Implementing Systemic Classification by Unification

C. S. Mellish

School of Cognitive Sciences, University of Sussex[1]

The "system networks" of Systemic Grammar provide a notation for declaring how combinations of properties may imply or be inconsistent with other combinations. Partial information about a linguistic entity can be recorded as a set of known properties, and a system network then enables one to infer which other properties follow from this and which other properties are incompatible with this. The possible descriptions allowed by a system network are partially ordered by the relationship of *subsumption*, where a description subsumes any description that is more specific than it, given the background constraints declared by the network. Given this partial ordering, the set of descriptions can be seen as forming a lattice with least upper bound and greatest lower bound operations. In a class of applications (such as parsing and generation) that require *incremental description refinement*, we are only really interested in forming new conjunctions (greatest lower bounds) and testing subsumption relationships.

If one factors out the complexity of variable renaming and introduces special top and bottom elements, the set of logical terms also forms a lattice (the lattice of Generalised Atomic Formulae –"GAF lattice") under the partial ordering relation "is equally or more instantiated than" (Reynolds (1970)). In this lattice, the greatest lower bound operation is unification (Robinson (1965)). Unification is a primitive operation in most logic programming systems and is also the basis of various grammatical formalisms. It is therefore a relatively well understood operation and can be efficiently implemented.

In this paper, we investigate to what extent it is possible to find structure-preserving mappings from the description spaces defined by system networks to sublattices of the GAF lattice. Where this is possible, we can use a fixed mapping from property names to logical terms to create terms that represent conjunctive descriptions (by unification) and to test subsumption (by testing "less instantiated than"). Incompatibility of descriptions is also indicated by unification failure. There are a number of reasons why it is interesting to investigate these possibilities:

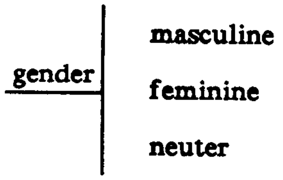
(1) They may result in more efficient or flexible ways of using system networks for inferencing.
(2) They may enable language analysers or generators which involve unification for other reasons (e.g., analysers for GPSG (Gazdar et al. (1985)) or Functional Unification Grammar (Kay (1984)) to build in feature co-occurrence tests using the same mechanism.
(3) They may enable us to make formal sense of various ad-hoc mechanisms used by logic programmers in natural language processing.
(4) By exposing the nature of the relevant description spaces, they may open various possibilities for the implementation of other classification tasks, e.g., concept learning (Mellish forthcoming).
(5) They may give us more insight into the semantics of system networks and the potential of unification.

1 Halliday's System Networks

System networks, as used in Halliday's Systemic Grammar (Hudson (1971), Kress (1976), Winograd (1983)) are a way of encoding the choices that must be made in the generation of a complex linguistic object and the interdependencies between them. There is actually nothing that makes such networks specific to linguistic applications, and so there is no reason why they cannot be applied to describing the choices involved in other complex situations.

A system network can be viewed as a graph, some of whose nodes are annotated with symbols representing properties. The nodes are tied together by the use of four different "connectives", which we shall designate by "|", "{", "}" and "]". In order to be precise about

0362-613X/ 88 /010040-51$03.00



exactly what system networks mean, we will present a logical interpretation, where each appearance of a "connective" in a network gives rise to a set of logical axioms relating the property symbols (interpreted as unary predicates) appearing with it.

A fundamental concept in system networks is that of the choice system. A choice system indicates that, if a certain "entry condition" holds, then the object described must have exactly one of the properties mentioned in the system. Choice systems are denoted by use of the "|" "connective". Thus, Figure 1 indicates that masculine, feminine and neuter are mutually exclusive and whenever an object has a gender it has one of these. In logic,

$\forall$ x: AMO {feminine(x) masculine(x) neuter(x)}
$\forall$ x: gender(x) $\equiv$ feminine(x) v masculine(x) v neuter(x)

where AMO ("at most one of") is defined by:

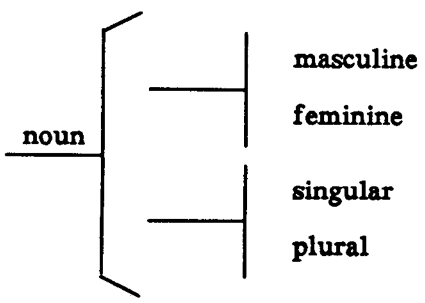
$$\text{AMO S} == \bigwedge_{\substack{s_1,s_2 \in S \\ s_1 \neq s_2}} \neg(s_1 \ \& \ s_2)$$

Incidentally, an alternative reading that might suggest itself, namely:

$\forall$ x: gender(x) $\equiv$ (AMO {feminine(x) masculine(x) neuter(x)} & feminine(x) v masculine(x) v neuter(x))

is not adequate, because it allows spurious models, for instance where there is an object "a" which satisfies "feminine(a)" and "masculine(a)" but not "gender (a)".

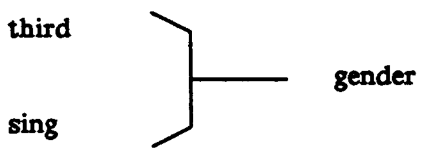
Sometimes more than one choice will be relevant, given the same entry conditions. This is indicated by the "{" "connective". For instance, as indicated in Figure 2, in some languages a noun may be either singular or plural, and also either masculine or feminine. Instances of the "{" connective can be translated into logic by



simply treating the entry condition of the "{" as that of all the networks introduced on the right hand side. Thus:

$\forall$ x: AMO {feminine(x) masculine(x)}
$\forall$ x: noun(x) $\equiv$ feminine(x) v masculine(x)
$\forall$ x: AMO {singular(x) plural(x)}
$\forall$x: noun(x) $\equiv$ singular(x) v plural(x)

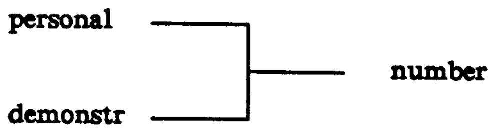
The final two connectives concern complex entry conditions into networks. A conjunctive entry condition is denoted by "}", as shown in Figure 3. This means



simply:

$\forall$ x: third(x) & sing(x) $\equiv$ gender(x)

Finally, "]" introduces a disjunctive entry condition, so that the example provided in Figure 4



means:

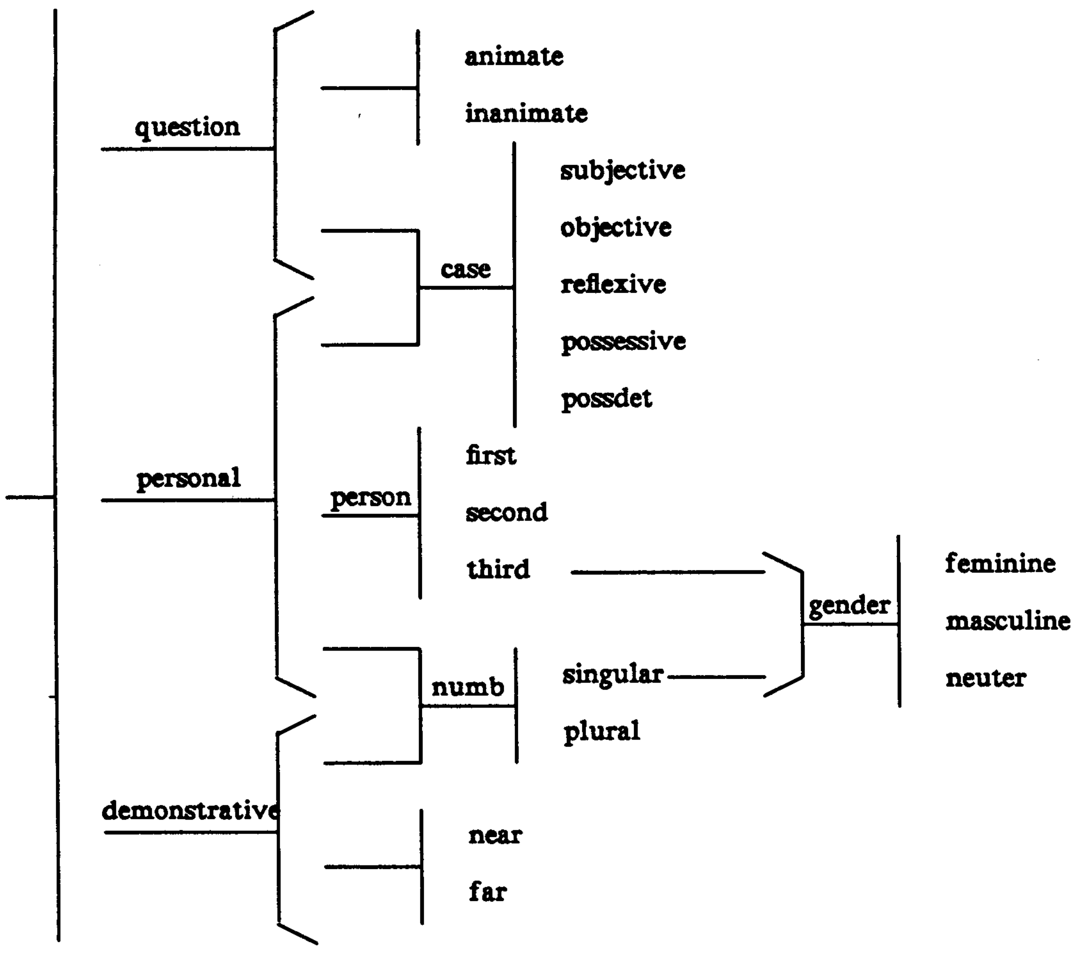
$\forall$ x: personal(x) v demonstr(x) $\equiv$ number(x)

By convention, uses of the four "connectives" can be connected together in any way, as long as "loops" are not created. That is, if one regards each connective as a set of arcs going from properties on its "left" to properties on its "right", the resulting directed graph must be acyclic.

As a larger example, Figure 5 depicts the system network for English pronouns presented in Winograd (1983)). Here is a logical translation of selected parts.

$\forall$ x: AMO {first(x) second(x) third(x)}
$\forall$ x: person(x) $\equiv$ first(x) v second(x) v third(x)
$\forall$ x: AMO {singular(x) plural(x)}
$\forall$ x: numb(x) $\equiv$ singular(x) v plural(x)
$\forall$ x: third(x) & singular(x) $\equiv$ gender(x)

It is important to note that in this paper we consider system networks as a self-contained notation for describing certain types of choices ("systemic choices") that are available in the construction of a complex (linguistic) object. We will be completely ignoring the philosophical differences between Systemic Grammar and other forms of generative grammar, and we will also completely ignore the other components that are required in a full Systemic Grammar, such as realisation rules.



2 Subsumption and the Lattice of Descriptions

The property symbols in a system network provide a basic vocabulary out of which descriptions can be built. The most obvious way to produce more complex descriptions is by conjunction and disjunction. The logical interpretation of such complex descriptions is straightforward, and we will often blur the distinction between a description and its interpretation. Thus:

masculine & singular corresponds to
  λx. masculine(x) & singular(x)
masculine v feminine corresponds to
  λx. masculine(x) v feminine(x)

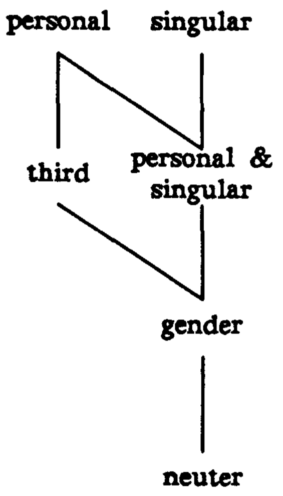
A fundamental relationship between descriptions is **subsumption**:

d_1 subsumes d_2 iff $\Sigma \models \forall$ x: $d_2(x) \supset d_1(x)$

where Σ is the set of axioms derived from the network. Note that our notion of subsumption depends vitally on Σ. This is a special case of what Buntine (1986) calls "generalised subsumption". Intuitively d_1 subsumes d_2 if, given the axioms Σ, d_1 is a more general description than d_2. That is, if d_1 describes all the objects accurately described by d_2 and maybe more. Subsumption is a partial ordering on descriptions, and the set of possible descriptions (properties and all possible finite conjunctions and disjunctions of descriptions made from them), ordered by subsumption, forms a lattice. In this lattice, the least upper bound of two descriptions is their disjunction and the greatest lower bound is their conjunction. Figure 6 is a picture of a portion of the lattice consisting of the descriptions derived from the pronoun network. In the picture, if there is a line going upwards from description d_2 to description d_1 then d_1 subsumes d_2. Two descriptions that are logically equivalent (e.g., "personal&singular" is the same as "singular&case") give rise to a single node in the diagram (technically, we are interested in the quotient lattice of the free lattice generated by the property symbols, with respect to the congruence relation defined by Σ). To find the node for the conjunction of two descriptions, one finds the highest node that is "below" both, i.e., the greatest lower bound. Similarly, to find the node for the disjunction of two descriptions, one finds the lowest node which is "above" both.

3 Incremental Description Refinement

The previous two sections introduced a simple language of descriptions and the use of system networks to express extra background information about (constraints on) the terms appearing in those descriptions. A notion of subsumption was defined which allowed this



background information to be taken into account. But what are the operations that we need to carry out on descriptions in practical natural language processing systems, and does the structure we have described support these?

In this paper, we will concentrate especially on a process that seems to arise in a number of contexts in natural language processing—**incremental description refinement**. Incremental description refinement (IDR) takes place when a **target description** is gradually being built of some individual, and information about this individual appears as a sequence of self-contained, independent **data descriptions**. For instance, the target could be the description of an English sentence and the data descriptions partial descriptions of this sentence like:

the sentence is passive
the sentence is declarative
the agent of the sentence is the speaker

At any point in an IDR process, the information that has accumulated so far may allow certain properties of the individual to be inferred, and so one would like to be able to interrogate the partial description that has been built. In particular, one would like to be able to answer questions about which descriptions are compatible and incompatible with the target description. To build an effective IDR system, one must have a way to represent the conjunction of an arbitrary set of pieces of information so that inconsistency and subsumption relationships with other descriptions can be easily detected. The term "incremental description refinement" was, we believe, originally coined by Bobrow and Webber (1980), but the notion of incrementally building descriptions has been influential in a number of AI projects.

