Default Representation in Constraint-based Frameworks

Alex Lascarides[*]
University of Edinburgh

Ann Copestake[†]
CSLI, Stanford University

Default unification has been used in several linguistic applications. Most of them have utilized defaults at a meta-level, as part of an extended description language. We propose that allowing default unification to be a fully integrated part of a typed feature structure system requires default unification to be a binary, order independent function, so that it acquires the perspicuity and declarativity familiar from normal unification-based frameworks. Furthermore, in order to respect the behavior of defaults, default unification should allow default reentrancies and values on more general types to be overridden by conflicting default information on more specific types. We define what we believe is the first version of default unification to fully satisfy these criteria, and argue that it can improve the representation of a range of phenomena in syntax, semantics and the lexico-pragmatic interface.

1. Introduction

The utility of defaults in linguistic representation has been widely discussed (for an overview, see Daelemans *et al.*, 1992). The most common linguistic application for default inheritance is to encode lexical generalizations (Boguraev and Pustejovsky, 1990; Briscoe *et al.*, 1990; Vossen and Copestake, 1993; Daelemans 1987; Evans and Gazdar 1989a,b, 1996; Flickinger *et al.*, 1985; Flickinger, 1987; Flickinger and Nerbonne, 1992; Kilgarriff, 1993; Krieger and Nerbonne, 1993; Sanfilippo, 1993; Shieber, 1986a, and others), but defaults have also been used for specification in syntactic theory (e.g., Gazdar, 1987; Shieber, 1986b), the analysis of gapping constructions (Kaplan, 1987) and of ellipsis (Grover *et al.*, 1994). In Lascarides *et al.* (1996), we argued for the role of defaults both in descriptions of the lexicon and grammar and to allow the linguistic component to make defeasible proposals to discourse processing/pragmatics. However, most current constraint-based systems either do not support defaults or only allow them at a meta-level, as part of an extended description language. Our aim is to allow defaults as a fully integrated part of a typed feature structure system.[1]

In general, although there have been several approaches to formalizing default inheritance within feature structure languages by defining an operation of default unification (some examples are cited in §1.2), these have failed to achieve the combination of perspicuity, declarativity and expressibility familiar from unification-based approaches to non-default inheritance. Lascarides *et al.* (1996) described a version of default unifica-

[*] Centre for Cognitive Science and Human Communication Research Centre, University of Edinburgh, 2, Buccleuch Place, Edinburgh EH8 9LW, Scotland, UK. E-mail: alex@cogsci.ed.ac.uk
[†] Center for the Study of Language and Information, Stanford University, Ventura Hall, Stanford, CA 94305, USA. E-mail: aac@csli.stanford.edu
[1] We will assume a very similar notion of types to Carpenter (1992), although we use the opposite polarity in hierarchies (i.e., for us the most general type is $\top$, top) — details are given in Copestake (1992).

tion which was more satisfactory in this respect. However, there were some problems with that approach, as we discuss in more detail in §1.2. Our aim here is to present an alternative version of default unification, which overcomes those flaws, and also to give detailed examples of various ways in which constraint-based systems can utilize defaults, and the requirements that these impose on the formalism.

1.1 Criteria for default unification

Lascarides *et al.* (1996) list a number of desiderata for default unification, which we repeat here (with some amendments to the justifications):

1. Non-default information can be distinguished from default information and is always preserved.
   We intend our representation language to supplement rather than supplant existing work using monotonic inheritance within frameworks such as HPSG. Non-default processing is crucial to parsing and generation in a unification-based framework, because unification failure is required to prevent ungrammatical structures. From this perspective, it seems reasonable to expect grammars and lexicons to have a monotonic backbone, which encodes the main architectural properties of the feature structures. Since construction of grammars and lexicons is error-prone, we believe that grammar writers will want to prevent accidental overriding of such structural information by ensuring that it is indefeasible. We therefore believe that it is desirable that defaults be explicitly marked, and, as shown by Young and Rounds (1993), this is a necessary condition for the order-independence of default unification (criterion 5, below).

2. Default unification never fails unless there is conflict in non-default information.
   The usual assumptions made about conflicting defaults is that they do *not* result in an inconsistent knowledge state (e.g., Reiter, 1980). So, it is clearly desirable that unification failure does not occur as a side-effect of the way default information is defined.

3. Default unification behaves like monotonic unification in the cases where monotonic unification would succeed.
   We want to use the same notions of unification, subsumption etc with respect to both default and non-default feature structures. This will enable us to use default unification to extend existing monotonic approaches to linguistic analysis, rather than replacing them.

4. Default unification returns a single result, deterministically.
   As we will describe in §3, there are definitions of default unification in which a disjunction of feature structures may be produced. While there is nothing wrong with such definitions formally, they are practically inconvenient, since they can result in a multiplication of structures, and they are only suitable for implementations which allow disjunction.

5. Default unification can be described using a binary, order independent (i.e., commutative and associative) operation.
   The main failing of most definitions of default unification is that they are not order-independent. Having an ordering dependent operation in the description language is inelegant, but it has been regarded as acceptable, because all the structures to be unified are in a fixed hierarchy and an inheritance order can therefore be imposed (there will be a difference between top-down vs.

bottom-up inheritance for instance). However, order-dependence undermines many of the arguments which have been put forward in favor of constraint-based grammar formalisms: that is, that processing can be seen as a uniform accumulation of constraints and is independent of the algorithm which controls evaluation order. For instance, an order-dependent operation causes problems for strategies such as lazy evaluation. More fundamentally, we have argued (e.g., in Lascarides and Copestake, in press) that it is necessary for the lexicon to propose default information which may be overridden by pragmatics. But in a discourse situation, it is impossible to predict which pieces of information are to be unified *in advance* of starting the discourse parsing process. So the interface between discourse processing and order dependent lexical processing would have to take into account the order in which the unification operations are done, which is impractical.

6. Defaults can be given a precedence ordering such that more specific information overrides less specific information.
   This is the usual assumption made about inheritance hierarchies, and is necessary if exceptions to a generalization themselves have exceptions. The approach to default unification that we will describe in this paper allows any defaults to be overridden by defaults which are associated with more specific types: thus priority ordering reflects the type hierarchy ordering. (In §6.2, we will mention other possibilities for imposing a priority order on defaults.)

Barring criterion 6, all of the above properties are necessary for making default unification behave as much like normal unification as possible, save that (default) information can be overridden. These criteria ensure that the default unification operation has properties familiar from monotonic unification, such as determinacy, the way information is accumulated, the conditions when unification fails, and order independence. Since this guarantees that default unification shares many of the properties of normal unification, a 'seamless transition' is possible between the monotonic approach to linguistic analysis supplied by normal unification, and the extension to these analyses provided by supplying default constraints and default unification operating over them. We will justify these assumptions with respect to particular linguistic examples in §4. In this paper, we define an order independent typed default unification operation called YADU (Yet Another Default Unification), which we believe is the first definition of default unification that fulfills all of the above criteria.

1.2 Previous definitions of default operations on feature structures

There have been a number of previous definitions of default unification, including those given by: van den Berg and Prüst (1991), Bouma (1990, 1992), Calder (1991), Carpenter (1993), Copestake (1992, 1993), Russell *et al.* (1991, 1993). These definitions were all based on Kaplan's sketch of priority union (Kaplan, 1987) and are asymmetric since one feature structure (FS) is taken to be indefeasible while the other is defeasible. This operation is not commutative or associative (Carpenter, 1993). Although there are some applications for which an asymmetric operation is useful (see §6.5) the order-dependency makes it undesirable as a basis for inheritance, as we discussed above. Furthermore, since these definitions do not allow for statements of precedence order in defaults, the inheritance hierarchy has to be stipulated separately.

Young and Rounds define an order independent version of default unification (Young and Rounds, 1993) using Reiter's default logic (Reiter, 1980) to model the operation. But it does not allow for precedence between defaults based on specificity (criterion 6, above). The most straightforward way of extending the definition to meet criterion 6 would be to extend Reiter's default logic so that it validates specificity. But as Asher and Morreau (1991), Lascarides and Asher (1993) and Lascarides *et al.* (1996) argue, all such extensions to Reiter's default logic either impose ordering constraints on the application of logical axioms in proofs (e.g., Konolige, 1988), or they impose a context-sensitive translation of the premises into the formal language (e.g,. Brewka, 1991). So extending Young and Rounds' definition in this way comes at the cost of an underlying formal semantics which has separately defined order constraints on the logical axioms, or context sensitive translation.

Lascarides *et al.* (1996) use conditional logic to extend Young and Rounds' definition to typed feature structures, describing an operation called Persistent Default Unification (PDU). They use a conditional logic precisely because these logics are able to validate nonmonotonic patterns of inference involving specificity without imposing ordering constraints on the application of axioms or imposing context sensitive translation on the premises. In order to allow default inheritance in type hierarchies, precedence of default values in PDU is determined by the specificity of the types at which the default is introduced. PDU thus meets criterion 6 for non-reentrant information. But it can't validate the overriding of default reentrancy by conflicting default values introduced by a more specific type. This is partly because the logic underlying PDU demands that the default values on paths are worked out independently of one another. Since PDU can't compare values on paths which, by default, share nodes, we were forced to make the design decision that default reentrancies always survive, and if the default values on the shared node that are introduced by a more specific type conflict, then the value on the shared node in the result is $\perp$, indicating unification failure.

This failure to fully meet criterion 6 restricts PDU's linguistic application, as we will show in §4. There are also problems in interpreting FSs containing $\perp$. Such structures cannot be treated as normal FSs and the operation, *DefFill*, which converts a partially defeasible FS to a monotonic one (necessary, for instance, when defaults are being used to allow concise description of classes of lexical entry, as we will discuss in §2 and §4) has a complex definition because of the need to allow for $\perp$. Furthermore, as we discuss briefly in §3.6.3, PDU can result in FSs which are not well-formed with respect to the distinction between simple values and FS values. We also found that the complexity of the definition of PDU made it difficult to use. These problems led us to develop the alternative definition, YADU, presented here. Like PDU, YADU can be formalized in a conditional logic, but in this paper we will give a definition in terms of an algebraic operation on FSs. These algebraic definitions are easier to follow, and provide much simpler proofs of theorems than the conditional logic (cf. the theorems for PDU in Lascarides *et al.* (1996)).

In the next section, we give an informal overview of YADU, by means of a worked example. This is followed in §3 by the formal definitions, some illustrative examples and an explicit comparison with PDU. In §4, we describe some linguistic examples in detail and discuss the requirements they impose on the default unification operation. §6 covers some alternative and extended definitions, including one which makes use of Carpenter's (1992) inequalities.

2. An Informal Overview of YADU

YADU is based on the intuitively simple idea of incorporating the maximal amount of default information, according to its priority. We will use a simple example in order to illustrate the operation and to contrast it with the PDU operation discussed in Lascarides *et al.* (1996). Suppose we wish to encode the following information about the suffixes of English verbs:

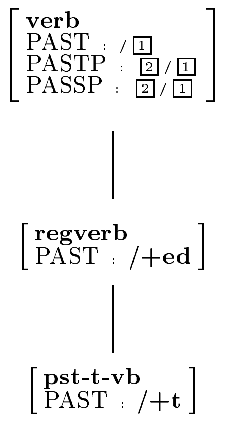


Figure 1
Constraint descriptions for a type hierarchy for inflections: informal notation

1. Past participle and passive participle suffixes are always the same.
2. Past tense suffixes are usually the same as participle suffixes.
3. Most verbs have the past suffix *+ed*.

We assume a paradigm style of encoding with separate features for each suffix: here the past tense suffix slot is indicated by the feature PAST, past participle by PASTP and passive participle by PASSP. Figure 1 shows a fragment of a type hierarchy which is intended to show informally how the generalizations above could be encoded in this way. The figure uses the conventional AVM notation for FSs, but with the addition of a slash, which indicates that material to its right is to be treated as default. All features have both non-default and default values, but where the non-default value is $\top$ (i.e., the most general type in the bounded complete partial order) we omit it: e.g. /+ed is equivalent to $\top$/+ed. We omit both the slash and the default value, if the non-default value is equal to the default value. Default reentrancy between nodes is indicated by slashes preceding the labels: e.g., in the constraint for **verb** in Figure 1, PASTP and PASSP are necessarily coindexed, while PAST is defeasibly coindexed with both of them. The intention of describing such a hierarchy is that verbs such as *walk* can be defined as **regverb**s, while *sleep*, for example, would be a **pst-t-verb**. We assume that inflectional rules are responsible for ensuring that the correct affix is realised, but we will not give details of such rules here.

However the slashed notation used in Figure 1 cannot in general contain sufficient information to ensure order-independence, as was shown in Lascarides *et al.* (1996). As we will show in more detail in §3, additional information is required which encodes part of the history of a series of default unifications. Lascarides *et al.* referred to the structure which encoded this information as a *tail*. In that paper, a tail was notationally equivalent to a set of atomic feature structures labeled with a type. An atomic FS is defined as in Carpenter (1993), i.e., it cannot be decomposed into simpler FS. Intuitively, an atomic FS in the tail represents a piece of default information which was introduced at some point in the series of default unifications that resulted in the current structure. The type which labels an atomic FS in a tail serves to prioritize the default information given by the atomic FS—the priority being determined by the position of this type in the type hierarchy (more specific types have higher priority).

$$\begin{tikzcd}
\begin{bmatrix}\textbf{verb}\\ \text{PAST} : \top \\ \text{PASTP} : \boxed{2} \\ \text{PASSP} : \boxed{2}\end{bmatrix} / \begin{bmatrix}\textbf{verb}\\ \text{PAST} : \boxed{1}\\ \text{PASTP} : \boxed{1}\\ \text{PASSP} : \boxed{1}\end{bmatrix} / \{\} \arrow[d, dash] \\
\begin{bmatrix}\textbf{regverb}\\ \text{PAST} : \top\end{bmatrix} / \begin{bmatrix}\textbf{regverb}\\ \text{PAST} : \textbf{+ed}\end{bmatrix} / \{\langle [\,\text{PAST} : \textbf{+ed}\,], \textbf{regverb}\rangle\} \arrow[d, dash] \\
\begin{bmatrix}\textbf{pst-t-verb}\\ \text{PAST} : \top\end{bmatrix} / \begin{bmatrix}\textbf{pst-t-verb}\\ \text{PAST} : \textbf{+t}\end{bmatrix} / \{\langle [\,\text{PAST} : \textbf{+t}\,], \textbf{pst-t-verb}\rangle\}
\end{tikzcd}$$

Figure 2
Type hierarchy using PDU

$$\begin{tikzcd}
\begin{bmatrix}\textbf{verb}\\ \text{PAST} : \top \\ \text{PASTP} : \boxed{2} \\ \text{PASSP} : \boxed{2}\end{bmatrix} / \begin{matrix}\{\langle \begin{bmatrix}\text{PAST} : \boxed{1}\\ \text{PASTP} : \boxed{1}\end{bmatrix}, \textbf{verb}\rangle, \\ \langle \begin{bmatrix}\text{PAST} : \boxed{1}\\ \text{PASSP} : \boxed{1}\end{bmatrix}, \textbf{verb}\rangle\}\end{matrix} \arrow[d, dash] \\
\begin{bmatrix}\textbf{regverb}\\ \text{PAST} : \top\end{bmatrix} / \{\langle [\,\text{PAST} : \textbf{+ed}\,], \textbf{regverb}\rangle\} \arrow[d, dash] \\
\begin{bmatrix}\textbf{pst-t-verb}\\ \text{PAST} : \top\end{bmatrix} / \{\langle [\,\text{PAST} : \textbf{+t}\,], \textbf{pst-t-verb}\rangle\}
\end{tikzcd}$$

Figure 3
Type hierarchy using YADU

In Lascarides *et al.* (1996), we used a tripartite structure consisting of an indefeasible typed feature structure, a defeasible TFS and the tail, written in a slashed notation: Indefeasible/Defeasible/Tail. The full PDU style representation corresponding to Figure 1 is shown in Figure 2 (the atomic FSs shown here are notationally equivalent to the path:value pairs used in Lascarides *et al.*).

Note that in Figure 2 the tails do not contain all the default information; in particular, they contain only path value structures, and do not contain equalities on paths (i.e., reentrancies). This contributed to PDU failing to achieve all the criteria mentioned in §1.1. YADU is different from PDU, in that it will achieve these criteria. In contrast, the tails in YADU contain information about reentrancy. This means that, unlike in PDU, it is not necessary to maintain the defeasible TFS as a substructure in the representation, since it can be calculated directly from the indefeasible structure and the tail. Thus for YADU we use bipartite structures, which we write Indefeasible/Tail. We will refer to these as typed default feature structures (TDFSs). The YADU representations corresponding to Figure 2 are shown in Figure 3.

There are two operations in YADU: one involves combining two TDFSs to form another TDFS, which we will notate $\overset{<}{\sqcap}$, and another takes a TDFS and returns a TFS, as discussed

$$\begin{bmatrix}\textbf{verb}\\ \text{PAST}:\top\\ \text{PASTP}:\boxed{2}\\ \text{PASSP}:\boxed{2}\end{bmatrix} / \begin{matrix}\{\langle\begin{bmatrix}\text{PAST}:\boxed{1}\\ \text{PASTP}:\boxed{1}\end{bmatrix},\textbf{verb}\rangle,\\ \langle\begin{bmatrix}\text{PAST}:\boxed{1}\\ \text{PASSP}:\boxed{1}\end{bmatrix},\textbf{verb}\rangle\}\end{matrix} \qquad \begin{bmatrix}\textbf{verb}\\ \text{PAST}:\boxed{1}\\ \text{PASTP}:\boxed{1}\\ \text{PASSP}:\boxed{1}\end{bmatrix}$$

$$\begin{bmatrix}\textbf{regverb}\\ \text{PAST}:\top\\ \text{PASTP}:\boxed{2}\\ \text{PASSP}:\boxed{2}\end{bmatrix} / \begin{matrix}\{\langle\begin{bmatrix}\text{PAST}:\boxed{1}\\ \text{PASTP}:\boxed{1}\end{bmatrix},\textbf{verb}\rangle,\\ \langle\begin{bmatrix}\text{PAST}:\boxed{1}\\ \text{PASSP}:\boxed{1}\end{bmatrix},\textbf{verb}\rangle,\\ \langle\begin{bmatrix}\text{PAST}:\textbf{+ed}\end{bmatrix},\textbf{regverb}\rangle\}\end{matrix} \qquad \begin{bmatrix}\textbf{regverb}\\ \text{PAST}:\boxed{1}\ \textbf{+ed}\\ \text{PASTP}:\boxed{1}\\ \text{PASSP}:\boxed{1}\end{bmatrix}$$

$$\begin{bmatrix}\textbf{pst-t-verb}\\ \text{PAST}:\top\\ \text{PASTP}:\boxed{2}\\ \text{PASSP}:\boxed{2}\end{bmatrix} / \begin{matrix}\{\langle\begin{bmatrix}\text{PAST}:\boxed{1}\\ \text{PASTP}:\boxed{1}\end{bmatrix},\textbf{verb}\rangle,\\ \langle\begin{bmatrix}\text{PAST}:\boxed{1}\\ \text{PASSP}:\boxed{1}\end{bmatrix},\textbf{verb}\rangle,\\ \langle\begin{bmatrix}\text{PAST}:\textbf{+ed}\end{bmatrix},\textbf{regverb}\rangle,\\ \langle\begin{bmatrix}\text{PAST}:\textbf{+t}\end{bmatrix},\textbf{pst-t-vb}\rangle\}\end{matrix} \qquad \begin{bmatrix}\textbf{pst-t-verb}\\ \text{PAST}:\boxed{1}\ \textbf{+t}\\ \text{PASTP}:\boxed{1}\\ \text{PASSP}:\boxed{1}\end{bmatrix}$$

Figure 4
TDFSs and default structures after inheritance using YADU

below. $\overset{<>}{\sqcap}$ corresponds to unifying the indefeasible TFSs of the input TDFSs using the normal definition of typed unification, and taking the set union of their tails, with removal of elements which are inconsistent with the combined indefeasible structure. The full TDFSs corresponding to the constraints in Figure 3 after inheritance are shown in the left-hand column of Figure 4.

