical troublemaker this was probably obvious from the outset.

The axioms derived from the pronoun network in Figure 5 are the following.

$$question \equiv animate \vee inanimate \tag{8}$$

$$demonstr \equiv near \vee far \tag{9}$$

$$question \vee personal \equiv subj \vee obj \vee refl \vee possess \vee possdet \tag{10}$$

$$personal \vee demonstr \equiv singular \vee plural \tag{11}$$

$$personal \equiv first \vee second \vee third \tag{12}$$

$$third \wedge singular \equiv feminine \vee masculine \vee neuter \tag{13}$$

The algorithm for translating system networks into a more standard form is now complete, so we move on to consider the ways in which this information can be used.

2.6 Computational Tractability

In the next section we address the question of whether the labeling schemes developed so far can form the basis of efficient algorithms for carrying out the task of systemic classification.

We start by showing that system networks can be used to express NP-hard problems. This is something of a disappointment, since the original goal of this work was to show that the network formalism could function as a low-power version of more general feature formalisms. The idea was that linguists who chose to express their grammars within the network formalism would be sacrificing a measure of notational convenience in order to gain a guarantee of acceptable computational behavior. Since we must currently assume that NP-hard problems are intractable, we are unable to provide the desired guarantee.

2.6.1 Systemic Classification is NP-Hard. In this section we show that the problem of systemic classification is at least as hard as problems known to be NP-hard. This is done by constructing a polynomial time mapping Π from instances of the NP-hard problem called 3SAT to networks that can be tricked into solving this problem for us. This is a standard technique from complexity theory. For an introduction to similar linguistic applications of complexity theory see Barton, Berwick, and Ristad (1988).

If there were a polynomial time algorithm for checking arbitrary system networks, it would follow that 3SAT could be solved by the composition of the mapping that constructs the network with the algorithm that checks the network. Since this composition is itself a polynomial time algorithm we would then have a polynomial time solution for 3SAT, and hence for all other problems of the complexity class $\mathcal{NP}$. Thus the successful construction of Π implies that systemic classification is itself NP-hard. (Whether NP-hard problems can be solved in polynomial time is still an open question at the time of writing, but, for the moment at least, they must be considered intractable). Kasper's demonstration that general disjunctive unification is NP-complete, taken together with the reducibility of systemic classification to that problem, indicates that

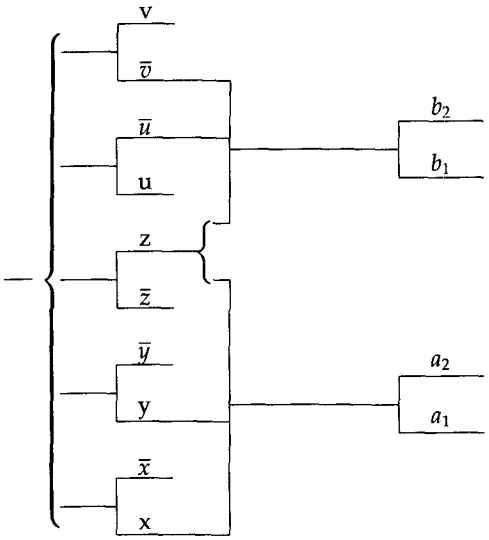

Figure 6
A network for 3SAT.

systemic classification is NP-complete as well as NP-hard. We think this upper bound on the complexity class to be of purely theoretical interest, since our parsers depend on efficient description-matching as a primitive, and an NP-hard problem is in our view already too expensive to be solved regularly in an efficient parsing system. We are aware that the complexity result only applies to the worst case, but in the absence of a satisfactory account of precisely what constraints a grammar writer has to observe to avoid the bad cases, we are uneasy about providing these grammar writers with a potentially explosive general mechanism. Our guess is that there is something about the human cognitive system that ensures that hard cases won't arise naturally, but it is dangerous to assume that they won't arise, not least because one possible way of using the technology provided here is as a target representation for automatic grammar development environments that present the user with higher level primitives. Whatever modularity constraints may be observed by a human writer, it is unclear that similar constraints will automatically hold over the output of an automatic grammar transformer.

The 3SAT problem. 3SAT is the problem of determining the satisfiability of a boolean formula, stated in conjunctive normal form, in which exactly three variables occur in each clause of the conjunction. These variables may either be positive or negated, and may be repeated from clause to clause. It is known that 3SAT is just as hard as the problem of satisfiability for general boolean formulae (Barton, Berwick, and Ristad provide a demonstration of this fact on pp. 52–55 of Barton, Berwick, and Ristad [1988]). A sample input formula for 3SAT is

$$(x \vee \overline{y} \vee \overline{z}) \wedge (y \vee z \vee u) \wedge (\overline{x} \vee y \vee u) \wedge (x \vee z \vee \overline{u}) \tag{14}$$

and the problem is to find an assignment of `true` and `false` to variables such that the whole expression comes out true.

A mapping from 3SAT instances to networks. We now construct the mapping Π that takes a 3SAT instance and constructs a network that can be used to solve that instance. The following steps are necessary. Let the name of the 3SAT instance be E and its length N_E.

- Make a list of the variable names used in E, counting positive and negative occurrences of a variable as the same. This can certainly be done in time polynomial in N_E using a standard sorting algorithm such as merge sort. Let the name of the list of variable names be V and its length N_V. Because the drawings are prone to get very intricate we use the example of the very simple expression

$$(x \vee y \vee z) \wedge (z \vee \overline{u} \vee \overline{v}) \tag{15}$$

  (which has already been illustrated in Figure 6)

- Construct a network consisting of a large *and* system feeding N_V parallel binary *choice* systems. Each *choice* system carries two labels, one corresponding to a variable name in V and the other formed by negating the label on the other branch of the system. The choice of prefix should be such that all labels on the resulting network are unique. This part of the process is polynomial in the length of V.

