# 9 The ACQUILEX LKB: An Introduction

ANN COPESTAKE, ANTONIO SANFILIPPO, TED BRISCOE AND
VALERIA DE PAIVA

## 9.1 Introduction

This chapter and those following describe the LKB, a lexical knowledge base system which has been designed as part of the ACQUILEX project to allow the representation of syntactic and semantic information semi-automatically extracted from machine readable dictionaries (MRDs) on a large scale. An overview of the ACQUILEX project is given by Briscoe (1991).

Although there has been previous work on building lexicons for Natural Language Processing (NLP) systems from MRDs (e.g. Carroll and Grover, 1989), most attempts at extracting semantic information have not made use of a formally defined representation language; typically a semantic network or a frame representation has been suggested, but the interpretation and functionality of the links has been left vague. Several networks based on taxonomies extracted from MRDs have been built (following Amsler, 1980) and these are useful for tasks such as sense-disambiguation, but are not directly utilisable as NLP lexicons. For a lexicon to be genuinely (re)usable, a declarative, formally specified, representation language is essential. A large lexicon has to be highly structured; it is necessary to be able to group lexical entries and to represent relationships between them, both in order to capture linguistic generalisations and to achieve consistency and conciseness. But, unless these notions of structure are properly specified, a lexicon based on them is in danger of being incomprehensible except (perhaps) to its creators. We therefore take semantic structuring seriously, and use taxonomic information as one of the ways of providing such structure, but we do this within the context of a formally specified representation language.

The LKB's knowledge representation language (LRL) can be viewed as an augmentation of a typed graph-based unification formalism with minimal default inheritance; default inheritance is formalised in terms of default unification of feature structures (see e.g. Carpenter, this volume). We chose to use a graph unification based representation language (e.g. Shieber, 1986) for the LKB, because this offered the flexibility to represent syntactic and semantic information, and the interaction between them, in a way which could be easily integrated with much current work on unification grammar, parsing and generation. In contrast to DATR (Evans and Gazdar, 1990) for example, the LRL has not been designed specifically for lexical representation. This made it much easier to

incorporate a parser in the LKB (which is almost essential for developing a type system and for testing lexical entries) and to experiment with notions such as lexical rules and inter-lingual links between lexical entries. Although this means that the LRL is perhaps too general for its main application, the type system provides a way of flexibly constraining the representation according to the particular linguistic treatment adopted.

The main structure of the lexicon is given by the type system. Our typed feature structure language is based on Carpenter's (1990, 1992) work on the HPSG formalism, although there are some significant differences. The type system can be regarded as a way of providing (non-default) inheritance, combined with error-checking. The notion of types, and features appropriate for a given type, gives some of the functionality of frame representation languages, such as KL-ONE; in particular, classification of a feature structure is possible.

We augment the typed feature structure language with a default inheritance mechanism. This can be used to organise the lexicon in a completely user-defined way, to allow morphological or syntactic information to be concisely specified, for example, as has been done with DATR and other systems (for example, Russell *et al.* and Krieger and Nerbonne's chapters in this volume). However much of the motivation behind our formalisation of default inheritance comes from consideration of the sense-disambiguated taxonomies semi-automatically derived from MRDs, which we are using to structure the LKB. The top level of the inheritance structure, which cannot be automatically derived from MRDs, is, in effect, given by the type system.

Thus the operations that the LRL supports are (default) inheritance, (default) unification and lexical rule and translation link application. It does not support any more general forms of inference and is thus designed specifically to support processes which concern lexical rather than general reasoning. The type system provides the non-default inheritance mechanism and constrains default inheritance. We use lexical rules as a further means of structuring the lexicon, in a flexible, user definable manner, but lexical rules are also constrained by the type system.

In the remainder of this introduction we provide an informal account of the type system and other aspects of the LKB including the lexical rule and translation link mechanisms. Other chapters in this volume discuss various aspects of the LRL in more depth and describe two applications in detail. De Paiva discusses the theoretical background to typed feature structures and the way that they are formalised in the LRL. Sanfilippo describes the type system which has been used in the ACQUILEX project to represent lexical entries for verbs, using information extracted from MRDs. Copestake completes the description of the LRL by considering the default unification and default inheritance mechanisms and the way they interact with the type system, and discusses the use of the default system in the representation of taxonomic information extracted from MRDs. Vossen and Copestake continue the discussion of the representation

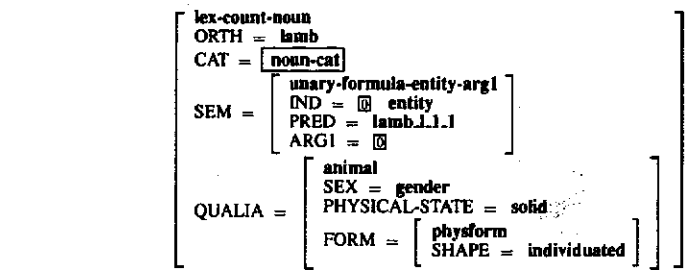



Figure 9.1: Simplified LKB lexical entry for *lamb*

of taxonomic information, and show how more problematic examples may be tackled. Appendix 13.8 is a bibliography of relevant papers produced under ACQUILEX. Appendix 13.8 gives a full description of the syntax of the LKB's feature structure description language.

## 9.2 An Informal Introduction to Typed Feature Structures

The feature structure shown in Figure 9.1 is a (highly simplified) example of a lexical entry in the LRL which illustrates the notational conventions which we will use in this group of chapters. Bold font is used for types, features are capitalised. A box round a type indicates that that portion of the feature structure is not shown. The lexical entry as shown has four components; ORTH is the orthography, CAT the syntactic information (not shown), SEM the formal semantic structure. This essentially corresponds to the expression, $\lambda x[\text{lamb\_L\_1\_1}(x)]$, where the predicate indicates that the sense corresponds to a particular dictionary sense, in this case *lamb*[1] *1* in LDOCE (*Longman Dictionary of Contemporary English*, Procter, 1978). The feature QUALIA introduces the lexical semantic information, the representation of which is loosely based on Pustejovsky's (1989, 1991) notion of qualia structure. The basic structure of the lexical entry is determined by the type system.

The type system can be described as having two components; the type hierarchy and the constraint system. The type hierarchy defines a partial ordering (notated $\sqsubseteq$) on the types and specifies which types are *consistent*. Only feature structures with mutually consistent types can be unified — two types which are unordered in the hierarchy are assumed to be inconsistent unless the user explicitly specifies a common subtype. Every *consistent* set of types $S \subseteq \mathsf{TYPE}$ has a unique greatest lower bound or meet (notation $\sqcap S$). This condition allows feature structures to be typed deterministically — if two feature structures of types **a** and **b** are unified the type of the result will be $\mathbf{a} \sqcap \mathbf{b}$, which must be unique if it exists. If $\mathbf{a} \sqcap \mathbf{b}$ does not exist unification fails. Thus in the fragment of a type hierarchy shown in Figure 9.2 **artifact** and **physical** are consistent; $\mathbf{artifact} \sqcap \mathbf{physical} = \mathbf{artifact\_physical}$. We will use a very simplified type sys-

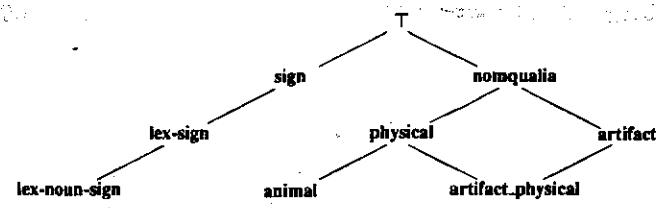



Figure 9.2: A fragment of a type hierarchy

tem in this introduction for ease of exposition; for more realistic type systems see the chapters by Sanfilippo and by Vossen and Copestake in this volume.

Our system differs somewhat from that described by Carpenter (1990, 1992) in that we adopt a different notion of well-formedness of typed feature structures. In our system every type must have exactly one associated feature structure which acts as a constraint on all feature structures of that type, by subsuming all well-formed feature structures of that type. The constraint also defines which features are *appropriate* for a particular type; a well-formed feature structure may only contain appropriate features. Constraints are inherited by all subtypes of a type, but a subtype may introduce new features (which will be inherited as appropriate features by all its subtypes). A constraint on a type is a well-formed feature structure of that type; all constraints must therefore be mutually consistent. Constraints can be seen as extending the PATR-II notion of templates (e.g. Shieber, 1986) in that the inheritance of constraints allows concise definitions of all feature structures, not just lexical entries; but in an untyped system, such as PATR-II, there is no restriction on the features that can occur in a feature structure.

For example the unexpanded constraint associated with the type **artifact** might be:

$$\begin{bmatrix} \textbf{artifact} \\ \text{TELIC} = \textbf{formula} \end{bmatrix}$$

This constraint states that any feature structure of type **artifact** must have a feature structure of type **formula** as the value for its TELIC (purpose) feature. The type **formula** is intended to represent a formula in predicate logic; it therefore would have a complex constraint itself:

$$\begin{bmatrix} \textbf{formula} \\ \text{IND} = \textbf{entity} \\ \text{PRED} = \textbf{logical-pred} \\ \text{ARG1} = \textbf{sem} \end{bmatrix}$$

The full constraint associated with a type is found by expanding the constraints associated with all the types inside the constraint feature structure, thus the expanded constraint for artifact would be:

$$\begin{bmatrix} \textbf{artifact} \\ \text{TELIC} = \begin{bmatrix} \textbf{formula} \\ \text{IND} = \textbf{entity} \\ \text{PRED} = \textbf{logical-pred} \\ \text{ARG1} = \textbf{sem} \end{bmatrix} \end{bmatrix}$$

The type **physical** might have constraint:

$$\begin{bmatrix} \textbf{physical} \\ \text{PHYSICAL-STATE} = \textbf{state} \\ \text{SHAPE} = \textbf{shape} \end{bmatrix}$$

Here **shape** and **state** are both atomic types, and have no appropriate features. For example the constraint on **state** is simply the atomic feature structure [**state**]. The constraint on **artifact_physical** will contain information inherited from both parents, thus:

$$\begin{bmatrix} \textbf{artifact\_physical} \\ \text{PHYSICAL-STATE} = \textbf{state} \\ \text{SHAPE} = \textbf{shape} \\ \text{TELIC} = \begin{bmatrix} \textbf{formula} \\ \text{IND} = \textbf{entity} \\ \text{PRED} = \textbf{logical-pred} \\ \text{ARG1} = \textbf{sem} \end{bmatrix} \end{bmatrix}$$

Given that **solid** is an atomic subtype of **state**, and that **entity** is an atomic subtype of **sem**, the feature structure below is well-formed. It contains all the appropriate features and no inappropriate ones, it is subsumed by the constraints on its type and all its substructures are well-formed.

$$\begin{bmatrix} \textbf{artifact\_physical} \\ \text{PHYSICAL-STATE} = \textbf{solid} \\ \text{SHAPE} = \textbf{shape} \\ \text{TELIC} = \begin{bmatrix} \textbf{formula} \\ \text{IND} = \boxed{0}\ \textbf{entity} \\ \text{PRED} = \textbf{logical-pred} \\ \text{ARG1} = \boxed{0} \end{bmatrix} \end{bmatrix}$$

Since the type system gives us a concept of a well-formed feature structure it follows that non well-formed feature structures can be detected, allowing error checking. This is particularly important for our particular application where mistakes may occur, either because of errors in the original dictionary entries, or because of problems in the automatic extraction processes.

Typing also allows for a form of classification; a feature may only be introduced as appropriate at one point in the type hierarchy (and will be inherited as an appropriate feature by all subtypes of that type); it follows from this that there is a unique maximal type for any set of features, and therefore an untyped feature structure can always be typed deterministically. For example, assuming the type system introduced above, the attribute value specification:

```
< PHYSICAL-STATE > = solid
< TELIC : IND > = < TELIC : ARG1 >
```

would be expanded out into the feature structure just shown. (Full details of the feature structure description language are given in Appendix B.) The type of the feature structure is determined automatically; since the features PHYSICAL-STATE

and TELIC are specified, its type has to be **artifact_physical** (or some subtype of that type).