But note that the tails may contain conflicting information. In general, for any operation which makes use of default information, we want to know which elements in the tail 'win'. We therefore also define an operation on a TDFS, *DefFS*, which returns a single default TFS (corresponding to the default structure in PDU). The default structures corresponding to the TDFSs are shown in the right-hand column in Figure 4. A default TFS is calculated from a TDFS by unifying in the maximal set of compatible elements of the union of the tails to the non-default TFS in priority order. In this case, the priority order is given by the ordering in the type hierarchy, that is **pst-t-verb** is strictly more specific than **regverb**, which is strictly more specific than **verb**. In the case where there are no conflicts in the tail, this straightforwardly amounts to the unification of the atomic FSs in the tail, as in the structures shown for **verb** and **regverb** in Figure 4. For **pst-t-verb**, the intermediate results of successively unifying in the tail elements according to their specificity ordering are shown below.

$$1.\begin{bmatrix}\text{PAST}:\textbf{+t}\end{bmatrix} \sqcap \begin{bmatrix}\textbf{pst-t-vb}\\ \text{PAST}:\top\\ \text{PASTP}:\boxed{2}\\ \text{PASSP}:\boxed{2}\end{bmatrix} = \begin{bmatrix}\textbf{pst-t-vb}\\ \text{PAST}:\textbf{+t}\\ \text{PASTP}:\boxed{2}\\ \text{PASSP}:\boxed{2}\end{bmatrix}$$

$$2.\begin{bmatrix}\text{PAST}:\textbf{+ed}\end{bmatrix} \text{ is incompatible with } \begin{bmatrix}\textbf{pst-t-vb}\\ \text{PAST}:\textbf{+t}\\ \text{PASTP}:\boxed{2}\\ \text{PASSP}:\boxed{2}\end{bmatrix}$$

$$3.\{\begin{bmatrix}\text{PAST}:\boxed{1}\\ \text{PASSP}:\boxed{1}\end{bmatrix},\begin{bmatrix}\text{PAST}:\boxed{1}\\ \text{PASTP}:\boxed{1}\end{bmatrix}\} \sqcap \begin{bmatrix}\textbf{pst-t-vb}\\ \text{PAST}:\textbf{+t}\\ \text{PASTP}:\boxed{2}\\ \text{PASSP}:\boxed{2}\end{bmatrix} = \begin{bmatrix}\textbf{pst-t-vb}\\ \text{PAST}:\boxed{1}\textbf{+t}\\ \text{PASTP}:\boxed{1}\\ \text{PASSP}:\boxed{1}\end{bmatrix}$$

Thus the conflict between the information on **regverb** and **pst-t-verb** is resolved in the favor of the latter, since it is more specific. This is the reason for separating **verb** and **reg-**

verb in the example above, since we want the *+t* value to override the *+ed* information on **regverb** while leaving intact the default reentrancy which was specified on **verb**. If we had not done this, there would have been a conflict between the defaults which was not resolved by priority. In such cases, we take the generalization of the competing defaults, but for simplicity we leave the details to §3, where the definition of YADU is given.

Note that in examples such as this one, defaults are being used in order to capture lexical generalizations, but the suffixes seen by the morphological analyzer must be non-defeasible, otherwise incorrect strings such as *sleeped* would be accepted, because the default *+t* could be overridden. We refer to such defaults as non-persistent since the tails of the TDFSs must be incorporated into the indefeasible structure at the interface between the lexicon and the rest of the system. This is the analogue of the operation called *DefFill* in Lascarides *et al.* (1996), but, in contrast to PDU, YADU *DefFill* simply amounts to taking the defeasible structure, constructed as described above, since this is guaranteed to be a valid TFS (unlike in PDU) which is more specific than the indefeasible structure. For our purposes, it is convenient to define *DefFill* as returning a TDFS with an empty tail (see §3.3 and §4). For instance:

$$
\textit{DefFill}\left(\begin{bmatrix}\textbf{pst-t-vb}\\ \text{PAST} : \top\\ \text{PASTP} : \boxed{2}\\ \text{PASSP} : \boxed{2}\end{bmatrix} \Bigg/ \begin{matrix}\{\langle\begin{bmatrix}\text{PAST} : \boxed{1}\\ \text{PASTP} : \boxed{1}\end{bmatrix}, \textbf{verb}\rangle,\\ \langle\begin{bmatrix}\text{PAST} : \boxed{1}\\ \text{PASSP} : \boxed{1}\end{bmatrix}, \textbf{verb}\rangle,\\ \langle\begin{bmatrix}\text{PAST} : \textbf{+ed}\end{bmatrix}, \textbf{regverb}\rangle,\\ \langle\begin{bmatrix}\text{PAST} : \textbf{+t}\end{bmatrix}, \textbf{pst-t-vb}\rangle\}\end{matrix}\right) = \begin{bmatrix}\textbf{pst-t-vb}\\ \text{PAST} : \boxed{1}\textbf{+t}\\ \text{PASTP} : \boxed{1}\\ \text{PASSP} : \boxed{1}\end{bmatrix} /\{\}
$$

It will be apparent that the tail notation for TDFSs can become quite cumbersome. However, for cases where there are no conflicts in the tails and where the priority order corresponds to the type of the indefeasible TFS, we can use the slash notation that we initially introduced in Figure 1. So this notation can be used as a shorthand for the description of type constraints, and in fact it is generally convenient to use it in the description language, rather than to specify tails individually. We will formalize the relationship between the abbreviatory notation and the full TDFSs in §3, and use it when discussing examples where possible.

Before concluding this section, we should note that there is a contrast in behavior between the PDU and YADU operation, in that with YADU a default value could override a default reentrancy, which was not possible in the earlier work. The reason for making the reentrancy between PAST and PASSP default on **verb** was that these forms differ in some classes of irregular verbs (e.g. *speak, spoke, spoken*). In PDU, the default reentrancy could only have been overridden by a non-default value. Arguably however, the specification of the vowel change and suffixation for this class ought also to be defeasible, because of a subclass of these verbs, such as *weave*, which are also found with the regular suffixation pattern (*wove/weaved*, *woven/weaved*) (cf, Russell *et al.*, 1993). Since such a treatment of dual forms rests on a number of somewhat controversial assumptions, we will not give details here. However we will discuss an alternative example where default values must survive when default reentrancies are overridden in §4.

3. Typed Default Feature Structures and YADU

In §1.1, we argued in favor of criterion 1 for default unification—non-default information can be distinguished from default information and is always preserved—on the basis that we want the default portions of linguistic analyses to extend rather than replace existing monotonic analyses. But the standard language of TFSs doesn't make such distinctions between default and non-default information. In the previous section, we informally ex-

$$\mathbf{a} \sqcap \mathbf{b} = \bot \qquad \mathbf{a} \sqcap \mathbf{c} = \bot \qquad \mathbf{b} \sqcap \mathbf{c} = \bot$$

$$\left(\begin{bmatrix} \mathrm{F} : \mathbf{c} \\ \mathrm{G} : \mathbf{c} \end{bmatrix} \overset{<}{\sqcap} \begin{bmatrix} \mathrm{F} : \mathbf{a} \\ \mathrm{G} : \mathbf{b} \end{bmatrix} \right) \overset{<}{\sqcap} \begin{bmatrix} \mathrm{F} : \boxed{1} \\ \mathrm{G} : \boxed{1} \end{bmatrix} = \begin{bmatrix} \mathrm{F} : \boxed{1}\mathbf{c} \\ \mathrm{G} : \boxed{1} \end{bmatrix}$$

$$\begin{bmatrix} \mathrm{F} : \mathbf{c} \\ \mathrm{G} : \mathbf{c} \end{bmatrix} \overset{<}{\sqcap} \left(\begin{bmatrix} \mathrm{F} : \mathbf{a} \\ \mathrm{G} : \mathbf{b} \end{bmatrix} \overset{<}{\sqcap} \begin{bmatrix} \mathrm{F} : \boxed{1} \\ \mathrm{G} : \boxed{1} \end{bmatrix} \right) = \begin{bmatrix} \mathrm{F} : \mathbf{c} \\ \mathrm{G} : \mathbf{c} \end{bmatrix}$$

Figure 5
Non-associativity of Asymmetric Default Unification

tended the TFS language with tails. In this section, we give the formal definition of this richer representation scheme of TDFSs.

But first, we consider the interplay between criterion 1 and criterion 5—default unification is a binary order independent operation. This is because the demands placed by order independence affect the kind of representation scheme we require.

3.1 Requirements for order independence

As we mentioned in §1.2, the versions of default unification which are based on Kaplan's priority union (e.g., Carpenter 1993) are binary operations on TFSs, where the LHS TFS represents non-default information and the RHS TFS is default information. Roughly speaking, the result of unifying these two TFSs together is: the LHS TFS unified with as much information as possible from the RHS TFS while remaining consistent. An example is shown in Figure 5. Clearly, such an operation is non-commutative, because by changing the order of the arguments, one changes which information is default and which is non-default. The examples in Figure 5 also show that such an operation is non-associative.

Our demand for order independence requires us to define a symmetric operation. But this means that we cannot treat one TFS as specifying wholly indefeasible information and the other as wholly defeasible, as one does in Figure 5. Such an operation is never commutative, because changing the order of arguments changes what's default. So, one must explicitly mark in the information structures which information is to be treated as default and which is non-default. Thus, as demonstrated in Young and Rounds (1993) and Lascarides *et al.* (1996), criterion 1 above is a prerequisite to criterion 5.

But to achieve both criteria 5 and 6, further extensions to the language of TFSs are necessary. We must also keep track of certain default information that was overridden during unification, and we must keep track of the specificity of this overridden information. This is because this information must sometimes influence the results of subsequent default unifications if it is to meet the specified criteria. To see this, suppose we didn't keep track of this information. And suppose we attempt to default unify the three TFSs given in Figure 6, according to the principles laid out earlier, that one incorporates a maximal amount of default information according to its priority into the result.[2] Then the operation would be non-associative, as shown.

In Figure 6, when TFS$_2$ and TFS$_3$ are default unified together first, the fact that the default value **d** appeared on the attribute F in TFS$_3$ is lost in the result. And yet it's compatible with the most specific default information given in TFS$_1$, that is subsequently unified with this result, and so this value **d** should be incorporated into the final result

2 In this example and those following, we adopt the convention that **a, b, c** refer to types which correspond to values in untyped FSs: that is types for which no feature is appropriate. We'll refer to these as *simple value types* or *simple values*. We use **t, u, v** for types which may label a node which has features.

$$\mathbf{t}_1 \sqsubset \mathbf{t}_2 \sqsubset \mathbf{t}_3 \qquad \mathbf{a} \sqcap \mathbf{b} = \bot \qquad \mathbf{d} \sqcap \mathbf{b} = \bot \qquad \mathbf{a} \sqcap \mathbf{d} = \mathbf{c}$$

$$\textsc{tfs}_1\text{:} \begin{bmatrix} \mathbf{t}_1 \\ \text{F} : /\mathbf{a} \end{bmatrix} \qquad \textsc{tfs}_2\text{:} \begin{bmatrix} \mathbf{t}_2 \\ \text{F} : /\mathbf{b} \end{bmatrix} \qquad \textsc{tfs}_3\text{:} \begin{bmatrix} \mathbf{t}_3 \\ \text{F} : /\mathbf{d} \end{bmatrix}$$

$$\begin{aligned}
\textsc{tfs}_1 \overset{<>}{\sqcap} \textsc{tfs}_2 &= \begin{bmatrix} \mathbf{t}_1 \\ \text{F} : /\mathbf{a} \end{bmatrix} \\
(\textsc{tfs}_1 \overset{<>}{\sqcap} \textsc{tfs}_2) \overset{<>}{\sqcap} \textsc{tfs}_3 &= \begin{bmatrix} \mathbf{t}_1 \\ \text{F} : /\mathbf{c} \end{bmatrix} \\
\textsc{tfs}_2 \overset{<>}{\sqcap} \textsc{tfs}_3 &= \begin{bmatrix} \mathbf{t}_2 \\ \text{F} : /\mathbf{b} \end{bmatrix} \\
\textsc{tfs}_1 \overset{<>}{\sqcap} (\textsc{tfs}_2 \overset{<>}{\sqcap} \textsc{tfs}_3) &= \begin{bmatrix} \mathbf{t}_1 \\ \text{F} : /\mathbf{a} \end{bmatrix}
\end{aligned}$$

Figure 6
Non-Associativity without Tails

too. The fact that it's not contributes to the order dependence. Ensuring that **d** plays the necessary part in the unification operation is possible if upon unifying TFS_2 and TFS_3, we record the fact that F:/**d** with specificity $\mathbf{t}_3$ was in the unification history, although it's currently overridden by conflicting default values. Subsequent default unification operations on this result can then check these 'records', to see if this information that was overridden earlier should now be incorporated into the current result. The *tails* that we introduced in §2 serve this 'book keeping' purpose.

This maintenance of a partial history of default information is the cost we must pay for being able to define default unification as a binary order independent operation. The assumption of a binary operation is problematic since by the very definition of nonmonotonicity, one cannot in general divide the premises into groups, work out the inferences from those groups, and expect the inferences to survive the whole. The only way to gain order independence, therefore, is to ensure that each time the binary operation is performed, it is influenced by the necessary pieces of information that are 'outside' the immediate context of the FSs it is unifying. Tails cover this necessary information by recording sufficient information for us to define the default unification operation so that it has all the desired properties. In Lascarides *et al.* (1996), we argued in formal detail why tails were necessary. Here, we have just hinted at the motivation, by means of Figure 6. The formal discussion is quite lengthy and technical, and since it detracts from the main purpose of this paper, which is to produce a more powerful default unification than was presented there, we refer the reader to Lascarides *et al.* (1996) for the relevant formal motivation. Indeed, the tails we have defined here contain more information than those in our previous paper: specifically tails here record default reentrancies, which were previously recorded only in the default structure, without associated specificity information. This 'bigger' tail is the cost of defining a more powerful operation than PDU; one in which default reentrancies can be overridden by default values. However this also means that YADU does not require that default TFSs be part of the TDFS.

The purpose of this section is to extend the representation of typed information to go beyond TFSs in the required manner. We do this in a similar way to Lascarides *et al.* (1996), but omitting the default structure. As we suggested in §2, from an informal perspective, a TDFS contains a typed feature structure (TFS) which specifies what is indefeasible and tails which specify defeasible information. For instance in (1), the value of the path F·G is **b** by default, but the existence of the paths F·G and H, and the value **a** for H are non-default.

(1) $$\begin{bmatrix} \mathbf{t} \\ \text{F} : \begin{bmatrix} \text{G} : \top \end{bmatrix} \\ \text{H} : \mathbf{a} \end{bmatrix} \Big/ \{\langle \begin{bmatrix} \text{F} : \begin{bmatrix} \text{G} : \mathbf{b} \end{bmatrix} \end{bmatrix}, \mathbf{t}\rangle\}$$

In general, a tail consists of a set of pairs, where the first member of the pair is an atomic FS (that is, a single path or path equivalence) and the second member is a type. The atomic FS in such a pair must record default information which (a) is compatible with the indefeasible information in the TDFS; and (b) was introduced by a TDFS F, that was used as an argument to (at least) one of the series of default unifications that resulted in the TDFS being considered. The second member of the pair is the root type of F. The position of this root type in the type hierarchy, which we assume to be a finite bounded complete partial order (Carpenter, 1992), determines the *specificity* or priority of the default information given by the atomic FS. For original TDFSs (i.e., those TDFSs that aren't derived through default unification), the tails are strictly default information, in the sense that unifying it with the indefeasible information returns something more specific.[3] So during a series of default unifications, tails will record (strictly) default information from original TDFSs together with its specificity. For example, suppose a TDFS has a tail of the following form:

$$\begin{array}{l}\{ \quad \langle \begin{bmatrix} \mathrm{F} : \begin{bmatrix} \mathrm{G} : \mathbf{b} \end{bmatrix} \end{bmatrix}, \mathbf{t}_1 \rangle, \\ \quad\ \langle \begin{bmatrix} \mathrm{F} : \mathbf{a} \end{bmatrix}, \mathbf{t}_2 \rangle, \\ \quad\ \langle \begin{bmatrix} \mathrm{F} : \boxed{1} \\ \mathrm{G} : \boxed{1} \end{bmatrix}, \mathbf{t}_3 \rangle \}\end{array}$$

This means that to produce the TDFS with this tail, we unified: a TDFS that had a root type $\mathbf{t}_1$ and the strictly defeasible path F:G:**b**; a TDFS that had root type $\mathbf{t}_2$ and strictly defeasible path F:**a**; and a TDFS that had root type $\mathbf{t}_3$ and strictly defeasible information that the F- and G-paths share the same value. These TDFSs may have contained further information; e.g., the latter TDFS may have contained the information that not only did F- and G- by default share the same values, whatever they are, but also that indefeasibly, F's value is **c** and G's value is **c** (where **c** is a supertype of **a** and **b**) as given in the TDFS below:

(2) $$\begin{bmatrix} \mathbf{t}_3 \\ \mathrm{F} : \mathbf{c} \\ \mathrm{G} : \mathbf{c} \end{bmatrix} \Big/ \ \{\langle \begin{bmatrix} \mathrm{F} : \boxed{1} \\ \mathrm{G} : \boxed{1} \end{bmatrix}, \mathbf{t}_3 \rangle\}$$

The point is that tails don't record *all* the information that forms part of the unification history; it only records the strictly default information that's compatible with the LHS TFS in the TDFS. We explain this in more detail below.

3.2 The definition of a TDFS

A TDFS is a TFS (which intuitively represents the indefeasible information), plus a tail (which intuitively is a record of default information that played a part in building the TDFS, and which is compatible with the indefeasible TFS). We use the definition of unification in the definition of TDFSs, to constrain the relationship between the TFS and its tail in the required way (i.e., to ensure any atomic FS in the tail is compatible with the TFS). And the TFS in a TDFS is adapted from Carpenter's (1992) definition. The relevant definitions follow:

Definition 1
The Type Hierarchy
A type hierarchy is a finite bounded complete partial order $\langle \mathsf{Type}, \sqsubseteq \rangle$.

Definition 2
Typed Feature Structures
A typed feature structure defined on a set of features Feat, a type hierarchy $\langle \mathsf{Type}, \sqsubseteq \rangle$ and a set of indices N is a tuple $\langle Q, r, \delta, \theta \rangle$, where:

- Q is a finite set of nodes,
- $r \in Q$ (this is the root node; see conditions 1 and 2 below)
- $\theta : Q \longrightarrow \mathsf{Type}$ is a partial typing function (this labels nodes with types).

3 So the elements of the tail of a derived TDFS (i.e., derived via default unification) are also strictly default, unless somewhere in the unification history one unified two TDFSs F_1 and F_2 where the indefeasible information in F_1 is strictly default in F_2.

- $\delta : Q \times \mathsf{Feat} \longrightarrow Q$ is a partial feature value function (this connects nodes with arcs labeled with features).
  Note that for notational convenience we may write a sequence of features $F_1 \ldots F_k$ as π, and $\delta(\ldots(\delta(n, F_1), F_2) \ldots F_k)$ as $\delta(n, \pi)$.)

Furthermore, the following must hold:

1. r isn't a δ-descendant.
2. All members of Q except r are δ-descendants of r.
3. There is no node n or path π such that $\delta(n, \pi) = n$.

(Conditions 1. and 2. ensure we have a directed graph rooted at r. Condition 3. ensures this directed graph is acyclic.)

The unification operation $\sqcap$ on TFSs is defined here along the lines of Carpenter (1993), in terms of subsumption on typed feature structures. First a little notation: Let F be a TFS. Then $\pi \equiv_F \pi'$ means that F contains path equivalence or reentrancy between the paths π and π' (i.e., $\delta(n, \pi) = \delta(n, \pi')$ where n is the root node of F); and $\mathcal{P}_F(\pi) = \sigma$ means that the type on the path π in F is σ (i.e., $\mathcal{P}_F(\pi) = \sigma$ if and only if $\theta(\delta(n, \pi)) = \sigma$, where n is the root node of F). Subsumption is then defined as follows:

Definition 3
Subsumption of typed feature structures
F subsumes F', written $F' \sqsubseteq F$, if and only if:

- $\pi \equiv_F \pi'$ implies $\pi \equiv_{F'} \pi'$
- $\mathcal{P}_F(\pi) = t$ implies $\mathcal{P}_{F'}(\pi) = t'$ and $t' \sqsubseteq t$

The subsumption hierarchy is a bounded complete partial order.

Unification is defined in terms of subsumption:

Definition 4
Unification
The unification $F \sqcap F'$ of two feature structures F and F' is taken to be the greatest lower bound of F and F' in the collection of feature structures ordered by subsumption.

Thus $F \sqcap F' = F''$ if and only if $F'' \sqsubseteq F$, $F'' \sqsubseteq F'$ and for every F''' such that $F''' \sqsubseteq F$ and $F''' \sqsubseteq F'$ it is also the case that $F''' \sqsubseteq F''$.