In natural language processing, IDR is relevant to both natural language parsing and generation. In parsing, it is natural to accumulate information about the structure of a phrase gradually as words are read. For instance, in the sentence

The hairy sheep was . . .

we know after reading the first three words that the gender of the subject noun phrase is "neuter" and after the next word we know that the number of that phrase is "singular". It is important in parsing that we be able to accumulate pieces of information of this kind and detect inconsistencies if they arise. In generation, it is natural to want to allow different semantic and pragmatic factors to provide separate constraints on a sentence to be generated. For instance, one pragmatic goal may force a sentence to be passive; another forces it to have a given surface subject. This conjunction of constraints may be inconsistent with certain choices of the main verb (e.g., "buy" vs. "sell"). Again there is a need to reason about partial descriptions that are built incrementally.

In formal terms, the operations involved in IDR are simple. At any point, the information known about the target description can be represented without loss by a single "partial description—the least upper bound of all the descriptions the target could be. Initially this is simply the most general description of all ("true"). When a new data description appears, the partial description is replaced by the greatest lower bound of it and the data description. This "algorithm" for IDR is in fact a special case of a more general classification algorithm given in Mellish (forthcoming). At any point, a contradiction is signaled by the partial description becoming the most specific description of all ("false"). Moreover, one can validly infer that the target is subsumed by a given description if the partial description is. The only operations that we need for IDR are subsumption checking and the computation of greatest lower bounds. This means that, in fact, we do not need the full lattice structure developed above—all we need is the meet semi-lattice (Birkhoff (1963)) that contains the possible data descriptions and all possible conjunctions of them.

The above description of IDR is not dependent on descriptions being related to system networks, and indeed IDR has been used in quite different contexts. In this paper, however, we will confine ourselves to this case, and consider IDR where the data descriptions are precisely the properties mentioned in a system network. We are thus concerned with ways of computing and testing subsumption between conjunctions of properties, given the background information provided by the network axioms.

4 Using Logical Terms to Encode Subsumption Relationships

The set of terms used in logic is partially ordered by the relation ("at least as instantiated as") where:

$t_1 \leq t_2$ iff
$t_1 = t_2\theta$ for some substitution θ

Thus for instance, the following statements are true:

f(y,z) ≤ x
f(a,x) ≤ f(y,z)
f(f(x)) ≤ f(y)

Just as we formed a lattice from descriptions, collapsing together two descriptions that were logically equivalent, so we can form a lattice from logical terms, collapsing together terms which are "variants" (i.e., terms t_1 and t_2 such that $t_1 \leq t_2$ and $t_2 \leq t_1$). This lattice, with a necessarily slightly altered version of ≤ defined on it and with special "top" and "bottom" terms added, is the lattice of Generalised Atomic Formulae ("GAF lattice") discussed by Reynolds (1970). In this lattice, the greatest lower bound operation is unification Robinson (1965), although for various sublattices the actual operation may be simpler.

The main point of this paper is to investigate when a structure-preserving mapping (1-1, 0-preserving meet homomorphism) can be found from the description lattice arising from a system network to the GAF lattice. Where this is possible, we can use the mapping from property names to logical terms to make terms that correspond to conjunctive descriptions (by unification) and to test subsumption (by testing "less instantiated than"). Incompatibility of descriptions is also indicated by unification failure.

We will initially illustrate the idea of encoding properties as logical terms by an example. Let us assume that we are given the above system network for English pronouns. We might come up with a mapping τ from properties mentioned in the network to logical terms which includes the following assignments (variables whose names are of no importance in the logical terms are denoted here by the symbol "_"):

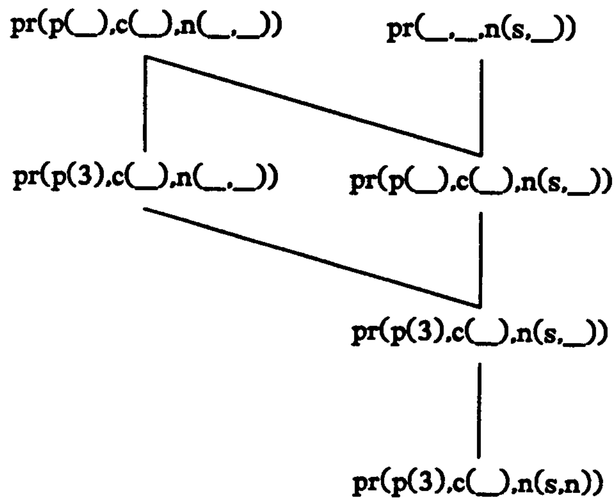
τ(animate)	= pr(q(an),c(_),no)
τ(case)	= pr(_,c(_),_)
τ(far)	= pr(d(far),no,n(_,_))
τ(gender)	= pr(p(3),c(_),n(s,_))
τ(neuter)	= pr(p(3),c(_),n(s,n))
τ(numb)	= pr(_,_,n(_,_))
τ(personal)	= pr(p(_),c(_),n(_,_))
τ(reflexive)	= pr(_,c(refl),_)
τ(singular)	= pr(_,_))
τ(third)	= pr(p(3),c(_),n(_,_))

This mapping is not purely random, but has been chosen so that the logical relationship of subsumption is "echoed" in the "degree of instantiation" of the terms. So the idea of the mapping τ is to have:

p_1 subsumes p_2 iff $\tau(p_1) \geq \tau(p_2)$

Thus, for instance, "gender" subsumes "neuter" and is translated into a term which is almost identical, but which is slightly less instantiated. The situation is similar with "case" and "reflexive". On the other hand, the terms from "reflexive" and "third" do not disagree on any non-variable component but neither is more instantiated than the other. This reflects the fact that the two properties are compatible but neither is more general than the other. In the case of the terms from "singular", "third" and "gender", the "gender" term is the least instantiated term that is more instantiated than each of the other two (it is the result of unifying the other two). This reflects the fact that "gender" is equivalent to the logical conjunction of "third" and "singular".

Figure 7 shows a portion of the GAF lattice whose structure mirrors the part of the pronoun description space shown in section 2. In the GAF lattice, one object is below another if it is more instantiated (less free), and the greatest lower bound is computed by unification. If



we can find a mapping such as τ, we can use operations in the GAF lattice to solve problems in the original description lattice. For instance, if we have an object which is "neuter" we might be interested to see whether it can also be "far". This is not possible, and is indicated by the fact that the two terms

pr(p(3),c(_),n(s,n))	(from "neuter")
pr(d(far),no,n(_,_))	(from "far")

do not unify (the conjunction is the "bottom", or "false" property). Similarly, we can establish that if an object is "singular" and "far" then it cannot be "animate". For, unifying the terms for "singular" and "far" we get:

pr(d(far),no,n(s,_))

which does not unify with the term for "animate". Using τ, we can also verify that "reflexive&gender" definitely implies "personal", that "singular" is compatible with "reflexive", and so on. It is worth noting that these inferences involving compatibility and properties that are not "maximally delicate" go beyond those allowed by previous systems such as Patten (1986).