- For every clause in E, add a ternary *disjunctive* system linking the lines of the network having the labels corresponding to the three symbols of the clause. This part of the process involves scanning down the N_V systems of the network once for each clause of E, and is therefore also polynomial in N_E.

- Finally, binary *choice* systems are attached to the outputs of all the *disjunctive* systems introduced in the last stage. These systems are labeled with generated labels distinct from those already used in the network. This step is clearly also polynomial in N_E, requiring the creation of a number of *choice* systems equal to the number of clauses in E.

The network given in Figure 6 is the one that would be produced from E. In order to use the constructed network to solve the satisfiability problem for E, we check an expression corresponding to the conjunction of all the three member clauses in E. This is built by choosing an arbitrary label from each of the rightmost *choice* systems. The conjunction of these labels is a consistent description whenever all the clauses of E can be satisfied by the same value assignment. The *choice* systems to the left of the disjunction express the facts that no variable can be simultaneously true and false. It should now be obvious that any correct checking algorithm will in fact succeed in just those circumstances where there is at least one value assignment for the variables of E

that makes E come out true. This means that systemic classification cannot be solved in polynomial time unless 3SAT, and hence all of $\mathcal{NP}$, can.

3. Algorithms for Consistency Checking

3.1 Isolating the Sources of Intractability

Given that there is no efficient way of applying all the constraints imposed by a system network, it would help if we could isolate the source or sources of the inefficiency. If this can be done, it might allow an algorithm for constraint application in which a cheap first stage does most of the work, calling on an expensive second stage only when it is needed. It turns out that the presence of *disjunctive* systems is the main cause of the inefficiency, so we now move to a method for the removal of these systems. The idea is that the nondisjunctive version of a disjunctive network expresses some, but not all, of the constraints present in the original network.

We have two equivalent representations for the constraints induced by a system network, and shall feel free to switch between the view of networks as geometric objects and the view of axiom sets as constraints on the structure of linguistic entities. We shall refer to diagrams of networks in the hope that this will help the reader gain intuitions about what is going on. Readers who find the diagrams unhelpful should ignore them.

There is a simple transformation that maps networks involving disjunction into nearly equivalent versions not containing any *disjunctive* systems. Although this temporarily ignores some of the information implicit in the disjunctive network, every description the transformed network would reject as inconsistent is also rejected by the full network from which it is derived. The right-hand sides of *choice* systems are left untouched by the transformation, so the exclusivity axioms of the original network continue to hold.

System networks form directed acyclic graphs, so it is possible to use the precedence relationship defined above to form a topologically sorted list of the atomic labels in the network. Precedence is not a total ordering on the atomic labels, so some other criterion, such as lexicographic ordering, is needed to produce an unambiguously ordered list. We call the position of a label in this list its depth. It does not matter what the extra criterion for ordering the depth list is, as long as it is consistently applied and no two labels have the same depth. By extension, *choice* systems also have a depth, defined to be the depth of the shallowest label in the right-hand side of the system. Atomic symbols in the sets of axioms derived from system networks inherit the depth of the corresponding labels in the network, and axioms inherit the depth of the corresponding *choice* systems.

The idea is that every fragment of the original network that flows out of the right of a *disjunctive* system is detached from that system and re-attached nearer the root of the network. This point is the point at which the various paths leading from the root to the *disjunctive* system most recently diverged. Although the formal details of how to do this in a general network are somewhat intricate, an illustration is provided in Figure 7. The two systems that have been moved are those that were originally fed by disjunctive systems in Figure 5, namely those which express choices for `CASE` and `NUMBER`. These have both been moved back from their original location and re-attached immediately to the right of the root of the network.

The new network now allows various combinations of features not allowed by the original network. It would, for example, permit the generation of a `reflexive, demonstrative, far, singular` pronoun, if one existed (it would presumably be *that-*