With this groundwork in place, we can now give the formal definition of TDFSs:

Definition 5
Typed Default Feature Structures
A typed default feature structure defined on a set of features Feat, a type hierarchy $\langle \mathsf{Type}, \sqsubseteq \rangle$ and a set of indices N is a tuple $\langle I, T \rangle$ where:

- I is a typed feature structure $\langle Q, r, \theta, \delta \rangle$ on a set of features Feat, a type hierarchy $\langle \mathsf{Type}, \sqsubseteq \rangle$ and a set of indices N, as defined in Definition 2 (i.e., it's a rooted directed acyclic graph);
- $\mathcal{T}$ is a tail: that is, it is a set of pairs, where:

—The first member of the pair is an atomic FS, as defined in (Carpenter, 1993);
that is, a single path, or a path equivalence; and
—the second member of the pair is a type;

Furthermore, for any element $\langle F, t\rangle \in T$, $I \sqcap F \neq \bot$. (This condition ensures that all the atomic FSs in the tails are compatible with the indefeasible information. However, these atomic FSs may be incompatible with each other).

Note that in this paper we make use of a notational convention, that a TDFS $\langle I, T\rangle$ can be written as I/T.

As we have mentioned before, we will use the tail of a TDFS as a device for keeping a record of the default information from the TDFSs which through default unification were used to form the TDFS in question. The definition of default unification we give below will ensure that tails do indeed provide this record keeping service. However, since tails can contain mutually incompatible information, a TDFS (and hence the result of the default unification operation), does not explicitly represent which default information in the tail 'wins'. So as well as defining default unification, we also define an operation on TDFSs, which we call *DefFS* (standing for default FS), which uses the indefeasible TFS and the tail to produce a TFS which represents the default information corresponding to that TDFS. That is, it determines which elements of the tail win. Because this TFS represents the default information, *DefFS*(I/T) is called a default TFS (relative to I/T).

3.3 The definition of $\overset{<>}{\sqcap}$ and DefFS

We now formally describe the operations $\overset{<>}{\sqcap}$ and *DefFS*, and in §3.7 and the Appendix we will prove that these have the properties we set out in §1.1.

The operation $\overset{<>}{\sqcap}$ operates over TDFSs to produce a TDFS. $\overset{<>}{\sqcap}$ is built up from a combination of set union and of the unification operation $\sqcap$ which operates on TFSs (that is, typed feature structures as defined in Carpenter (1992)). To define $\overset{<>}{\sqcap}$, we also need to talk about the first and second members of the pairs in the tail:

Definition 6
Projection of Tails
Let T be a tail; i.e., a member of atomic FSs $\times$ Type. Then:

$$\begin{aligned} \wp_{fs}(T) = &\quad \{\phi : \langle \phi, t\rangle \in T\} \text{ and} \\ \wp_t(T) = &\quad \{t : \langle \phi, t\rangle \in T\} \end{aligned}$$

Informally, $\overset{<>}{\sqcap}$ takes two TDFSs, and produces a new TDFS I/T, where I is the (monotonic) unification of the TFSs in the argument TDFSs, and T is the union of the tails, with all information that's incompatible with I removed. In other words, $\overset{<>}{\sqcap}$ serves to accumulate the indefeasible information in I, and accumulate the defaults that are compatible with I in T. The formal definition is as follows:

Definition 7
$\overset{<>}{\sqcap}$
Let $F_1 =_{def} I_1/T^1$ and $F_2 =_{def} I_2/T^2$ be two TDFSs, and let $F_{12} =_{def} F_1 \overset{<>}{\sqcap} F_2$. Furthermore, assume $F_{12} =_{def} I_{12}/T^{12}$. Then I_{12} and T^{12} are calculated as follows:

1. The Indefeasible Part:
$I_{12} = I_1 \sqcap I_2$

That is, the indefeasible TFS is the unification of the indefeasible parts of the arguments.

2. The Tail T^{12}:
$T^{12} =_{def} (T^1 \cup T^2) \setminus Bot^{12}$
where Bot^{12} are all the elements T of $T^1 \cup T^2$ where the atomic FS $\wp_{fs}(T)$ is incompatible with I_{12}. That is:

$$Bot^{12} = \{T \in (T^1 \cup T^2) : \wp_{fs}(T) \sqcap I_{12} = \bot\}$$

$F_1 \overset{<>}{\sqcap} F_2$ returns a TDFS F_{12}. F_{12} represents the indefeasible information in both F_1 and F_2. Also, through the tail T^{12}, it represents the defeasible information which can potentially play a role in defining what holds by default—either in F_{12} itself, or in some other TDFS which is built from F_{12} via $\overset{<>}{\sqcap}$. As we mentioned before, we need to define a further operation which computes which elements in the tail T^{12} provide the winning default information relative to F_{12}. This is done in the *DefFS* operation given below, which computes a (default) TFS from a TDFS.

DefFS is built up from a combination of the unification and generalization operations $\sqcap$ and $\sqcup$ which operate on TFSs. Generalization is the opposite of unification. The generalization of two feature structures is defined to be the most specific feature structure which contains only information found in both feature structures.

Definition 8
Generalization
The generalization $F \sqcup F'$ of two feature structures is defined to be their lowest upper bound in the subsumption ordering.

Thus $F \sqcup F' = F''$ if and only if $F \sqsubseteq F''$, $F' \sqsubseteq F''$ and for every F''' such that $F \sqsubseteq F'''$ and $F' \sqsubseteq F'''$ then $F'' \sqsubseteq F'''$.

The operation *DefFS* on TDFSs also uses an extended version of Carpenter's (1993) credulous default unification, which in turn is defined in terms of $\sqcap$. We extend it here, in that the second argument of the operation will be a set of atomic FSs that may be mutually incompatible, instead of it being a single FS.

Carpenter's definition of credulous default unification is as follows:

Definition 9
Carpenter's Definition of Credulous Default Unification
The result of credulously adding the default information in G to the strict information in F is given by:

$F \overset{<}{\sqcap}_c G = \{F \sqcap G' : G' \sqsupseteq G$ is maximally specific wrt the subsumption hierarchy such that $F \sqcap G'$ is defined$\}$

An example of the operation is given in (3).

(3) Where $\mathbf{t}' \sqsubset \mathbf{t}$:

$$\begin{bmatrix} \mathbf{t} \\ \text{F} : \mathbf{a} \\ \text{G} : \top \end{bmatrix} \overset{<}{\sqcap}_c \begin{bmatrix} \mathbf{t}' \\ \text{F} : \boxed{1}\ \mathbf{b} \\ \text{G} : \boxed{1} \end{bmatrix} = \left\{ \begin{bmatrix} \mathbf{t}' \\ \text{F} : \mathbf{a} \\ \text{G} : \mathbf{b} \end{bmatrix}, \begin{bmatrix} \mathbf{t}' \\ \text{F} : \boxed{1}\ \mathbf{a} \\ \text{G} : \boxed{1} \end{bmatrix} \right\}$$

One can see that credulous default unification is asymmetric, and doesn't return a single result deterministically. So it doesn't meet the criteria we specified in §1.1. Nevertheless, *DefFS* will have the desired properties, even though its defined in terms of $\overset{<}{\sqcap}_c$. The extended definition which works on a set of atomic FSs is given below:

$$F_1 \overset{<}{\sqcap}_{ca} \{G_1, \ldots, G_n\} = \{F_1 \sqcap F_2 : \; F_2 \text{ is the unification of a maximal subset of } \{G_1, \ldots, G_n\} \text{ such that } F_1 \sqcap F_2 \text{ is defined}\}$$

We will want to iterate credulous default unification in what happens below, and for this purpose, we define $\overset{<}{\sqcap}_{cs}$ as follows:

Definition 10
Credulous Default Unification $\overset{<}{\sqcap}_{cs}$ on Sets
Let $\mathcal{F}_1$ be a *set* of TFSs $\{F_1, \ldots, F_n\}$, and $\mathcal{G}_2$ a set of atomic FSs. Then

$$\mathcal{F}_1 \overset{<}{\sqcap}_{cs} \mathcal{G}_2 = \{F_1 \overset{<}{\sqcap}_{ca} \mathcal{G}_2, \ldots, F_n \overset{<}{\sqcap}_{ca} \mathcal{G}_2\}$$

The definition of *DefFS* will manipulate tails in a certain way, and the following way of partitioning tails plays an important role in defining how defaults behave under *DefFS*:

Definition 11
The Specificity Partition of a Tail
Let T be a tail. Then $\mu_1, \ldots \mu_m$ is a Specificity Partition of T if:

$$T = \mu_1 \cup \ldots \cup \mu_m$$

and

$$\begin{aligned} \mu_1 = \; & \{\langle \phi, t\rangle \in T : \; \forall \langle \phi', t'\rangle \in T, t' \not\sqsubset t\} \\ \mu_i = \; & \{\langle \phi, t\rangle \in T : \; \exists \langle \phi', t'\rangle \in \mu_{i-1} \text{ where } t' \sqsubset t \text{ and} \\ & \qquad \forall \langle \phi'', t''\rangle \in T \setminus (\mu_1 \cup \ldots \cup \mu_{i-1}), t'' \not\sqsubset t\} \quad 2 \leq i \leq m \end{aligned}$$

Intuitively, μ_1 is the set of pairs in T with the most specific types, μ_2 is the set of pairs in T with the next most specific types, and so on. Partitioning tails in order of specificity enables us to define *DefFS* so that default information on more specific types overrides that on more general types.

Having set up this groundwork, we're now in a position to define *DefFS* on TDFSs in terms of $\sqcup$ and $\overset{<}{\sqcap}_{cs}$ on TFSs.

Definition 12
The Operation DefFS
Let F be a TDFS I/T. Then

$$\mathit{DefFS}(F) = \sqcup((I \overset{<}{\sqcap}_{cs} \wp_{fs}(\mu_1)) \overset{<}{\sqcap}_{cs} \ldots \overset{<}{\sqcap}_{cs} \wp_{fs}(\mu_n))$$

Where $\langle \mu_1, \ldots, \mu_n\rangle$ is a specificity partition on T.

According to this definition, $\mathit{DefFS}(F)$ is the generalization of the 'credulous default unification' of: I; the atomic FSs in μ_1; ...; the atomic FSs in μ_m, in that order. By notational convention, we occasionally write D_i for $\mathit{DefFS}(F_i)$. So $\mathit{DefFS}(F_{12}) =_{def} \mathit{DefFS}(I_{12}/T^{12}) =_{def} \mathit{DefFS}(F_1 \overset{<>}{\sqcap} F_2) =_{def} D_{12}$.

Note that in Lascarides *et al.* (1996), we defined the default structure as part of the TDFS. We could have done this here too,[4] but the current formulation makes the contrast

4 and in fact did so in earlier drafts of this paper ...

with PDU clearer, since in YADU the default structure does not carry any information additional to the tail—it is calculated purely in order to see which parts of the tail win, in the current structure. We will prove formally that $\overset{<>}{\sqcap}$ together with this definition *DefFS* for computing default structures achieve the effects we desire in §3.7 and the appendix. Note further that this means the implementation of the $\overset{<>}{\sqcap}$ operation itself is a trivial extension to ordinary unification. Constructing the default structure is more complex, but this is not a necessary part of the actual unification step. For instance, in the use of defaults described in §2, the default structure is only calculated once for a given lexical entry: it is not necessary to calculate default structures for intermediate types in the hierarchy, for instance. We will discuss this further in §5.

For completeness, we also define the operation *DefFill*, which takes a TDFS and returns a TDFS with the default information incorporated. This is required for non-persistent defaults, as discussed in §4.

Definition 13
The Operation DefFill
Let F be a TDFS I/T. Then

$$\mathit{DefFill}(F) = \mathit{DefFS}(F)/\{\}$$

With respect to the definitions we have given in this section, we should note that we're using an extremely weak notion of typing, and not imposing any appropriateness conditions on the TFSs or requiring that unification maintain well-formedness. The basic strategy for making the definitions of $\overset{<>}{\sqcap}$ and *DefFS* respect well-formedness conditions on types is straightforward, since the set of well-formed TDFSs can be defined as the set of TDFSs for which the indefeasible component meets the applicable constraints, and the use of $\sqcap$ in the definitions can be replaced by a variant of unification which maintains well-formedness, hence guaranteeing that the default feature structure is also well-formed. However, we omit the details, since we want to keep the definition of the default operation as general as possible, and there are a number of different constraint languages within feature structure frameworks (e.g., Alshawi *et al.* (1991), Carpenter (1992), Dörre and Eisle (1991), Emele and Zajac (1990), Gerdemann and King (1993), Krieger and Schäfer (1994), de Paiva (1993), Smolka (1989)). In fact, in the examples in this paper, the only use we make of types is functionally equivalent to the use of templates in untyped FS systems, so going into the details of well-formedness with respect to one particular system seems redundant.

3.4 Unification at non-root nodes

In grammars based on the use of (T)FSs, it is often necessary to be able to unify structures at nodes other than the root. For example, if grammar rules or schemata are described as TFSs (as we will assume in §4), then the instantiation of a rule by a daughter will involve unifying a substructure TFS in the rule with the complete TFS for the daughter. This is straightforward for TFSs, because $\delta(r,\pi)$, if defined, is itself the root node of a TFS:

Definition 14
SubTFS
Let $I = \langle Q, r, \delta, \theta\rangle$ be a TFS, and let $\delta(r,\pi)$ be defined. Then the sub-TFS I' of I that's rooted at $\delta(r,\pi)$, written SubTFS(I,π), is the TFS $\langle Q', \delta(r,\pi), \delta', \theta'\rangle$, where:

1. $Q' \subseteq Q$ are those nodes that are δ-descendants of $\delta(r,\pi)$;
2. δ' is the projection of δ onto Q'; and

3. θ' is the projection of θ onto Q'.

For TDFSs, the situation is slightly more complex, because the definition of substructure has to include relevant elements of the tail, as well as the indefeasible (sub)structure.

Definition 15
SubTDFS
Let F be a TDFS I/T and π be a path. Then

$$SubTDFS(F,\pi) = SubTFS(I,\pi)/SubTail(T,\pi)$$

Where

$$SubTail(T,\pi) = \{\langle F',t\rangle \in T : \begin{array}{l}\text{There is an element } \langle F,t\rangle \in T \text{ such that}\\ r \text{ is the root node of } F \text{ and } \delta(r,\pi) \text{ is defined; and}\\ F' = SubTFS(F,\pi)\}\end{array}$$

Similarly, we can define the prefixation of a TDFS by a path, by utilizing the following definition:

Definition 16
SuperTFS
Let $I = \langle Q, r, \delta, \theta\rangle$ be a TFS and $\pi = F_1 \dots F_n$ be a path. Then $SuperTFS(I,\pi)$, which is the TFS prefixed with the path π, is $\langle Q', r', \delta', \theta'\rangle$, such that:

1. $Q' = Q \cup r' \cup \{q_1, \dots, q_{n-1}\}$

2. δ' is a partial feature value function defined on Q', such that $\delta'(r', F_1) = q_1$, $\delta'(q_1, F_2) = q_2, \dots,$ $\delta'(q_{n-2}, F_{n-1}) = q_{n-1}, \delta'(q_{n-1}, F_n) = r$; and $\delta'(q, F) = \delta(q, F)$ for all $q \in Q$ and all features F.

3. θ' is a partial typing function on Q', such that $\theta(q) = \theta(q)$ for all $q \in Q$, and is undefined otherwise.

Definition 17
SuperTDFS
Let F be a TDFS I/T and π be a path. Then

$$\begin{array}{rl} SuperTDFS(F,\pi) = & SuperTFS(I,\pi)/T' \text{ such that}\\ T' = & \{\langle F',t\rangle : \langle F,t\rangle \in T \text{ and } F' = SuperTFS(F,\pi)\}\end{array}$$

Note that in both *SubTDFS* and *SuperTDFS*, the specificity of the tail elements is left unchanged and thus the specificity of a tail element may not be in a partial order relationship with the root type of the indefeasible structure.

3.5 Basic TDFSs and abbreviatory conventions

When initially specifying TDFSs, as type constraints for instance, it is useful to allow tails to be constructed from pairs of TFSs, where the first member of the pair is regarded as indefeasible and the second member is treated as defeasible. Obviously it only makes sense to do this if the default TFS is compatible with the indefeasible structure, so that the defeasible information FS_D augments the indefeasible information FS_I in some respects. We will assume that the tail is the most specific strictly default information that's specified by the indefeasible and defeasible TFSs (you can compute this by checking which information is in $FS_I \sqcap FS_D$ and absent in FS_I), plus the priority given by the (default) root type (which means that the type on the root node of either FS_I or FS_D must be defined). We will call the TDFS that's constructed in this way from the indefeasible TFS FS_I and defeasible TFS FS_D a *basic* TDFS. Basic TDFSs are special TDFSs, because there will be no conflict in the tails (since the tail is derived from two compatible TFSs).

Definition 18
Basic TDFSs
Let FS_I and FS_D be typed feature structures, where FS_I is regarded as indefeasible and FS_D as defeasible. Furthermore, suppose that the root node of FS_I is typed, or the root node of FS_D is typed, and that $FS_I \sqcap FS_D \neq \bot$ (so FS_I and FS_D are compatible). Then the basic TDFS $BasicTDFS(FS_I, FS_D)$ of FS_I and FS_D is the TDFS FS_I/T, such that:

$$\begin{array}{ll} T = \{\langle F, t\rangle : & t \text{ is the root type on } FS_D \sqcap FS_I \text{, and } F \text{ is an atomic TFS such that:} \\ & \text{(a) } FS_I \not\sqsubseteq F; \\ & \text{(b) } FS_D \sqcap FS_I \sqsubseteq F \text{; and} \\ & \text{(c) there's no other atomic FS } F' \text{ such that } F' \sqsubset F \text{ and } F' \text{ satisfies} \\ & \text{conditions (a) and (b)}\} \end{array}$$

Note that the basic TDFS derived from FS_I and FS_D is indeed a TDFS, since FS_I and T satisfy all the conditions given in Definition 5. In particular, for any $\langle F, t\rangle \in T$, $F \sqcap FS_I \neq \bot$, since $FS_D \sqcap FS_I \neq \bot$. and $I_D \sqcap FS_I \sqsubseteq F \sqcap FS_I$. It's also important to stress that in general, $DefFS(BasicTDFS(FS_I, FS_D)) \neq FS_D$. To see this, note that it follows from the definition of *DefFS* and the fact that all elements in T are compatible with FS_I that $DefFS(FS_I/T) \sqsubseteq FS_I$. But it's not necessarily the case that $FS_D \sqsubseteq FS_I$ because FS_D will not in general repeat the information in FS_I. However, we will see in §3.7 and the appendix that $FS_I \sqcap FS_D = DefFS(BasicTDFS(FS_I, FS_D))$.

For basic TDFSs, we can make some abbreviations, as suggested in §2. That is we can use a single AVM representation where we have *IndefeasibleValue*/*DefeasibleValue* (*IndefeasibleValue* is omitted when it's $\top$, and the slash is omitted when *IndefeasibleValue* = *DefeasibleValue*). Thus, for example, (4) can be represented as (5).

(4) $$\begin{bmatrix} \mathbf{t} \\ \text{F} : \begin{bmatrix} \text{G} : \top \end{bmatrix} \\ \text{H} : \mathbf{a} \end{bmatrix} \Big/ \{\langle \begin{bmatrix} \text{F} : \begin{bmatrix} \text{G} : \mathbf{b} \end{bmatrix} \end{bmatrix}, \mathbf{t}\rangle\}$$

(5) $$\begin{bmatrix} \mathbf{t} \\ \text{F} : \begin{bmatrix} \text{G} : /\mathbf{b} \end{bmatrix} \\ \text{H} : \mathbf{a} \end{bmatrix}$$

Where appropriate, we will use such abbreviations in the examples which follow.

3.6 Examples

We will prove that the operation has the properties specified in §1.1 in the next section. But first, we will illustrate how the operation works by means of specific examples.

3.6.1 Specificity. The following example demonstrates that default reentrancies are overridden by conflicting default information of a more specific type.