### *9.2.1 Limitations of Error Checking and Classification*

Error checking and classification with respect to a type system in the LKB are computationally efficient but have limitations. One disadvantage of the system as described is that it is not possible in general to enforce co-occurrence restrictions, even of a quite limited sort. For example, Sanfilippo's representation of verb semantics in the LKB (Sanfilippo, this volume) involves using thematic roles and encoding restrictions on arguments of a predicate by sorting the variables. In order to do this a type **theta-formula** is defined to have the following constraint:

$$\begin{bmatrix} \textbf{theta-formula} \\ \text{IND} = \boxed{1}\ \textbf{eve} \\ \text{PRED} = \textbf{theta-relation} \\ \text{ARG1} = \boxed{1} \\ \text{ARG2} = \textbf{obj} \end{bmatrix}$$

To classify psychological predicates thematic predicates such as **theta-sentient** are used; in this case the second argument to any formula whose predicate is **theta-sentient** should denote a sentient entity; i.e. if the value of PRED is **theta-sentient** then the value of ARG2 is **e-sentient**. But the nearest we could get to achieving this would be to define a subtype of **theta-formula**, e.g. **theta-sentient-formula**, with constraint:

$$\begin{bmatrix} \textbf{theta-sentient-formula} \\ \text{IND} = \boxed{1}\ \textbf{eve} \\ \text{PRED} = \textbf{theta-sentient} \\ \text{ARG1} = \boxed{1} \\ \text{ARG2} = \textbf{e-sentient} \end{bmatrix}$$

and to define other subtypes for the other possible theta relations. This does not really achieve the desired result, however — for example:

$$\begin{bmatrix} \textbf{theta-formula} \\ \text{IND} = \boxed{1}\ \textbf{eve} \\ \text{PRED} = \textbf{theta-sentient} \\ \text{ARG1} = \boxed{1} \\ \text{ARG2} = \textbf{e-plant} \end{bmatrix}$$

is still a well-formed feature structure, despite the fact that it cannot be extended to be a well-formed structure with a type corresponding to that of any leaf node in the type hierarchy (assuming that **e-plant** $\sqcap$ **e-sentient** $= \bot$). This seems undesirable; the type system is supposed to be complete, so intuitively we might expect such a feature structure to be ill-formed in some sense. It seems clear that we cannot check for such cases efficiently in general, because to do so would, in the worst case, involve attempting to unify the feature structure with the constraints of all leaf types which were subtypes of its type.

We refer to a feature structure which can be extended to a well-formed structure where every type is a leaf type as 'ultimately well-formed', and we can

enforce such co-occurrence restrictions when automatically acquiring lexical entries from the MRDs by checking for ultimate well-formedness. This does not impose an unreasonable overhead in practice, since a lexical entry need only be acquired and checked once.

A related issue is that classification of a feature structure with respect to a type system is also limited, in that the procedure only takes account of the top level features in a structure and not their values. Even if the only subtype of **binary-formula** which had a value for PRED which was compatible with **theta-relation** was **theta-formula**, the following feature structure would be classified as a **binary-formula** rather than a **theta-formula**:

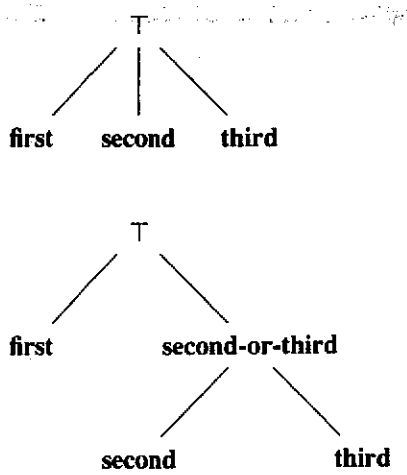
$$\begin{bmatrix} \textbf{top} \\ \text{IND} = \boxed{1}\ \textbf{entity} \\ \text{PRED} = \textbf{theta-relation} \\ \text{ARG1} = \boxed{1} \\ \text{ARG2} = \textbf{obj} \end{bmatrix}$$

Again, although full classification would be expensive computationally, allowing such a procedure to be invoked when lexical entries are being created is a practical option which has considerable advantages in allowing augmentation of automatically acquired information.

### *9.2.2 Extensions to the Language*

A type system in the LKB has, essentially, to be fully defined before lexical entries can be built. This causes obvious problems with respect to atomic types representing orthography and predicate names, for example, where it is unrealistic to assume that the complete set can be known in advance. To get round this we allow any string as a valid LKB type; all strings are assumed to be subtypes of the predefined atomic type **string**, but to be unordered with respect to one another. Particular features such as ORTH, which are specified as having value **string**, thus in effect take arbitrary string values.

Although many feature structure based languages allow disjunctive feature structures, we have avoided this in the LKB. Arbitrary disjunction can result in a computationally intractable system and it is not clear whether it is in fact necessary, given that the type system can be set up in a way which, in effect, allows a constrained form of disjunction. For example, given the types shown in Figure 9.2, rather than stating that a feature had value **animal** or **artifact_physical**, we would state its value to be **physical**. In general new types might have to be created in order to do this; thus given, for example, that the type **person** was defined to have the subtypes **first**, **second**, **third**, a new type would have to be inserted in the hierarchy in order to express the equivalent of the disjunction **second** or **third**.




In the particular case of atomic types we do allow disjunction in the language, with an effect which is formally similar to creating an additional type in this way. We express this as a list of values, for example, **(second third)**.

The flexibility of the LKB is enhanced by allowing feature structures to be described in terms of other feature structures. Particular feature structures may have identifiers associated with them; feature structures representing complete lexical entries are identified by a combination of orthography plus sense information. Lexical and grammar rules also have associated names, and in general any feature structure may be defined with an associated identifier. We refer to all such named feature structures as *psorts*; the significance of this is the use of such structures in the description of other feature structures. Feature structure descriptions are not just local; relationships between feature structures may be set up in a variety of ways. The simplest of these relationships conceptually is non-default inheritance, (notated <=); a feature structure may be described as inheriting information from a psort. For example, we could define the feature structure corresponding to the lexical entry for *ewe* as inheriting its qualia structure from a particular sense of *sheep*, but further specifying the SEX to be **female**:

```
<> = lex-count-noun
< QUALIA > <= sheep_L_0_1 < QUALIA >
< QUALIA : SEX > = female
```

Non-default inheritance is simply implemented by unification of the feature structure with a copy of the relevant part of the psort.[1]

[1] Formally the ordinary attribute value language may be regarded as describing feature structures, and the feature structure which is built is the minimal satisfier of this set of descriptions (see de Paiva, this volume, for example). Non-default inheritance from another feature structure is equivalent to adding the set of descriptions of which it is the minimal satisfier to the locally defined set.

We also allow a feature structure to be specified as being identical (modulo alphabetic variance) to a psort (notated as ==).[2] (The non-default inheritance relationship can be seen as a constraint that the daughter feature structure is subsumed by the psort feature structure; the equality relationship corresponds to a mutual subsumption constraint.) The default inheritance relationship (notated as <) allows for values to be overridden, thus we could (albeit somewhat perversely) specify *ram* as inheriting information by default from *ewe*, but override the value for SEX:

```
<> = lex-count-noun
< QUALIA > < ewe_L_0_0 < QUALIA >
< QUALIA : SEX > = male
```

Psort feature structures may also be combined by unification and generalisation,[3] or transformed by lexical rule application. For details, and some examples of use of these operations in the lexical representation language, see Copestake (this volume) and Vossen and Copestake (this volume).

## 9.3 Lexical Rules

The lexical rule mechanism, and the translation link mechanism which will be discussed in the next section, involve no further extensions to the LRL, but indicate how typing and inheritance may be applied to feature structures other than lexical entries. We encode grammar and lexical rules as typed feature structures, which represent relationships between two or more signs. Here we will just consider lexical rules; further details can be found in Copestake and Briscoe (1991) and Briscoe and Copestake (1991). A lexical rule is a feature structure of type **lexical-rule** which is a subtype of **rule**. The expanded constraints for the types are:

$$\begin{bmatrix} \textbf{rule} \\ 0 = \textbf{sign} \\ 1 = \textbf{sign} \end{bmatrix} \quad \begin{bmatrix} \textbf{lexical\_rule} \\ 0 = \textbf{lex\_sign} \\ 1 = \textbf{lex\_sign} \end{bmatrix}$$

Thus all lexical rules have to have the features 0 and 1 which must both have values which are of type **lex_sign**. Lexical rules can be regarded as a means of generating new lexical signs; if a lexical entry can be unified with the feature structure at the end of the path <1> in the lexical rule then the feature structure at the end of the path <0> is a new lexical sign. Alternatively they can be regarded statically, as expressing the relationship between two existing lexical signs. We use lexical rules both to represent morphological derivation and sense extension.

[2] Earlier versions of the LKB used == for non-default inheritance.

[3] The generalisation operation is the opposite of unification; it produces a feature structure which contains only the information which is common to both of its arguments (see, for example, de Paiva, this volume, for a formal definition). In contrast to disjunction it always yields a single feature structure; this may be equivalent to the disjunction, but in general will be less specific.

An example of a productive sense extension process which we represent as a lexical rule is that which transforms animal denoting (count) nouns to (mass) nouns denoting their meat (e.g. *lamb*). Because lexical rules are typed feature structures, we can make use of the LKB's inheritance mechanism in their representation. In this case we regard the animal/meat sense extension as a special case of 'grinding'. It is well known that any count noun denoting a physical object can be used in a mass sense to denote a substance derived from that object, when it occurs in a sufficiently marked context. We refer to this as 'grinding' because the context normally suggested is the 'Universal Grinder' (see Pelletier and Schubert, 1986). So if *a table* is ground up the result can be referred to as *table* (*there was table all over the floor*). Several regular sense extensions can be regarded as special cases of 'grinding', where the extension may have become conventionalised; besides the animal/meat examples, trees used for wood (*beech*) have a sense denoting the wood, and so forth.

A general type for grinding lexical rules can be specified in the LKB as follows:

**grinding** $\sqsubseteq$ **lexical_rule**

$$\left[\begin{array}{l}\textbf{grinding}\\ 1 = \left[\begin{array}{l}\textbf{lex-count-noun}\\ \text{ORTH} = \boxed{1}\\ \text{QUALIA} = \left[\begin{array}{l}\textbf{physical}\\ \text{FORM} = \left[\begin{array}{l}\textbf{physform}\\ \text{SHAPE} = \textbf{individuated}\end{array}\right]\end{array}\right]\end{array}\right]\\ 0 = \left[\begin{array}{l}\textbf{lex-uncount-noun}\\ \text{ORTH} = \boxed{1}\\ \text{QUALIA} = \left[\begin{array}{l}\textbf{physical}\\ \text{FORM} = \left[\begin{array}{l}\textbf{physform}\\ \text{SHAPE} = \textbf{unindividuated}\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right]$$

The effect of this rule is to transform a count noun with the qualia structure properties appropriate to an individuated physical object into a mass noun with properties appropriate for a substance. Thus the core component of grinding is a linguistic operation which affects syntactic realisation, such as the ability to appear without a determiner, correlated with an abstract and underspecified semantic operation.

We specialise the grinding rule to allow for cases such as the animal/meat regular sense extension explicitly. The typed framework provides us with a natural method of characterising the subparts of the lexicon to which such rules should apply. The lexical rules can, in effect, be parametrised by inheritance in the type system. Thus animal_grinding can be described as follows:

animal_grinding

$$\left[\begin{array}{l}\textbf{grinding}\\ 1 = \left[\text{QUALIA} = \textbf{animal}\right]\\ 0 = \left[\text{QUALIA} = \textbf{food\_substance}\right]\end{array}\right]$$

We treat the lexical rule as fully productive across the appropriate subset of the lexicon and account for cases of 'blocking', where an existing lexeme (e.g. *pork*) appears to render the sense extension highly marked (c.f. *pig*), by a separate mechanism which detects the presence of an existing lexical entry comparable to the extended sense (see Briscoe *et al.*, in press). (This relies on the lexical semantic specification of lexical entries being considerably more fine-grained than that shown here.) Thus, the use of *pig* to mean the meat is possible, but tends to suggest that the substance is in some way inferior, or that the speaker is adverse to (this type of) meat.

## 9.4 Representation of Translation Equivalence

In this section we complete the overview of the LKB by introducing the techniques which we use for encoding translation equivalence between lexical entries. A more detailed description, and a discussion of how translation links might be used in machine translation is given in Copestake *et al.* (1992) and Sanfilippo *et al.* (1992).

We define lexical translation equivalence in terms of cross-linguistic links, *tlinks*, between the lexical entries in the monolingual lexicons. In general there may be a many-to-many equivalence between word senses, but each possibility is represented by a single tlink. In the simplest and commonest cases unmodified pairs of lexical entries can be treated as translation equivalents, and it is unnecessary to augment the monolingual information, other than simply to assert that a link is present (see **simple-tlink**, below). However in general we have to allow for 'mismatches' such as differences in argument ordering, plurality, and specificity of reference, and for 'lexical gaps', where a word sense in one language has to be translated by a phrase in the other. The tlink mechanism allows the monolingual information to be augmented with translation specific information, in a variety of ways, in order to cope with such problems. We use inheritance from both lexical entries and rules in tlinks; this makes them compact while ensuring that the multilingual and monolingual components are compatible.

A tlink is simply a feature structure of type **tlink**, which is to be read as stating that two feature structures (the 'output structures') are to be regarded as translation equivalents. The tlink encodes the relationship between the input word senses and these output structures; it can be viewed as describing how lexical entries may be transformed into translation equivalent pairs. A complete tlink is essentially a relationship between two **rules** (as defined above) where the rule inputs have been instantiated by the representations of the word senses in the source and target languages which are to be linked and where the rule outputs are translation equivalent. A level of indirection is thus involved in stating the equivalence between lexical entries, and this allows 'mismatches' to be treated.