This seems to work well, but is there a principled way to produce such mappings into the GAF lattice that is guaranteed to yield correct results? For correctness, we need the following to be true for all possible properties $p_1, p_2, \ldots p_{n+1}$ named in the system network:

$$\Sigma \models \forall x\text{: } p_1(x) \ \&\ p_2(x) \ \&\ldots\&\ p_n(x) \supset p_{n+1}(x)$$
$$\text{iff } \tau(p_1) \text{ II } \tau(p_2) \text{ II } \ldots \tau(p_n) \leq \tau(p_{n+1})$$
$$\Sigma \models \forall x\text{: } p_1(x) \ \&\ p_2(x) \ \&\ldots\&\ p_n(x) = F$$
$$\text{iff } \{\tau(p_1)\tau(p_2) \ldots \tau(p_n)\} \text{ is not unifiable}$$

where Σ is the set of logical axioms derived from the network and II is unification (greatest lower bound in the GAF lattice). A sufficient condition for this would be for τ to be a 1-1, 0-preserving meet-homomorphism (Birkhoff (1963)).

In fact, the above mapping does not echo in the GAF lattice the result:

$$\Sigma \ \models \forall \text{ x: case(x) \& numb(x)} \supset \text{personal(x)}$$

and cannot be straightforwardly extended to do so. We thus need to investigate under what conditions such mappings exist and what algorithms might enable us to discover them.

5 A "Brute Force" Translation

That such mappings always exist, and that inefficient algorithms exist for discovering them, is demonstrated by a more general result. If we have a finite set of propositional symbols and an arbitrary set of axioms mentioning these symbols then there is a method of encoding those symbols as logical terms so that conjunctions are computed by unification and subsumption between conjunctive descriptions (relative to the axioms) is reflected in "degree of instantiation". We can apply this result to the system networks domain because the logical axioms, although not strictly propositional, are equivalent to propositional axioms if we only consider one object being described at a time.

The way to construct this encoding is to consider all possible models of the axioms and the truth assignments made to the symbols in them. In the case of the pronouns network, there are 54 models of the corresponding logical axioms. The following example shows the truth assignments made by one of them (this corresponds to the pronoun "he").

animate = F,	case = T,	demonstrative = F,	far = F,
feminine = F,	first = F,	gender = T,	inanimate = F,
masculine = T,	near = F,	neuter = F,	numb = T,
objective = F,	person = T,	personal = T,	plural = F,
possessive = F,	possdet = F,	pronoun = T,	question = F,
reflexive = F,	second = F,	singular = T,	subjective = T,
third = T			

If there are n possible models, each property is represented by a logical term of arity n + 1. Each such term also has the same function symbol, the constant "0" as its first argument and the constant "1" as its last argument. The other arguments, for a given property, are to be derived as follows. Each argument starts off as a distinct variable, and then, for each model i which assigns "F" to the property, the ith argument is unified with the i + 1st argument. In a situation where there are five possible models, the following example includes some of the terms that might result (again we use "_" for variables with uninteresting names).

```
      1   2   3   4   5
      |   |   |   |   |
f(0,  _,  _,  X,  X,  1)  (property excludes only model 4)
f(0,  0,  _,  X,  X,  1)  (property excludes models 1 and 4)
f(0,  X,  X,  X,  1,  1)  (property excludes models 2, 3, 5)
```

Each such term represents the models that the property excludes by the instantiation of its arguments. When two such terms are unified, the result is a term that encodes exclusion of the union of the models excluded by the terms individually. If the two terms together exclude all possible models, the unification fails (this occurs if we attempt to unify the last two of the above terms). This is as desired, since excluding all models amounts to incompatibility of descriptions. The technique used here, which amounts to encoding sets as terms, was, we believe, first developed by Alain Colmerauer in another context. In fact, we can make do with only n arguments.

In general, given a set of n property symbols, there can be as many as 2^n different models, and so this "brute force" approach is unattractive. In a situation where there are very few models of the logical axioms, however, the encoding scheme might be quite practical. For instance, for the pronoun network the number of models (54) is much smaller than the worst case for the same number of properties (2^{25}). The theoretical maximum number of models is reduced by a factor of roughly $2^r/r$ for every "|" connective with r symbols on its right appearing in the system network. Moreover, the theoretical maximum is reduced by a factor of roughly 2 for each property appearing on the right hand side of a "{". The encoding technique is analogous to using bit strings to encode sets, the only advantage being that a contradictory conjunction is flagged immediately by a unification failure. On the other hand, the terms make use of repeated variables, and so the potential for exploiting parallelism in the unifications is restricted.

System networks do not, however, correspond to arbitrary sets of logical axioms. Only certain kinds of axioms can come out of a system network (although a useful characterisation eludes us). We might therefore hope that, given the extra restrictions, there are better encoding techniques available, in particular encoding techniques that do not require the use of repeated variables. We will therefore investigate whether structure-preserving mappings can be found from systemic descriptions to elements of the lattice of generalised atomic formulae without repeated variables, which we call GAF_0.

6 Encoding for the Connectives "|", and "{"

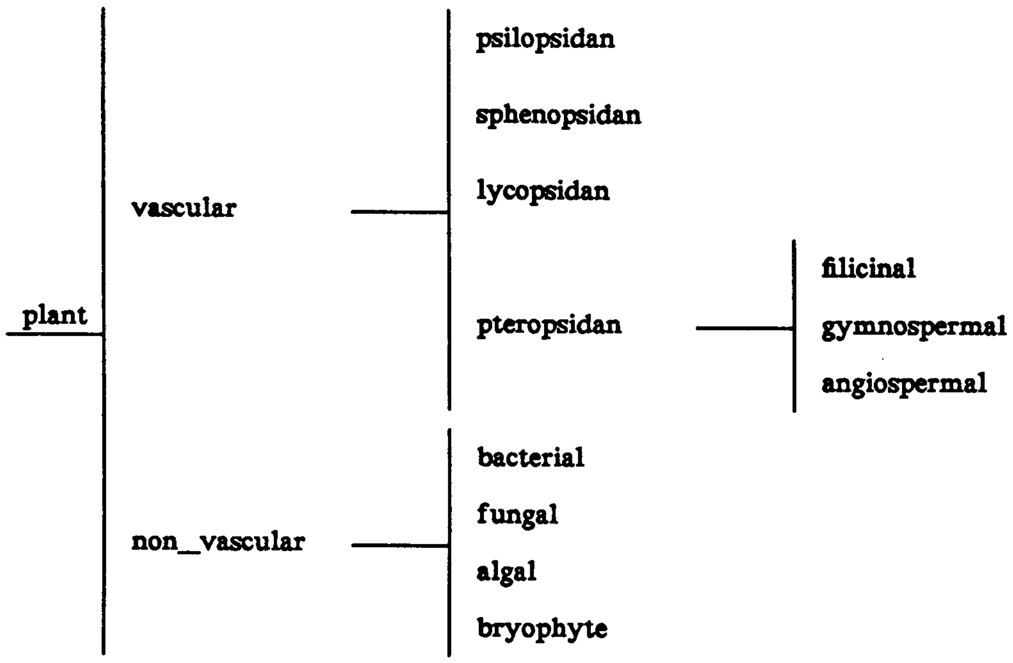
If the only connective used in a system network is "|", a particularly simple encoding scheme is feasible. This technique has been used in an ad hoc way in a number of language processing systems written in Prolog, and probably originates from Dahl (1977). Consider the example network depicted in Figure 8, expressing the top levels of classification normally used for the plant kingdom. For such a simple network, we can use nesting as a means of capturing the subsumption relationships in logical terms. Thus each property, apart from properties at the "leaves" of the tree, is associated with a unary function symbol, and the argument of a function symbol is used for a more "fine grained" description if that is available:

τ(plant)	= plant(_)
τ(vascular)	= plant(vascular(_))
τ(pteropsidan)	= plant(vascular(pteropsidan(_)))
τ(angiospermal)	= plant(vascular(pteropsidan(angiospermal)))

Where there are alternatives at a given level in the classification, these are indicated simply by different function symbols appearing in the relevant argument positions. Thus, for instance:

τ(non_vascular)	= plant(non_vascular(_))
τ(sphenopsidan)	= plant(vascular(sphenopsidan))
τ(filicinal)	= plant(vascular(pteropsidan(filicinal)))
τ(fungal)	= plant(non_vascular(fungal))

If the "{" connective is also allowed in system networks, this means that a given property may have refinements along several independent dimensions. In the logical terms, this can be allowed for by giving the function symbols more than one argument. Such a system has been used by McCord (1986). Figure 9 depicts a version of the "verb" network used by Winograd (1972), simplified to use only "|" and "{". One possible translation from the above property symbols into terms includes the following:



τ(vb)	= vb(_,_)
τ(vprt)	= vb(vprt,_)
τ(aux)	= vb(aux(_,_,_),_)
τ(neg)	= vb(aux(neg,_,_),_)
τ(be)	= vb(aux(_,_,be),_)
τ(pres)	= vb(_,pres)

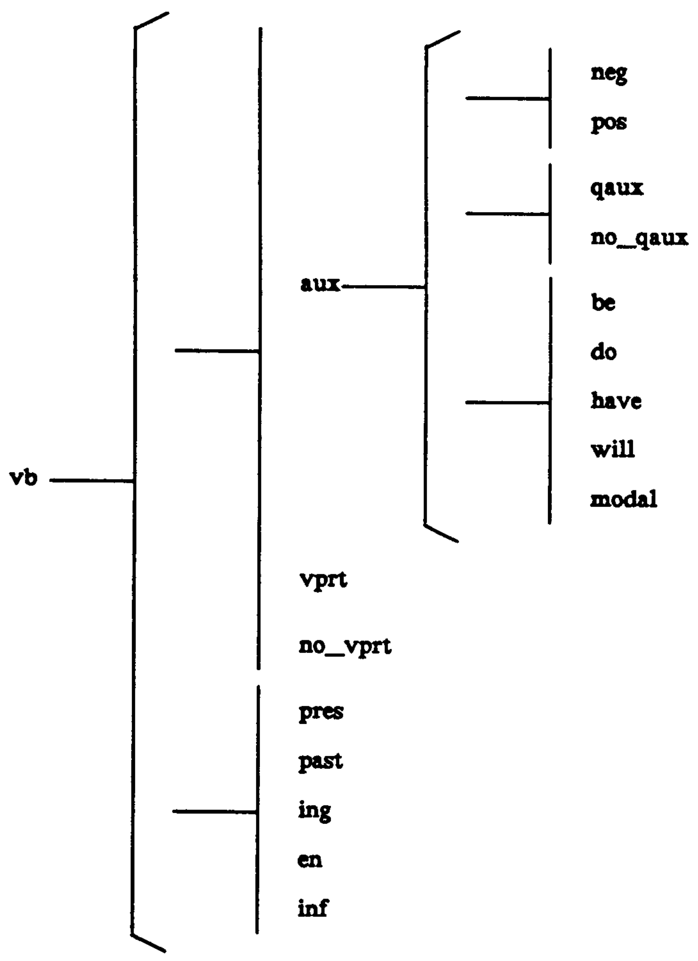
Notice how the "aux" function symbol has arity 3; this corresponds to the three independent sub-classifications introduced by the "{" connective. Similarly, "vb" has arity 2. The translation from symbols to terms is again fairly straightforward.

Instead of going into detail about the relatively simple problems of dealing with system networks containing only "|" and "{" connectives, we shall concentrate on an algorithm which also allows "}" connectives. This algorithm was used in essence by Bundy et al. (1982), although it has not been described before.

7 Introducing "}"

In this section we describe a generalised translation algorithm which associates elements of GAF_0 with properties displayed in a system network. The algorithm works for networks using the connectives "|", "{" and "}", but does not handle networks that use "]".

For this algorithm, we require in advance a function Λ which associates with each node n of the system



network (a possibly labeled location corresponding to one or more "ends" of connectives) a function symbol Λ (n). Λ is required to be 1-1, except that it should map all the nodes appearing around a "{" connective to the same function symbol. Such a function is easy to define for a given network; for instance, most nodes can be simply mapped to the names associated with them in the network. The translation algorithm associates with each node of the network a pair ⟨C, P⟩. The values for C and P are:

C = a set of "constraints" describing the term
P = a "path" indicating a (possibly embedded) component of the term which can become further instantiated for terms representing properties subsumed by the current property

A **path** is simply a sequence of alternating function symbols and numbers, either empty or starting with a function symbol, indicating a specific position in a term and the function symbols that appear on the route from the outside of the term to this position. A symbol in the sequence indicates a function symbol that is present, whereas a number selects one of the argument positions of the last function symbol. For instance, in

f(a, g(X,h(b),i(d,e,j(c))))

the symbol "c" appears at the place indicated by the path ⟨f,2,g,3,i,3,j,1⟩ and the symbol "b" at the place indicated by ⟨f,2,g,2,h,1⟩. We will make use of two basic types of **extensions** of a path P:

```
an extension of P to f, for some function symbol f:
    If P is ⟨⟩ then ⟨f⟩
    Otherwise if P is ⟨$p_1$, $p_2$, . . . i⟩ for some number i
        then ⟨$p_1$, $p_2$, . . . i,f⟩
    Otherwise if P is ⟨$p_1$, $p_2$, . . . s⟩ for some symbol s
        then ⟨$p_1$, $p_2$, . . . s,i,f⟩ for some number i
an extension of P beyond f, for some function symbol f:
    If P is ⟨⟩ then ⟨f,i⟩ for some number i
    Otherwise if P is ⟨$p_1$, $p_2$, . . . i⟩ for some number i
        then ⟨$p_1$, $p_2$, . . . i,f,j⟩ for some number j
    Otherwise if P is ⟨$p_1$, $p_2$, . . . s⟩ for some symbol s
        then ⟨$p_1$, $p_2$, . . . s,i⟩ for some number i
```