Where $\mathbf{t} \sqsubset \mathbf{t}'$ and $\mathbf{a} \sqcap \mathbf{b} = \bot$

We will calculate:

$$\begin{bmatrix}\mathbf{t}\\ \text{F} : /\mathbf{a}\\ \text{G} : /\mathbf{b}\end{bmatrix} \overset{<>}{\sqcap} \begin{bmatrix}\mathbf{t}'\\ \text{F} : /\boxed{1}\\ \text{G} : /\boxed{1}\end{bmatrix}$$

and the defeasible result of this default unification. I.e.:

$$\mathit{DefFS}(\begin{bmatrix}\mathbf{t}\\ \text{F} : /\mathbf{a}\\ \text{G} : /\mathbf{b}\end{bmatrix} \overset{<>}{\sqcap} \begin{bmatrix}\mathbf{t}'\\ \text{F} : /\boxed{1}\\ \text{G} : /\boxed{1}\end{bmatrix})$$

- $I_{12} = \begin{bmatrix}\mathbf{t}\\ \text{F} : \top\\ \text{G} : \top\end{bmatrix} \sqcap \begin{bmatrix}\mathbf{t}'\\ \text{F} : \top\\ \text{G} : \top\end{bmatrix} = \begin{bmatrix}\mathbf{t}\\ \text{F} : \top\\ \text{G} : \top\end{bmatrix}$

- $$\begin{array}{lll} T^{12} & = & (T^1 \cup T^2) \setminus \mathit{Bot}^{12} \\ T^1 & = & \{\langle [\,\text{F} : \mathbf{a}\,], \mathbf{t}\rangle, \langle [\,\text{G} : \mathbf{b}\,], \mathbf{t}\rangle\} \\ T^2 & = & \{\langle \begin{bmatrix}\text{F} : \boxed{1}\\ \text{G} : \boxed{1}\end{bmatrix}, \mathbf{t}'\rangle\} \\ \mathit{Bot}^{12} & = & \emptyset \\ \text{So} & & \\ T^{12} & = & (T^1 \cup T^2) \setminus \mathit{Bot}^{12} \\ & = & \{\langle [\,\text{F} : \mathbf{a}\,], \mathbf{t}\rangle, \langle [\,\text{G} : \mathbf{b}\,], \mathbf{t}\rangle, \langle \begin{bmatrix}\text{F} : \boxed{1}\\ \text{G} : \boxed{1}\end{bmatrix}, \mathbf{t}'\rangle\} \end{array}$$

Therefore

$$\begin{bmatrix}\mathbf{t}\\ \text{F} : /\mathbf{a}\\ \text{G} : /\mathbf{b}\end{bmatrix} \overset{<>}{\sqcap} \begin{bmatrix}\mathbf{t}'\\ \text{F} : /\boxed{1}\\ \text{G} : /\boxed{1}\end{bmatrix} = \begin{bmatrix}\mathbf{t}\\ \text{F} : \top\\ \text{G} : \top\end{bmatrix} \Big/ \begin{array}{l}\{\langle [\,\text{F} : \mathbf{a}\,], \mathbf{t}\rangle, \langle [\,\text{G} : \mathbf{b}\,], \mathbf{t}\rangle, \\ \quad \langle \begin{bmatrix}\text{F} : \boxed{1}\\ \text{G} : \boxed{1}\end{bmatrix}, \mathbf{t}'\rangle\}\end{array}$$

We now calculate the resulting defeasible structure:

- The specificity partition of T^{12} is $\langle \mu_1, \mu_2\rangle$, where:

$$\begin{array}{ll} \mu_1 = & \{\langle [\,\text{F} : \mathbf{a}\,], \mathbf{t}\rangle, \langle [\,\text{G} : \mathbf{b}\,], \mathbf{t}\rangle\} \\ \mu_2 = & \{\langle \begin{bmatrix}\text{F} : \boxed{1}\\ \text{G} : \boxed{1}\end{bmatrix}, \mathbf{t}'\rangle\} \end{array}$$

- So

$$\begin{array}{lll} D_{12} =_{def} \mathit{DefFS}(I_{12}/T^{12}) & = & \sqcup((I_{12} \overset{<}{\sqcap}_{cs} \wp_{fs}(\mu_1)) \overset{<}{\sqcap}_{cs} \wp_{fs}(\mu_2)) \\ & = & \sqcup\Bigg(\Bigg(\begin{bmatrix}\mathbf{t}\\ \text{F} : \top\\ \text{G} : \top\end{bmatrix} \overset{<}{\sqcap}_{cs} \{\ [\,\text{F} : \mathbf{a}\,], [\,\text{G} : \mathbf{b}\,]\ \} \Bigg) \\ & & \qquad \overset{<}{\sqcap}_{cs} \begin{bmatrix}\text{F} : \boxed{1}\\ \text{G} : \boxed{1}\end{bmatrix} \Bigg) \\ & = & \sqcup\Bigg(\begin{bmatrix}\mathbf{t}\\ \text{F} : \mathbf{a}\\ \text{G} : \mathbf{b}\end{bmatrix} \overset{<}{\sqcap}_{cs} \begin{bmatrix}\text{F} : \boxed{1}\\ \text{G} : \boxed{1}\end{bmatrix} \Bigg) \\ & = & \begin{bmatrix}\mathbf{t}\\ \text{F} : \mathbf{a}\\ \text{G} : \mathbf{b}\end{bmatrix} \end{array}$$

Hence

$$\begin{array}{lll} D_{12} & = & \mathit{DefFS}(\begin{bmatrix}\mathbf{t}\\ \text{F} : /\mathbf{a}\\ \text{G} : /\mathbf{b}\end{bmatrix} \overset{<>}{\sqcap} \begin{bmatrix}\mathbf{t}'\\ \text{F} : /\boxed{1}\\ \text{G} : /\boxed{1}\end{bmatrix}) \\ & = & \mathit{DefFS}(\begin{bmatrix}\mathbf{t}\\ \text{F} : \top\\ \text{G} : \top\end{bmatrix} \Big/ \ \{\langle [\,\text{F} : \mathbf{a}\,], \mathbf{t}\rangle, \langle [\,\text{G} : \mathbf{b}\,], \mathbf{t}\rangle, \langle \begin{bmatrix}\text{F} : \boxed{1}\\ \text{G} : \boxed{1}\end{bmatrix}, \mathbf{t}'\rangle\}\ ) \\ & = & \begin{bmatrix}\mathbf{t}\\ \text{F} : \mathbf{a}\\ \text{G} : \mathbf{b}\end{bmatrix} \end{array}$$

3.6.2 The Nixon Diamond. The credulous unification operation $\overset{<}{\sqcap}_{cs}$ doesn't generally return a deterministic result. However, $\overset{<>}{\sqcap}$ does, even though it's defined in terms of $\overset{<}{\sqcap}_{cs}$. This is because $\overset{<>}{\sqcap}$ generalizes over the credulous options. The following example illustrates this.

Where $\mathbf{t}$ and $\mathbf{t}'$ are in no specificity ordering, and

$$\begin{aligned} \mathbf{t} \sqcap \mathbf{t}' &= \mathbf{t}'' \\ \mathbf{a} \sqcap \mathbf{b} &= \bot \\ \mathbf{a} \sqcup \mathbf{b} &= \mathbf{c} \end{aligned}$$

We will calculate:

$$\begin{bmatrix} \mathbf{t} \\ \text{F} : /\mathbf{a} \\ \text{G} : /\mathbf{b} \end{bmatrix} \overset{<>}{\sqcap} \begin{bmatrix} \mathbf{t}' \\ \text{F} : /\boxed{1} \\ \text{G} : /\boxed{1} \end{bmatrix}$$

and the defeasible result of this unification. I.e.:

$$\mathit{DefFS}(\begin{bmatrix} \mathbf{t} \\ \text{F} : /\mathbf{a} \\ \text{G} : /\mathbf{b} \end{bmatrix} \overset{<>}{\sqcap} \begin{bmatrix} \mathbf{t}' \\ \text{F} : /\boxed{1} \\ \text{G} : /\boxed{1} \end{bmatrix})$$

- $$\begin{aligned} I_{12} = & \begin{bmatrix} \mathbf{t} \\ \text{F} : \top \\ \text{G} : \top \end{bmatrix} \sqcap \begin{bmatrix} \mathbf{t}' \\ \text{F} : \top \\ \text{G} : \top \end{bmatrix} \\ = & \begin{bmatrix} \mathbf{t}'' \\ \text{F} : \top \\ \text{G} : \top \end{bmatrix} \end{aligned}$$

- $$\begin{aligned} T^{12} &= (T^1 \cup T^2) \setminus \mathit{Bot}^{12} \\ T^1 &= \{\langle [\text{F} : \mathbf{a}], \mathbf{t}\rangle, \langle [\text{G} : \mathbf{b}], \mathbf{t}\rangle\} \\ T^2 &= \{\langle \begin{bmatrix} \text{F} : \boxed{1} \\ \text{G} : \boxed{1} \end{bmatrix}, \mathbf{t}'\rangle\} \\ \mathit{Bot}^{12} &= \emptyset \\ \text{So} & \\ T^{12} &= (T^1 \cup T^2) \setminus \mathit{Bot}^{12} \\ &= \{\langle [\text{F} : \mathbf{a}], \mathbf{t}\rangle, \langle [\text{G} : \mathbf{b}], \mathbf{t}\rangle, \langle \begin{bmatrix} \text{F} : \boxed{1} \\ \text{G} : \boxed{1} \end{bmatrix}, \mathbf{t}'\rangle\} \end{aligned}$$

Hence

$$\begin{bmatrix} \mathbf{t} \\ \text{F} : /\mathbf{a} \\ \text{G} : /\mathbf{b} \end{bmatrix} \overset{<>}{\sqcap} \begin{bmatrix} \mathbf{t}' \\ \text{F} : /\boxed{1} \\ \text{G} : /\boxed{1} \end{bmatrix} = \begin{bmatrix} \mathbf{t}'' \\ \text{F} : \top \\ \text{G} : \top \end{bmatrix} \Big/ \begin{array}{l} \{\langle [\text{F} : \mathbf{a}], \mathbf{t}\rangle, \langle [\text{G} : \mathbf{b}], \mathbf{t}\rangle, \\ \quad \langle \begin{bmatrix} \text{F} : \boxed{1} \\ \text{G} : \boxed{1} \end{bmatrix}, \mathbf{t}'\rangle\} \end{array}$$

We now calculate the resulting defeasible structure:

- The specificity partition of T^{12} is T^{12} itself (since both $\mathbf{t}$ and $\mathbf{t}'$ are the most specific types in T^{12}).

- $$\begin{aligned} D_{12} =_{def} \mathit{DefFS}(F_{12}) &= \sqcup((I_{12} \overset{<}{\sqcap}_{cs} \wp_{fs}(T^{12}))) \\ &= \sqcup\left(\begin{bmatrix} \mathbf{t}'' \\ \text{F} : \top \\ \text{G} : \top \end{bmatrix} \overset{<}{\sqcap}_{cs} \right. \\ &\qquad \left. \left\{ [\text{F} : \mathbf{a}], [\text{G} : \mathbf{b}], \begin{bmatrix} \text{F} : \boxed{1} \\ \text{G} : \boxed{1} \end{bmatrix} \right\} \right) \\ &= \sqcup\left(\left\{ \begin{bmatrix} \mathbf{t}'' \\ \text{F} : \mathbf{a} \\ \text{G} : \mathbf{b} \end{bmatrix}, \begin{bmatrix} \mathbf{t}'' \\ \text{F} : \boxed{1}\mathbf{a} \\ \text{G} : \boxed{1} \end{bmatrix}, \begin{bmatrix} \mathbf{t}'' \\ \text{F} : \boxed{1}\mathbf{b} \\ \text{G} : \boxed{1} \end{bmatrix} \right\} \right) \\ &= \begin{bmatrix} \mathbf{t}'' \\ \text{F} : \mathbf{c} \\ \text{G} : \mathbf{c} \end{bmatrix} \end{aligned}$$

Hence

$$DefFS(\begin{bmatrix}\mathbf{t}\\ \text{F} : /\mathbf{a}\\ \text{G} : /\mathbf{b}\end{bmatrix} \overset{<>}{\sqcap} \begin{bmatrix}\mathbf{t}'\\ \text{F} : /\boxed{1}\\ \text{G} : /\boxed{1}\end{bmatrix}) = \begin{bmatrix}\mathbf{t}''\\ \text{F} : \mathbf{c}\\ \text{G} : \mathbf{c}\end{bmatrix}$$

3.6.3 Atomic values and feature values. The default unification operation PDU defined in Lascarides *et al.* (1996) did not respect even the weakest notion of well-formedness: that there are some atomic types for which no features are appropriate. Thus the formalism is too weak even to be able to define an analogue of untyped FS systems, where nodes which have (atomic) values cannot also have features. For example, even if **a** is intended to be a simple value type (i.e., **a** has no appropriate features and no subtypes with appropriate features) the following is true:

Where $\mathbf{t} \sqsubset \mathbf{t}'$ and $\mathbf{a} \sqcap \mathbf{u} = \bot$

$$\begin{bmatrix}\mathbf{t}\\ \text{F} : /\mathbf{a}\end{bmatrix} \text{PDU} \begin{bmatrix}\mathbf{t}'\\ \text{F} : /\begin{bmatrix}\mathbf{u}\\ \text{G} : \mathbf{b}\end{bmatrix}\end{bmatrix} = \begin{bmatrix}\mathbf{t}\\ \text{F} : /\begin{bmatrix}\mathbf{a}\\ \text{G} : \mathbf{b}\end{bmatrix}\end{bmatrix}$$

This problem with PDU stems from the fact that the defeasible results of the operation for each node were calculated independently. In this example, the result for the path F:G is independent of the outcome for the path F. This allowed the use of a polynomial algorithm, but this was at the cost of producing default TFS which were not well-formed (both in the way outlined above and also in that the result could contain nodes typed $\bot$) which causes problems for the definition of *DefFill* (i.e., the operation for producing a well-formed TFS from a TDFS). In contrast, YADU is not calculated on a path-by-path basis: but atomic FSs are incorporated into the default result only if they unify with the complete structure. We'll demonstrate here that as a consequence, we don't get ill-formed results of the sort demonstrated above, although the cost of this is that the known algorithms for calculating default TFSs in YADU are worst-case factorial in complexity, as discussed in §5.

So, let's calculate the following:

Where $\mathbf{t} \sqsubset \mathbf{t}'$ and $\mathbf{a} \sqcap \mathbf{u} = \bot$

$$\begin{bmatrix}\mathbf{t}\\ \text{F} : /\mathbf{a}\end{bmatrix} \overset{<>}{\sqcap} \begin{bmatrix}\mathbf{t}'\\ \text{F} : /\begin{bmatrix}\mathbf{u}\\ \text{G} : \mathbf{b}\end{bmatrix}\end{bmatrix}$$

and the defeasible result of this. I.e.,:

$$DefFS(\begin{bmatrix}\mathbf{t}\\ \text{F} : /\mathbf{a}\end{bmatrix} \overset{<>}{\sqcap} \begin{bmatrix}\mathbf{t}'\\ \text{F} : /\begin{bmatrix}\mathbf{u}\\ \text{G} : \mathbf{b}\end{bmatrix}\end{bmatrix})$$

- $I_{12} = I_1 \sqcap I_2 = \begin{bmatrix}\mathbf{t}\\ \text{F} : \top\end{bmatrix}$

- $$\begin{aligned}
T^{12} &= (T^1 \cup T^2) \setminus Bot^{12}\\
T^1 &= \{\langle[\text{F} : \mathbf{a}], \mathbf{t}\rangle\}\\
T^2 &= \{\langle[\text{F} : \mathbf{u}], \mathbf{t}'\rangle, \langle\left[\text{F} : [\text{G} : \mathbf{b}]\right], \mathbf{t}'\rangle\}\\
Bot^{12} &= \emptyset\\
\text{So}\\
T^{12} &= (T^1 \cup T^2) \setminus Bot^{12}\\
&= \{\langle[\text{F} : \mathbf{a}], \mathbf{t}\rangle, \langle[\text{F} : \mathbf{u}], \mathbf{t}'\rangle, \langle\left[\text{F} : [\text{G} : \mathbf{b}]\right], \mathbf{t}'\rangle\}
\end{aligned}$$

Therefore the specificity partition of T^{12} is $\langle \mu_1, \mu_2\rangle$, where:

$$\begin{aligned}
\mu_1 = &\ \{\langle[\text{F} : \mathbf{a}], \mathbf{t}\rangle\}\\
\mu_2 = &\ \{\langle[\text{F} : \mathbf{u}], \mathbf{t}'\rangle, \langle\left[\text{F} : [\text{G} : \mathbf{b}]\right], \mathbf{t}'\rangle\}
\end{aligned}$$

- $$\begin{aligned} D_{12} = &\ \sqcup(((I_{12} \overset{<}{\sqcap}_{cs} \wp_{fs}(\mu_1)) \overset{<}{\sqcap}_{cs} \wp_{fs}(\mu_2)) \\ = &\ \sqcup\Big(\Big(\begin{bmatrix} \mathbf{t} \\ \text{F} : \top \end{bmatrix} \overset{<}{\sqcap}_{cs} \{ \begin{bmatrix} \text{F} : \mathbf{a} \end{bmatrix} \} \Big) \\ &\qquad \overset{<}{\sqcap}_{cs} \Big\{ \begin{bmatrix} \text{F} : \mathbf{u} \end{bmatrix}, \begin{bmatrix} \text{F} : \begin{bmatrix} \text{G} : \mathbf{b} \end{bmatrix} \end{bmatrix} \Big\} \Big) \\ = &\ \sqcup\Big(\begin{bmatrix} \mathbf{t} \\ \text{F} : \mathbf{a} \end{bmatrix} \overset{<}{\sqcap}_{cs} \Big\{ \begin{bmatrix} \text{F} : \mathbf{u} \end{bmatrix}, \begin{bmatrix} \text{F} : \begin{bmatrix} \text{G} : \mathbf{b} \end{bmatrix} \end{bmatrix} \Big\} \Big) \\ = &\ \begin{bmatrix} \mathbf{t} \\ \text{F} : \mathbf{a} \end{bmatrix} \end{aligned}$$
  (because $\begin{bmatrix} \mathbf{t} \\ \text{F} : \mathbf{a} \end{bmatrix} \sqcap \begin{bmatrix} \text{F} : \mathbf{u} \end{bmatrix}$ and $\begin{bmatrix} \mathbf{t} \\ \text{F} : \mathbf{a} \end{bmatrix} \sqcap \begin{bmatrix} \text{F} : \begin{bmatrix} \text{G} : \mathbf{b} \end{bmatrix} \end{bmatrix}$ both fail)

Hence

$$\begin{aligned} \mathit{DefFS}(\begin{bmatrix} \mathbf{t} \\ \text{F} : /\mathbf{a} \end{bmatrix} \overset{<>}{\sqcap} \begin{bmatrix} \mathbf{t}' \\ \text{F} : / \begin{bmatrix} \mathbf{u} \\ \text{G} : \mathbf{b} \end{bmatrix} \end{bmatrix}) = &\ \mathit{DefFS}(\begin{bmatrix} \mathbf{t} \\ \text{F} : \top \end{bmatrix} \Big/ \begin{array}{l} \{\langle \begin{bmatrix} \text{F} : \mathbf{a} \end{bmatrix}, \mathbf{t} \rangle, \\ \langle \begin{bmatrix} \text{F} : \mathbf{u} \end{bmatrix}, \mathbf{t}' \rangle, \\ \langle \begin{bmatrix} \text{F} : \begin{bmatrix} \text{G} : \mathbf{b} \end{bmatrix} \end{bmatrix}, \mathbf{t}' \rangle \} \end{array}) \\ = &\ \begin{bmatrix} \mathbf{t} \\ \text{F} : \mathbf{a} \end{bmatrix} \end{aligned}$$

3.7 The properties of $\overset{<>}{\sqcap}$ and *DefFS*

We now state the theorems that demonstrate that defaults have the properties that we specified in §1.1. The proofs for these theorems are given in the appendix. The rest of the paper can be understood without reading the appendix. In what follows, we continue to use the notational convention that $F_i =_{def} I_i/T^i$ and $\mathit{DefFS}(F_i) = D_i$.

First, criterion 1 is satisfied: non-default information can be distinguished from default information and is always preserved. Non-default and default information are distinguished as a direct result of our definition of TDFSs. $\overset{<>}{\sqcap}$ preserves all non-default information, because the non-default result of $\overset{<>}{\sqcap}$ is defined in terms of $\sqcap$ on the non-default parts of the arguments, and $\sqcap$ preserves information.