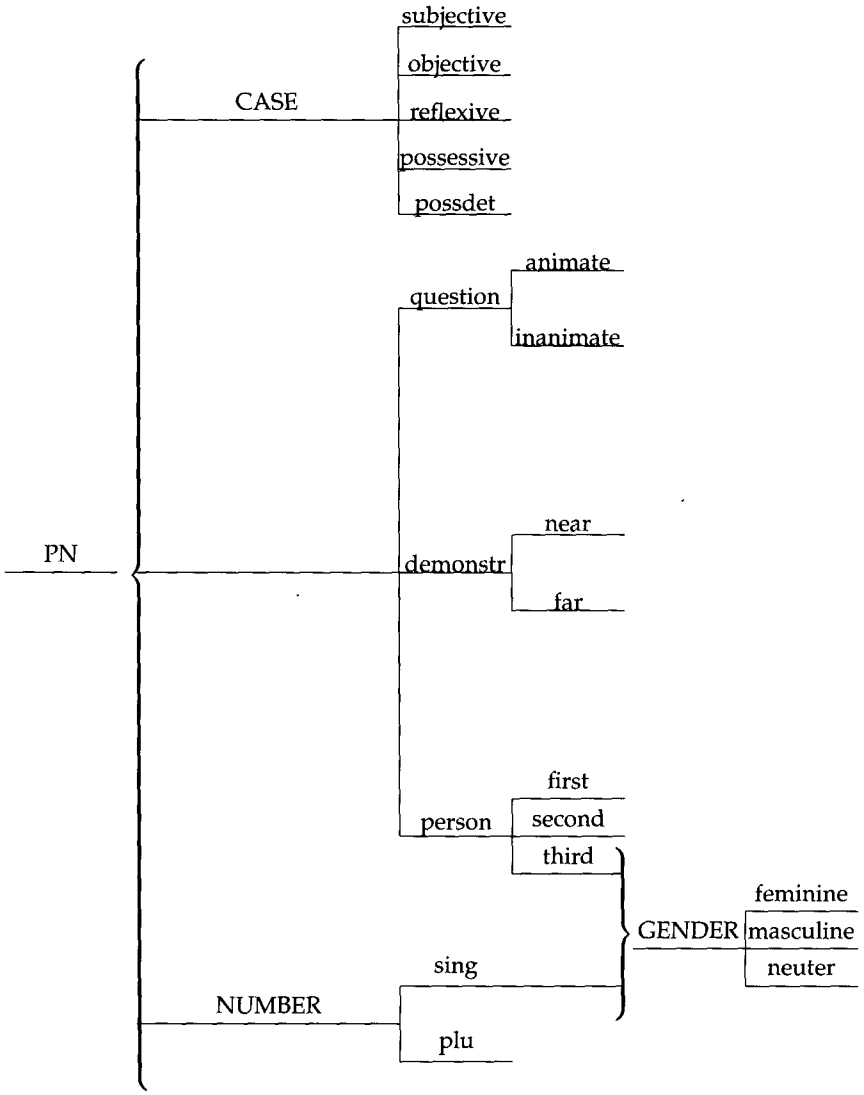

Figure 7
A simplified network.

self). Some of the constraints expressed in the original network are lost in the transformed version.

3.2 Relaxing Sets of Axioms

Relaxation is the formal counterpart of the network rewriting described in the last section. For clarity, it is defined over sets of axioms rather than over networks, simply because it is far easier to achieve the necessary clarity when working in the domain of propositional formulae.

In a set of axioms, the *precedes* relationship between labels in the axioms is defined in much the same way as it was for labels in a network, namely, as the smallest relation

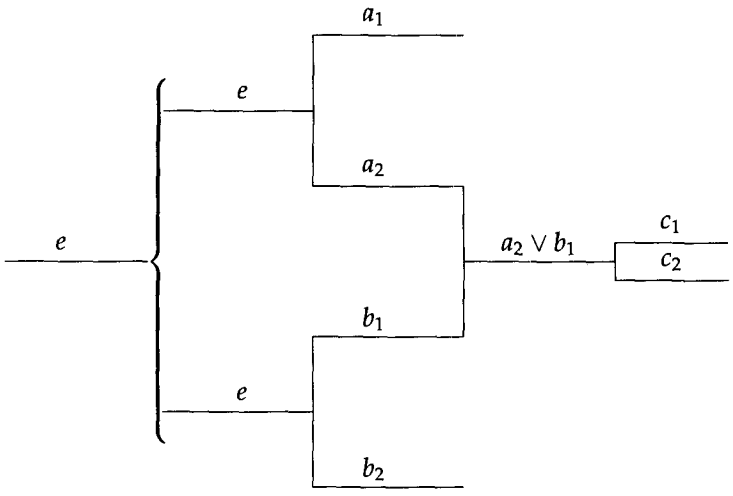

Figure 8
A small example network.

such that:

1. l_1 precedes l_2 if
$$l_1 \in LHS(a), l_2 \in RHS(a)$$
for some axiom a.
2. l_1 precedes l_2 if for any label l_3 in the set of axioms l_1 precedes l_3 and l_3 precedes l_2.

The common predecessors of a set of symbols S are defined to be the members of that set $P(S)$ that precede every member of S. Every fully elaborated description containing any member of S must also contain every member of $P(S)$. The deepest common predecessors of S are those members of $P(S)$ having no successor in $P(S)$. The transformed set of axioms is formed by replacing every axiom involving disjunction in the LHS with a rule made up of the same RHS and a new LHS consisting of the conjunction of a specially generated identifying feature with the deepest common predecessors of the elements of the old LHS. As an example we use the network given in Figure 8, which was used by Mellish to demonstrate the necessity for repeated variables in the terms representing systemic descriptions. It has been exhaustively labeled, showing the disjunction introduced by the presence of a disjunctive system feeding the rightmost *choice* system. This network demonstrates that the consistency of descriptions is not always deriveable from information about the pairwise consistency of individual features. For example, while a_1 is consistent with b_2 and also with c_1, it does not follow that the combined description $a_1 \wedge b_2 \wedge c_1$ is also consistent.

In Figure 9 the result of peeling away the disjunction is shown. Immediately preceding the *choice* system for c_1 and c_2 is a degenerate *choice* system with one entry and one exit. This system is there in order that the special marker feature genfeat can be inserted on the right-hand side of a choice system, and goes hand in hand with the definition of basic labelings given earlier.

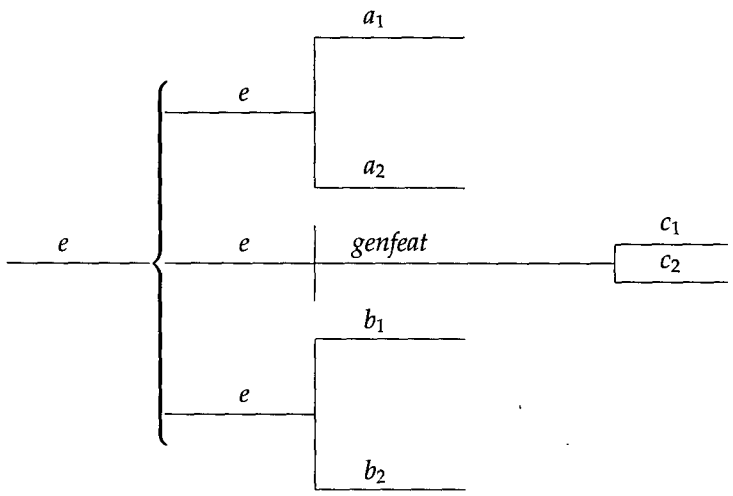

Figure 9
A transformed version of the example.