For instance, for the path ⟨f,2,g⟩, two possible extensions *to* h are ⟨f,2,g,33,h⟩ and ⟨f,2,g,1,h⟩. For the path ⟨f,1,g,2⟩ two possible extensions *beyond* h are ⟨f,1,g,2,h,4⟩ and ⟨f,1,g,2,h,6⟩. Finally, we will have two ways of describing paths that differ, according to the type of the first component where they disagree:

P1 and P2 are **independent** iff their first disagreement is on a number
P1 and P2 are **inconsistent** iff their first disagreement is on a function symbol

Thus ⟨f,2,g,3,h⟩ is independent of ⟨f,2,g,4,d,3⟩, and ⟨f,2,g,5,a⟩ is inconsistent with ⟨f,2,h,4⟩.

A **constraint** is a path used to specify that particular function symbols must appear at particular places in a term. A term is correctly described by a constraint if the path makes sense (i.e., all the relevant components exist) and the given function symbols do indeed appear at the relevant places. A term is correctly described by a set of constraints if it is correctly described by all of them. Thus, for instance, the set of constraints:

{ ⟨g,1,f,2,a⟩, ⟨g,1,f,3,b⟩, ⟨g,1,f⟩, ⟨g⟩ }

correctly describes any of the terms:

g(f(z,a,b(c)),h(1))
g(f(w(x),a(d,f,g),b,x))
g(f(w(x),a(g),b),x)

The term that is the translation of a node in a system network is obtained from the set of constraints as follows. First of all, each function symbol (element of the range of Λ) is taken to have the same arity (number of arguments) in all the terms derived from the network. This is the minimal arity such that all the constraints attached to all the nodes are satisfiable. Secondly, with arities fixed in this way, the translation of a property is the most general (least instantiated) term that is correctly described by the constraints at the property's node. Thus, for example, if the arities of a, b, f and g were 1, 0, 3 and 2 respectively (note that the arity of f must be at least 3), the following term would be the "solution" of the above set of constraints:

g(f(_,a(_),b),_)

The following defines a space of possible translation algorithms. For each place where alternatives are allowed, it does not matter which the algorithm specifies.

(1) Record ⟨{⟨f⟩},⟨f⟩⟩ as the value for the leftmost node, where f is the value of Λ for that node
(2) Until there is no connective all of whose left nodes have translations but none of whose right nodes do, do the following:
(2.1) Select one such connective
(2.2) If the connective is "|" and the left hand node has translation ⟨C,P⟩, assign to the ith node n_i on the right hand side the translation:

$\langle C \cup \{p_i\}, p_i\rangle$

where, for each i, p_i is an extension of P to Λ (n_i), and where p_i is inconsistent with p_j if $i \neq j$.

If the connective is "{" and the left hand node L has translation ⟨C, P⟩, assign to the ith node on the right hand side the translation:

$\langle C, p_i\rangle$

where, for each i, p_i is an extension of P beyond Λ (L) and where p_i and p_j are independent if $i \neq j$.

If the connective is "}" and the two left hand nodes have translations $\langle C_1, P_1\rangle$ and $\langle C_2, P_2\rangle$, assign to the node on the right hand side the translation:

$\langle C_1 \cup C_2, P_i\rangle$

with either $i = 1$ or $i = 2$.

The basic idea is that as one goes to the right in the network, the constraints on a node are the constraints on the nodes to its immediate left, together with possibly extra constraints to differentiate it from them (if they properly subsume it). At each point, the path component of a node's translation indicates where nodes further to the right can be further instantiated. If the connective is "|", the extra constraints added for each node on the right amount to forcing the place indicated by the path to have the function symbol associated with that node. Since the paths used for the right hand nodes are inconsistent, the terms generated for the nodes will be incompatible. If the connective is "{", the nodes on the right hand side must have the same constraints as the node on the left. On the other hand, each is given a different, slightly longer, path so that the term can be further instantiated in several independent ways.

The above algorithm, although not optimal in the sense of generating the smallest possible terms, is the basis of one that has generated correct results in practice. In addition, we have a proof of correctness for it. The proof hinges on the reduction of the correctness criterion given in section 4 to the following simpler one. The translation τ is correct if for all properties p_1, p_2 and p_3 named in the network:

$\Sigma \models \forall$ x: $p_1(x) \supset p_2(x)$
iff $\tau(p_1) \leq \tau(p_2)$
$\Sigma \models \forall$ x: $p_1(x) \supset p_2(x)$
iff $\Sigma \models p_2 = p_4 \& p_5$ for some p_4,p_5 in the network with p_2, p_4 and p_5 all logically distinct
or $\Lambda(p_2)$ appears in $\tau(p_1)$
$\Sigma \models \forall$ x: $p_1(x)$ & $p_2(x)$ = F
iff $\{\tau(p_1)\tau(p_2)\}$ is not unifiable
$\Sigma \models \forall$ x: $p_1(x)$ & $p_2(x) = p_3(x)$
iff $\tau(p_1)$ II $\tau(p_2) = \tau(p_3)$

for some function Λ is introduced above. The reduction can be made because of various special properties that hold of system networks that only use "|", "{" and "}":

If $\Sigma \models \forall$ x: $p_1(x) = p_2(x) \& p_3(x)$, for properties p_1, p_2 and p_3 mentioned in the network and logically distinct, then p_1 is equivalent to a property on the right hand side of a "}" connective, and p_2 and p_3 are equivalent to the properties appearing on the left hand side.

If $\Sigma \models \forall$ x: $p_1(x) \& p_2(x) \& . . . p_n(x) \supset$ p(x), for properties p_1, . . . p_n, p mentioned in the network, then either $\Sigma \models \forall$ x: $p_i(x) \supset$ p(x) for some i, or p is equivalent to a property appearing on the right of a "}".

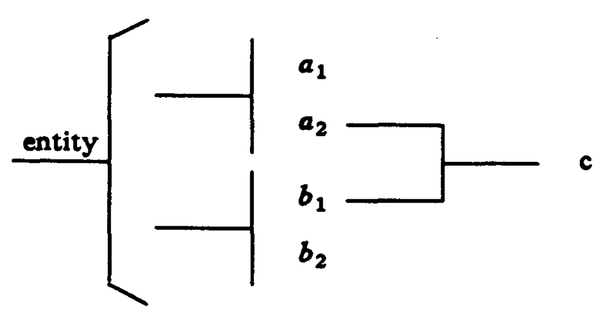
If P is a set of properties mentioned in the network and for every "|" connective either no elements of P lie to the right of that connective or all elements lie to the right of the same right hand branch, then the set P is compatible.