Second, criterion 2 is satisfied: default unification never fails unless there is conflict in non-default information. We assume $F_1 \overset{<>}{\sqcap} F_2$ fails if and only if $F_1 \overset{<>}{\sqcap} F_2 = \bot/\emptyset$. But $I_{12} = \bot$ if only if $I_1 \sqcap I_2 = \bot$. That is, $\overset{<>}{\sqcap}$ fails only if there is conflict in the non-default information I_1 and I_2. And $T^{12} = \emptyset$ whenever $I_{12} = \bot$ by the definition of T^{12} given in $\overset{<>}{\sqcap}$. So default unification fails only if monotonic unification does. And note that when $F_1 \overset{<>}{\sqcap} F_2$ fails, $D_{12} =_{def} \mathit{DefFS}(\bot/\emptyset) = \bot$.

We have already shown in §3.6 that the definitions of $\overset{<>}{\sqcap}$ and *DefFS* satisfy criterion 6: defaults can be given a precedence ordering such that more specific information overrides less specific information.

Now all that's left are criteria 3, 4 and 5. We start with criterion 5. We have divided the operation of computing the default result of two TDFSs into two operations—$\overset{<>}{\sqcap}$ and then *DefFS*. So default unification is order independent if $\overset{<>}{\sqcap}$ is a commutative and associative binary operation. For then, the default result *DefFS* will be the same, regardless of the order in which the TDFSs were unified, since the argument to *DefFS* will be the same. Roughly speaking, the proof that $\overset{<>}{\sqcap}$ is commutative and associative is done in parts: we prove that the indefeasible part of $\overset{<>}{\sqcap}$ is commutative and associative; and we prove that the definition of tails given by $\overset{<>}{\sqcap}$ is commutative and associative. The relevant lemmas and theorems that support this are specified below, and they are proved in the appendix.

Lemmas 1 and 2 ensure that as far as tails are concerned, the order of arguments of TDFSs to $\overset{<>}{\sqcap}$ doesn't matter:

Lemma 1

Tails are commutative. That is $T^{12} = T^{21}$.

Lemma 2
Tails are associative. That is: $T^{(12)3} = T^{1(23)}$

Lemmas 3 and 4 are necessary for proving both criterion 4 (default unification returns a single result, deterministically), and criterion 5 (default unification is order independent). Lemmas 3 and 4 ensure that the default unification on TDFSs and the default results of the operation both return a single result deterministically, as required by criterion 4.

Lemma 3
$\overset{<>}{\sqcap}$ is a function. That is TDFS$_1 \overset{<>}{\sqcap}$ TDFS$_2$ is a unique TDFS.

Lemma 4
DefFS is a function. That is, *DefFS*(TDFS) is a unique TFS.

These lemmas contribute to the proofs of lemmas lemmas 5 and 6, which in turn are used to prove theorem 1, which ensures criterion 5 is met:

Lemma 5
$\overset{<>}{\sqcap}$ is commutative. That is:

$$\begin{array}{rcl} F^{12} & =_{def} & I_{12}/T^{12} \\ & = & F^{21} \\ & =_{def} & I_{21}/T^{21} \end{array}$$

Lemma 6
$\overset{<>}{\sqcap}$ is associative. That is:

$$\begin{array}{rcl} F_{(12)3} & =_{def} & I_{(12)3}/T^{(12)3} \\ & = & F_{1(23)} \\ & =_{def} & I_{1(23)}/T^{1(23)} \end{array}$$

Theorem 1
$\overset{<>}{\sqcap}$ is an order independent binary function.

Note that it follows immediately from this theorem that no matter what order a set of TDFSs are default unified together, the default result as given by the operation *DefFS* is the same. In particular, $D_{12} = D_{21}$, and $D_{(12)3} = D_{1(23)}$.

We now turn to criterion 3 from §1.1: Default unification behaves like monotonic unification in the cases where monotonic unification would succeed. This criterion is essentially about cases where there is no conflict in the arguments to be default unified. When there's no conflict, $\overset{<>}{\sqcap}$ should give the same results as $\sqcap$. In our case, this needs explication for two reasons. First, we have distinguished default from non-default information in our TDFSs—as required by criterion 1—and so $\overset{<>}{\sqcap}$ and $\sqcap$ work on different types of arguments: the former is defined on TDFSs whereas the latter is defined on TFSs. So, we must refine what we mean by: $\overset{<>}{\sqcap}$ gives the same result as $\sqcap$. Here we assume that it means that when there's no conflict between the arguments, $\overset{<>}{\sqcap}$ and *DefFS* should give the same results as $\sqcap$ for the indefeasible TFSs and the defeasible TFSs respectively. That is:

$$\begin{aligned} \text{Where } F_{12} =_{def} & \quad I_1/T_1 \overset{<>}{\sqcap} I_2/T_2 \\ \text{and } \mathit{DefFS}(F_i) =_{def} & \quad D_i \\ \\ I_{12} = & \quad I_1 \sqcap I_2, \text{ and} \\ D_{12} = & \quad D_1 \sqcap D_2 \end{aligned}$$

In other words, on the defeasible part, $\mathit{DefFS}(F_1 \overset{<>}{\sqcap} F_2) = \mathit{DefFS}(F_1) \sqcap \mathit{DefFS}(F_2)$. Note that $I_{12} = I_1 \sqcap I_2$ follows trivially from the definition of $\overset{<>}{\sqcap}$, regardless of whether there is conflict or not. So we must show that when there's no conflict between the arguments to $\overset{<>}{\sqcap}$, $D_{12} = D_1 \sqcap D_2$.

The second thing that must be explicated is the conditions under which the arguments to $\overset{<>}{\sqcap}$—namely, two TDFSs—are said to conflict. When default unification is defined in terms of TFSs (e.g., Carpenter, 1993), the concept of conflict is clear: the TFSs conflict if their unification is $\bot$. However, because we have demarcated default from non-default information in TDFSs to satisfy criterion 1, and we have introduced tails into TDFSs to satisfy criterion 5, our definition of when two TDFSs conflict is necessarily a little more complex. So what should conflict mean in this case? There are at least two alternatives. First, two TFDSs conflict if $D_1 \sqcap D_2 = \bot$. In other words, they conflict if $\mathit{DefFS}(F_1) \sqcap \mathit{DefFS}(F_2) = \bot$. Second, two TDFSs conflict if the information in the two tails conflict, either with each other or with the accumulated indefeasible information I_{12}. That is, the two TDFSs conflict if $I_{12} \sqcap \wp_{fs}(T^1) \sqcap \wp_{fs}(T^2) = \bot$. Note that given that T^1 potentially contains mutually incompatible information (in the TFS sense), and similarly for T^2, and in general T^i contains more information than D_i (by the definition of *DefFS*), the latter condition of conflict is much weaker than the former condition. As it turns out, we can prove that $D_{12} = D_1 \sqcap D_2$ (i.e., $\overset{<>}{\sqcap}$ behaves like $\sqcap$ when there's no conflict) only in the case where there is no conflict in the latter sense. This is given in theorem 2 below.

Theorem 2
The Typed Unification Property
Let F_1, F_2 be TDFSs. And suppose

$$I_{12} \sqcap \wp_{fs}(T^1) \sqcap \wp_{fs}(T^2) \neq \bot$$

Then

$$D_{12} = D_1 \sqcap D_2$$

In other words:

$$\mathit{DefFS}(F_1 \overset{<>}{\sqcap} F_2) = \mathit{DefFS}(F_1) \sqcap \mathit{DefFS}(F_2)$$

In general, $\overset{<>}{\sqcap}$ doesn't reduce to $\sqcap$ when there's no conflict between the defeasible TFSs (i.e., when $D_1 \sqcap D_2 \neq \bot$). The following example demonstrates this:

Where

$$\begin{aligned} & \mathbf{t1} \sqsubset \mathbf{t2} \sqsubset \mathbf{t3} \\ & \mathbf{a} \sqcap \mathbf{c} = \bot \\ & \mathbf{b} \sqcap \mathbf{c} = \bot \\ & \mathbf{a} \sqcup \mathbf{b} = \top \end{aligned}$$

$$\begin{bmatrix} \mathbf{t1} \\ \text{F} : \top \\ \text{G} : \top \end{bmatrix} \Big/ \begin{array}{l} \{\langle \begin{bmatrix} \text{F} : \boxed{1} \\ \text{G} : \boxed{1} \end{bmatrix}, \mathbf{t1} \rangle, \\ \langle [\, \text{F} : \mathbf{a} \,], \mathbf{t2} \rangle, \ \ \langle [\, \text{G} : \mathbf{b} \,], \mathbf{t2} \rangle \} \end{array} \ \overset{<>}{\sqcap} \begin{bmatrix} \mathbf{t3} \\ \text{F} : \top \\ \text{G} : \top \end{bmatrix} / \ \{\langle [\, \text{F} : \mathbf{c} \,], \mathbf{t3} \rangle\}$$

We just show how to calculate the defeasible TFS $D_{12} =_{def} \mathit{DefFS}(F_1 \overset{<>}{\sqcap} F_2)$:

$$\begin{aligned}
D_{12} = \; & \bigsqcup\left(\left(\left(\begin{bmatrix} \mathbf{t}_1 \\ \text{F} : \top \\ \text{G} : \top \end{bmatrix} \overset{<}{\sqcap}_{cs} \left\{ \begin{bmatrix} \text{F} : \boxed{1} \\ \text{G} : \boxed{1} \end{bmatrix} \right\} \right) \overset{<}{\sqcap}_{cs} \left\{ \begin{bmatrix} \text{F} : \mathbf{a} \end{bmatrix}, \begin{bmatrix} \text{G} : \mathbf{b} \end{bmatrix} \right\} \right) \right. \\
& \qquad\qquad \left. \overset{<}{\sqcap}_{cs} \left\{ \begin{bmatrix} \text{F} : \mathbf{c} \end{bmatrix} \right\} \right) \\
= \; & \bigsqcup\left(\left(\begin{bmatrix} \mathbf{t}_1 \\ \text{F} : \boxed{1} \\ \text{G} : \boxed{1} \end{bmatrix} \overset{<}{\sqcap}_{cs} \left\{ \begin{bmatrix} \text{F} : \mathbf{a} \end{bmatrix}, \begin{bmatrix} \text{G} : \mathbf{b} \end{bmatrix} \right\} \right) \overset{<}{\sqcap}_{cs} \left\{ \begin{bmatrix} \text{F} : \mathbf{c} \end{bmatrix} \right\} \right) \\
= \; & \bigsqcup\left(\left\{ \begin{bmatrix} \mathbf{t}_1 \\ \text{F} : \boxed{1}\mathbf{a} \\ \text{G} : \boxed{1} \end{bmatrix}, \begin{bmatrix} \mathbf{t}_1 \\ \text{F} : \boxed{1}\mathbf{b} \\ \text{G} : \boxed{1} \end{bmatrix} \right\} \overset{<}{\sqcap}_{cs} \left\{ \begin{bmatrix} \text{F} : \mathbf{c} \end{bmatrix} \right\} \right) \\
= \; & \begin{bmatrix} \mathbf{t}_1 \\ \text{F} : \boxed{1}\mathbf{a} \\ \text{G} : \boxed{1} \end{bmatrix} \sqcup \begin{bmatrix} \mathbf{t}_1 \\ \text{F} : \boxed{1}\mathbf{b} \\ \text{G} : \boxed{1} \end{bmatrix} \\
= \; & \begin{bmatrix} \mathbf{t}_1 \\ \text{F} : \boxed{1} \\ \text{G} : \boxed{1} \end{bmatrix}
\end{aligned}$$

Thus $D_{12} \neq D_1 \sqcap D_2$ in this case, even though

$$D_1 \sqcap D_2 = \begin{bmatrix} \mathbf{t}_1 \\ \text{F} : \boxed{1}\mathbf{c} \\ \text{G} : \boxed{1} \end{bmatrix} \neq \bot$$

We have suggested that because TDFSs contain more information than the defaults that win, the notion of 'conflict' is more complex than in Carpenter (1992, 1993) and Bouma (1992). Their definitions of default unification correspond to the situations where we are unifying basic TDFSs. This is because these record default information but, like Carpenter's and Bouma's accounts, they don't track overridden default values from the unification history, (because there hasn't been any unification history). We can prove that the typed unification property for basic TDFSs is one where $\overset{<>}{\sqcap}$ reduces to $\sqcap$ when there is no conflict among the TFS components. The corollary 1 below follows from the above theorem and the following lemma:

Lemma 7
DefFS for Basic TDFSs
Let F be a basic TDFS I/T derived from the indefeasible TFS I and the defeasible TFS I_D. Then

$$\mathit{DefFS}(F) =_{def} D = I \sqcap I_D$$

Corollary 1
The Typed Unification Property for Basic TDFSs
If TDFS_1 and TDFS_2 are basic TDFSs derived from I_1 and I_{D_1}, and I_2 and I_{D_2} respectively, and $I_1 \sqcap I_{D_1} \sqcap I_2 \sqcap I_{D_2} \neq \bot$ (i.e., $D_1 \sqcap D_2 \neq \bot$), then

$$\begin{aligned} D_{12} = \; & D_1 \sqcap D_2 \\ = \; & I_1 \sqcap I_{D_1} \sqcap I_2 \sqcap I_{D_2} \end{aligned}$$

Thus criterion 3 holds, proviso the refinements it required because of the more complex information structures (i.e., TDFSs) that are necessary in order to meet the other criteria.

3.8 Comparison between PDU and YADU

For convenience, we summarize the major similarities and differences between PDU and YADU here.

1. Both PDU and YADU can be defined using conditional logic. However, unlike PDU, YADU has a simple interpretation as an algebraic operation, which is what we have given here. This considerably simplifies the proofs that the operation has the desired properties.

2. PDU does not fully satisfy our criterion 6 from section §1.1, because default coindexations cannot be overridden by default values. In contrast, YADU allows coindexations to be overridden by default values. This is because (a) unlike PDU, the defaults on paths are *not* computed independently of one another, and (b) unlike PDU, YADU keeps track of overridden default reentrancies in the tails.

3. Furthermore, because tails are extended this way in YADU compared with PDU, one needn't define the default structure in a TDFS separately; it is computable directly from the indefeasible TFS and the tail.

4. PDU can result in default TFSs which are not well-formed, in that they contain $\bot$, or violate a very basic notion of well-formedness. This necessitates a complex *DefFill* operation in order to utilize the default structures. In contrast, in YADU, the default structures are guaranteed to be well-formed and *DefFill* is basically equivalent to treating the default structure given by *DefFS* as being non-default.

5. YADU can be extended to handle default inequalities (see §6.1 below); PDU cannot be extended this way.

4. Linguistic Examples

In this section, we go through some linguistic examples where defaults can be utilized. The aim of this section is to illustrate ways in which the use of defaults can simplify and enhance monotonic grammars, concentrating on syntax and (compositional and lexical) semantics. In particular, of course, our aim is to show the linguistic utility of the particular assumptions we have made in defining YADU. We do not have space here to systematically illustrate the ways in which defaults may be utilized in a grammar, though the papers listed in §1 give many more examples, which can, in general, also be modeled within the current framework (with the main exception being some uses which require asymmetric defaults, discussed in §6.5, below). We have tried to give examples which have not, to our knowledge, been extensively discussed in the previous default unification literature.

We assume an HPSG-like framework, but we have tried to give an explanation in enough detail for readers who are relatively unfamiliar with HPSG to follow the examples. In HPSG, the basic linguistic structure is the *sign*. Signs may be lexical or phrasal, but always correspond to constraints which may be represented using typed feature structures. In what follows, we will instead use TDFSs, and take various liberties with the feature geometry for ease of exposition. However, although much simplified, the treatment in the grammar fragments here is substantially based on that assumed in the English Resource Grammar (ERG) which is under development at CSLI (Flickinger *et al.*, in preparation). The ERG itself has been developed without making use of defaults up to this point, using the DISCO/PAGE system (Uszkoreit *et al.* 1994), but it also runs within the LKB system (Copestake, 1992). YADU has been implemented within the latter system, replacing the earlier version of default unification described in Copestake (1993).

Before giving detailed examples however, we must make some further remarks on the assumptions we are making about persistent and non-persistent defaults in these fragments. In the example of inflectional morphology which we gave in §2, we mentioned that the defaults could not persist beyond the lexicon, since the values for the suffixes must be treated as hard information by the parser/generator. More generally, we distinguish between defaults which are used in the description of some object in the grammar (that is, a lexical sign, a lexical rule or a grammar rule/schema), which we refer to as *non-persistent* defaults, and defaults which are intended to contribute to the output of the grammar (in particular, the logical form), which we term *persistent* defaults. In fact, most of the examples we will discuss here make use of non-persistent defaults, and where we intend defaults to be persistent we will distinguish this with a subscript p following the slash in the case of the abbreviatory notation, and with an annotation on the tail in the full description.[5]

In the examples below, as in §2, we assume that non-persistent defaults may be part of type constraints. This is in line with the practice followed in the ERG, where some types are purely lexical, that is, they are only utilized in the description of lexical entries and are irrelevant to the parser/generator. However, some grammar writers deprecate such use of types, and capture the corresponding generalizations with macros or templates. For our current purposes, this controversy is of limited relevance, since the same TDFSs which we are describing as constraints on types could alternatively be used as the values of templates. The only requirement is that, in order to have a compatible notion of specificity, the templates must form a partial order (which will be naturally true if they are used as an inheritance hierarchy) so that template order can replace type order in the determination of specificity in YADU (as discussed in §6.2). Under these circumstances, non-persistent defaults would most naturally be seen as part of the description language. For further discussion of the type/template distinction, in the context of a default inheritance system, see Lascarides *et al.* (1996).

4.1 Modals

One very straightforward example of the utility of defaults in descriptions of syntactic behavior is the treatment of *ought* in English. This behaves as a modal verb in most respects: it inverts, it can be negated without *do*, it takes the contracted negation *oughtn't* and it does not have distinct inflected forms (though we will ignore morphological issues here). However, unlike most modals, it requires the *to*-infinitive, as shown in (6a–d), rather than the base form of the verb. The modal *is to* behaves similarly, as shown in (6e), although it inflects.[6]

(6)a. I ought to finish this paper.
b. * I ought finish this paper.
c. You oughtn't to disagree with the boss.
d. Ought we to use this example?
e. You are not to argue.

Here we sketch a treatment of modals which allows *ought* to be an exception to the general class, just with respect to the value of the attribute on its complement

5 In principle at least, there can be a mixture of defaults of varying persistence in the tail of a TDFS, and the *DefFill* operation must be defined so that it is sensitive to this distinction and only incorporates defaults of the appropriate persistence. To do this, we can define a partial order on persistence markers and add a persistence marker as an extra component to the definition of a tail. *DefFill* is then defined relative to a particular persistence, and incorporates all tails which are marked as having that persistence or any persistence which is prior in the partial order. Since none of the examples which follow make use of mixed persistence tails, we ignore this complication here.

6 For some speakers the infinitive without *to* is possible or even preferred with *ought* in non-assertive contexts. For others, *ought* does not share the other properties of modals we have listed here. We will ignore these dialect variations here, and simply assume the grammaticality pattern shown in (6).

$$\begin{bmatrix}\textbf{modal} & \\ \text{SYNSEM} : & \begin{bmatrix}\text{HEAD AUX} : \textbf{true} \\ \text{VAL COMPS} : \left\langle \begin{bmatrix}\text{HEAD VFORM} : /\textbf{bse} \\ \text{VAL COMPS} : \langle\rangle\end{bmatrix} \right\rangle\end{bmatrix}\end{bmatrix}$$

Figure 7
Constraint on type **modal**

$$\begin{bmatrix}\textbf{head-comp-phrase} & \\ \text{ORTH} : \textbf{could sleep} & \\ \text{SYNSEM} : & \begin{bmatrix}\text{HEAD} : \boxed{1} \\ \text{VAL COMPS} : \langle\rangle\end{bmatrix} \\ \text{HD-DTR} : & \begin{bmatrix}\textbf{modal} & \\ \text{ORTH} : \textbf{could} & \\ \text{SYNSEM} : & \begin{bmatrix}\text{HEAD} : \boxed{1}\begin{bmatrix}\text{AUX} : \textbf{true}\end{bmatrix} \\ \text{VAL COMPS} : \langle \boxed{2} \rangle\end{bmatrix}\end{bmatrix} \\ \text{NON-HD-DTRS} : & \left\langle \begin{bmatrix}\text{ORTH} : \textbf{sleep} & \\ \text{SYNSEM} : \boxed{2} & \begin{bmatrix}\text{HEAD VFORM} : \textbf{bse} \\ \text{VAL COMPS} : \langle\rangle\end{bmatrix}\end{bmatrix} \right\rangle\end{bmatrix}$$

Figure 8
Structure of an HPSG-style phrase for *could sleep*

specification which specifies whether *ought* expects a base or infinitival complement. We also briefly introduce the syntactic feature geometry which we will also assume for all the examples which follow.