A record is kept of the correspondences between the features introduced to stand for left-hand sides and the expansions for which they stand. This is used, if necessary, in a second stage of checking. In the example above we would have had to note that

$$genfeat \equiv c_1 \vee c_2$$

and to retain this information for future checking.

3.3 Overview of the Checking Algorithm

Our consistency checking algorithm works by implementing a sort of parallel search, working back from the right of the network toward its root. The run-time behavior of the algorithm depends on the properties of the networks used. If the network involves *disjunctive* systems then true parallelism is required, with multiple processes constructing mutually inconsistent partial descriptions on the basis of the same seed description. Since parallel processes may need private copies of part or all of the target expression, this approach will involve us in potentially exponential amounts of work copying data structures.

If *disjunctive* systems are not present, cheaper approaches are possible, because we can organize matters so as to require no more than one copy of the target expression. This allows us to demonstrate the tractability of consistency checking for system networks lacking disjunction, while the checking procedure for full system networks remains just as costly as we would expect given its equivalence to 3SAT.

In what follows we shall be introducing a way of decomposing sets of axioms into components corresponding to a nondisjunctive network and a group of disjunctive subnetworks. This involves the construction of new expressions based on the axioms of the original network. Strictly we ought not to use the term 'axiom' for the results of our manipulations, but we shall continue to do so, since the algorithms apply equally well to networks that never had *disjunctive* systems in the first place as to the transformed versions of networks that did.

Our algorithms decompose the problem of systemic classification into three stages:

1. An off-line step in which a general system network is broken down into a relaxed version lacking disjunction and a set of constraints expressing the information missing from the relaxed version of the network.
2. A first stage of checking in which the relaxed form of the network is applied to an input expression.
3. A second (optional) stage in which the information missing from the relaxed network is re-applied to the results of the first check.

In the compilation step the axioms from the original network are converted into an ordered list of relaxed axioms. These axioms contain special generated features that act as hooks for a group of aliasing rules that have the job of expressing the information present in the original axioms but absent from their relaxed counterparts.

The first stage of checking involves successive inspection of the relaxed axioms for applicability to the expression being checked. If the right-hand side of an axiom matches the expression then the left-hand side of the axiom is combined to form an extension of the expression. Because the compilation step has organized the axioms in order of depth this algorithm is able to make all legitimate deductions from the relaxed form of the network, and also to fail if the expression given is outlawed by this relaxed form.

In the second stage of checking we use the output of the first stage, attempting to discharge any generated features that have been introduced. In the worst case this will involve polynomial expense, since the problem is that of simultaneously satisfying a number of constraints each of which may individually be satisfiable in a number of different ways.

Although the difficulty of this constraint satisfaction task depends both on the structure of the network and on the details of the expression being checked, we can make a certain amount of progress with this task even in the compilation step. This is because the structure of the network may tell us that the constraint satisfaction problem can be decomposed into subtasks that will never interact whatever the nature of the input expression. By inspection of a system network we can derive a metric that indicates the possible computational cost of applying the information contained in that network, as well as an indication of how much of that information can be recovered without getting involved in the costly search processes that can be occasioned by the presence of disjunctive systems.

3.4 Representation

This section deals with the way in which expressions and axioms are represented for use in the checking algorithm. As mentioned previously this presentation is somewhat abstract, and describes idealized data structures rather than their somewhat more intricate realizations.

For the network in Figure 8 the original set of axioms is

$$e \equiv a1 \vee a2 \tag{16}$$

$$e \equiv b1 \vee b2 \tag{17}$$

$$a2 \vee b1 \equiv c1 \vee c2 \tag{18}$$

and the checking rules relevant to the first stage are

$$e \equiv a1 \vee a2 \tag{19}$$

$$e \equiv b1 \vee b2 \tag{20}$$

$$e \wedge genfeat1 \equiv c1 \vee c2 \tag{21}$$

3.4.1 Representation of Expressions. All boolean expressions can be reduced to the form of a disjunction of conjunctions of (possibly negated) atomic symbols. The expressions to be checked will consist either of simple conjunctions of unnegated atomic symbols, or, if we need to support subsumption checking, conjunctions of a single negated atomic symbol with a number of unnegated symbols. The algorithm will in fact handle any number of negated atomic symbols, but we do not make any use of this capability. An appropriate representation of such expressions is as a vector of slots capable of taking three values along each dimension. Each of the N slots corresponds to one and only one of the N atomic symbols in the set of axioms, and the order of the slots is the order of increasing depth given by the depth relationships in the network. We shall illustrate these vectors using strings of symbol specifications. Within such a string there are three possible types of symbol specification:

- the presence of a positive specification for a symbol is represented using a boldface capital letter ($\mathbf{A_1}$)
- the presence of a negated specification for a symbol is represented with a boldface capital letter with a line above ($\overline{\mathbf{A_1}}$)
- the absence of any specification for a symbol is represented using a lighter lower case font (a_1). Explicit representation of absent symbols means that every symbol in the repertoire must appear (in some form) in every vector.

For the example network in Figure 8 the correspondence between expressions and their representations is as follows.

$$a_1 \wedge b_2$$

is represented by

$$(c_2, c_1, B_2, b_1, a_2, A_1, e)$$

and

$$\neg c_1$$

is represented by

$$(c_2, \overline{C_1}, b_2, b_1, a_2, a_1, e)$$

The elements of a symbol vector are implicitly conjoined, and there is no mechanism for expressing disjunction.

3.4.2 Representation of Axioms. The right-hand sides of axioms consist of ranges of atomic symbols having neighboring depths. In principle one can represent these ranges by specifying a starting depth and a number of symbols, or one can use a slightly more intuitive representation as vectors of implicitly disjoined feature specifications. The individual feature specifications would then be represented in the way used above for those in target expressions.

For the purposes of the algorithm all we actually need to know is the range of depths over which the mutually exclusive alternatives in the axiom extend, since we are primarily interested in checking how many members the right-hand side of an axiom shares with the target expression.