The type **tlink** is defined as follows:

```
tlink (top)
< TLINK-ID > = tlink-id
< SFS > = rule
< TFS > = rule
< SFS : 0 : SEM : IND > = < TFS : 0 : SEM : IND >.
```

The third line indicates equivalence of the variables in the two output structures in the particular monolingual encoding of semantic information that we are currently adopting. For all tlinks the feature structures at the end of the paths < SFS : 0 > and < TFS : 0 > will be translation equivalent. For all tlinks at least the paths < SFS : 1 > and < TFS : 1 > have to be instantiated by lexical entries to produce the complete tlink.[4]

By defining types of tlinks the concept of translation equivalence can be constrained and generalisations can be encoded. The commonest and simplest cases of translation equivalence can be represented as **simple-tlinks**.

```
simple-tlink (tlink)
< SFS : 0 > = < SFS : 1 >
< TFS : 0 > = < TFS : 1 >.
```

A **simple-tlink** is applicable in the case where two lexical entries which denote single place predicates (nouns etc.) are straightforwardly translation equivalent, without any transformation being necessary. (For verbs more argument equivalence specifications are necessary; see Sanfilippo *et al.*, 1992.) Like lexical rules tlinks can be regarded statically or dynamically; given a feature structure in one language, and an appropriate tlink, unification with the feature structure at the end of the appropriate path (e.g. < SFS : 0 > ) in the tlink will result in the feature structure at the end of the other output path being returned (e.g. < TFS : 0 > ).

Assuming that the LDOCE sense *chocolate* 1 4, is translation equivalent to the Van Dale *chocolade* 0 2, we would have the tlink:

```
simple-tlink
< SFS : 1 > <= chocolate_L_1_4 <>
< TFS : 1 > <= chocolade_V_0_2 <> .
```

where <= indicates non-default inheritance from the lexical entries.

Some restrictions on translation can be expressed by making the target or source feature structures more specific. For example, both *maestro* and *maestra* in Spanish can be translated as *teacher* in English; the restriction that *maestro* denotes a male teacher and *maestra* a female one can be encoded as follows:

```
simple-tlink
< SFS : 1 > <= teacher_1 <>
< TFS : 1 > <= maestro_1 <>
```

[4] Tlinks and lexical rules are both symmetrical and reversible; we use the terminology source (sfs), target (tfs), input (1) and output (0) solely for ease of exposition.

```
< SFS : 0 : QUALIA : SEX > = male.

simple-tlink
< SFS : 1 > <= teacher_1 <>
< TFS : 1 > <= maestra_1 <>
< SFS : 0 : QUALIA : SEX > = female.
```

Alternatively we can define a type **human-tlink** and state as a constraint that the values for the SEX feature must be equivalent in the translation equivalent feature structures.

```
human-tlink (simple-tlink)
< SFS : 0 : QUALIA : SEX > = < TFS : 0 : QUALIA : SEX >.
```

The restrictions would then follow, assuming that the Spanish lexical entries were appropriately instantiated, and would apply to the whole class.

Somewhat rarer and more complex cases of linking arise when the changes to the feature structures are those such as pluralisation, which is a process that has to be represented separately, and which has to be viewed as a transformation of a feature structure rather then a simple restriction. For example, a lexical/morphological rule for plural formation, which would be required anyway for the monolingual grammar, can be used in a tlink: we encode the idea that the equivalence is to be defined between a basic lexical entry and a lexical entry after rule application by instantiating one half of the tlink with the appropriate lexical rule.

For example, *furniture* can be encoded as translation equivalent to the plural *muebles* by specifying that the named rule 'plural' has to be applied to the base sense in Spanish.

```
tlink
< SFS : 1 > <= furniture_1 <>
< TFS : 1 > <= mueble_1 <>
< SFS : 0 > = < SFS : 1 >
< TFS > <= plural <> .
```

This tlink can be represented diagrammatically as shown in Figure 9.3; unlabelled arrows indicate token identity between FSs. Since the singular form of *mueble* would not unify with the feature structure at the end of the output path < TFS : 0 >, a translation of *mueble* as *furniture* would not be generated by this tlink.

In some cases the existence of a tlink between two lexical items implies a further translation relationship. For example, a similar sense extension rule to that of animal_grinding described in the previous section applies to Italian (Östling, 1991) but in Dutch a compound is generally used (*lam*, *lamsvlees*), although the semantic process is apparently equivalent. To represent the relationship between these lexical rules we define the type **tlink-rule**:

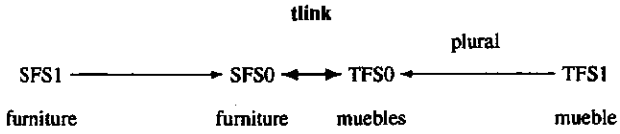



Figure 9.3: Diagrammatic representation of translation link

```
tlink-rule (top)
< ID > = tlink-rule-id
< T0 > = tlink
< T1 > = tlink
< SRULE > = lexical-rule
< SRULE : 1 > = < T0 : SFS : 1 >
< SRULE : 0 > = < T1 : SFS : 1 >
< TRULE > = lexical-rule
< TRULE : 1 > = < T0 : TFS : 1 >
< TRULE : 0 > = < T1 : TFS : 1 >
```

By stating that the lexical rule for animal-grinding is linked with that for compounding with *vlees*, we can, for example, automatically generate the relationship between *lamb_2* and *lamsvlees* from a simple tlink between *lamb_1* and *lam*; see Figure 9.4.[5] There are many examples of such correspondences; for example, the English sense extension between trees and their fruits (*pear*, etc.) is mirrored in Italian with a gender distinction; the trees are masculine but the fruits feminine (*pero*, *pera*).[6]

## 9.5 Conclusion

In this introduction we have outlined the functionality of the ACQUILEX LKB. A description of the software system and its use in the ACQUILEX project is given in Copestake (1992b). In the remaining chapters we give a more detailed description of the LRL and some of the uses to which it has been put.

The aim of the ACQUILEX project is to demonstrate that substantial amounts of lexical information can be acquired semi-automatically from MRDs and represented in a way that makes it usable by a range of NLP systems. The first essential for this is a well-defined representation language, which is efficiently

[5] Since we are just making use of the monolingual sense extension mechanism here we can rely on that to handle cases where the sense extension is blocked.

[6] It does not necessarily matter for translation purposes whether the rule can fully predict the effects of the sense extension; even if the rule is used statically to encode the regular aspects of the relationship between two lexicalised items, an appropriate translation link will be generated if the monolingual processes are sufficiently similar.

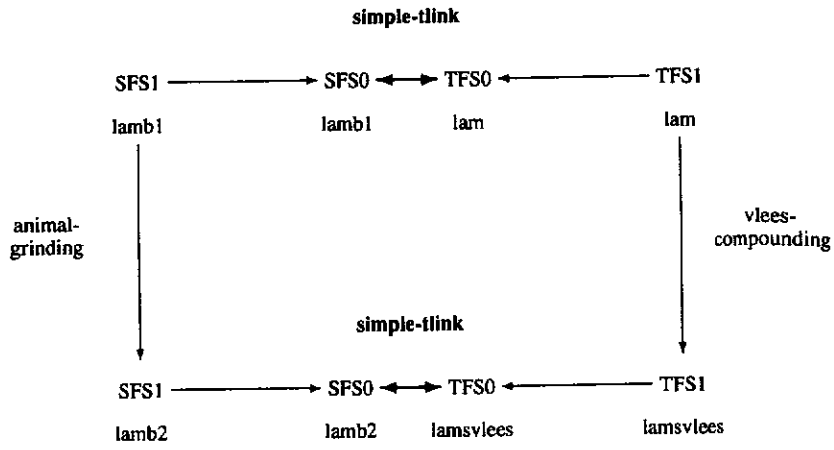



Figure 9.4: Tlink rule for animal grinding and compounding with *vlees*

implementable. We have chosen to use an LRL which is relatively 'theory-neutral' in the same sense as PATR-II (Shieber, 1986); it could be used to implement different linguistic theories. The second essential requirement is to have some theory of the data to be represented which can be encoded in the LRL. Sanfilippo's chapter describes one such theory for verbs; it also illustrates one advantage that our LRL has over PATR-II, in that the type system makes the encoding of the theory more explicit. It would be, however, possible to make use of the information encoded even if a different treatment were adopted; for example, deriving a verb lexicon for a system which did not rely on theta roles to express semantic argument structure would be straightforward, because this information could simply be ignored.

Clearly linguistic theories, their encoding in the LRL and even the LRL itself may have to be modified in response to the data. It is not currently possible to construct a large lexicon which incorporates lexical semantic information without developing the linguistic theory, since formal lexical semantics is a relatively undeveloped field. In our discussion of defaults in the LRL we will show that there are problematic areas, where modifications are required. However, even if further changes are made to the LRL, most, if not all, of the existing data will be reusable, because the current language has been explicitly specified.

### Acknowledgements

This work and that reported in the subsequent chapters was supported by Esprit grant BRA 3030. We are grateful to our colleagues on ACQUILEX in the

Universities of Pisa and Amsterdam, University College Dublin and the Universitat Politecnica de Catalunya, Barcelona, for discussions on the LKB, to John Carroll for his advice and help on the design and construction of the software and to Bob Carpenter for his detailed comments on our use of typed feature structures.

# 10 Types and Constraints in the LKB

VALERIA DE PAIVA

## Introduction

This chapter describes — from a mathematical perspective — the system of typed feature structures used in the ACQUILEX Lexical Knowledge Base (LKB). We concentrate on describing the type system the LKB takes as input, making explicit the necessary conditions on the type hierarchy and explaining how — mathematically — our system of constraints works. It is assumed that the reader is familiar with basic unification-based formalisms like PATR-II, as explained in Shieber (1986). It must also be said from the start that our approach draws heavily on the work on *typed feature structures* by Carpenter (1990, 1992).

The LKB works basically through unification on (typed) feature structures. Since most of the time we deal with *typed* feature structures (defined in section 10.2) we will normally drop the qualifier and talk about feature structures. When necessary, to make a distinction, we refer to structures in PATR-II and similar systems as *untyped* feature structures. Feature structures are defined over a (fixed) *finite* set of features FEAT and over a (fixed) type hierarchy $\langle \mathsf{TYPE}, \sqsubseteq \rangle$. Given FEAT and $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ we can define $\mathcal{F}$ the collection of all feature structures over FEAT and $\langle \mathsf{TYPE}, \sqsubseteq \rangle$. But we are interested in feature structures which are *well-formed* with respect to a set of constraints. To describe constraints and well-formedness of feature structures we specify a function $C : \mathsf{TYPE} \rightarrow \mathcal{F}$, which corresponds to an association of a constraint feature structure $C(t_i)$ to each type $t_i$ in the type hierarchy TYPE. The constraint feature structure $C(t_i)$ imposes conditions on all well-formed feature structures of type $t_i$. We call the combination of FEAT, $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ and the constraint function $C$ the *type system.*

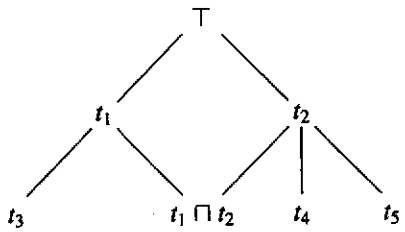
Initially we define the type hierarchies $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ we deal with and then formalise our notion of feature structures and some operations over them. Next we describe our kind of constraints and what it means for a feature structure to be well-formed in our system. Then we discuss briefly internal and external logics of feature structures. A short section concludes comparing this with related work, especially Carpenter's.

## 10.1 The Type Hierarchy

The type hierarchy is a partially ordered set (or poset) $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ with two extra properties. Before describing these properties we recall that if $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ is a poset, it satisfies:

- (reflexivity) For any $t$ in TYPE, $t \sqsubseteq t$.
- (anti-symmetry) If $t \sqsubseteq s$ and $s \sqsubseteq t$ in $\langle \mathsf{TYPE}, \sqsubseteq \rangle$, then $s = t$.
- (transitivity) If $t_1 \sqsubseteq t_2$ and $t_2 \sqsubseteq t_3$ then $t_1 \sqsubseteq t_3$.

We adopt the convention that the most general type appears at the top of any diagram. The type hierarchy is ordered by $\sqsubseteq$ (which can be read 'is more specific than'). For example:




It is a straightforward consequence of the definition of a poset that the order '$\sqsubseteq$' has *no cycles*, i.e if $t_1 \sqsubset t_2$ then $t_2 \not\sqsubseteq t_1$ — where we write $t_1 \sqsubset t_2$ for $t_1 \sqsubseteq t_2$ and $t_1 \neq t_2$. (Suppose it had a cycle, i.e. $t_2 \sqsubseteq t_1$; then using $t_1 \sqsubseteq t_2$ and anti-symmetry we have $t_1 = t_2$, a contradiction!)