Figure 7 shows a constraint specification for a type **modal**, which we assume is the lexical type of all modal verbs including *ought*. This constraint is responsible for the characteristic behavior of modals mentioned above. Because we are ignoring morphology, the figure only shows the SYNSEM (syntax/semantics) part of the lexical sign, and we will ignore the semantics of modals for this example. Subcategorisation properties of signs are specified via valence (VAL) features, which describe a list of specifications with which the SYNSEM values of other signs may be unified. For verbs, the relevant valence features are SUBJ (subject) and COMPS (complements), though we have systematically omitted SUBJ in this example for simplicity. The constraint given on **modal** for the value of the COMPS list means that it may only contain a single element. That element must have a verbal HEAD, a single member subject list and an empty complements list: i.e., it must be a verb phrase. Thus the constraint means that modals are restricted to taking a single verb phrase complement, which we assume here is obligatory. The value of the VFORM of the complement is specified by default to be **bse**, which is true of verbs and verb phrases where the head is a base form, but not of infinitivals (which have VFORM **inf**).

Figure 8 illustrates the complement selection mechanism in operation. It shows a simple instance of a head complement phrase consisting of a modal (*could*) and its verb phrase complement (*sleep*). As we mentioned above, phrases are represented as TDFSs. Generalizations about phrases, or schemata, are represented as constraints on types and correspond to grammar rules in other frameworks. Here **head-comp-phrase** is a type, corresponding to head-complement phrases, as we will discuss in more detail in §4.2. Headed phrases have features HD-DTR, indicating the head of the phrase, which in this case corresponds to the structure for the modal verb *could*, and NON-HD-DTRS, which corresponds to a list of the other daughters. Here NON-HD-DTRS is a singleton, corresponding to the verb phrase *sleep*. The head-complement schema constrains the list of SYNSEMs of the NON-HD-DTRS signs to be equal to the COMPS list of the head daughter.

Description for *could*:
$$\begin{bmatrix} \textbf{modal} \\ \text{ORTH} : \textbf{could} \end{bmatrix}$$

Structure for *could*:
$$\begin{bmatrix} \textbf{modal} \\ \text{ORTH} : \textbf{could} \\ \text{SYNSEM} : \begin{bmatrix} \text{HEAD AUX} : \textbf{true} \\ \text{VAL COMPS} : \left\langle \begin{bmatrix} \text{HEAD VFORM} : \textbf{bse} \\ \text{VAL COMPS} : \langle\rangle \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix}$$

Description for *ought*:
$$\begin{bmatrix} \textbf{modal} \\ \text{ORTH} : \textbf{ought} \\ \text{SYNSEM VAL COMPS} : \left\langle \begin{bmatrix} \text{HEAD VFORM} : \textbf{inf} \end{bmatrix} \right\rangle \end{bmatrix}$$

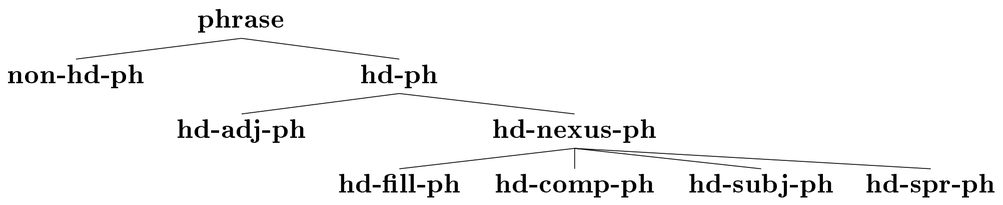
Structure for *ought*:
$$\begin{bmatrix} \textbf{modal} \\ \text{ORTH} : \textbf{ought} \\ \text{SYNSEM} : \begin{bmatrix} \text{HEAD AUX} : \textbf{true} \\ \text{VAL COMPS} : \left\langle \begin{bmatrix} \text{HEAD VFORM} : \textbf{inf} \\ \text{VAL COMPS} : \langle\rangle \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix}$$

Figure 9
Examples of lexical description for modal verbs

The constraint for **modal** in Figure 7 also specifies that the HEAD AUX value of modal signs is **true** indefeasibly. This controls the behavior with respect to various lexical rules. For instance, negation is implemented via a lexical rule which, among other things, adds a structure compatible with *not* to the beginning of the complement list of the verb. This lexical rule only applies to verbs such as auxiliaries and modals which, unlike other verbs, have a value for SYNSEM HEAD AUX compatible with **true**. The AUX feature also controls the application of the inversion lexical rule.

The lexical specification for *could*, which we take as an example of a normal modal, is also shown in Figure 9. The only idiosyncratic information given here is the morphology (though of course in the full entry there is also a specification of the semantics). Thus the TFS for *could* inherits all the information from the constraint on the type **modal**. Note however that the default for the VFORM of the complement must be non-persistent, and thus the actual TFS for the lexical sign is calculated via DefFill, with the result that is shown in Figure 9. The structure for *could sleep*, which is derived from the sign shown in Figure 9, is shown in Figure 8 (for details on constructing this, see §4.2). In contrast, the lexical specification for *ought* overrides the default value inherited from **modal** with the non-default value **inf** for the VFORM of its complement. So *ought* cannot combine with *sleep* in the manner shown in Figure 8, as required.

This is a very simple example but it illustrates that the use of defaults allows the grammar to capture the generalization that most modals take a base form complement. It is possible to devise a monotonic encoding which would produce the same lexical FSs but the various behaviors characteristic of modals would have to be split into different constraints, so that *ought* could inherit some but not all of them. This would not capture the intuition that *ought* is the exceptional case. Furthermore a monotonic encoding requires extra types. With the example as shown here, it appears that the monotonic encoding would require two additional types compared to the default encoding, one for *ought* and *is to* (and also *used to*, for speakers for whom this is a modal) and the other for all other modals. However, in the full English Resource Grammar, seven types are duplicated to allow for *ought.* The real gain in conciseness from allowing defaults is therefore more significant than our simplified example suggests.



non-hd-ph	non-headed-phrase
hd-ph	headed-phrase
hd-adj-ph	head-adjunct-phrase
hd-nexus-ph	head-nexus-phrase
hd-fill-ph	head-filler-phrase
hd-comp-ph	head-complement-phrase
hd-subj-ph	head-subject-phrase
hd-spr-ph	head-specifier-phrase

Figure 10
Hierarchy of phrases from Sag (1997)

4.2 Defaults in constructions

Default inheritance can also be utilized in the description of rules, schemata or constructions within the grammar itself. Recent work within HPSG has demonstrated the utility of describing a hierarchy of phrasal types in a manner analogous to the more familiar lexical hierarchy. As we mentioned in the previous example, in HPSG, phrasal types play a similar role to rules in other frameworks, systematically relating the mother of a phrase to its daughters. Sag (1997), in an account of relative clause constructions, defines a general type **phrase**, from which various subtypes of phrase inherit (Figure 10). Sag uses defaults in his description of the phrasal hierarchy, but we go through the example in detail here, in order to demonstrate more formally how default unification operates in this case.

We will not go into the full details of the hierarchy here, since our main concern is to demonstrate the way in which defaults are used. So we will just consider the constraints on **headed-phrase** (shown in Figure 11) and the way these are inherited by some of its subtypes.[7] The Head Feature Principle (HFP), which states that the HEAD value of the mother is identical to that of the head daughter, is unchanged from earlier work on HPSG (e.g., Pollard and Sag (1994:34)) and is not default. In contrast the Valence Principle (VALP) and the Empty Complement Constraint (ECC) are both stated in terms of defaults. VALP is a reformulation of the Valence Principle in Pollard and Sag (1994:348). Here SUBJ (subject), SPEC (specifier) and COMPS (complements) are all valence features, and the effect of the constraint is to specify that these should be identical on the head daughter of a phrase and the mother, unless a more specific phrase type overrides this. The exceptions to the generalization that valence information is identical on the head daughter and mother are the phrase types where the individual valence features are satisfied. For example, in a head-complement phrase, the complements of the head of the phrase are instantiated and the COMPS list of the mother will be empty (as shown above in the example in Figure 8). ECC is a simpler constraint which states that, by default, headed-phrases have head daughters with an empty complement list. It is important to

7 The constraints are presented slightly differently from Sag (1997), since he sometimes omits features on paths when specifying constraints, but for the sake of clarity we show full paths here. We have also added the feature VAL, as in the ERG and the example above.

Head Feature Principle (HFP): $\begin{bmatrix}\text{SYNSEM HEAD} : \boxed{1}\\ \text{HD-DTR SYNSEM HEAD} : \boxed{1}\end{bmatrix}$

Valence Principle (VALP): $\begin{bmatrix}\text{SYNSEM VAL} : \begin{bmatrix}\text{SUBJ} : /\boxed{1}\\ \text{SPR} : /\boxed{2}\\ \text{COMPS} : /\boxed{3}\end{bmatrix}\\ \text{HD-DTR SYNSEM VAL} : \begin{bmatrix}\text{SUBJ} : /\boxed{1}\\ \text{SPR} : /\boxed{2}\\ \text{COMPS} : /\boxed{3}\end{bmatrix}\end{bmatrix}$

Empty Complement Constraint (ECC): $\begin{bmatrix}\text{HD-DTR SYNSEM VAL COMPS} : /\langle\rangle\end{bmatrix}$

Figure 11
The HFP, VALP and ECC constraints (abbreviatory notation)

$$\begin{bmatrix}\textbf{hd-phr} & \\ \text{SS} : & \begin{bmatrix}\text{HEAD} : \boxed{1}\\ \text{VAL} : \begin{bmatrix}\text{SUBJ} : \textbf{list}\\ \text{SPR} : \textbf{list}\\ \text{COMPS} : \textbf{list}\end{bmatrix}\end{bmatrix}\\ \text{HD-DTR SS} : & \begin{bmatrix}\text{HEAD} : \boxed{1}\\ \text{VAL} : \begin{bmatrix}\text{SUBJ} : \textbf{list}\\ \text{SPR} : \textbf{list}\\ \text{COMPS} : \textbf{list}\end{bmatrix}\end{bmatrix}\end{bmatrix} \Big/ \begin{array}{l}\{\langle\begin{bmatrix}\text{SS VAL SUBJ} : \boxed{1}\\ \text{HD-DTR SS VAL SUBJ} : \boxed{1}\end{bmatrix}, \textbf{hd-phr}\rangle,\\ \langle\begin{bmatrix}\text{SS VAL SPR} : \boxed{1}\\ \text{HD-DTR SS VAL SPR} : \boxed{1}\end{bmatrix}, \textbf{hd-phr}\rangle,\\ \langle\begin{bmatrix}\text{SS VAL COMPS} : \boxed{1}\\ \text{HD-DTR SS VAL COMPS} : \boxed{1}\end{bmatrix}, \textbf{hd-phr}\rangle,\\ \langle\begin{bmatrix}\text{HD-DTR SS VAL COMPS} : \langle\rangle\end{bmatrix}, \textbf{hd-phr}\rangle\}\end{array}$$

Figure 12
Expressing the HFP, VALP and ECC constraints as a single TDFS (SS here is an abbreviation for SYNSEM)

note that although Sag states HFP, VALP and ECC as three separate constraints, this is equivalent to treating their conjunction as a single constraint on the type **headed-phrase**, as shown in Figure 12 (for clarity we use the full indefeasible/tail notation for a TDFS). Note that one advantage of order-independence in our definition of defaults is that the result of specifying constraints individually is guaranteed to be equivalent to stating them as a single TDFS, leading to greater perspicuity in the definition of a grammar.

In Figure 13, we show the constraint specifications on the types **head-nexus-phrase**, **head-complement-phrase** and **head-specifier-phrase**, as given by Sag (with modifications as above). The constraint on **head-nexus-phrase** refers to how the semantic content is shared between mother and head daughter (a default version of this constraint which actually removes the need for the type **head-nexus-phrase** will be discussed in §6.1). For **head-complement-phrase** and **head-specifier-phrase**, the attribute NON-HD-DTRS corresponds to a list of the daughters of the phrase excluding the head. As in Figure 8, elements of the appropriate valence feature are instantiated by the SYNSEMs of the non-head daughters.

There is a complication here: the constraint on **head-complements-phrase** given by Sag is intended to be read as specifying that an arbitrary number of complements (possibly zero) correspond to the value of the COMPS feature. In fact, this cannot be directly implemented as written in the framework we assume here. The required effect can be achieved with a recursive type or, on the assumption that the complements list can contain no more than four elements, multiple subtypes can be specified, each with a fixed number of complements. For simplicity, we have assumed the latter encoding style here, which we illustrate with two schemata in Figure 13, for the zero complement (**head-zero-comp-phrase**) and the one complement cases (**head-one-comp-phrase**).

Figure 14 shows the final structures for the schemata for the head-complement-phrases and head-specifier-phrase, after inheritance from the supertypes and making the default properties non-default.[8] Note that in the head-one-complement-phrase schema,

8 Sag (1997) suggests that the constraints are converted from default to non-default for all maximally

the ECC is overridden, as is the part of VALP which concerns the coindexation between the complements of the head-daughter and the mother, while in head-specifier-phrase, the coindexation of the specifier has been overridden.

One interesting point is that the specification language discussed in §3.5 allows for a very succinct encoding of constraints which, like the valence principle, state that a set of features are identical by default. Figure 15 shows this alternative encoding for VALP: the default TFS states one coindexation, between the VAL features for mother and head daughter. Because the definition of *BasicTDFS* states that a path equivalence will be present in the tail for each case where a node can be reached by two distinct paths, the TDFS contains tail elements indicating a path equivalence not only for VAL, but also for all paths which extend VAL, i.e., SUBJ, COMPS and SPR (for clarity, we have shown the type **list** explicitly, rather than using the angle-bracket notation: **list** has no appropriate features). Thus the TDFS has one extra tail element compared to the version we gave in Figure 12, but this extra element will be overridden in all of the schemata (except head-zero-comp-phrase, if we use the encoding assumed in Figure 13).[9]

Thus the use of defaults allows a more concise specification of the phrase hierarchy, enabling generalizations to be captured which would be obscured with a purely monotonic treatment. Although these constraints could be captured in a system without defaults, they would either have to be stipulated redundantly, in multiple points in the hierarchy, or extra types would have to be introduced so that multiple inheritance could be used to distribute properties appropriately. This latter option would considerably complicate the hierarchy for the set of default constraints which Sag considers. In neither case could the valence principle be stated as a single generalization. The only way in which this would be possible in a monotonic system would be if the constraint language were enriched. This example shows the utility of allowing overriding of default path equivalence statements. This would not be possible with some previous versions of default unification, including Russell *et al.*(1991, 1993) and the second version of the LKB default unification given in Copestake (1993). It is also not allowed in Young and Rounds (1993), which is the only order-independent version of default unification of which we are aware apart from PDU and YADU.

4.3 Agreement and semantic plurality

We turn now to an example which involves a more complex use of YADU, in which it is necessary to allow default values to survive when default path equivalence statements are overridden. The example concerns the relationship of number agreement to semantic notions of plurality and massness in English nouns. It is partially based on a treatment given in Copestake (1992), but the use of YADU enables a more satisfactory encoding of the central intuition, which is that agreement usually, but not always, follows the semantics.

To explain the account, we must first briefly introduce the style of semantic encoding used in the ERG. This is known as Minimal Recursion Semantics (MRS) and is described in detail in Copestake *et al.* (1995), though for the sake of simplicity we will ignore most of the details in these examples, including all discussion of quantification. The semantics

specific phrasal types, which means that these must have a special status. A more straightforward approach is to assume that the terminal schemata are analogous to lexical entries, rather than being types in their own right.

9 This encoding style also has potential in the representation of lexical rules, where these are represented as single TDFS with an input and an output feature, since the default TFS in the specification can state that the input is equal to the output. This allows a much more succinct description than is possible in an unextended monotonic language, where feature values must be explicitly duplicated.

head-nexus-phrase
$$\begin{bmatrix} \text{SYNSEM CONT} : \boxed{1} \\ \text{HD-DTR SYNSEM CONT} : \boxed{1} \end{bmatrix}$$

head-spr-phrase
$$\begin{bmatrix} \text{SYNSEM VAL SPR} : \langle\rangle \\ \text{HD-DTR SYNSEM VAL SPR} : \langle \boxed{1} \rangle \\ \text{NON-HD-DTRS} : \langle \begin{bmatrix} \text{SYNSEM} : \boxed{1} \end{bmatrix} \rangle \end{bmatrix}$$

head-comp-phrase
(…notation)
$$\begin{bmatrix} \text{SYNSEM VAL COMPS} : \langle\rangle \\ \text{HD-DTR SYNSEM VAL COMPS} : \langle \boxed{1}, \ldots, \boxed{n} \rangle \\ \text{NON-HD-DTRS} : \langle \begin{bmatrix} \text{SYNSEM} : \boxed{1} \end{bmatrix}, \ldots, \begin{bmatrix} \text{SYNSEM} : \boxed{n} \end{bmatrix} \rangle \end{bmatrix}$$

head-zero-comp-phrase
$$\begin{bmatrix} \text{SYNSEM VAL COMPS} : \langle\rangle \\ \text{HD-DTR SYNSEM VAL COMPS} : \langle\rangle \\ \text{NON-HD-DTRS} : \langle\rangle \end{bmatrix}$$

head-one-comp-phrase
$$\begin{bmatrix} \text{SYNSEM VAL COMPS} : \langle\rangle \\ \text{HD-DTR SYNSEM VAL COMPS} : \langle \boxed{1} \rangle \\ \text{NON-HD-DTRS} : \langle \begin{bmatrix} \text{SYNSEM} : \boxed{1} \end{bmatrix} \rangle \end{bmatrix}$$

Figure 13
Constraints on phrases

head-zero-comp-phrase schema
$$\begin{bmatrix} \text{SYNSEM} : \begin{bmatrix} \text{HEAD} : \boxed{1} \\ \text{VAL} : \begin{bmatrix} \text{SUBJ} : \boxed{2} \\ \text{SPR} : \boxed{3} \\ \text{COMPS} : \langle\rangle \end{bmatrix} \\ \text{CONT} : \boxed{4} \end{bmatrix} \\ \text{HD-DTR SYNSEM} : \begin{bmatrix} \text{HEAD} : \boxed{1} \\ \text{VAL} : \begin{bmatrix} \text{SUBJ} : \boxed{2} \\ \text{SPR} : \boxed{3} \\ \text{COMPS} : \langle\rangle \end{bmatrix} \\ \text{CONT} : \boxed{4} \end{bmatrix} \\ \text{NON-HD-DTRS} : \langle\rangle \end{bmatrix}$$

head-one-comp-phrase schema
$$\begin{bmatrix} \text{SYNSEM} : \begin{bmatrix} \text{HEAD} : \boxed{1} \\ \text{VAL} : \begin{bmatrix} \text{SUBJ} : \boxed{2} \\ \text{SPR} : \boxed{3} \\ \text{COMPS} : \langle\rangle \end{bmatrix} \\ \text{CONT} : \boxed{4} \end{bmatrix} \\ \text{HD-DTR SYNSEM} : \begin{bmatrix} \text{HEAD} : \boxed{1} \\ \text{VAL} : \begin{bmatrix} \text{SUBJ} : \boxed{2} \\ \text{SPR} : \boxed{3} \\ \text{COMPS} : \langle \boxed{5} \rangle \end{bmatrix} \\ \text{CONT} : \boxed{4} \end{bmatrix} \\ \text{NON-HD-DTRS} : \langle \begin{bmatrix} \text{SYNSEM} : \boxed{5} \end{bmatrix} \rangle \end{bmatrix}$$

head-spr-phrase schema
$$\begin{bmatrix} \text{SYNSEM} : \begin{bmatrix} \text{HEAD} : \boxed{1} \\ \text{VAL} : \begin{bmatrix} \text{SUBJ} : \boxed{2} \\ \text{SPR} : \langle\rangle \\ \text{COMPS} : \boxed{3} \end{bmatrix} \\ \text{CONT} : \boxed{4} \end{bmatrix} \\ \text{HD-DTR SYNSEM} : \begin{bmatrix} \text{HEAD} : \boxed{1} \\ \text{VAL} : \begin{bmatrix} \text{SUBJ} : \boxed{2} \\ \text{SPR} : \langle \boxed{5} \rangle \\ \text{COMPS} : \langle\rangle \end{bmatrix} \\ \text{CONT} : \boxed{4} \end{bmatrix} \\ \text{NON-HD-DTRS} : \langle \begin{bmatrix} \text{SYNSEM} : \boxed{5} \end{bmatrix} \rangle \end{bmatrix}$$