The left-hand sides of axioms are more problematic, requiring the representation of arbitrary boolean expressions in disjunctive normal form. However, for the simplified sets of axioms in which we are primarily interested, this form reduces to a single conjunction of atomic symbols, which can readily be represented using the scheme for target expressions given above.

Examples of axioms and their representation follow.

$$e \equiv a_1 \vee a_2$$

is represented by

$$(c_2, c_1, b_2, b_1, a_2, a_1, e) \equiv (c_2, c_1, b_2, b_1, A_2, A_1, e)$$

while

$$a_2 \vee b_1 \equiv c_1 \vee c_2$$

is represented by

$$(c_2, c_1, b_2, b_1, A_2, a_1, e) \vee (c_2, c_1, b_2, B_1, a_2, a_1, e) \equiv (C_2, C_1, b_2, b_1, a_2, a_1, e)$$

In order to convert these axioms into nondisjunctive equivalents we transform the disjunctive axiom by generating a new feature for its left-hand side, giving

$$e \wedge genfeat1 \equiv c_1 \vee c_2$$

which is represented by an eight-element vector (because we have introduced an extra feature name, and need an extra slot). The axiom becomes

$$(c_2, c_1, b_2, b_1, a_2, a_1, GENFEAT1, E) \equiv (C_2, C_1, b_2, b_1, a_2, a_1, genfeat1, e)$$

This illustrates the fact that the representations of expressions in the first and second stage of checking differ slightly because of the introduction of generated features.

The checking algorithm organizes the axioms into a list of decreasing depth. The axiom at the head of this list (initially, the deepest axiom of all) is the current axiom. In the set of axioms given above the appropriate order would be axiom 21, axiom 20, then axiom 19.

3.5 The First Stage of Checking

The current axiom a is applicable to the target expression E if E contains a positive occurrence of at least one of the elements of $RHS(a)$. In this case we have to distinguish between the case where the intersection contains exactly one element and that where it contains more than one. In the latter case we have a violation of the exclusivity axiom associated with the accessibility axiom that we have explicitly represented. The target expression is therefore inconsistent with the network from which the axioms were derived.

If a is applicable and does not lead directly to an inconsistency we need to insert the material from $LHS(a)$ into E, forming a new expression E'. This is fairly straightforward for networks lacking disjunction in the left-hand side of axioms. The substitution can

fail if the target expression already includes a negated specification for any of the features contained in the left-hand side of the axiom, but otherwise the substitution results in the generation of an expression E', which is simply the conjunction of E and $LHS(a)$. At this stage we do not need to check for mutually exclusive features, since this kind of failure will be detected in a later iteration of the procedure.

If $LHS(a)$ is a disjunction containing multiple elements, as can happen if we are dealing with a full system network, rather than a relaxed version that has been transformed to remove the disjunction, then the algorithm requires the setting up of as many parallel processes as there are elements in the disjunction, as well as extensive copying of those parts of the target expression that may yet be modified. Each process may attempt to write a different combination of features in its own personal copy of the relevant parts of the target expression, and it is crucial that the activities of the separate processes, whether or not they actually run in parallel, be prevented from interfering with one another, as would happen if they modified a common copy of the same data structure. This is obviously the expensive part of the checking procedure, since there can be a very considerable number of quasi-parallel or parallel processes if we happen to be dealing with an awkward case.

3.6 The Cost of the First Stage of Checking

It is possible to code up the algorithm for the first stage of checking as an algorithm operating on a two-tape Turing machine, in which one input tape represents the axioms used and the other represents the expression to be checked. Analysis of this algorithm reveals that it is the substitution step that will prove most costly. It turns out that the possible space needed to represent the axiom tape is proportional to the square of the number of atomic labels in the set of axioms, and that the Turing machine needs no more than three passes over this tape to complete the necessary substitutions. In any event both time and space bounds for the first stage of checking are certainly polynomial in the number of labels in the network being checked.[6]

3.7 The Second Stage of Checking

The second stage of checking has the job of re-imposing the constraints that were ignored in the first stage. By inserting generated features, we have effectively marked the locations at which extra material may have to be introduced. While it would obviously work to simply throw away the results of the first stage of checking, and re-check the original expression (using the original axioms) from scratch, it is more satisfactory to turn the work we have already done to our advantage. This section describes one way of doing this. The parsing system adopts a different, but in essence equivalent, approach, building PROLOG terms as the output of the first stage, then compiling the alias rules into clauses of a procedure that attempts to find a consistent substitution for generated features.

In Winograd's pronoun network (shown in Figure 1) the first stage of checking for the description

$$subjective \wedge singular \tag{22}$$

would yield

$$subjective \wedge singular \wedge genfeat1 \wedge genfeat2 \wedge pn \tag{23}$$

6 Although we introduce a potentially large number of extra labels in the transformation from disjunctive to nondisjunctive networks, these can be omitted without affecting the correctness of the first stage of checking. The labels are introduced as a convenience for use during the second stage of checking, rather than as an essential part of the first stage. In any case, as pointed out by an anonymous referee, the number of these labels is bounded by the number of choice systems in the network as a whole.

which contains two generated features,[7] one corresponding to the expression

$$genfeat1 \equiv question \vee personal \tag{24}$$

and the other to

$$genfeat2 \equiv personal \vee demonstr. \tag{25}$$

In this case it should be obvious that all we have to do is choose compatible values from the disjunctive parts of expressions 24 and 25, and that this is achieved by choosing `personal` in both cases. This has no further consequences. In this simple case we find that the fullest description we can be sure of on the basis of what we know is:

$$subjective \wedge singular \wedge personal \wedge pn. \tag{26}$$

Unfortunately, looking for noncontradictory choices of substitution for generated features will not always gain us the information which we need to know. We illustrate this with reference to Figure 10, which provides a network for the pronoun system of an imaginary language closely related to English.