Following Carpenter we call a subset $S \subseteq \mathsf{TYPE}$ *consistent*[1] iff there is some $t_0$ in TYPE such that $t_0 \sqsubseteq t$ for any $t$ in $S$. In the example above, for instance, the sets $\{t_1, t_2\}$ and $\{t_2, t_4\}$ are consistent sets, but $\{t_4, t_5\}$ is not, so the first two sets have meets, respectively $t_1 \sqcap t_2$ and $t_4$, while the third set has not. Then we can define:

**Definition 1** *A type hierarchy $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ is a (non-empty) poset with two extra properties:*

1. *Every* consistent *set of types $S \subseteq \mathsf{TYPE}$ has a* unique *greatest lower bound or meet (notation $\sqcap S$).*
2. *The partial order $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ has no unary branches, i.e. no type may have exactly one immediate subtype. If $t_2 \sqsubset t_1$ and there is no intermediate type $s$ such that $t_2 \sqsubset s$ and $s \sqsubset t_1$ then there must be some other subtype $t_3$ such that $t_3 \sqsubset t_1$ and $t_3 \not\sqsubseteq t_2$.*

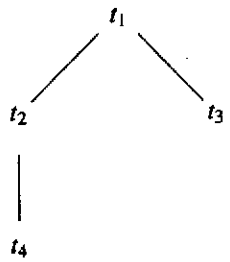
Note that the empty set is (vacuously) consistent, as for any $t_0$ in TYPE it satisfies the condition that $t_0 \sqsubseteq t$ for all $t$'s in the empty set. Hence the partial

[1] The usual term in Lattice Theory is bounded, but consistent seems more expressive.

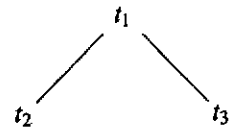
order $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ must have a maximal element $\top$ which is the meet of the (consistent) empty set, $\top = \sqcap\emptyset$. This element $\top$ is such that $t \sqsubseteq \top$ for any $t$ in TYPE. The first property says that the type hierarchy $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ is (the dual of) a bounded complete poset, cf. definition in Gunther and Scott (1991). This property could be re-stated as saying that $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ is a 'consistently complete meet-semilattice'.[2]

If $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ is finite then all (non-empty) joins are defined. Thus we have a poset $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ with two operations, a *partial* operation of taking binary meets $\sqcap$ - or greatest lower bounds - and a total operation of taking joins $\sqcup$ - or lowest upper bounds.

The prohibition of unary branches means that posets like



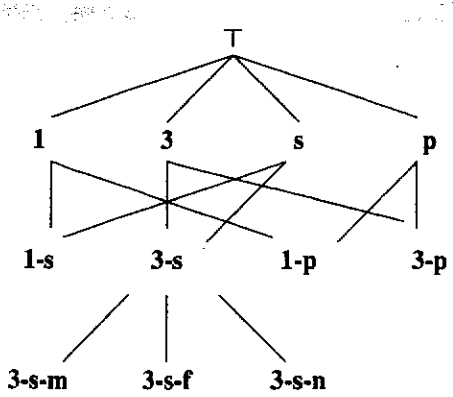

are not allowed. The no-unary-branching condition is desirable because the type system must be 'intuitively complete', where by complete we mean that whatever is said in the partial-order is all that can be said about the types being described. Hence if we say




the interpretation we have in mind is that $t_2$ things are $t_1$ and $t_3$ things are $t_1$ and things which are $t_1$ are either $t_2$ or $t_3$ but nothing else. Thus if we did have the situation above where $t_4$ is the only subtype of $t_2$ we would be stating that everything which was of type $t_2$ was also of type $t_4$ (as well as the inverse). To specify both in the hierarchy could lead to inconsistency (with respect to the specification of constraints, for example) so unary branches are disallowed.

We can make the meet $\sqcap$ operation total if we add the join of the empty-set $\bot = \sqcup\emptyset$ to $\langle \mathsf{TYPE}, \sqsubseteq \rangle$. But even if we do add $\bot$ to make $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ a lattice, this lattice need not be distributive, not even modular, as the example below from Carpenter (1990) shows

[2] Note that a consistently complete meet-semilattice is *not* a meet-semilattice, since it does not have all binary meets, only the *consistent* ones.




Adding $\perp$ to the poset above, we have:

$$\textbf{(3-s-m} \sqcup \textbf{3-s-f)} \sqcap \textbf{3-s-n} = \textbf{3-s-n}$$

$$\neq \textbf{(3-s-m} \sqcap \textbf{3-s-n)} \sqcup \textbf{(3-s-f} \sqcap \textbf{3-s-n)} = \perp$$

Some implementations of systems similar to ours assume a lattice of types and a lattice of feature structures. This can always be achieved by a process of completion of the partial order and several different completion processes are possible; see, for instance, Davey and Priestley (1990). If we do add *only* $\perp$ to $\langle \mathsf{TYPE}, \sqsubseteq \rangle$, we call the resulting type hierarchy $\langle \mathsf{TYPE}, \sqsubseteq \rangle_\perp$. In this case we have an inclusion,

$$\langle \mathsf{TYPE}, \sqsubseteq \rangle \xrightarrow{incl} \langle \mathsf{TYPE}, \sqsubseteq \rangle_\perp$$

Condition 1 on the definition of the type hierarchy $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ seems necessary for the constructions we want to make, at least if one insists on a unique value for the unification of feature structures. Condition 2, on the other hand, is interesting, but not necessary. In his most recent work Carpenter drops this condition and, in effect, so do we, since the introduction of unary branches cannot practically be avoided while developing a type system.

## 10.2 Feature Structures

In this section we define formally the feature structures we shall be dealing with and compare our definition with the traditional (untyped) PATR-II style one, as in Moshier and Rounds (1987). We define the collection $\mathcal{F}$ of feature structures over the (fixed) set of features FEAT and the (fixed) type hierarchy $\langle \mathsf{TYPE}, \sqsubseteq \rangle$. Our feature structures are an acyclic variant of Carpenter's (typed) *quasi-feature structures.*

**Definition 2** *A* feature structure *is a tuple* $F = \langle Q, q_0, \delta, \alpha \rangle$ *where*

- $Q$ *is a (non-empty) finite set of (connected, acyclic) nodes;*

- $q_0 \in Q$ *is the initial (or root) node;*
- $\alpha: Q \rightarrow$ TYPE *is a* total *node typing function and* $\langle$TYPE, $\sqsubseteq\rangle$ *is a type hierarchy as in the previous section;*
- $\delta: Q \times$ FEAT $\rightarrow Q$ *is a partial transition function, where* FEAT *is a (non-empty) finite set.*

*The collection of all possible feature structures for a given set* FEAT *and poset* $\langle$TYPE, $\sqsubseteq\rangle$ *is denoted* $\mathcal{F}$.

An example of a feature structure $F_1$ in attribute-value matrix notation is:

$$\begin{bmatrix} \textbf{phrase} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix}$$

In this case the set of nodes consists of $\{q_0, q_1, q_2, q_3\}$, where $\alpha(q_0) =$ **phrase**, $\alpha(q_1) =$ **agr**, $\alpha(q_2) =$ **1**, $\alpha(q_3) =$ **sing**, $\delta(q_0, \text{AGR}) = q_1$, etc. A notational convention is that types are written in **boldface** and features are written in SMALL CAPITALS within attribute-value matrices (with the exception of the type $\top$). In mathematical definitions $t$'s are used as variables for types, $f$'s as variables for features and $F$'s as variables for feature structures.

The intuition behind this definition goes back to Kasper and Rounds' formalisation of the logic of feature structures; the main idea being that an attribute-value matrix like:

$$\begin{bmatrix} \text{AGR} = \begin{bmatrix} \text{PERS} = \textbf{3} \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix}$$

could be thought of as a deterministic automaton (Kasper and Rounds, 1986). By a 'connected set of nodes' we mean that every node $q \in Q$ is reachable from the initial node $q_0$ by using the transition function $\delta$. More precisely, there exists a sequence of features $\langle f_1 \ldots f_{n-1}\rangle$ in FEAT$^*$ and a sequence of nodes $\langle q_0, q_1, \ldots q_n\rangle$ such that $\delta(q_i, f_{i+1}) = q_{i+1}$ and $q_n = q$.

Recall that in the traditional definition of a feature structure as in, for instance, Carpenter's paper in this volume (after Moshier and Rounds, 1987), one has a *partial (injective) atomic value* function $\alpha$ from nodes to atoms. But only nodes for which no features are defined by the transition function can have atomic values, so that if $\alpha(q)$ is defined then $\delta(q, f)$ is undefined for all $f$ in FEAT. Some types in the definition above will correspond to the 'nodes that do not have features' in the traditional definition and we shall call them *atomic* types. For instance **sing** and **1** in the example above are atomic types.

The main differences between the traditional definition and the one above are that:

- In our definition *all* nodes, not only *some* of the terminal ones, have types.
- The set of types TYPE is now endowed with a partial order.

People of a very abstract turn of mind could write the Moshier-Rounds definition as a triple of functions,

$$1 \xrightarrow{q_0} Q \times \mathsf{FEAT} \xrightarrow{\delta} Q \xrightarrow{\alpha} \mathsf{ATOMS}$$

where a map $1 \xrightarrow{q_0} Q$ picks up one object, $q_0$, in the set $Q$; the arrows $\rightharpoonup$ for $\delta$ and $\alpha$ are partial maps; $\alpha$ is injective and the domain of definition of $\alpha$ is given by

$$dom(\alpha) = \{q \in Q \mid \delta(q,f) \text{ is undefined } \forall f \in \mathsf{FEAT}\}$$

They could also write our definition as

$$1 \xrightarrow{q_0} Q \times \mathsf{FEAT} \xrightarrow{\delta} Q \xrightarrow{\alpha} \langle \mathsf{TYPE}, \sqsubseteq \rangle$$

where, in contrast, the function $\alpha$ is total and TYPE is endowed with a partial order. Pollard and Moshier's (1990) *ordinary feature structures* are slightly different in that the function $\alpha$ is partial, non-terminal nodes can have *SORTS* (and *SORTS* may have a partial ordering on them) and the acyclicity condition is dropped. In all cases one should remember that the set $Q$ is 'rooted' (or connected) by the transition function.

One of the immediate consequences of our definition is that, as every feature structure has a unique initial node $q_0$, every feature structure has a type. We say that

**Definition 3** *The* type of the feature structure $F = \langle Q, q_0, \delta, \alpha \rangle$ *is the type of its initial node, that is,* $\alpha(q_0)$.

Note that this definition induces a function *type-of*: $\mathcal{F} \rightharpoonup$ TYPE. For example the type of feature structure $F_1$ in the example above is **phrase**.

Corresponding to the distinction between atomic and non-atomic types we have atomic and non-atomic feature structures. The feature structure $F_1$ in the example above is a non-atomic feature structure, whereas the feature structure consisting of the single type [**sing**] is an atomic one.

One similarity between the definitions above is that they can be extended to 'paths' $\pi$ in FEAT*. That is, every feature structure $F$ over FEAT gives rise to a map

$$Q \times \mathsf{FEAT}^* \xrightarrow{\delta^*} Q$$

where

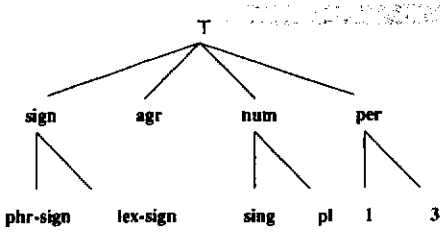



Figure 10.1: An example of a type hierarchy

- $\delta^*(q, \lambda) = q$ if $\lambda$ is the empty path,
- $\delta^*(q, \pi.f) = \delta^*(\delta^*(q, \pi), f)$.

## 10.3 Subsumption of Feature Structures

In this section we describe the order on the collection $\mathcal{F}$ of feature structures and describe the operation of restricting a feature structure $F$ to a node $q$ or a path $\pi$ over FEAT.

It is clear that, as in the traditional setting, we have a natural order in the collection of feature structures $\mathcal{F}$. We call this order $\sqsubseteq$, overloading the symbol. Intuitively $\sqsubseteq$ means is subsumed by ('is-more-specific-than'). Subsumption is like usual subsumption of feature structures, with the added condition that the order on types is 'preserved' (see precise definition below).

For example, given the type hierarchy in Figure 10.1, we have:

$$\left[\begin{array}{l}\textbf{phr-sign}\\ \text{AGR} = \left[\begin{array}{l}\textbf{agr}\\ \text{PERS} = \textbf{1}\\ \text{NUM} = \textbf{sing}\end{array}\right]\end{array}\right] \sqsubseteq \left[\begin{array}{l}\textbf{sign}\\ \text{AGR} = \left[\begin{array}{l}\textbf{agr}\\ \text{PERS} = \textbf{1}\\ \text{NUM} = \top\end{array}\right]\end{array}\right]$$

If $F_1$ and $F_2$ are feature structures of types $t_1$ and $t_2$ respectively, then $F_1 \sqsubseteq F_2$ only if $t_1 \sqsubseteq t_2$. Another example:

$$\left[\begin{array}{l}\textbf{agr}\\ \text{PERS} = \textbf{1}\\ \text{NUM} = \textbf{sing}\end{array}\right] \sqsubseteq \left[\begin{array}{l}\textbf{agr}\\ \text{PERS} = \textbf{1}\end{array}\right]$$

But note that the subsumption order is not simply a containment order. For example, in the feature structures below, $F_1$ contains $F_2$, but $F_1 \not\sqsubseteq F_2$.

$$F_1 = \left[\begin{array}{l}\textbf{sign}\\ \text{AGR} = \left[\begin{array}{l}\textbf{agr}\\ \text{PERS} = \textbf{1}\\ \text{NUM} = \textbf{sing}\end{array}\right]\end{array}\right] \qquad F_2 = \left[\begin{array}{l}\textbf{agr}\\ \text{PERS} = \textbf{1}\end{array}\right]$$

This order in the collection of feature structures $\mathcal{F}$ is mathematically expressed using feature structure morphisms, following Moshier and Rounds.