Figure 14
Expanded schemata

$$
\mathit{BasicTDFS}(\begin{bmatrix}\textbf{hd-phr} & \\ \mathrm{SS} : & \begin{bmatrix}\mathrm{HEAD} : \boxed{1} \\ \mathrm{VAL} : \begin{bmatrix}\mathrm{SUBJ} : \textbf{list} \\ \mathrm{SPR} : \textbf{list} \\ \mathrm{CPS} : \textbf{list}\end{bmatrix}\end{bmatrix} \\ \mathrm{HD\text{-}DTR\ SS} : & \begin{bmatrix}\mathrm{HEAD} : \boxed{1} \\ \mathrm{VAL} : \begin{bmatrix}\mathrm{SUBJ} : \textbf{list} \\ \mathrm{SPR} : \textbf{list} \\ \mathrm{CPS} : \textbf{list}\end{bmatrix}\end{bmatrix}\end{bmatrix}, \begin{bmatrix}\mathrm{SS\ VAL} : \boxed{1} \\ \mathrm{HD\text{-}DTR\ SS\ VAL} : \boxed{1}\big[\mathrm{CPS} : \langle\rangle\big]\end{bmatrix})
$$

$$
=
$$

$$
\begin{bmatrix}\textbf{hd-phr} & \\ \mathrm{SS} : & \begin{bmatrix}\mathrm{HEAD} : \boxed{1} \\ \mathrm{VAL} : \begin{bmatrix}\mathrm{SUBJ} : \textbf{list} \\ \mathrm{SPR} : \textbf{list} \\ \mathrm{CPS} : \textbf{list}\end{bmatrix}\end{bmatrix} \\ \mathrm{HD\text{-}DTR\ SS} : & \begin{bmatrix}\mathrm{HEAD} : \boxed{1} \\ \mathrm{VAL} : \begin{bmatrix}\mathrm{SUBJ} : \textbf{list} \\ \mathrm{SPR} : \textbf{list} \\ \mathrm{CPS} : \textbf{list}\end{bmatrix}\end{bmatrix}\end{bmatrix} \Big/ \begin{array}{l}\{\langle\begin{bmatrix}\mathrm{SS\ VAL} : \boxed{1} \\ \mathrm{HD\text{-}DTR\ SS\ VAL} : \boxed{1}\end{bmatrix}, \textbf{hd-phr}\rangle, \\ \langle\begin{bmatrix}\mathrm{SS\ VAL\ SUBJ} : \boxed{1} \\ \mathrm{HD\text{-}DTR\ SS\ VAL\ SUBJ} : \boxed{1}\end{bmatrix}, \textbf{hd-phr}\rangle, \\ \langle\begin{bmatrix}\mathrm{SS\ VAL\ SPR} : \boxed{1} \\ \mathrm{HD\text{-}DTR\ SS\ VAL\ SPR} : \boxed{1}\end{bmatrix}, \textbf{hd-phr}\rangle, \\ \langle\begin{bmatrix}\mathrm{SS\ VAL\ CPS} : \boxed{1} \\ \mathrm{HD\text{-}DTR\ SS\ VAL\ CPS} : \boxed{1}\end{bmatrix}, \textbf{hd-phr}\rangle, \\ \langle\big[\mathrm{HD\text{-}DTR\ SS\ VAL\ CPS} : /\langle\rangle\big]\rangle\}\end{array}
$$

Figure 15
Encoding VALP using the Basic TDFS notation (SYNSEM is abbreviated SS, COMPS is CPS)

(i.e., the value for the CONTENT feature) for the lexical sign for *dog* is shown in (7a). This has an interpretation roughly equivalent to that of (7b).

(7)a. $\begin{bmatrix} \text{INDEX} : \boxed{1}\left[\text{AGR NUM} : \textbf{sg}\right] \\ \text{LISZT} : \left\langle \begin{bmatrix} \textbf{\_dog\_rel} \\ \text{INST} : \boxed{1} \end{bmatrix} \right\rangle \end{bmatrix}$

b. $\lambda x[\text{dog}(x)]$

The feature INDEX in (7a) indicates the equivalent of the lambda variable in (7b). Features on the index indicate agreement, in the usual way in HPSG: here the noun will agree with a verb that takes singular agreement (we only consider number agreement here). The LISZT contains a list of relations, which is a singleton here as for most lexical signs (composition of the semantics proceeds by appending LISZTs). The type of the relation, **_dog_rel**, indicates the predicate name. The initial underscore in the type name is a notational convention to indicate a lexical predicate, so that we can for instance distinguish the type **noun_rel**, which is a general type for noun relations in the type hierarchy, and **_noun_rel** which is the relation corresponding to the noun *noun*. The argument to the predicate is indicated by INST, which is coindexed with the value of INDEX.

This structure has to be augmented to represent semantic plurality and individuation (i.e., the mass/count distinction) since the AGR NUM feature will not always make the appropriate distinctions. Although in normal count nouns (such as *dog* in its most usual interpretation) number agreement and semantics are in step, this is not true for all nouns. Some nouns, such as *scissors* or *trousers*, that denote bipartite objects have obligatory plural agreement. Other count nouns, such as *gallows* and *barracks*, show variable agreement, even when referring to a single object (Quirk *et al.*, 1985). Mass terms usually take singular agreement, but there are exceptions, such as *clothes*, which although it behaves semantically as a mass term, nonetheless takes plural agreement. Here we will assume that semantic plurality/individuation is indicated by a predicate modifier, as illustrated in 8.

(8)	single entity	$\lambda x[{}^{S}\text{truth}(x)]$	One truth is self-evident.
	plural entity	$\lambda x[{}^{P}\text{truth}(x)]$	Some truths are self-evident.
	unindividuated entity	$\lambda x[{}^{M}\text{truth}(x)]$	There is much truth in that.

For simplicity, we have assumed that all occurrences of noun predicates are modified either by M, P or S (corresponding to mass, singular and plural) though it does not matter if one of these subcases is taken to correspond to the unmodified predicate, or if the structure assumed is more complex, since all that matters for current purposes is that there is some three-way distinction in the formal semantics. Similarly, we need not go into the details of the corresponding models (though see Krifka (1987) for example).

One way of capturing the distinction in MRS is to add another feature to the relation to record the predicate modifier, which we will call PLMOD. The values of this slot are irrelevant from the perspective of the formal semantics, as long as we can make a three-way distinction. However, in order to facilitate the provision of the default relationship between semantic values and agreement, we will use the values **sg**, **pl** and **mass**, where the first two types also correspond to possible values of the AGR NUM slot. The hierarchy for these atomic types is shown at the top of Figure 16.

We can now capture the generalizations about semantics and agreement as shown in Figure 16. The type **noun-sign** is a subtype of **sign**, and **mass-sign** is a subtype of **noun-sign** (along with types for count nouns, pair nouns and so on, which are not shown here). Encoding the generalizations at the **sign** level allows inflectional information to be included, though we do not show this here. But another reason for not using the

rel hierarchy is that we want types such as **_truth_rel** to be neutral between mass and count. The default generalization about nouns is that the value of the PLMOD and the AGR NUM paths are coindexed. This default will remain intact for ordinary count nouns. However, for mass nouns, as shown in Figure 16, there is a default value for the AGR NUM feature of **sg** and a non-default value of **mass** for the PLMOD feature. The lexical entries for *clothing* and *clothes* both inherit from **mass-noun**, but the latter has AGR NUM of **pl**. We make the usual assumption that *DefFill* operates on these lexical entries, to give the results shown in Figure 17. So for this example, we have made use of default reentrancy, and the fact that a default value can survive on a node for which a default path equivalence was overridden, which was not the case in PDU.[10]

4.4 Persistent defaults and the interface between the lexicon and pragmatics

The utility of persistent defaults in the interface between the lexicon and pragmatics was discussed in some detail in Lascarides *et al.* (1996), and from a more linguistic perspective in Lascarides and Copestake (in press). However, since this sort of use was an important part of the motivation for developing PDU and YADU, we will give one example here, which was not discussed in the former paper and was only considered briefly and informally in the latter.

Verbs such as *drink*, *eat* and *bake* have both intransitive and strict transitive uses. Both uses have very similar meanings, since for these verbs intransitive use implies a patient (in contrast to *kick*, for instance), but as pointed out by Fillmore (1986) and others, the intransitive uses have more specific default interpretations. In the absence of information to the contrary, (9a) means (9b), (10a) means (10b), and (11a) means (11b):

(9) a. John drinks all the time.
 b. John drinks alcohol all the time.

(10) a. We've already eaten.
 b. We've already eaten a meal.

(11) a. I spent yesterday afternoon baking.
 b. I spent yesterday afternoon baking cookies, cakes or bread. (as opposed to ham, apples or potatoes, for example)

These defaults can be overridden by arbitrary background information, for example:

(12) As long as we're baking anyway, we may as well do the ham now too.
(due to Silverstein, cited in Fillmore (1986))

A purely pragmatic explanation for the default for *bake* seems implausible, since there is no evident real world explanation. For instance it would be difficult to claim that people usually bake flour-based products as opposed to natural ones. A historical justification could be suggested, since the Oxford English Dictionary (2cd edition) says of *bake*:

> primarily used of preparing bread, then of potatoes, apples, the flesh of animals.

10 Note that it is not actually the case for this example that a default value is overriding a default path equivalence, although this is allowed in YADU, as we have discussed. The value of AGR NUM is indefeasibly **agrnum** which is incompatible with the indefeasible value **mass** for the PLMOD of **mass-noun**s. However, in PDU this conflict would have resulted in the value $\perp$ on the relevant node in the default structure, and the default value **sg** on AGR NUM would thus in effect have been ignored.

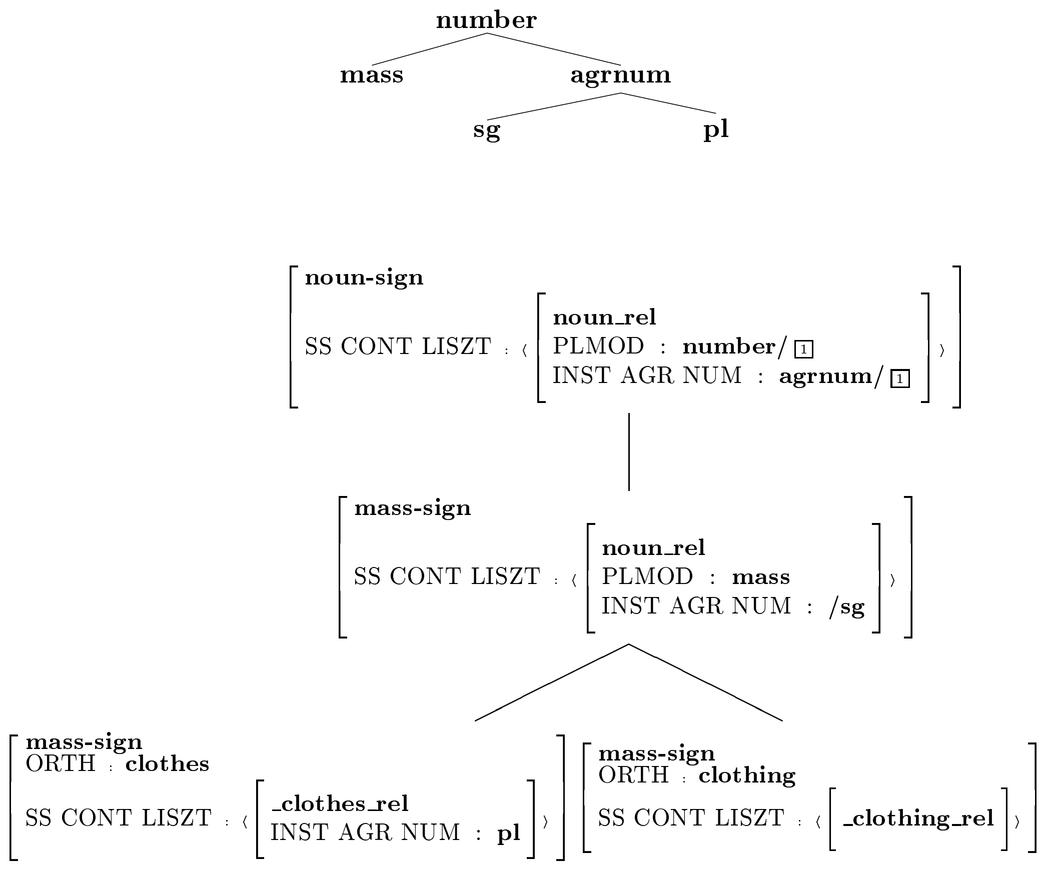

number
mass
agrnum
sg
pl
noun-sign
SS CONT LISZT : ⟨ noun_rel PLMOD : number/1 INST AGR NUM : agrnum/1 ⟩
mass-sign
SS CONT LISZT : ⟨ noun_rel PLMOD : mass INST AGR NUM : /sg ⟩
mass-sign
ORTH : clothes
SS CONT LISZT : ⟨ _clothes_rel INST AGR NUM : pl ⟩
mass-sign
ORTH : clothing
SS CONT LISZT : ⟨ _clothing_rel ⟩

Figure 16
Relating agreement and semantics for nouns (SYNSEM is abbreviated SS)

$$\begin{bmatrix} \textbf{mass-sign} \\ \text{ORTH} : \textbf{clothes} \\ \text{SYNSEM CONT LISZT} : \left\langle \begin{bmatrix} \textbf{\_clothes\_rel} \\ \text{PLMOD} : \textbf{mass} \\ \text{INST AGR NUM} : \textbf{pl} \end{bmatrix} \right\rangle \end{bmatrix}$$

$$\begin{bmatrix} \textbf{mass-sign} \\ \text{ORTH} : \textbf{clothing} \\ \text{SYNSEM CONT LISZT} : \left\langle \begin{bmatrix} \textbf{\_clothing\_rel} \\ \text{PLMOD} : \textbf{mass} \\ \text{INST AGR NUM} : \textbf{sg} \end{bmatrix} \right\rangle \end{bmatrix}$$

Figure 17
Expanded and DefFilled lexical signs

$$\left[\begin{array}{l} \text{ORTH} : \textbf{bake} \\ \text{SYNSEM} : \left[\begin{array}{l} \text{VAL COMPS} : \left\langle \left[\begin{array}{l} \text{OPT} : \textbf{true} \\ \text{HEAD} : \textbf{noun} \\ \text{VAL COMPS} : \langle\rangle \\ \text{CONT} : \left[\begin{array}{l} \text{INDEX} : \boxed{y} \\ \text{LISZT} : \boxed{1} /_p \left\langle \left[\begin{array}{l} \textbf{flour-based\_rel} \\ \text{INST} : \boxed{y} \end{array}\right] \right\rangle \end{array}\right] \end{array}\right] \right\rangle \\ \text{CONT LISZT} : \left\langle \left[\begin{array}{l} \textbf{\_bake\_rel} \\ \text{ARG1} : \boxed{x} \\ \text{ARG2} : \boxed{y} \end{array}\right] \right\rangle \oplus \boxed{1} \end{array}\right] \end{array}\right]$$

Figure 18
Simplified representation for intransitive and transitive *bake*

However, synchronically, the default interpretation seems to have become lexicalised: even if the speaker often cooks potatoes by baking but very rarely prepares bread or cakes, (11a) still implies (11b). This implies that the default is associated conventionally with the word *bake*, rather than arising as a consequence of its meaning. The assumption we make about the boundary between the lexicon and pragmatics is that the lexicon is responsible for encoding the relationship between word-forms and meanings, and that pragmatics only has access to meanings. Under these assumptions, the default must be encoded lexically, but in such a way that it can be overridden in the right discourse context. This motivates the use of a persistent default: that is, one which is not converted to hard information, in contrast to the descriptive defaults discussed in the previous examples (for further details and justification of these general assumptions, see Lascarides and Copestake, in press).

One way of describing an entry for *bake* which covers both transitive and intransitive uses is sketched in Figure 18. We first go over the indefeasible part of the structure, which will be the same for other verbs which take an optional noun phrase complement. The VAL COMPS value is a singleton, which is constrained to have HEAD **noun** and an empty complements list: i.e., to be a noun phrase. However, unlike in our earlier examples, we have also shown the feature OPT on the complement. A complement is treated as optional if its value of OPT is (compatible with) **true** (the details of the schemata which achieve this are not relevant here). As before we use the MRS style of encoding semantics in TFSs. The INDEX value of the complement is coindexed with the object slot in the main relation for the semantics of the verb. The MRS structure for the verb relation corresponds to bake(x, y) where x is coindexed with the index on the subject, and y with the object. The $\oplus$ in the CONT LISZT value of the verb is shorthand for a complex feature structure which has the effect of appending the noun phrase semantics to that of the verb.

The default part of the structure concerns the semantics for the noun phrase complement, which conventionally would not be instantiated. Here, however, the LISZT is stated, by default, to be the singleton relation **flour-based_rel**.[11] Note the subscript p, indicat-

11 In a full treatment the default semantics would also contain a quantifier. In fact the implicit object must have narrow scope existential quantification. Combining YADU with a semantic representation capable of representing quantifier scope in an underspecified fashion means that this can be made to follow from a general default assignment of scope. However the details are outside the scope of this paper.

ing that the default is persistent: that is, it is not subject to the lexical *DefFill.* The effect of this is that the object of *bake* is given a default semantics indicating that it is a flour-based substance. We assume that **flour-based_rel** is a type which is incompatible with all lexical relational types, and thus any explicit object will override this specification. However, in an intransitive use the default semantics will be retained, giving a representation which can be represented in a linear notation as: bake$(e, x, y) \wedge$ /flour-based(y). The manner in which this can be overridden by pragmatics in examples such as (12) is outside the scope of this paper, but is discussed in Lascarides and Copestake (in press).

With a suitably rich constraint language, one could devise a monotonic encoding which would allow for an underspecified entry for *bake* which could be specialised either to have an obligatory complement or to be strictly intransitive with the additional **flour-based_rel**. However such a treatment would not allow for overriding by pragmatics in contexts such as (12). Furthermore, it should be clear that this sort of use of defaults is only possible if defaults can be distinguished from indefeasible parts of the structure, and if they persist beyond the lexicon, and that an approach such as PDU or YADU is therefore required for such examples.

5. Theoretical and Practical Complexity Issues

When discussing the complexity properties of YADU, we have to distinguish between the $\overset{<>}{\sqcap}$ operation, which involves the combination of two TDFSs, and *DefFS*, the calculation of the default structure from a TDFS. One reason for drawing this distinction is that, in a practical system, it may be necessary to carry out the former operation much more frequently than the latter. As we mentioned in §3, it is only necessary to construct the default feature structure at some interface: for example, at the interface between the lexicon and the parser/generator (as in the examples in §4.1, §4.2 and §4.3) or between the grammar and the pragmatic component (as in the examples in §4.4). In fact, only one *DefFS* operation is necessary per lexical entry (or other grammar object), regardless of the depth or complexity of the inheritance hierarchy. Similarly, one *DefFS* is required per parse for the use of persistant defaults. This is fortunate, since the combination operation has considerably better complexity properties than the calculation of the default structure.

$\overset{<>}{\sqcap}$ simply involves unification of typed feature structures, set union of tails and removal of tail elements which are incompatible with the indefeasible structure. Checking path-value tail elements for unifiability with the indefeasible structure involves checking one pair of types to see if they are compatible. Checking path-equivalence elements is roughly equivalent to unifying the relevant parts of the indefeasible structure: in the worst case this could amount to unifying two TFSs of $(n-1)/2$ nodes each per tail-element, where n is the number of nodes in the indefeasible structure. But, although we defined $\overset{<>}{\sqcap}$ as requiring the elimination of tail elements which are incompatible with the default, we could equivalently have left this to the *DefFS* operation, and simply accumulated tail elements via set union. This is an insignificant overhead on normal unification, since the tail elements have very simple structures.

In contrast, the worst case properties of the calculation of the default TFS are unpleasant. Recall that this operation involves partitioning the tail and then carrying out a step similar to asymmetric default unification as described by Carpenter (1993) for each partition. The only known algorithms for computing asymmetric default unification are factorial in the number of atomic FSs in the worst case (Carpenter 1993). Thus for a partition with t tail elements, the worst case performance is proportional to $t!$ in the number of individual unification operations, where each unifcation could involve up to $(n-1)/2$ nodes, as above.