The path leading from pn to the choice between `first`, `second`, and `third` involves passing through an extra *choice* system encoding the alternation between `honorific` and `familiar`, so we would have inserted `familiar` rather than `personal`. The diagram abbreviates `personal`, `honorific`, and `familiar` to `pers`, `hon`, and `fam` respectively. In the imaginary language in question the choice of an honorific makes it unnecessary to consider further features of the pronoun. Translating the network produces

$$subjective \wedge singular \wedge familiar \wedge pn \tag{27}$$

We now need to know whether a pronoun that is `familiar` is necessarily also **personal**, which involves the re-use of the axiom,

$$personal \equiv honorific \vee familiar \tag{28}$$

and we also need

$$pn \equiv question \vee personal \vee demonstr$$

to ensure that the ill-formed

$$\ldots familiar \wedge question \ldots$$

does not get accepted.

In the worst case it actually turns out that nearly all the axioms present in the original network can be needed for the second stage of checking. In more typical networks, such as the one constructed by glueing together the networks from Houghton's dialog generation application, only a few axioms will be involved in the second stage of checking, and this subset of the axioms can be picked out as a preprocessing step, which we will shortly describe.

7 The pn in expression 23 arises because the left-hand sides of the transformed axioms are made from the conjunction of the generated feature and the deepest enclosing features, rather than simply the generated features on their own.

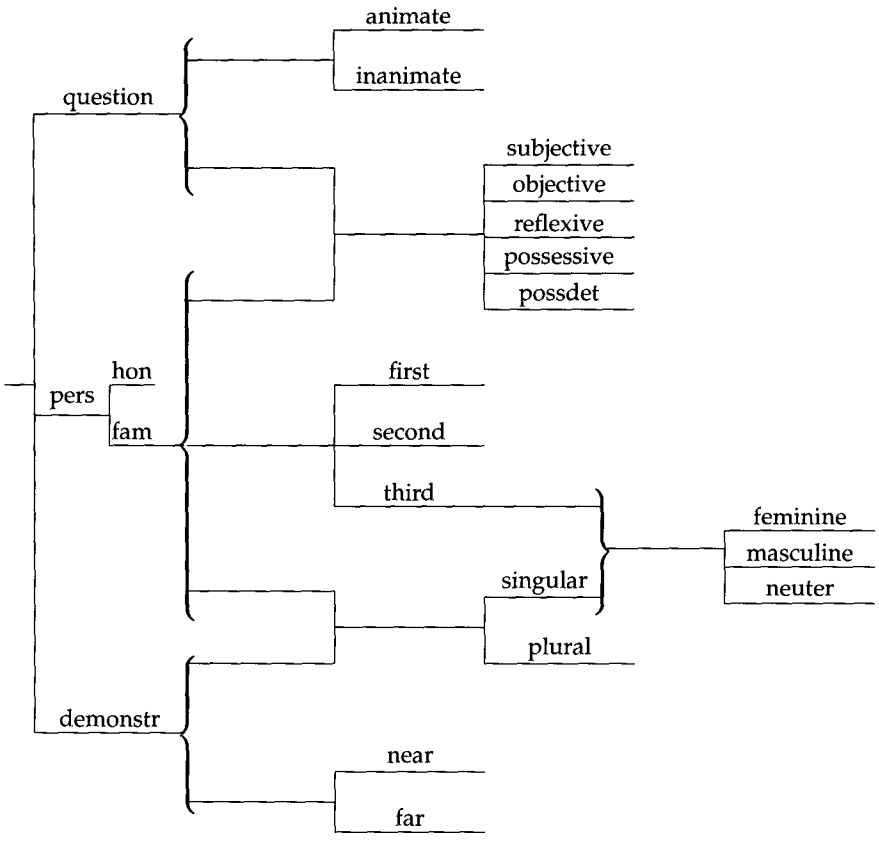

Figure 10
The pronouns of an imaginary language.

3.8 Islands of Uncertainty

One way of viewing the transformation that originally removed disjunction is as a device for temporarily allowing an island of uncertainty to persist in part of the network. The job of the second stage of checking is to deal with this uncertainty, and the potential complexity arises from the possibility that many interacting axioms contribute to the uncertainty. Often there will be several small and independent islands of uncertainty, each involving only a few axioms, and it will be possible to decompose the checking task into more manageable portions.

We now show how to characterize the subsets of axioms involved in each island of uncertainty. The starting point for this operation is the original network, not the transformed version containing generated features. The relevant portion of the network for any given generated feature is that falling between the corresponding *disjunctive* system and the deepest common predecessors of the left-hand lines of that system. Anything nearer to the root than that will have been checked in the first stage of checking. So in order to pick out the potentially relevant axioms for each generated feature, we form a list of labels starting with those involved in the left-hand side of the *disjunctive* system and working back to the deepest common predecessors, which

do not need to be included. This list is then used to choose axioms; every axiom that has a right-hand side containing one of the features on the list must be included.

In the case of Winograd's network (Figure 5), the list for the *disjunctive* systems is

[question,personal,demonstr]

and the only relevant axiom is

$$pn \equiv question \vee personal \vee demonstr \tag{29}$$

For the imaginary language introduced above we would construct the list

[familiar,question,personal,demonstr]

and we therefore need the axiom

$$personal \equiv familar \vee honorific \tag{30}$$

as well as axiom 29.