**Definition 4** *Given feature structures $F_1$ and $F_2$, $\langle Q_1, q_0, \delta_1, \alpha_1\rangle$ and $\langle Q_2, q_0', \delta_2, \alpha_2\rangle$, respectively, in $\mathcal{F}$ we say a total map $h: Q_1 \to Q_2$ is a* feature structure morphism *iff*

- *$h$ sends the initial node $q_0$ of $F_1$ to the initial node $q_0'$ of $F_2$, that is, $h(q_0) = q_0'$.*
- *$h$ preserves the partial map structure of $F_1$, that is, the following diagram 'commutes',*

$$\begin{array}{ccc} Q_1 \times \mathsf{FEAT} & \xrightarrow{\delta_1} & Q_1 \\ {\scriptstyle h \times \mathsf{FEAT}}\downarrow & & \downarrow{\scriptstyle h} \\ Q_2 \times \mathsf{FEAT} & \xrightarrow[\delta_2]{} & Q_2 \end{array}$$

  *this means that if $\delta_1(q,f)$ is defined (written as '$\delta_1(q,f)\ \downarrow$') then $\delta_2(h(q),f)\downarrow$ and $h(\delta_1(q,f)) = \delta_2(h(q),f)$.*
- *$h$ 'preserves' the order in $\langle \mathsf{TYPE}, \sqsubseteq \rangle$, that is, $\alpha_2(h(q)) \sqsubseteq \alpha_1(q)$.*

*For feature structures $F_1$ and $F_2$ in $\mathcal{F}$, we say $F_1 \sqsubseteq F_2$ iff there is a* feature structure morphism *$h: F_2 \to F_1$.*

Note the 'opposite' of the Carpenter (or Pollard and Moshier, 1990) order in the definition above. With our definition of feature structures, the least informative feature structure is $[\top]$, that is, $F \sqsubseteq [\top]$ for any $F$ in $\mathcal{F}$.

This notion of morphism is a natural extension of the definition of homomorphism of untyped feature structures in Moshier and Rounds. The main difference is that for untyped feature structures, if $\alpha_1(q)$ is defined, then $\alpha_2(h(q))$ is defined and *equal* (rather than less than or equal) to $\alpha_1(q)$.

Looking at the definition of morphism of feature structures abstractly we have:

$$\begin{array}{ccccccc} 1 & \longrightarrow & Q_1 \times \mathsf{FEAT} & \xrightarrow{\delta_1} & Q_1 & \xrightarrow{\alpha_1} & \mathsf{TYPE} \\ \| & & {\scriptstyle h \times \mathsf{FEAT}}\downarrow & & {\scriptstyle h}\downarrow & \sqsubseteq & \| \\ 1 & \longrightarrow & Q_2 \times \mathsf{FEAT} & \xrightarrow[\delta_2]{} & Q_2 & \xrightarrow[\alpha_2]{} & \mathsf{TYPE} \end{array}$$

First note that the order in $\mathcal{F}$ is *not* a partial order, but only a *pre-order*. We can have $F_1 \sqsubseteq F_2$ and $F_2 \sqsubseteq F_1$ without $F_1$ and $F_2$ being the same; in this case they are called alphabetic variants, which we write as $F_1 \sim F_2$, following Carpenter. We can make this pre-order a poset by taking equivalence relations the usual way. The equivalence classes of feature structures are called by Moshier (1988) *abstract feature structures.*

We need some extra easy definitions. Given a feature structure $F = \langle Q, q_0, \delta, \alpha \rangle$ and a node $q$ in $Q$ we can define $F|_q$, the restriction of $F$ to $q$, as the feature structure that starts in $q$ and is the restriction of $F$ - as a partial map. More formally:

**Definition 5** *Given a feature structure $F$ of the form $\langle Q, q_0, \delta, \alpha \rangle$ and a node $q$ in $Q$ we define $F|_q$ the restriction of $F$ to $q$, as the feature structure $F' = \langle Q', q_0', \delta', \alpha' \rangle$, such that:*

1. *The new initial node $q_0'$ is $q$,*
2. *The set of nodes $Q'$ is the subset of the set $Q$ of nodes of $F$, reachable from $q$, i.e.*

$$Q' = \{q \in Q \mid \delta(q, \pi) \text{ is defined, for all } \pi \in \mathsf{FEAT}^*\}$$

3. *The transition function $\delta'(q,f)$ is the restriction of $\delta$ to $Q'$.*
4. *The typing function $\alpha'$ is the restriction of $\alpha$ to $Q'$.*

We can also define the restriction $F@\pi$ of a feature structure $F$ to a path $\pi \in \mathsf{FEAT}^*$. The definition above would only change in the two first clauses; the new initial node is $q_0' = \delta(q_0, \pi)$ and the new set of nodes is the subset of $Q$ reachable from $q_0'$. Viewed this way each feature $f$ or path $\pi$ determines a partial function from feature structures to feature structures, the basis of other formalisations of feature structure logics (cf. Smolka, 1988).

Another definition extracts features from a node.

**Definition 6** *Given a feature structure $F$ and a node $q$ in $F$, we define* features of the node $q$ in $F$ *or $Feat(\langle F, q \rangle)$, as the set of features labelling the edges coming out of the node $q$. Thus if $F$ is given by $\langle Q, q_0, \delta, \alpha \rangle$ and $\delta(q,f)$ is defined then the feature $f$ is in $Feat(\langle F, q \rangle)$ or*

$$Feat(\langle F, q \rangle) = \{f \in \mathsf{FEAT} \mid \delta(q,f) \text{ is defined}\}$$

*We call $Feat_0(F)$ the set of features that appear on the top level of the feature structure $F$, that is, $Feat_0(F) = Feat(\langle F, q_0 \rangle)$.*

It is reasonable to ask *why* do we want the morphisms in $\mathcal{F}$ in the direction of definition 4, which is not the direction chosen by Carpenter or Pollard and Moshier. The same question could be asked about the direction of the order on TYPE. The answer is given by the use of feature structures. The main operation one wants to perform with feature structures is *unification*, which we describe in the next section. Unification of $F_1$ and $F_2$ is the *conjunction* of the information contained in $F_1$ and $F_2$. Taking the order on TYPE and the morphisms as defined here, unification corresponds to 'meet' $\sqcap$ in the pre-order $\mathcal{F}$, the natural choice for logical conjunction.

## 10.4 Operations on Feature Structures

We want to define two main operations on feature structures, *generalisation* and *unification*. We give algebraic definitions of both unification and generalisation, following Carpenter, but since algebraically *generalisation* is easier, we start with this operation.

The operation of generalisation $\sqcup\colon \mathcal{F} \times \mathcal{F} \rightarrow \mathcal{F}$ is much more natural from the algebraic viewpoint than the more useful unification. Given feature structures $F_1$ and $F_2$, respectively, $\langle Q_1, q_0, \delta_1, \alpha_1 \rangle$ and $\langle Q_2, q_0', \delta_2, \alpha_2 \rangle$ in $\mathcal{F}$, we can take the product $\delta_1 \times \delta_2$ of the partial maps $\delta_1$ and $\delta_2$ and transform it in a 'product of automata' as follows:

$$\begin{array}{ccccc} 1 & & & & \mathsf{TYPE} \\ \downarrow{\scriptstyle \langle q_0, q_0' \rangle} & & & & \uparrow{\scriptstyle \sqcup} \\ (Q_1 \otimes Q_2) \times \mathsf{FEAT} & \xrightarrow[\delta_1 \otimes \delta_2]{} & Q_1 \otimes Q_2 & \xrightarrow[\alpha_1 \times \alpha_2]{} & \mathsf{TYPE} \times \mathsf{TYPE} \end{array}$$

To be precise:

**Definition 7** *Given feature structures $F_1$ and $F_2$ their generalisation is the feature structure $F_1 \sqcup F_2$ given by:*

$$F_1 \sqcup F_2 = \langle Q_1 \otimes Q_2, \langle q_0, q_0' \rangle, \delta_1 \otimes \delta_2, \alpha_1 \otimes \alpha_2 \rangle$$

- *The initial node of $F_1 \sqcup F_2$ is the pair $\langle q_0, q_0' \rangle$.*
- *The transition function $\delta_1 \otimes \delta_2$ of $F_1 \sqcup F_2$ is the restriction of the product function $\delta_1 \times \delta_2$ given by the composition:*

  $$Q_1 \times Q_2 \times \mathsf{FEAT} \xrightarrow{\Delta} Q_1 \times Q_2 \times \mathsf{FEAT} \times \mathsf{FEAT} \xrightarrow{\delta_1 \times \delta_2} Q_1 \times Q_2$$

  *Thus $\delta_1 \otimes \delta_2$ is given by restricting $\delta_1 \times \delta_2$ to the pairs*

  $$\langle \delta_1(q_1, f), \delta_2(q_2, f') \rangle$$

  *where the feature 'read' is the same, i.e $f = f'$.*
- *The set of nodes $Q_1 \otimes Q_2$, is the subset of $Q_1 \times Q_2$ rooted by the transition function $\delta_1 \otimes \delta_2$ above. Thus $(q_1, q_2)$ is in $Q_1 \otimes Q_2$, if there exists a path $\pi \in \mathsf{FEAT}^*$ such that*

  $$\delta_1 \otimes \delta_2(\langle q_0, q_0' \rangle, \pi) = (q_1, q_2)$$

- *The typing function $\alpha_1 \otimes \alpha_2$ is given by the composition of the product $\alpha_1 \times \alpha_2$ with the function generalisation on types $\sqcup\colon \mathsf{TYPE} \times \mathsf{TYPE} \rightarrow \mathsf{TYPE}$ restricted to the nodes in $Q_1 \otimes Q_2$.*

An easy example should help to make things clear. Suppose we have $F_1$ and $F_2$ below and we know **phr-sign** $\sqsubseteq$ **sign**,

$$F_1 = \begin{bmatrix} \textbf{phr-sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{sing} \end{bmatrix} \\ \text{RGA} = \top \end{bmatrix} \quad F_2 = \begin{bmatrix} \textbf{sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{pl} \end{bmatrix} \end{bmatrix}$$

Generalising, we end up with $F_1 \sqcup F_2$ given by:

$$F_1 \sqcup F_2 = \begin{bmatrix} \textbf{sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{num} \end{bmatrix} \end{bmatrix}$$

Thus generalisation corresponds to taking the product of the partial maps restricting to the diagonal in FEAT and making the resulting structure 'rooted', i.e. getting rid of the unreachable nodes. In the example above we have 20 nodes in $Q_1 \times Q_2$, but 16 are isolated, thus only 4 appear in $Q_1 \otimes Q_2$.

Note that $F_1 \sqcup F_2$ is the *lowest upper bound* of $F_1$ and $F_2$ in the subsumption order. That is, $F_1 \sqsubseteq F_1 \sqcup F_2$ and $F_2 \sqsubseteq F_1 \sqcup F_2$ and if $F_1 \sqsubseteq G$ and $F_2 \sqsubseteq G$ then $F_1 \sqcup F_2 \sqsubseteq G$. Generalisation is a *total* function; the feature structure [T] will always be an upper bound.

Another operation we could define looks very much like generalisation, but uses the 'meet' $\sqcap$ operation on types, which makes it a partial map, if we use the type hierarchy $\langle \text{TYPE}, \sqsubseteq \rangle$,

$$\oplus : \mathcal{F} \times \mathcal{F} \rightharpoonup \mathcal{F}$$

$$\begin{array}{ccccc} 1 & & & & \text{TYPE} \\ \downarrow{\scriptstyle \langle q_0, q_0' \rangle} & & & & \uparrow{\scriptstyle \sqcap} \\ (Q_1 \otimes Q_2) \times \text{FEAT} & \xrightarrow[\delta_1 \otimes \delta_2]{} & Q_1 \otimes Q_2 & \xrightarrow[\alpha_1 \times \alpha_2]{} & \text{TYPE} \times \text{TYPE} \end{array}$$

But if we use $\langle \text{TYPE}, \sqsubseteq \rangle_\perp$ it is another total operation. In the example above, we end up with $F_1 \oplus F_2$ as

$$\begin{bmatrix} \textbf{phr-sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \perp \end{bmatrix} \\ \text{RGA} = \top \end{bmatrix}$$

The operation $\oplus$ has not been discussed in the literature, probably because it is not clear that it has any linguistic utility.

### *10.4.1 Unification*

Unification of feature structures is defined as a *partial* function denoted by $\sqcap: \mathcal{F} \times \mathcal{F} \rightharpoonup \mathcal{F}$. The definition for (typed) feature structures follows broadly the definition for untyped feature structures. Carpenter presents a very simple algorithm — attributed to Moshier — to compute it. Intuitively, the difference from untyped feature structures unification is that:

> If $F_1$ and $F_2$ are feature structures of types $t_1$ and $t_2$ respectively, then $F_1 \sqcap F_2$ has to have type $t_1 \sqcap t_2$. Thus if $t_1 \sqcap t_2$ does not exist then unification fails.