None of the schemes in this section or the previous section require repeated variables in the logical terms, and so there is nothing to prevent unification tackling different components in parallel. Of course, if one had a great deal of parallelism available, one could use it to implement a naive network searcher for testing compatibility of two properties (e.g., by searching for the rightmost node that appears to the left of both and seeing whether the connective at that point is "|". The implementation via unification would then be unnecessary (at least for simple cases). Thus to a certain extent the "compilation" of tasks like compatibility- and subsumption-checking into unification tasks can be viewed as a special case of compiling OR-parallel programs into AND-parallel programs (Codish and Shapiro (1986), Ueda (1986)). The checking can be even faster if the unifications are further compiled (Warren (1977)). For instance, in the verb network, checking that a description of a verb is compatible with "pres" simply amounts to seeing whether the third component of the term is not a function symbol different from "pres".

8 Introducing "]"

We have intentionally left consideration of the "]" connective to its own section. This is because we have no general algorithm (apart from the above "brute force" algorithm) for translating system networks with "]" connectives into logical terms in a way that will correctly mirror the subsumption relationships. Indeed, there is no reason to believe that there are straightforward algorithms for this.

First of all, we can easily demonstrate through an example that networks containing "]"'s cannot be implemented using only GAF_0. Thus, something like the full power of unification is necessary, rather than the simple subcases we have considered up to now. Consider the system network depicted in Figure 10. The



subsumption relationships between properties and conjunctions of properties are shown in Figure 11. This

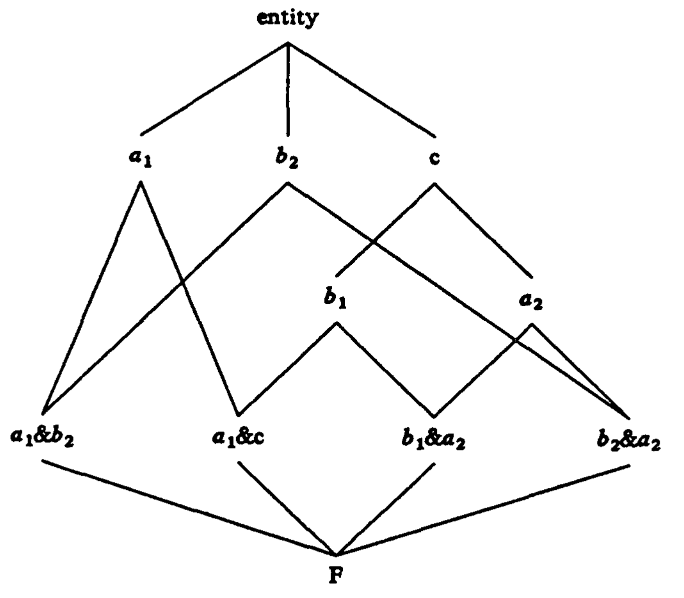
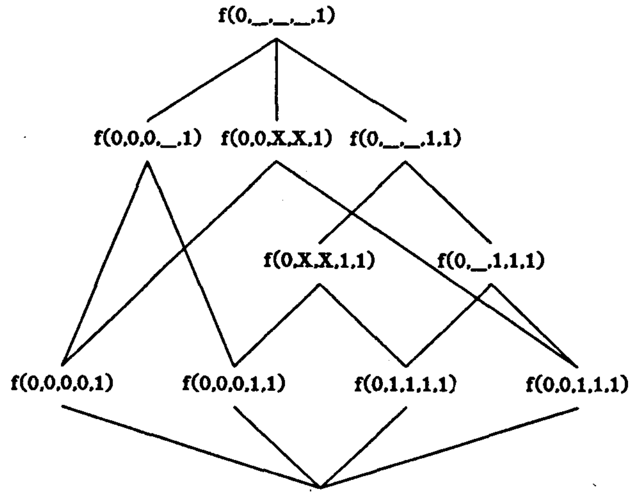


diagram must be read with some care, as we have only shown the elements of the meet-semilattice consisting of the data elements and their conjunctions. This is all that we need for an IDR application, but it means that, although we can read off from the diagram information about greatest lower bounds, we cannot obtain from it information about least upper bounds. Now $a_1\&c$ and $b_1\&a_2$ are incompatible, i.e., their greatest lower bound is "F". Therefore the terms representing these two descriptions cannot unify. If we are not allowing terms with repeated variables, the only way that two terms can fail to unify is by there being some component where the two terms take on different non-variable values. Let $a_1\&c$ take the value "y" in this component and $b_1\&a_2$ take the value "n" (we do not have to assume that "y" and "n" are atomic, merely that their "top level" function symbols are different). We can then make various inferences about what other terms should have in this component:

$a_1\&c$	– y	(hypothesis)
$b_1\&a_2$	– n	(hypothesis)
b_1	–_	since it is above $a_1\&c$ and $b_1\&a_2$
c	–_	since it is above b_1
a_2	– n	so as to make $b_1\&a_2$ when unified with b_1
$b_2\&a_2$	– n	since it is below a_2
a_1	– y	so to make $a_1\&c$ when unified with c
$a_1\&b_2$	– y	since it is below a_1
b_2	–_	since it is above $b_2\&a_2$ and $a_1\&b_2$

Now $b_2\&a_2$, as well as being the greatest lower bound of b_2 and a_2, is also the greatest lower bound of b_2 and c. However, unifying the terms for b_2 and c, the above specifies that the particular component will have the value "_". This conflicts with the value already inferred for $b_2\&a_2$ –"n". Therefore, under the constraint of avoiding repeated variables, it is not possible to produce a valid mapping from descriptions to terms.

Figure 12 shows the subsumption relationships again, with the descriptions substituted by logical terms obtained by the "brute force" method. Of course, repeated variables have to be used several times.

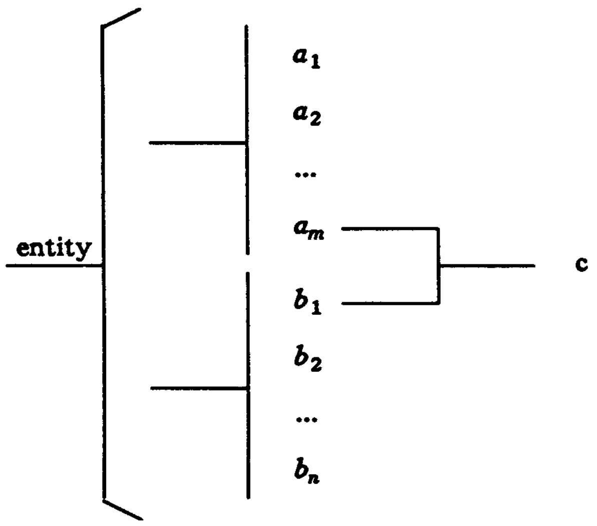


It is natural to ask what it is about descriptions arising from networks with "]" that cannot be modelled in GAF_0. There is at least one property of GAF_0 (and any sub-semilattice of it that contains the bottom element) which is not echoed in description spaces like our examples. Since descriptions from "]" networks can violate this property, it follows that GAF_0 cannot be used accurately to model the description spaces. The particular property is that compatibility is **pairwise determined**:

For all sets of elements $\{t_1, t_2, \ldots t_n\}$
$\{t_1, t_2, \ldots t_n\}$ compatible
iff t_i is compatible with t_j for each i, j.