For every generated feature in the network we collect a set of relevant axioms. Ideally the sets of axioms will be disjoint, which means that the islands of uncertainty induced by the disjunctive left-hand sides can be handled independently. In fact the examples we are using represent the opposite case, in which different generated features are associated with nondisjoint (actually identical) sets of axioms. Our method for carrying out the substitutions collects nondisjoint sets of axioms into larger sets. The procedure operates on structures of the form

$$I_i = \{E_i, A_i\}$$

where E_i is a set of expressions relating generated features to disjunctive left-hand sides, and A_i is the union of the sets of axioms found to be relevant to these left-hand sides. The sets of axioms A_i must be disjoint. Each I_i can therefore be processed separately.[8]

We construct ways of substituting for generated features by replacing each generated feature with an element of the corresponding disjunctive left-hand side. The number of possible substitutions depends both on the number of generated features in the network and on the way they interact. The more sharing there is between the sets of axioms corresponding to the generated features the greater will be the number of distinct ways of substituting for the generated features.

In the first example we have to consider the following possibilities:

$$genfeat1 = question \quad genfeat2 = personal \tag{31}$$
$$genfeat1 = question \quad genfeat2 = demonstr \tag{32}$$
$$genfeat1 = personal \quad genfeat2 = personal \tag{33}$$
$$genfeat1 = personal \quad genfeat2 = demonstr \tag{34}$$

of which only the expression 33 satisfies the exclusivity axiom for axiom 29. Since there are no other relevant axioms either for *genfeat*1 or for *genfeat*2 the check is complete.

8 There may be more efficient algorithms than this, but we have not yet found the need to explore them.

In the second example, axiom 29 does not do quite as much for us, because the possible combinations of substitutions are:

$$genfeat1 = question \quad genfeat2 = familiar \tag{35}$$
$$genfeat1 = question \quad genfeat2 = demonstr \tag{36}$$
$$genfeat1 = familiar \quad genfeat2 = familiar \tag{37}$$
$$genfeat1 = familiar \quad genfeat2 = demonstr \tag{38}$$

only one of which (expression 36) can be eliminated by checking against the exclusivity axioms. Instead we find ourselves checking

$$subjective \wedge singular \wedge familiar \wedge question \wedge pn \tag{39}$$
$$subjective \wedge singular \wedge familiar \wedge pn \tag{40}$$
$$subjective \wedge singular \wedge familiar \wedge demonstr \wedge pn \tag{41}$$

against the axioms. In fact the presence of `familiar` combines with axiom 30 to demand the addition of `personal`. `Demonstr` and `question` are incompatible with `personal` so only the expression produced from expression 40 can satisfy axiom 29. The full expansion of this expression is

$$subjective \wedge singular \wedge familiar \wedge personal \wedge pn \tag{42}$$

which is the required answer.

It may seem unnecessarily elaborate to separate out the effects of disjunctive axioms, but a major reason for doing so is that the partitioning of the set of axioms that is involved can be carried out as a compilation step, which both prepares the axioms for the simpler first stage of checking and gives us an informal measure of the extent to which we are going to be able to decompose the potentially expensive second stage of the problem into small and independent problems.

In the set of axioms coming from the network in Figure 6, we find that every introduced feature interacts with every other one, producing a compile time clue that if we are unlucky we shall have to check an unreasonably large number of possible ways of carrying out the substitution. If a situation like this arose in a natural language application we should probably begin to look for a way of redesigning the network, and consider whether the costly accuracy produced by the second stage of checking actually helps our application much or at least start to suspect that the application should not be expected to produce reliable real-time response.

For Houghton's networks it turns out that the second stage of checking is rarely needed at all, so we have implemented only a simple version of the second stage of checking. In this implementation we make the pessimistic assumption that all generated features may interact. The more sophisticated approach outlined above will certainly produce significant improvements for some networks, but we do not currently have access to a large corpus of independently constructed networks, so cannot tell for sure whether the effort of implementing the slightly trickier algorithm would in fact be justified by linguistic practice.

4. Conclusions

This paper has explored the potential of system networks as a classification tool for linguists. The results of this enterprise can be divided into a group of technical results about networks as such and a collection of less formal ideas about the practical implications of these results for the working computational linguist.

The technical results come first.

1. The problem of systemic classification is an inherently hard one, for which computational tractability cannot be guaranteed.
2. The source of the intractability is the use of *disjunctive* systems in the networks. Restricting the algorithm to use only *choice, and,* and *conjunctive* systems gains us tractability at the expense of some loss of expressive power. Mellish (1988) reports an application in which this route was taken with satisfactory results. It is not yet clear whether the restriction to nondisjunctive networks will be acceptable as a basis for large natural language grammars.

Note that the networks are not the only possible source of disjunction in a large natural language system, since when realization rules are taken into account the choice systems of the system networks can stand for disjunctions of general feature structures.[9] This objection carries most force in the context of a purely constraint based system like Kasper's, since in our hybrid parser each line of analysis is committed to a particular choice of realization rule before feature matching is necessary. Effectively the control regime of the nondeterministic parser is dealing with some of the search that is necessary in order to resolve the disjunctions implicit in the grammar. The chart-parsing part of this involves only a number of feature-matching operations polynomial in the size of the grammar and the length of the input string,[10] so the complexity of the overall parsing process hinges on the efficiency of the feature-matching algorithm involved. The actual performance of the parser may be much better than that indicated by the theoretical properties of the algorithm, but if so the efficiency depends upon contingent properties of the grammar used, so it would be unwise to provide a definitive guarantee that a parser will maintain acceptable performance as the grammar is developed and extended.

For practical purposes the following results are probably of more interest.