If in $F_1$ and $F_2$ below **phr-sign** $\sqsubseteq$ **sign**:

$$F_1 = \begin{bmatrix} \textbf{phr-sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \end{bmatrix} \end{bmatrix} \quad F_2 = \begin{bmatrix} \textbf{sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{NUM} = \textbf{pl} \end{bmatrix} \end{bmatrix}$$

Then

$$F_1 \sqcap F_2 = \begin{bmatrix} \textbf{phr-sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{pl} \end{bmatrix} \end{bmatrix}$$

But if $F_1$ and $F_2$ are as below and **sing** $\sqcap$ **pl** $= \bot$:

$$F_1 = \begin{bmatrix} \textbf{phr-sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix} \quad F_2 = \begin{bmatrix} \textbf{sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{pl} \end{bmatrix} \end{bmatrix}$$

The unification $F_1 \sqcap F_2$ fails, as the information $F_1$ and $F_2$ convey about the feature NUM is not consistent. Finally unifying

$$F_1 = \begin{bmatrix} \textbf{phr-sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix} \quad F_2 = \begin{bmatrix} \textbf{sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \top \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix}$$

we end up with:

$$F_1 \sqcap F_2 = \begin{bmatrix} \textbf{phr-sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix}$$

Now we define unification algebraically in two steps. Recall that to unify feature structures $F_1$ and $F_2$ we want to 'union' the partial maps $\delta_1$ and $\delta_2$, making sure that

- the two initial nodes are made the 'same';
- if a feature $f$ appears in both feature structures in a consistent way, this feature appears only once in the unification.

Trying to make the initial nodes the same, we have to check that the relation $\delta_3$ that arises from this 'identification' and subsequent ones is really a partial map, not a relation. Note that, given two feature structures $F_1$ and $F_2$, we can (without loss of generality) consider the sets of nodes $Q_1$ and $Q_2$ disjoint. We then write $Q_1 + Q_2$ for the disjoint union of $Q_1$ and $Q_2$.

Define the union $\delta_1 + \delta_2$ of the transition functions $\delta_1$ and $\delta_2$ by

$$(Q_1 + Q_2) \times \mathsf{FEAT} \xrightarrow{\delta_1 + \delta_2} (Q_1 + Q_2)$$

As a graph the partial map $\delta_1 + \delta_2$ is disconnected; it has two initial nodes, and a feature $f$ may appear in both components. Thus to make the initial nodes the same we define an equivalence relation on the set of nodes $Q_1 + Q_2$. Given feature structures $F_1 = \langle Q_1, q_0, \delta_1, \alpha_1 \rangle$ and $F_2 = \langle Q_2, q_0', \delta_2, \alpha_2 \rangle$ in $\mathcal{F}$, we define the equivalence relation '$\bowtie$' on the set $Q_1 + Q_2$ as the least equivalence relation such that:

- $q_0 \bowtie q_0'$;
- $\delta_1(q,f) \bowtie \delta_2(q',f)$ iff both are defined and $q \bowtie q'$.

Because we need to identify nodes in a coherent fashion, the unification operation is more complicated from the algebraic point-of-view than the operation of generalisation. One observation is that having a (partial or not) map $f: A \to B$ and an equivalence relation on $A$, we could define an induced map $[f]$ on the equivalence classes of $A$ by saying that $[f]([a]) = [f(a)]$ if whenever $a \sim a'$ then $f(a) = f(a')$. This is to take the identity equivalence relation on $B$ and it is exactly what is done with unification of untyped feature structures, where $B$ is the set of *ATOMS* and the map $\alpha$ names the atomic nodes. When we merge the graphs of $F_1$ and $F_2$ (as untyped feature structures), we say $F_1 \sqcap F_2$ is defined if $\alpha([q']) = a$ for any $q' \in [q]$. But since we have a partial order on the set of types TYPE there are more possibilities.

We define the unification of typed feature structures $F_1$ and $F_2$ as follows:

**Definition 8** *Given feature structures* $F_1 = \langle Q_1, q_0, \delta_1, \alpha_1 \rangle$ *and* $F_2 = \langle Q_2, q_0', \delta_2, \alpha_2 \rangle$ *in* $\mathcal{F}$, *their unification* $F_1 \sqcap F_2$ *is the feature structure*

$$F_1 \sqcap F_2 = \langle Q_{\bowtie}, [q_0], \delta_{\bowtie}, \alpha_{\bowtie} \rangle$$

- *The set of nodes* $Q_{\bowtie}$ *is given by the set of equivalence classes* $(Q_1 + Q_2)/\bowtie$.
- *The new initial node is the equivalence class* $[q_0]$.
- *The transition function* $\delta_{\bowtie}$ *is given by the equivalence class of the union of the transition functions* $\delta_1 + \delta_2$, *when it is defined, that is:*

$$\delta_{\bowtie}([q],f) = [\delta_1 + \delta_2(q,f)] \text{ if } \delta_1 + \delta_2(q,f) \text{ is defined}$$

- *The new typing function $\alpha_{\bowtie}$ is the 'meet' of the types in the equivalence class of $q$, that is $\alpha_{\bowtie}([q]) = \sqcap\{\alpha_i(q') | q' \bowtie q\}$*

*provided that $F_1 \sqcap F_2$ is not cyclic. If $F_1 \sqcap F_2$ is cyclic we say that unification fails.*

In the same way we could do generalisation with $\sqcup$ or $\sqcap$ on types, we can do unification with either. Looking at it from the graph-theoretical viewpoint we are glueing or merging the graphs, if they are consistent, and then choosing either $\sqcup$ or $\sqcap$ for the result type. The operation described above — true unification — chooses the meet $\sqcap$ of types. We could as well define an operation $\oslash: \mathcal{F} \times \mathcal{F} \rightharpoonup \mathcal{F}$, doing unification of graphs but choosing the join $\sqcup$ of types.

Unification could also be defined through the subsumption order of feature structures, which is a theorem in Carpenter (1992). The unification may FAIL, but if it does succeed, the result of the unification is the meet (or *greatest lower bound*) of the feature structures being unified. Thus $F_1 \sqcap F_2 \sqsubseteq F_1$ and $F_1 \sqcap F_2 \sqsubseteq F_2$ and if $G \sqsubseteq F_1$ and $G \sqsubseteq F_2$, then $G \sqsubseteq F_1 \sqcap F_2$. That is intuitively reasonable, as the unification gives us the *conjunction* of the information in $F_1$ and $F_2$, if they are consistent.

It is worth noting that a product is used for 'join' of information and a coproduct, albeit a complicated one, is used for a 'meet' of information. This is reminiscent of the situation in Domain Theory; the similarity between feature structures and domains has been pointed out and used by several people in different ways; see Pereira and Shieber (1984), Carpenter (1990, 1992) and Pollard and Moshier (1990) .

### *10.4.2 Comparing Feature Structures*

The structure on the type hierarchy TYPE repeats itself on the collection of feature structures $\mathcal{F}$, which is why we have used the same symbols. Thus $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ is a partial order, where $\sqcap$ is called unification of types and $\sqcup$ is called generalisation of types. Also $\langle \mathcal{F}, \sqsubseteq \rangle$ is a pre-order, where $\sqcap$ is given by unification and $\sqcup$ is given by generalisation. The same way two types are consistent if $t_1 \sqcap t_2$ exists, we say that $F_1$ and $F_2$ are consistent if their unification $F_1 \sqcap F_2$ exists. Moreover $\langle \mathcal{F}, \sqsubseteq \rangle$ is a bounded complete pre-order. If $F \sqsubseteq F_1$ and $F \sqsubseteq F_2$ then $F_1 \sqcap F_2$ exists and $F \sqsubseteq F_1 \sqcap F_2$. If we deal with $\langle \mathsf{TYPE}, \sqsubseteq \rangle_\perp$ we can say that types are consistent if $t_1 \sqcap t_2$ exists and is different from $\perp$.

Apart from being typed, the feature structures above are very similar to the traditional ones in Shieber's book (1986). In particular, we do not support *cyclic* feature structures, so, as mentioned before, the set of nodes $Q$ is an *acyclic* connected or rooted graph. There are two main reasons to allow cyclic feature structures. One is implementational, since the check for cycles (no occurs-in check) during unification is computationally expensive. The other one is more conceptual, as mathematically one of the problems with the assumption

that feature structures are acyclic is that you can start with two acyclic feature structures, and their unification is cyclic. This problem can be 'solved' by checking for cyclicity a posteriori, which is not very elegant.

On the other hand, if one accepts cyclic feature structures, apart from problems with checking for well-formedness (next section), one does not have a meet semi-lattice if the set of nodes is finite (cf. Pollard and Moshier, 1990, p. 297). Also, as Pollard and Sag (1987) put it

> In general, cyclic graphs present certain mathematical and computational complexities which are best avoided, although linguistic applications for them have been suggested from time to time.

One of the differences between the feature structures here and the ones in PATR-II is that, because of the type hierarchy, we can support in the formalism disjunction of atomic values. That happens because we can 'complete' the hierarchy $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ with more 'generic' types. For example we can add a type **num** above the types **sing** and **pl**, which stands for either of the types singular or plural. In the traditional definition of a feature structure, since $\delta: Q \times \mathsf{FEAT} \rightarrow Q$ is a partial map, to say that the feature NUMBER could have values **sing** or **pl** on a node $q$ would not be possible — a partial map cannot have two values at some node. Another way to deal with this problem is to introduce a notion of *set-valued* feature structure. This is done, using distinct, but similar, approaches in Pollard and Moshier (1990).

If we write $\langle \mathcal{UF}, \sqsubseteq \rangle$ for PATR-II untyped feature structures (using the Moshier–Rounds definition) and their subsumption order, then we have a map that 'forgets' the (non-atomic) types and the ordering among them

$$\langle \mathcal{F}, \sqsubseteq \rangle \xrightarrow{Erase} \langle \mathcal{UF}, \sqsubseteq \rangle$$

but preserves subsumption. We also have a function

$$\langle \mathcal{UF}, \sqsubseteq \rangle \xrightarrow{trivtyp} \langle \mathcal{F}, \sqsubseteq \rangle$$

which assigns the trivial type '$\top$' to every non-terminal node.

## 10.5 Constraints

So far the typing of feature structures is only providing an ordering on values. Any arbitrary assignment of types is possible and this is intuitively too unconstrained. Types should tell you which features to expect, in principle. Thus the idea here is to 'carve out' from the pre-order of all feature structures $\langle \mathcal{F}, \sqsubseteq \rangle$ a subset, the subset of the *well-formed* feature structures $\langle \mathcal{WF}, \sqsubseteq \rangle$ and these will be well-formed with respect to a given constraining function.

Here we depart substantially from Carpenter's work. Carpenter describes an 'appropriateness specification', that is, a partial map

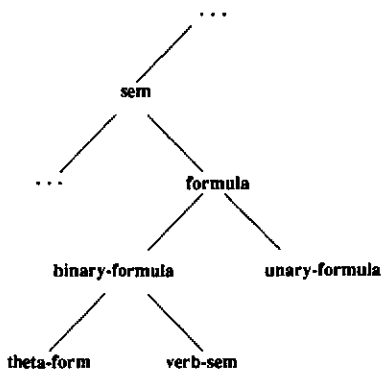



Figure 10.2: Fragment of a type hierarchy

$$\mathit{Approp}: \mathsf{TYPE} \times \mathsf{FEAT} \rightharpoonup \mathsf{TYPE}$$

(satisfying some conditions) which says that for certain types some features are appropriate and yield some other types. This is one possible formalisation of the idea in Pollard and Sag (1987).

The partial map *Approp* is equivalent to a total function

$$\bar{\mathit{A}}\mathit{pprop}: \mathsf{TYPE} \rightarrow [\mathsf{FEAT} \rightharpoonup \mathsf{TYPE}]$$

which corresponds to associating to each type a list of its appropriate features with types. But a list of features with types can be seen as a very simple (one-level only) feature structure. For example, if we have a fragment of a type hierarchy as in Figure 10.2, then an appropriateness specification for the type **formula**, which has features IND, PRED and ARG1, could be:

$$\begin{bmatrix} \textbf{formula} \\ \text{IND} = \textbf{entity} \\ \text{PRED} = \textbf{logical-pred} \\ \text{ARG1} = \textbf{sem} \end{bmatrix}$$

and by the way we wrote the list, one can see that it is, in fact, a very simple feature structure. Hence the idea is to generalise appropriateness specifications, by using feature structures instead of one-level only ones.

We generalise the idea of appropriateness specification by associating with each type a whole feature structure in our constraint specification function. Thus every type in $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ must have exactly one associated feature structure which acts as a constraint on all feature structures of that type. This associated feature structure is given by the function

$$C: \langle \mathsf{TYPE}, \sqsubseteq \rangle \rightarrow \langle \mathcal{F}, \sqsubseteq \rangle$$

but one can think about the constraint specification function $C$ as the set of basic feature structures $C(t_1), C(t_2), \ldots, C(t_k)$ — the constraint feature structures — corresponding to the enumeration of the types $t_1, t_2, \ldots, t_k$ in $\langle \mathsf{TYPE}, \sqsubseteq \rangle$.