(we say that a set of objects is compatible if their greatest lower bound is not the "bottom" element). In our example description space, the set $\{a_1, b_2, c\}$ is not compatible, and yet each pair of elements taken from it is compatible. Thus compatibility is not determined pairwise in the description space.

It seems unlikely in this example that one could come up with an encoding in the full GAF that would improve on the "brute force" approach. Indeed, this may often be the case, as networks containing "]"s are very complex to process. Consider, for instance, Figure 13 which is an extension to the above network.



In choosing logical terms for $a_1, a_2, \ldots a_m, b_1, b_2, \ldots b_n$, one has to reflect the fact that

c & a_i	implies b_1	$(i = 1..m\text{-}1)$
c & b_i	implies a_m	$(i = 2..n)$
a_i & b_j & c	is contradictory	$(i = 1..m\text{-}1, j = 2..n)$

This certainly prevents the sub-classifications represented by the "a"s and the "b"s from being translated independently, in the way that could be done in our portion of Winograd's verb network.

9 Related Work

A piece of work that is closely related in many ways is Kasper's work (Kasper (1986)) on translating Systemic Grammars into functional unification grammars. Kasper describes a way of mapping a system network and feature choices into a functional description of FUG (Kay (1984)). Such a mapping induces a mapping from systemic descriptions to functional descriptions in such a way that conjunction of descriptions is computed by (functional) unification of the functional descriptions. Kasper's work represents an extension of ours in that it treats more than the systemic choices of a Systemic Grammar, and we are currently investigating how our framework can best be extended to deal with these other aspects. Unfortunately, the functional description that Kasper builds for anything but a complete (maximally specific) systemic description will contain disjunctions, and the complexity of functional unification is seriously affected by the presence of disjunctions. Moreover, Kasper actually needs to extend the notion of functional description in order to produce translations for networks with complex entry conditions. Since logical unification corresponds closely to functional unification when there are no disjunctive descriptions or ordering patterns, our methods can be viewed as techniques for mapping systemic descriptions into disjunction-free, conventional functional descriptions. Whereas Kasper's primary aim is for a mapping between a systemic *Grammar* and a Functional Unification *Grammar* (with a mapping between individual descriptions induced by this), in our case there is only a description mapping. It is hard to believe that a unification grammar allowing exactly those functional descriptions corresponding to the terms generated by our "brute force" method would be of much interest.

Another related piece of work is that of Gazdar et al., (forthcoming), which seeks to produce a uniform formalism for describing legal categories in grammatical formalisms. In the case of Systemic Grammar, a "category" is a set of feature specifications. The work of Gazdar and his colleagues envisages an implementation of category conjunction by a kind of unification which is similar to logical unification, except that there are no repeated variables. As with Kasper's work, the approach is to provide a translation for the grammar of legal descriptions. In this case, however, one product of the translation is a set of global constraints on legal descriptions. Unfortunately, unification does not preserve the validity of these constraints and is not therefore a correct implementation of greatest lower bound in the subsumption lattice. Our approach can be roughly characterised as an attempt to find alternative translations where unification is correct because the global constraints have been "built in". To do this, unfortunately, we need to allow repeated variables.

Acknowledgements

This work was carried out as part of a project on natural language generation financed by SERC grant GR/D/

08876. I am currently supported by an SERC Advanced Fellowship.

References

Birkhoff, Garrett. 1963 *Lattice Theory*. American Mathematical Society.

Bundy, Alan; Byrd, Lawrence and Mellish, Chris. 1982 Special Purpose, but Domain Independent, Inference Mechanisms. In Steels, Luc and Campbell, John, Eds., *Progress in Artificial Intelligence*. John Wiley, Chichester, UK: 93–111.

Buntine, Wray. 1986 Generalised Subsumption and its Applications to Induction and Redundancy. In Steels, Luc and DuBoulay, Ben, Eds., *Procs of the Seventh European Conference on Artificial Intelligence*. Brighton, 1986.

Codish, Michael and Shapiro, Ehud. 1986 Compiling OR-Parallelism into AND-Parallelism. In Shapiro, Ehud, Ed., *Procs of the Third International Conference on Logic Programming*. Springer Verlag.

Dahl, Veronica. 1977 Un Systeme Deductif d'Interrogation de Banques de Donnes en Espagnol. PhD dissertation, Groupe d'Intelligence Artificielle, University of Marseille-Luminy.

Gazdar, Gerald; Klein, Ewan; Pullum, Geoffrey and Sag, Ivan. 1985 *Generalised Phrase Structure Grammar*. Blackwell.

Gazdar, Gerald; Pullum, Geoffrey; Carpenter, Robert; Klein, Ewan; Hukari, Thomas and Levine, Robert. (to appear) Category Structures. *Computational Linguistics*.

Hudson, Richard A. 1971 *English Complex Sentences: An Introduction to Systemic Grammar*. North Holland.

Kasper, Robert. 1986 Systemic Grammar and Functional Unification Grammar. In Benson, J. and Greaves, W., Eds., *Selected papers from the Twelfth International Systemics Workshop*. Ablex, Norwood, NJ, USA.

Kay, Martin. 1984 Functional Unification Grammar: A Formalism for Machine Translation. In *Procs of COLING-84*. Stanford, July 1984.

Kress, Gunther, Ed. 1976 *Halliday: System and Function in Language*. Oxford University Press.

McCord, Michael C. 1986 Design of a Prolog-based Machine Translation System. In Shapiro, Ehud, Ed., *Procs of the Third International Conference on Logic Programming*. Springer Verlag.

Mellish, Chris S. (forthcoming) Version Spaces, Description Spaces and Unification. Research paper, Dept of Artificial Intelligence, University of Edinburgh.

Patten, Terry A. 1986 Interpreting Systemic Grammar as a Computational Representation: A Problem Solving Approach to Text Generation. PhD thesis, University of Edinburgh.

Reynolds, John C. 1970 Transformational Systems and the Algebraic Structure of Atomic Formulas. In Meltzer, Bernard and Michie, Donald, Eds., *Machine Intelligence 5*. Edinburgh University Press.

Robinson, J. Alan. 1965 A Machine-Oriented Logic Based on the Resolution Principle. *JACM* 12, 23–41.

Ueda, Kazunori. 1986 Making Exhaustive Search Programs Deterministic. In Shapiro, Ehud, Ed., *Procs of the Third International Conference on Logic Programming*. Springer Verlag.

Warren, David H. D. 1977 Implementing Prolog: Compiling Predicate Logic Programs: Research Reports 39 and 40, Department of Artificial Intelligence, University of Edinburgh.

Winograd, Terry. 1972 *Understanding Natural Language*. Academic Press.

Winograd, Terry. 1983 *Language as a Cognitive Process. Volume 1: Syntax*. Addison Wesley.

Notes

1. Author's current address: Department of Artificial Intelligence, University of Edinburgh, 80 South Bridge, EDINBURGH EH1 1HN, Scotland.