1. A given system network can be compiled into a set of axioms without reference to the particular expressions to be checked with these axioms. Although this process is potentially expensive, it can be carried out off-line.
2. The potentially expensive part of systemic classification can be separated from a cheaper first stage. The first stage is comparable in cost to taxonomic classification, and can be implemented efficiently using current technology. The second stage may turn out to be exponentially expensive in the worst case, but for some applications, such as the parsers we have implemented, the second stage proves unnecessary. The issue of how the algorithms would perform on yet larger grammars is as yet unexplored.
3. Even the second stage can often be rendered more manageable by the decomposition of the algorithm into two stages, since the 'islands' of potential intractability tend to be isolated from one another, and can

9 This was pointed out to us by one of the *Computational Linguistics* referees.
10 These results, and the reasons for treating them with some caution, are covered by Barton, Berwick, and Ristad (1988), particularly in Chapters 7 and 8.

often be treated as independent subproblems. In the worst case axioms *are* interdependent, and general algorithms for the unification of disjunctive constraints, such as the ones of Kasper (1987b) and Eisele and Dörre (1988) are likely to out-perform the techniques described here. We see our algorithms more as illustrations of the nature of the problem than as competitors of the more sophisticated algorithms presented by others.

4. The representations used by our algorithms are close to those present in the networks themselves, so systems built on our algorithms should be easier to debug than those in which the networks are compiled into low-level formalisms. As a by-product of the production of these representations we can obtain an informal measure of the likely cost of using a particular set of networks.

We have provided a preliminary analysis of the nature of the information contained in system networks, indicated why the flexible use of this information is likely to be costly, and sketched techniques that will somewhat alleviate the undesirable consequences of this situation. While we are unable to provide the linguist with the desired guarantee of computational tractability in the general case, we have provided tools and techniques that will aid linguists and implementors in the production of efficient grammars built in the systemic formalism.

References

Barton, G. Edward; Berwick, Robert C.; and Ristad, Eric Sven. (1988). *Computational Complexity and Natural Language.* Cambridge, MA: The MIT Press.

Brachman, Ronald J., and Schmolze, James G. (1985). "An overview of the KL-ONE knowledge representation system." *Cognitive Science,* **9**(2):171–216.

Davey, Anthony. (1978). *Discourse Production: A Computer Model of Some Aspects of a Speaker.* Edinburgh, U.K.: Edinburgh University Press.

Eisele, Andreas, and Dörre, Jochen. (1988). "Unification of disjunctive feature descriptions." In *Proceedings, 25th Annual Meeting of the Association for Computational Linguistics.* Buffalo, NY.

Halliday, M. A. K. (1976). "The form of a functional grammar." In *Halliday: System and Function in Language,* edited by G. R. Kress, 7–25. Oxford, U.K.: Oxford University Press.

Halliday, M. A. K. (1975). *Learning How to Mean — Explorations in the Development of Language.* London: Edward Arnold.

Houghton, George A. (1986). "The production of language in dialogue: A computational model." Doctoral dissertation, University of Sussex.

Houghton, George A., and Isard, Stephen D. (1987). "Why to speak, what to say and how to say it: modelling language production in discourse." In *Modelling Cognition,* edited by P. Morris, 249–267, New York: John Wiley.

Hudson, Richard A. (1971). *English Complex Sentences.* Dordrecht: North Holland.

Kaczmarek, Thomas S.; Bates, Raymond; and Robins, Gabriel. (1986). "Recent developments in NIKL." In *Proceedings, 5th National Conference of the American Association for Artificial Intelligence.* Philadelphia, PA.

Kasper, Robert T. (1988). "An experimental parser for systemic grammars." In *Proceedings, 12th International Conference on Computational Linguistics (COLING 88).* Budapest.

Kasper, Robert T. (1987). "Feature structures: A logical theory with application to language analysis." Doctoral dissertation, University of Michigan.

Kasper, Robert T. (1987). "A unification method for disjunctive feature descriptions." In *Proceedings, 25th Annual Meeting of the Association for Computational Linguistics.* Stanford, CA.

von Luck, Kai; Nebel, Bernhard; Peltason, Christoph; and Schmiedel, Albrecht. (1986). "BACK to consistency and incompleteness." In *GWAI-85: 9th German Workshop on Artificial Intelligence,* edited by H. Stoyan, 245–257, Berlin: Springer Verlag.

MacGregor, Robert, and Bates, Raymond. (1987). "The loom knowledge representation language." Technical

Report ISI/RS-87-188 Information Sciences Institute, University of Southern California.

Mann, William C., and Matthiessen, Christian. (1985). "Nigel: A systemic grammar for text generation." In *Systemic Perspectives on Discourse*, edited by R. O. Freedle, Ablex.

McCord, Michael. (1977). "Procedural systemic grammars." *International Journal of Man-Machine Studies*, **9**(3):255–286.

Mellish, C. S. (1988). "Implementing systemic classification by unification." *Computational Linguistics*, **14**(1):40–51.

Nebel, Bernhard. (1990). "Reasoning and Revision in Hybrid Representation Systems." Number 422 in *Lecture Notes in Artificial Intelligence*. Berlin: Springer Verlag.

Patten, Terry, and Ritchie, Graeme. (1986). A Formal Model of Systemic Grammar. Research Paper 290, Department of A.I., Edinburgh University.

Pereira, Fernando A., and Shieber, Stuart M. (1987). *Prolog and Natural Language Analysis*. CSLI Lecture Notes No. 10. Center for the Study of Language and Information, Stanford, CA.

Power, Richard. (1979). The organization of purposeful dialogue. *Linguistics*. **17**(1–2):107–152.

Reynolds, John C. (1970). "Transformational systems and the algebraic structure of atomic formulae." In *Machine Intelligence*, Volume 5, edited by Bernard Meltzer and Donald Michie. Edinburgh, U.K.: Edinburgh University Press.

Shieber, Stuart M. (1986). *An Introduction to Unification Based Approaches to Grammar*. CSLI Lecture Notes, Number 4. Center for the Study of Language and Information, Stanford, CA.

Winograd, Terry. (1972). *Understanding Natural Language*. New York: Academic Press.

Winograd, Terry. (1983). *Language as a Cognitive Process: Volume 1: Syntax*. Reading, MA: Addison-Wesley.