We think of appropriateness conditions as being (indirectly specified) information which can be extracted from the constraint feature structures $C(t_i)$ by reading only their first level. We actually refer to the 'appropriate features' of a type meaning the top level features of the constraint feature structures $C(t_i)$. Clearly, having feature structures as constraints on types gives us an extra degree of flexibility, as we can impose *re-entrant* constraints.

Any association of types to (their appropriate) features must satisfy some conditions. We proceed to describe the conditions we impose on the constraining function. Similar conditions are imposed by Carpenter and by Pollard and Sag (1987).

The constraints imposed on a type are inherited by all subtypes of this type. In mathematical terms that means that the function $C$ is monotonic, a very reasonable assumption, since its domain is the poset $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ and its codomain the pre-order $\mathcal{F}$ ordered by subsumption $\langle \mathcal{F}, \sqsubseteq \rangle$. Thus:

**Monotonicity** Given types $t_1$ and $t_2$ if $t_1 \sqsubseteq t_2$ then $C(t_1) \sqsubseteq C(t_2)$

Of course, a subtype may introduce new features — thus if we have the same fragment of a type hierarchy as before and the type **formula** had as its constraint feature structure the previous example, then its subtype **binary-formula** could have as constraint:

$$\begin{bmatrix} \textbf{binary-formula} \\ \text{IND} = \textbf{entity} \\ \text{PRED} = \textbf{logical-pred} \\ \text{ARG1} = \textbf{sem} \\ \text{ARG2} = \textbf{sem} \end{bmatrix}$$

But not all monotonic functions $C : \langle \mathsf{TYPE}, \sqsubseteq \rangle \rightarrow \mathcal{F}$ determine a constraint function. Another obvious condition on constraints is:

**Type** For a given type $t$, if $\langle Q, q_0, \delta, \alpha \rangle$ is the feature structure $F$ given by $C(t)$ then $\alpha(q_0) = t$.

Mathematically this means that composing the function $C$ with the function *type-of* gives the identity on the set TYPE; in other words we have a retraction,

$$\mathsf{TYPE} \underset{C}{\overset{\textit{type-of}}{\leftrightarrows}} \mathcal{F}$$

The condition **Type** is part of the 'modelling convention' in Pollard and Moshier (1990). We also want a condition saying that a feature can only be introduced at one (maximal) point in the type hierarchy — it will be inherited as an appropriate feature by subtypes of that type. (This condition allows us to carry out type inference; see the next section.) Recall from section 10.3 that $Feat_0(F)$ is the set of features that appear on the top level of the feature structure $F$ and that $F|_q$ is the feature structure $F$ starting from the node $q$.

Given a type $t \in \mathsf{TYPE}$ and a candidate constraint function $C(t)$ let the set of *appropriate features of the type* $t$ be the set of features $AppFeat(t)$ that appear on the top level of the constraint $C(t)$, that is, $Feat_0(C(t))$.

> **Maximal Introduction** Given $AppFeat$ obtained from $C(t)$, say $C$ satisfies a maximal introduction condition if for every feature $f \in \mathsf{FEAT}$ there is a unique type $t = Maxtype(f)$ such that $f \in AppFeat(t)$ and there is no type $s$ such that $t \sqsubset s$ and $f \in Appfeat(s)$.

An appropriateness specification has to satisfy two conditions; the first corresponds to **Monotonicity** and the second to **Maximal Introduction**. Our condition **Type** is not necessary in Carpenter's approach because for each type he gives directly the list of appropriate features and their types.

Another condition on the constraining function $C$ seems very reasonable. This condition says that the constraining feature structures $C(t_i)$ must be compatible with each other.

> **Compatibility** If $C(t_1) = F_1$ and some $t_2$ appears in $F_1$, that is, if $F_1$ is the feature structure $\langle Q_1, q_0, \delta_1, \alpha_1 \rangle$ and $\alpha_1(q) = t_2$ for some $q$ in $Q_1$, then $C(t_2) = F_2$ is such that $F_1|_q \sqsubseteq F_2$. Moreover, $Feat_0(F_1|_q) = Feat_0(F_2)$.

The compatibility condition is reminiscent of Sheaf Theory (Tennison, 1975), as it says that *where* the constraining feature structures $C(t_1), C(t_2), \ldots, C(t_k)$ overlap they agree with each other.

The compatability condition implies that no constraint feature structure $C(t) = F$ can strictly contain a feature structure of type $t$ or any subtype of $t$. That is, if $F$ is given by $\langle Q, q_0, \delta, \alpha \rangle$, then for all non-initial nodes $q \in Q$, $q \neq q_0$ the type of the node $\alpha(q) \not\sqsubseteq t$. If such a node existed it would have to be the initial node of a feature structure $F|_q$ which was more specific than $F$, i.e. $F|_q \sqsubseteq F$, and would therefore itself have to contain such a node, and so on. Thus such a constraint could only be satisfied by a cyclic or infinite structure, and we disallow both of these possibilities.

Note that consistency of the constraining feature structures $C(t_i)$, for *consistent types* $t_i$ is enforced simply by monotonicity of the function $C$. If types $t_1$ and $t_2$ are consistent — as types — $t_1 \sqcap t_2$ exists and $t_1 \sqcap t_2 \sqsubseteq t_1$ and $t_1 \sqcap t_2 \sqsubseteq t_2$. Since the constraining function $C$ is monotonic $C(t_1 \sqcap t_2) \sqsubseteq C(t_1)$ and $C(t_1 \sqcap t_2) \sqsubseteq C(t_2)$. Thus the unification of $C(t_1)$ and $C(t_2)$ as feature structures, $C(t_1) \sqcap C(t_2)$ exists ($\mathcal{F}$ is bounded complete) and is such that $C(t_1 \sqcap t_2) \sqsubseteq C(t_1) \sqcap C(t_2)$. Thus the constraint feature structures $C(t_1)$ and $C(t_2)$ are consistent as feature structures, simply by monotonicity of $C$.

**Definition 9** *A function* $C: \langle \mathsf{TYPE}, \sqsubseteq \rangle \rightarrow \langle \mathcal{F}, \sqsubseteq \rangle$ *is a constraint specification function with respect to* FEAT *and* $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ *if it satisfies* **Monotonicity**,**Type**, **Maximal Introduction** *and* **Compatibility**.

**Definition 10** *A feature structure $F = \langle Q, q_0, \delta, \alpha \rangle$ in the collection of feature structures $\mathcal{F}$ is a* well-formed feature structure *with respect to a given constraint specification $C$ iff for all $q \in Q$,*

- $F|_q \sqsubseteq C(\alpha(q))$ *and*
- $Feat_0(F|_q) = Feat_0(C(\alpha(q)))$.

*We call the collection of all well-formed feature structures $\mathcal{WF}$. $\mathcal{WF}$ is a pre-order, as the order in $\langle \mathcal{F}, \sqsubseteq \rangle$ restricts to $\mathcal{WF}$.*

Recap:

1. We wanted to carve out from the collection of all feature structures $\langle \mathcal{F}, \sqsubseteq \rangle$ a collection of well-formed ones $\langle \mathcal{WF}, \sqsubseteq \rangle$ with good properties.
2. To define well-formed feature structures we use a constraining function $C: \mathsf{TYPE} \to \langle \mathcal{F}, \sqsubseteq \rangle$. To calculate whether any feature structure $F$ is well-formed we have to calculate some subsumptions and some sets of features.
3. But not any function $C: \mathsf{TYPE} \to \mathcal{F}$ is a constraining function. To be a constraining function $C$ must satisfy the four conditions **Monotonicity**, **Type**, **Compatibility** and **Maximal Introduction**.

Note that the constraint feature structures $C(t_i)$ are all well-formed by definition, using the compatibility condition, but the definitions are not circular, as the process of checking compatibility of $C(t_i)$'s terminates at the atomic types. Also the function *AppFeat* that we used to define **Maximal Introduction** could be obtained by forgetting some information present in *AppSpec*, namely the target type.

### *10.5.1 Type Checking and Type Inferencing*

As mentioned by Copestake *et al.* (this volume) the maximal introduction condition on features makes a form of type inference possible, whereby a feature structure is given the most general type which is consistent with its top level features. As each feature in FEAT has a maximal type $Maxtype(f)$ at which it can be introduced, given a set of features $S \subseteq \mathsf{FEAT}$, either the set $T = \{Maxtype(f) \mid f \in S\}$ is inconsistent or it has a greatest lower bound $\sqcap T$ where that set of features $S$ will become valid. To show that one uses the bounded completeness of TYPE again. This is interesting because we are not assuming any structure on the set of features, FEAT, but the maximal introduction condition induces a notion of 'consistency' of sets of features.

We have a *Well-formed Inference* proposition, analogous to Carpenter's Type Inference theorem, which says:

**Proposition 1** *Given a constraint specification function $C$ there is a partial map*

$$Fill: \langle \mathcal{F}, \sqsubseteq \rangle \to \langle \mathcal{WF}, \sqsubseteq \rangle$$

*such that for each $F$ in $\langle \mathcal{F}, \sqsubseteq \rangle$, Fill returns a well-formed feature structure $F' = Fill(F)$ or fails.*

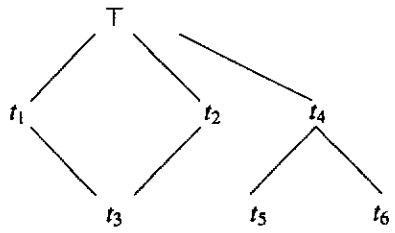
But note that the procedure to transform any feature structure into a well-formed one may fail.

### *10.5.2 Unification of Well-Formed Feature Structures*

Unification of two well-formed feature structures will involve, in general, unifying with the constraint feature structure associated with the meet of their types in order to produce a well-formed result.

If $F_1$ and $F_2$ are well-formed feature structures of types $t_1$ and $t_2$ respectively, then $F_1 \sqcap F_2$, if it exists, has type $t_1 \sqcap t_2$. Since $F_1$ and $F_2$ are well-formed, in particular we know that $F_1 \sqsubseteq C(t_1)$ and $F_2 \sqsubseteq C(t_2)$. Thus if $F_1$ and $F_2$ are consistent, $F_1 \sqcap F_2 \sqsubseteq C(t_1) \sqcap C(t_2)$. But to be well-formed $F_1 \sqcap F_2$ has to satisfy $F_1 \sqcap F_2 \sqsubseteq C(t_1 \sqcap t_2)$ and $C(t_1 \sqcap t_2)$ might be more specific than $C(t_1) \sqcap C(t_2)$.

Consider the following example of a type hierarchy:




Assume that the types $t_4$, $t_5$ and $t_6$ are atomic (i.e. they have constraints $[t_4]$, $[t_5]$ and $[t_6]$, respectively) and the constraints on types $t_1$, $t_2$ and $t_3$ are:

$$C(t_1) = \begin{bmatrix} t_1 \\ f_1 = t_4 \end{bmatrix} \quad C(t_2) = \begin{bmatrix} t_2 \\ f_2 = \top \end{bmatrix} \quad C(t_3) = \begin{bmatrix} t_3 \\ f_1 = t_5 \\ f_2 = \top \\ f_3 = \top \end{bmatrix}$$

We then have

$$C(t_1) \sqcap C(t_2) = \begin{bmatrix} t_3 \\ f_1 = t_4 \\ f_2 = \top \end{bmatrix}$$

Thus $t_3 = t_1 \sqcap t_2$ but $C(t_3) \sqsubset C(t_1) \sqcap C(t_2)$ If we have the following well-formed feature structures

$$F_1 = \begin{bmatrix} t_1 \\ f_1 = t_6 \end{bmatrix} \quad F_2 = \begin{bmatrix} t_2 \\ f_2 = \top \end{bmatrix}$$

then their unification exists:

$$F_1 \sqcap F_2 = \begin{bmatrix} t_3 \\ f_1 = t_6 \\ f_2 = \top \end{bmatrix}$$

But $F_1 \sqcap F_2$ is not a well-formed feature structure of type $t_3$, as $F_1 \sqcap F_2 \not\sqsubseteq C(t_3$ Moreover it cannot be extended to a well-formed feature structure, because it value for $f_1$ is inconsistent with the constraint for $t_3$. Note that the same situatio could arise with Carpenter's appropriateness specifications.

The problem of starting with well-formed feature structures and not getting well-formed unification can be solved by saying that the well-formed unificatio of $F_1$ and $F_2$ is the well-formed feature structure $F_1 \sqcap F_2 \sqcap C(t_1 \sqcap t_2)$, if exists. Another possibility would be to ask $C$ to preserve meets $C(t_1 \sqcap t_2) \sim C(t_1) \sqcap C(t_2)$, but this is too constraining, since it would also have to appl to the cases where $t_1$ and $t_2$ were not immediate parents of $t_1 \sqcap t_2$. Thus th unification operation is not a closed operation in $\mathcal{WF}$. Well-formed unificatio of well-formed feature structures will result in a structure which is totally wel typed (strongly typed) in Carpenter's sense in that all the features which ar possible for that type will be present in the feature structure.

This example illustrates that although the ordering on constraints given b subsumption must be consistent with the type hierarchy, that is, $t_1 \sqsubseteq t_2$ implie $C(t_1) \sqsubseteq C(t_2)$, we do not have that $C(t_1 \sqcap t_2) \sim C(t_1) \sqcap C(t_2)$ nor that $t_1 \sqsubset$ implies $C(t_1) \sqsubset C(t_2)$.

## 10.6 Internal and External Logics

One can think about logic in the context of feature structures in two rath different ways. One way is to think about the collection $\mathcal{F}$ as a set with son algebraic operations and try and see how these operations compare with th algebraic interpretations of traditional logical connectives. In this sense every s which has the structure of a Boolean algebra is a model of classical proposition logic, any set which has a Heyting algebra structure is a model of intuitionist propositional logic, any meet-semilattice is a model of a logic of conjunctio etc. That is what we are calling the 'internal logic', as it is logical structure th is already present in the algebraic definitions.

The second way is to produce a logical calculus (or a set of formulas) fro the feature structures. Thus we can read the paths in feature structures as atom formulae and *add* the traditional logical connectives linking these formulae; th was Kasper and Rounds' approach in their seminal paper (1986). Subseque work has been done to add more logical connectives; for instance, Moshi and Rounds (1987) add intuitionistic implication and negation to the logic feature structures. To make a clear distinction between the feature structur and the formulae built using the same attributes one talks about the language attribute-value 'descriptions'. Descriptions are then a neat notation for pickii up feature structures and we can talk about disjunctive descriptions — even they cannot be represented by a single feature structure.

### *10.6.1 Internal Logic*

When the collection of feature structures is regarded as a set with algebraic structure, where we look for the intrinsic logical operators, an unusual propositional logic emerges, where conjunction (or unification) is partial; that is, conjunction only exists for certain pairs of feature structures, the consistent ones. Recall that $[\top]$ behaves as the identity for unification as $F \sqcap [\top] = [\top] \sqcap F = F$ for any $F$ in $\mathcal{F}$.

Note that even with partial conjunction, we could talk about one feature structure implying another $F_1 \Rightarrow F_2$, where we would define $F_1 \Rightarrow F_2$ as the greatest (or least informative) feature structure $X$ such that $F_1 \sqcap X \sqsubseteq F_2$, when $F_1 \sqcap X$ is defined (Pollard and Sag, 1987). Then we have a (closed) logic of 'partial implication and partial conjunction'.

Generalisation gives us a 'kind of' disjunction. But the logical operation given by generalisation in $\mathcal{F}$ is not logical disjunction. For instance, assume that $F_1$ and $F_2$ are as below, and **1**, **2**, **3** are immediate subtypes of **per**,

$$F_1 = \left[\begin{array}{l}\textbf{phr-sign} \\ \text{AGR} = \left[\begin{array}{l}\textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{sing}\end{array}\right]\end{array}\right] \quad F_2 = \left[\begin{array}{l}\textbf{phr-sign} \\ \text{AGR} = \left[\begin{array}{l}\textbf{agr} \\ \text{PERS} = \textbf{2} \\ \text{NUM} = \textbf{sing}\end{array}\right]\end{array}\right]$$

Then intuitively one expects that $F_1 \vee F_2$ should be

$$\left[\begin{array}{l}\textbf{phr-sign} \\ \text{AGR} = \left[\begin{array}{l}\textbf{agr} \\ \text{PERS} = \textbf{'1} \vee \textbf{2'} \\ \text{NUM} = \textbf{sing}\end{array}\right]\end{array}\right]$$

But $F_1 \sqcup F_2$ is

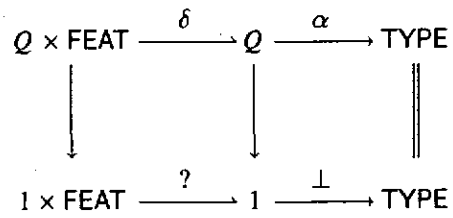
$$\left[\begin{array}{l}\textbf{phr-sign} \\ \text{AGR} = \left[\begin{array}{l}\textbf{agr} \\ \text{PERS} = \textbf{per} \\ \text{NUM} = \textbf{sing}\end{array}\right]\end{array}\right]$$

Hence $(F_1 \vee F_2) \sqsubseteq (F_1 \sqcup F_2)$, which means that $\sqcup$ is not fine-grained enough to model disjunction of information. Thinking about the internal logic of feature structures in $\mathcal{F}$ one is reduced to a logic of partial conjunction, partial implication and total (but very strange) 'form-of-disjunction'.

We can make conjunction total, by adding an inconsistent feature structure. Supposing we have $\langle \mathsf{TYPE}, \sqsubseteq \rangle_\perp$ instead of $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ we have an atomic feature structure $[\perp]$. We could use this feature structure to make unification total, that is, we could define $F_1 \sqcap F_2 = [\perp]$, if $F_1 \sqcap F_2$ fails. This is analogous to the situation with types.

But $[\perp]$ is not an identity for generalisation. If an identity $I$ for generalisation existed it should satisfy $I \sqsubseteq F$, for all $F$'s in $\mathcal{F}$, the characteristic of *false*, the identity for disjunction, hence we should have a morphism of feature structures

$F \rightarrow I$. We also want to complete the definition of $I$ by saying which is the partial map $?: 1 \times \mathsf{FEAT} \rightarrow 1$ making the diagram below commute.




But there is only one map $Q \rightarrow 1$ and if the diagram above were a morphism of feature structures '$?(!(q),f)$' would have to be defined and equal to '$!(\delta(q,f))$' for any feature $f \in \mathsf{FEAT}$. That means that the partial map '?' would have $?(*,f) = *$ for all features and that is not a partial map, hence not a feature structure.[3] Even if $[\bot]$ is not ideal as 'the' inconsistent feature structure, say, $F_1$ and $F_2$ are consistent if $F_1 \sqcap F_2$ exists and is different from $[\bot]$.

Thus if we use $\langle \mathsf{TYPE}, \sqsubseteq \rangle_\bot$ we have total conjunction, but no constant *false*, and disjunction and generalisation are not the same — a very poor logical set up. But of course there are external logics. One of the first external logics was described by Kasper and Rounds and is the subject of the next section.

### *10.6.2 Logic of Descriptions*

In this section we introduce a restriction of the logical attribute-value language that several researchers, notably Kasper and Rounds (1986, 1990) and Carpenter (1992) employ to *describe* feature structures. Much more powerful description languages are used by researchers who prefer *feature algebras*, and a comparison between these two approaches is not attempted here.

**Definition 11** *The set of* descriptions *over the poset* $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ *of types and the collection* FEAT *of features is the least set* DESC *such that*

- $t \in \mathsf{DESC}$ *if* $t \in \mathsf{TYPE}$
- $\pi : \phi \in \mathsf{DESC}$ *if* $\pi \in \mathsf{FEAT}^*$ *and* $\phi \in \mathsf{DESC}$
- $\pi_1 \doteq \pi_2 \in \mathsf{DESC}$ *if* $\pi_1, \pi_2 \in \mathsf{FEAT}^*$
- $\phi \wedge \psi \in \mathsf{DESC}$ *if* $\phi$ *and* $\psi \in \mathsf{DESC}$

The idea of providing descriptions as formulae of a logic to be satisfied by some feature structures is introduced by Pereira and Shieber (1984), but they, as well as many other researchers, have a richer set of formulae.

[3] Note that we assume that the set FEAT has at least two elements.

Since we restricted the formulae in DESC to the $\wedge$-fragment of propositional logic in the definition above, it does not matter how satisfaction is defined, as the $\wedge$-fragment of classical logic is equivalent to the $\wedge$-fragment of intuitionistic logic. But if we want to add disjunction or implication or negation to DESC, a choice of logical framework becomes necessary. Also different notions of satisfaction will lead to different logical formalisms, which explains why there are so many papers in the literature on this topic.

**Definition 12** *The* satisfaction relation *relates the collection of feature structures $\mathcal{F}$ and the set of descriptions* DESC. *It is the least relation '$\models$' such that, if $F$ is the feature structure $\langle Q, q_0, \delta, \alpha\rangle$ and $\phi \in$ DESC*

- $F \models t$ *if* $t \in$ TYPE *and* $\alpha(q_0) \sqsubseteq t$
- $F \models \pi{:}\phi$ *if* $F@\pi \models \phi$
- $F \models \pi_1 \doteq \pi_2$ *if* $\delta(q_0, \pi_1) = \delta(q_0, \pi_2)$
- $F \models \phi \wedge \psi$ *if* $F \models \phi$ *and* $F \models \psi$

Note that the type $\top$, which is already in DESC by definition, behaves as the constant *true* for this logic. It is satisfied by any feature structure $F \models \top$, because for all $F \in \mathcal{F}$, $\alpha(q_0) \sqsubseteq \top$. Recall as well the following usual logical definition,

**Definition 13** *Consider the set of all feature structures that satisfy a certain description $\phi$, that is $Sat(\phi) = \{F \in \mathcal{F} | F \models \phi\}$.*

*If $\phi$ is a formula in* DESC, *say that $\phi$ is* satisfiable *if there exists a feature structure $F$ that satisfies it, that is the set $Sat(\phi)$ is not empty.*

We have the traditional result:

**Proposition 2** *If $F_1 \models \phi$ and $F_2 \sqsubseteq F_1$ then $F_2 \models \phi$ (monotonicity).*

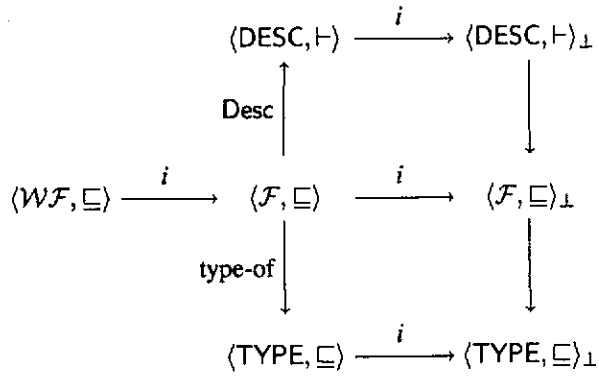
*For every satisfiable formula $\phi$ there is a most general feature structure $MGSat(\phi)$ that satisfies it.*

*For any feature structure in $\mathcal{F}$ there is a description* Desc($F$) *such that*

$$F \cong MGSat(\mathsf{Desc}(F))$$

The results and definitions above are all in Kasper and Rounds, the only reason to recall them here is to remind the reader that these results are another way of expressing the existence of the internal logic. In other words, within this small fragment of propositional logic, we have a notion of entailment '$\vdash$' and as entailment is reflexive and transitive, we can think of $\langle$DESC, $\vdash\rangle$ as a pre-order, where meet $\sqcap$ is given by conjunction and $\top$ is the constant *true*. Clearly, given any feature structure $F$, we can write it as a big conjunction of descriptions, hence the existence of the the function Desc in the proposition above and the

diagram below:




Having established a minimum common denominator one could extend the set of descriptions to accommodate several formalisms. If we choose intuitionistic logic, we can have the Moshier and Rounds (1987) approach. If we choose three-valued logic we can use Dawar and Vijay-Shanker's (1989) formalism. If we choose classical logic, we can have the Kasper and Rounds (1986, 1990) system and Carpenter's (1990, 1992) system (both have classical disjunction). Still using classical logic but at right angles, we have Smolka's (1988, 1989) and Johnson's (1988) systems where DESC has variables and negation. Reape (1991) also adds variables, but he wants to consider the features in paths of the form $\langle f_1 : \phi \rangle$ as modal (possibility)[4] operators, thus getting a poly-modal logic.

## 10.7 Conclusion

We presented a rigorous *mathematical* definition of a system of well-formed typed feature structures. Our system is very similar to Carpenter's system, but we allow more expressive constraints to be made over types. In our system each type $t_i$ is constrained by a whole, possibly re-entrant, constraint feature structure $C(t_i)$, while using Carpenter's appropriateness specifications each type is constrained by a list of features and types — a one-level only feature structure. On the other hand, Carpenter allows (general) 'disjunctive constraints' which we do not handle at the moment. It should be noted that we could not use Carpenter's appropriateness specifications alone to give the sort of functionality for inheritance in the type system that we needed, thus the generalisation to well-formedness.

We also pointed out where somewhat different choices could be made in the formalisation of typed feature structures and presented two new operations that

[4] This also seems to be the case for the Category Structures of Gazdar *et al.* (1988).

one could consider over feature structures looking at them from the mathematical viewpoint only. Finally we briefly discussed one of the reasons why there are many different logics of feature structures in the literature, but for a survey of these logics the reader is referred to the substantial works of Carpenter, Reape and Johnson.

**Acknowledgements**

Many of the basic ideas and intuitions in this chapter are due to Ann Copestake, Ted Briscoe and Antonio Sanfilippo. The way in which these ideas have been formalised is the responsibility of the author, as are any mistakes in the formalisation.