In practice, YADU is much better behaved on realistic examples than this would suggest. The first point to make is that, as far as operations on TFSs themselves are concerned, the *DefFS* operation can be implemented as a series of normal unification steps. One consequence of this is that the use of YADU does not incur any significant overhead with respect to ordinary unification when there are no tail elements. The only additional requirement for YADU is that there be a slot at the top level of an object's representation to store its tail. We mention this because it contrasts with some other extensions to TFS formalisms: implementations of disjunction, for example, generally require that the TFS data structures and unification algorithms be considerably more complex, with a consequent overhead in performance even for non-disjunctive structures. Another mitigating factor is that it is easy for a grammar writer to tell in advance whether a particular use of defaults is likely to be computationally expensive. The worst-case complexity behaviour only occurs in situations where there are interactions between path equivalence statements which are not resolved by specificity. While it is possible to invent pathological examples which have very deleterious performance characteristics, it is not clear that comparable cases will often arise in real grammars, especially if defaults are being used in a rather conservative fashion to extend a monotonic core grammar. Consider the example in §4.1 for instance. There is only a single path value element in the tails of any of the structures described, and this will straightforwardly either conflict or be compatible with an indefeasible atomic value. Indeed in examples like this, the reduction in the numbers of types involved compared to the purely monotonic encoding could potentially lead to an efficiency gain.

As we mentioned above, since PDU is polynomial, it has much better worst-case behaviour than YADU. However, making a realistic comparison is not straightforward. The combination operation in PDU is more complex than in YADU, since it is necessary to calculate default TFSs at every stage as well as tails. The constant overhead compared to ordinary unification is higher and the implementation of PDU is trickier than YADU. Furthermore, there is a trade-off between complexity behaviour and intuitiveness. The reason that PDU has better complexity behaviour is that it *always* accumulates default reentrancies. If there is a clash with default values, the default reentrancies win — if there is a clash with indefeasible values, the coindexed nodes in the default structure are set to $\perp$, indicating inconsistency, and the *DefFill* operation must subsequently take care of constructing a valid TFS by removing the default path equivalences. However, this can lead to cases where potentially valid default path equivalences are removed.

Thus in PDU the sort of examples which lead to the factorial worst-case complexity in YADU are treated specially, in that maximal information is not incorporated from the default structure. Roughly speaking, for these corner cases, there is a trade-off between the complexity behaviour in YADU, and the complex behaviour of PDU.[12] But our main practical reason for preferring YADU over PDU is that PDU can behave in unintuitive ways in examples where YADU would have non-problematic complexity behaviour. It is also worth noting that YADU will not be slow if the tail partitions are kept small, which is something the grammar writer can control.

6. Extensions and Alternatives

In this section, we briefly consider some variants on the definition of YADU which have utility in specific circumstances.

12 Similar remarks also apply to the contrast between Bouma's (1990, 1992) and Carpenter's (1993) versions of asymmetric default unification.

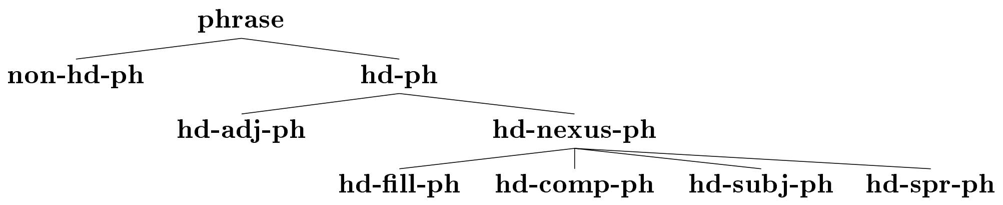


Figure 19
Hierarchy of phrases from Sag (1997)

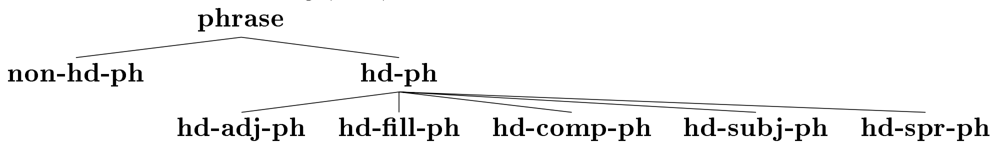


Figure 20
Simplification of hierarchy

$$\begin{bmatrix} \text{SYNSEM CONT} : / \boxed{1} \\ \text{HD-DTR SYNSEM CONT} : / \boxed{1} \end{bmatrix}$$

Figure 21
Default version of the semantics principle

$$\text{head-adjunct-phrase} \quad \begin{bmatrix} \text{SYNSEM CONT} : \boxed{1} \\ \text{HD-DTR SYNSEM CONT} : \boxed{2} \\ \text{NON-HD-DTRS SYNSEM CONT} : \boxed{1} \end{bmatrix}$$
$$\boxed{1} \not\sim \boxed{2}$$

Figure 22
Inequalities overriding default equalities

6.1 Inequalities

We have come across a number of cases where it would be useful to be able to override a default reentrancy without specifying conflicting values for the paths involved. For example, consider the type hierarchy shown in Figure 10 and repeated in Figure 19 for convenience. For most of the subtypes of **headed-phrase**, the CONTENT of the mother should be equivalent to the CONTENT of the head daughter. This holds for **head-subject-phrase**, **head-comps-phrase** and **head-specifier-phrase** and their subtypes, but it is not true for **head-adjunct-phrase**s, where the content value of the mother is equal to the content value of the single non-head-daughter. It would seem natural to specify the generalisation on the supertype **headed-phrase** as a default constraint, as shown in Figure 21, and to override the default on the subtype **head-adjunct-phrase**. This would allow the simplification of the hierarchy as shown in Figure 20. However, in standard YADU, there is no way to express the idea that the coindexation between mother and non-head-daughter should hold instead of the coindexation between mother and head-daughter, since, as far as this structure goes, these coindexations are mutually compatible. Of course the CONTENT values of the head- and non-head- daughters should not be unified in any instantiation of this schema, but since the range of values for each is indefinitely large, there is no way of giving them mutually incompatible types. Thus the type **head-nexus-phrase** had to be introduced, as a place to state a monotonic constraint on the relationship between semantics values but this type is otherwise unmotivated and somewhat unintuitive.

This sort of situation can be avoided by making use of *inequalities*, as defined by Carpenter (1992). Intuitively, what is required in order to specify the constraint in Figure 21 on **headed-phrase** is to say that the constraint on the schema head-adjunct-phrase stipulates explicitly that its head-daughter content is not equal to the content on the mother, as shown in Figure 22.

To achieve this formally takes only a very minor modification to the definitions already given. First, one must change the definition of TFSs and tails, so that they include inequalities. The relation $\not\leftrightarrow \subseteq Q \times Q$ is added to the tuple that currently defines TFSs (Definition 2), and a fifth condition is added to the four that are already in that definition, which ensures that $\not\leftrightarrow$ is a relation of the right sort (see Carpenter, 1992):

5. $\not\leftrightarrow \subseteq Q \times Q$ *is an anti-reflexive and symmetric relation.*

Atomic FSs in tails are extended, so that they include path inequalities (as well as the existing path:values and path equalities). The definition of TDFSs is the same as before, except that it's now based on this new definition of TFSs and the new tails.

Second, one changes the definition of subsumption as in Carpenter (1992). First, some notation: $\pi \not\equiv_F \pi'$ means $\delta(r, \pi) \not\leftrightarrow \delta(r, \pi')$, where r is the root node of the TFS.

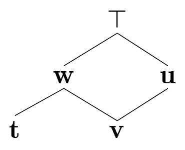
Definition 19
Inequated Subsumption
F subsumes F', written $F' \sqsubseteq F$, if and only if:

- $\pi \equiv_F \pi'$ implies $\pi \equiv_{F'} \pi'$
- $\pi \not\equiv_F \pi'$ implies $\pi \not\equiv_{F'} \pi'$
- $\mathcal{P}_F(\pi) = t$ implies $\mathcal{P}_{F'}(\pi) = t'$ and $t' \sqsubseteq t$

The definitions of $\sqcap$ and $\sqcup$ remain the same, save that the new notion of inequated subsumption is used. The resulting operations are still well-behaved, in that they are order independent, and return a single result deterministically (see Carpenter, 1992). The definitions of $\overset{<>}{\sqcap}$ and *DefFS DefFill* and *BasicTDFS* all remain the same. And the lemmas and theorems given in §3.7 still hold, with the proofs unchanged as given in the appendix. These proofs still hold, largely because they depend only on the well-behaved nature of set union, $\sqcap$ and $\sqcup$.

Note that an inequality can arise in a derived TDFS (or its corresponding default TFS) only if there was that inequality in one of the TDFSs (or tails) that were used to build it via $\overset{<>}{\sqcap}$. Inequalities that weren't explicitly in this input never arise in the result. Consequently, this corresponds to a relatively weak notion of negation. One might learn through $\sqcap$ or through $\overset{<>}{\sqcap}$ that two nodes can't be equal because they have incompatible types. But this doesn't mean that these nodes stand in the inequality relation defined by $\not\leftrightarrow$. However, one can always convert a TFS into a unique most general fully inequated TFS, as defined in Carpenter (1992) (where a fully inequated TFS is one where any two incompatibly typed nodes in the TFS stand in the inequality relation defined by $\not\leftrightarrow$). Thus one can define a version of *DefFS* which always outputs a unique, fully inequated default TFS also. Furthermore, every TDFS has a unique most general fully inequated TDFS: it amounts to the unique most general fully inequated TFS, plus the tail.

As far as we are aware, no other version of default unification has been specified which allows for inequalities. In particular, PDU cannot be extended straightforwardly to handle inequalities, because it's computed on a path by path basis. Consequently, one gets an ill-formed TDFS when one attempts to PDU a TDFS with an indefeasible path equality, and a TDFS with a default inequality on the same paths. We think that the fact that incorporating default inequalities is possible with such a small change to the definition of YADU attests to its elegance.



Standard definition $\mathit{DefFS}(\mathit{BasicTDFS}([\,\text{F} : \top\,], [\,\text{F} : \mathbf{t}\,]) \overset{<>}{\sqcap} [\,\text{F} : \mathbf{u}\,]/\{\}) = [\,\text{F} : \top\,]$

Fine-grained definition $\mathit{DefFS}(\mathit{BasicTDFS}([\,\text{F} : \top\,], [\,\text{F} : \mathbf{t}\,]) \overset{<>}{\sqcap} [\,\text{F} : \mathbf{u}\,]/\{\}) = [\,\text{F} : \mathbf{v}\,]$

Figure 23
Effect of fine-grain tails

6.2 Specificity ordering

Note that although we have consistently used the type hierarchy to give a specificity ordering to tail elements, the only real requirement to be able to define *DefFS* is that the tail elements have specificity markers which are in a partial order. Hence the defaults that 'win' in a TDFS could be determined by an ordering other than the type hierarchy. In fact, any partial order could be utilised: all that is necessary is to indicate the specificity in the tails and to make the definition of the partition of tails sensitive to the relevant partial order. Specifically, the second member of the pairs in the tails, which we have defined as types, should be replaced with specificity information of the relevant sort, and the specificity partition of a tail defined accordingly. The definitions of $\overset{<>}{\sqcap}$ and *DefFS* then proceed as before.

The fact that prioritisation of defaults need not be linked to the type hierarchy means that it is straightforward to adapt YADU to untyped feature structures or, in general, to a system where some form of templatic inheritance is used instead of the type hierarchy. It also might be useful if an ordering is given by some component extrinsic to the FSs, such as an ordering based on probabilities. It would even be possible to add a finer-grain of specificity to the type hierarchy, by creating a specificity ordering of tail elements within types, for instance so that for a type t, specificity markers $t^1, t^2 \ldots t^n$ were defined so that within-type priority following numerical ordering. The potential utility of this is shown by the example in §2, where the two types **verb** and **regverb** were distinguished simply in order to acheive the correct prioritorisation. An alternative would have been to use a single type **verb** with two distinct specificity markers $\mathbf{verb}^1$ and $\mathbf{verb}^2$ to get the desired priorities on the defaults.

6.3 Fine-grained structures

One point that we glossed over slightly is the use of atomic FSs within a typed framework (as opposed to the untyped FSs assumed in Carpenter (1993)). In the definition for *BasicTDFS* given in §3, we assumed that if a path π had a value $\mathbf{t}$, then there would be one corresponding path-value atomic FS in the tail. But there is another possibility, which is to have additional structures in the tail, corresponding to each supertype of $\mathbf{t}$: e.g., if $\mathbf{w}$ were a supertype of $\mathbf{t}$, then there would also be an atomic FS in the tail where the path π was associated with the value $\mathbf{w}$. This would give a finer-grained notion of maximal incorporation of information, since there might be a situation where $\mathbf{t}$ was incompatible with a type $\mathbf{u}$ in the non-default FS (or it was incompatible with a more specific default FS) but where $\mathbf{u} \sqcap \mathbf{w}$ resulted in some more specific type $\mathbf{v}$, which would survive in the YADU result (see Figure 23).

To extend tails this way, one must change the definition of basic TDFSs, to remove the condition (c) from the original definition, which ensured that only the most specific information was included in the tail. So the new definition is:

Definition 20
Fine-Grained Basic TDFSs
Let I and I_D be typed feature structures, where I is regarded as indefeasible and I_D as defeasible. Furthermore, suppose that $I \sqcap I_D \neq \bot$ (so I and I_D are compatible). Then the fine-grained basic TDFS $\mathit{BasicTDFS}(I, I_D)$ of I and I_D is the TDFS I/T, such that:

$$\begin{array}{ll} T = \{\langle F, t\rangle : & t \text{ is the root type on } I_D \sqcap I\text{, and } F \text{ is an atomic TFS such that:} \\ & \text{(a) } I \not\sqsubseteq F; \\ & \text{(b) } I_D \sqcap I \sqsubseteq F\} \end{array}$$

The existing definitions of $\overset{<>}{\sqcap}$ and *DefFS* will then provide the finer-grained notion of maximal incorporation of default information, from these fine-grained basic TDFSs.

Extending tails this way is useful for the treatment of lexical rules, as discussed in Briscoe and Copestake (1995). However it has the obvious disadvantage of considerably increasing the number of atomic FSs which must be considered, with adverse effects on efficiency.

6.4 Credulous YADU

Another way in which the definition could be varied would be to omit the generalization step from *DefFS*, which ensures that the default result of a TDFS is a single TFS, and to have a credulous variant of *DefFS* instead, which would be analogous to Carpenter's (1993) credulous asymmetric default unification:

Definition 21
Credulous DefFS
Let F be a TDFS I/T. Then

$$\mathit{DefFS}(F) = (I \overset{<}{\sqcap}_{cs} \wp_{fs}(\mu_1)) \overset{<}{\sqcap}_{cs} \ldots \overset{<}{\sqcap}_{cs} \wp_{fs}(\mu_n)$$

Where $\langle \mu_1, \ldots, \mu_n \rangle$ is a specificity partition on T.

We argued in §1.1 that a unique result is preferable in order to avoid multiplication of disjunctive structures, but disjunctive results might be useful in cases where there are multiple alternative structures (e.g., in modeling *dreamed/dreamt*: see Russell *et al.*, 1993).

6.5 Asymmetric default unification

We should point out that although we believe order-independent default unification is preferable to asymmetric default unification for many applications, there are situations where the latter is required. YADU could not replace asymmetric default unification in Grover *et al.*'s (1994) treatment of ellipsis. It is also not directly suitable for encoding lexical rules: it is conventional to write lexical rules using a sort of default notation which is intended to be interpreted as meaning that the output of the rule is identical to the input except where otherwise specified, but formalizing this calls for an asymmetric notion of default (see Briscoe and Copestake, 1995). Similarly, Copestake (1992) argues that it is useful to be able to encode irregular lexical entries as inheriting by default from the output of lexical rule application (e.g., the entry for *children* could inherit from the result of applying a lexical rule for plural formation to the entry for *child* but override the orthography). This requires asymmetric default unification, where the TFS which results from the application of the lexical rule is treated as defeasible and the specification on the lexical entry is treated as hard information. The current LKB implementation thus allows both types of default unification (which is straightforward, since YADU is implemented using a series of asymmetric default unification operations).

7. Conclusion

We have argued that for default unification to achieve the combination of perspicuity and declarativity familiar from normal unification, default unification should share some of its properties— such as determinacy and order independence. At the same time, default unification should respect the behavior of defaults, such as the overriding of default information by more specific conflicting defaults. We have also argued here and elsewhere (Lascarides and Copestake, in press) that some linguistic phenomena suggest that there are conventional default constraints which persist beyond the lexicon, and are potentially overridden by more open-ended reasoning with (default) pragmatic knowledge in a discourse context. This requires a definition of default unification where the default results of unification are marked as default, and thus distinguished from the indefeasible results. We provided a definition of default unification known as YADU, which intuitively models the incorporation of the maximal amount of default information into the result, by adapting Carpenter's (1993) version of asymmetric default unification to the situation where default and non-default information is distinguished in a single structure and defaults may have different priorities. Our definition was formally proven to meet the above requirements. We suggested that such a definition of default unification can improve the declarativity of existing uses of default inheritance within the lexicon because it does not require one to pre-specify the order in which information is to be accumulated.

Despite YADU's factorial worst-case complexity behavior, its use does not significantly decrease overall system performance when compared to a monotonic encoding for the examples we have tried in the LKB system. These results are preliminary and obviously only true relative to our particular implementation and style of grammar encoding, but they lead us to believe that the worst-case complexity behavior does not preclude the use of YADU in typed feature structure implementations. Although we only discussed a few examples in §4, we believe these illustrate the potential utility of defaults in a range of different contexts within a grammar and lexicon. We hope to report on a comparison between the monotonic and YADU versions of the English Resource Grammar in a later paper.

Acknowledgments
This work is sponsored by a grant funded by ESRC UK (grant number R000236052), and by the ESRC-funded Human Communication Research Centre, University of Edinburgh. This material is also in part based upon work supported by the ESPRIT Acquilex-II, project BR-7315, grant to Cambridge University and National Science Foundation under grant number IRI-9612682, to Stanford University. We would like to thank Ted Briscoe, Dan Flickinger, Ivan Sag, Hidetoshi Sirai and three anonymous reviewers for helpful comments on previous drafts.

References

Alshawi, Hiyan, Doug J. Arnold, Rolf Backofen, David M. Carter, Jeremy Lindop, Klaus Netter, Stephen G. Pulman, Junichi Tsujii, Hans Uszkoreit. (1991). "Eurotra ET6/1: Rule Formalism And Virtual Machine Design Study (final report)." CEC, Luxembourg.

Asher, Nicholas and Michael Morreau. (1991). Common Sense Entailment: A Modal Theory of Nonmonotonic Reasoning. *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, Sydney, Australia.

van den Berg, Martin and Hub Prüst. (1991). Common Denominators and Default Unification. *Proceedings of the First Meeting, Computational Linguistics in the Netherlands (CLIN-91)*, Utrecht: 1–16.

Brewka, Gerhard. (1991). Cumulative

Default Logic: In Defense of Nonmonotonic Inference Rules. *Artificial Intelligence* 50(2): 183–205.

Boguraev, Bran and James Pustejovsky. (1990). Lexical Ambiguity and the Role of Knowledge Representation in Lexicon Design. *Proceedings of the 13th International Conference on Computational Linguistics (COLING-90)*, Helsinki: 36–42.

Bouma, Gosse. (1990). Defaults in Unification Grammar. *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics (ACL-90)*, Pittsburgh, USA: 165–173.

Bouma, Gosse. (1992). Feature Structures and Nonmonotonicity. *Computational Linguistics* 18(2): 183–204.

Briscoe, Edward J., Ann Copestake and Bran Boguraev. (1990). Enjoy the Paper: Lexical Semantics via Lexicology. *Proceedings of the 13th International Conference on Computational Linguistics (COLING-90)*, Helsinki: 42–47.

Briscoe, Edward J. and Ann Copestake. (1995). "Dative Constructions as Lexical Rules in the TDFS framework." Acquilex-II Working Papers 78, University of Cambridge Computer Laboratory, Cambridge, England.

Calder, Jo. (1991). "Some Notes on Priority Union." Paper presented at the ACQUILEX Workshop on Default Inheritance in the Lexicon, Cambridge, England.

Carpenter, Bob. (1992). *The Logic of Typed Feature Structures.* Cambridge University Press, Cambridge, England.

Carpenter, Bob. (1993). "Skeptical And Credulous Default Unification With Application To Templates And Inheritance." In *Inheritance, Defaults and the Lexicon,* edited by Edward J. Briscoe, Ann Copestake and Valeria de Paiva. Cambridge University Press, Cambridge, England, 13–37.

Copestake, Ann. (1992). The Representation of Lexical Semantic Information. D.Phil. dissertation, University of Sussex, Brighton, England. Cognitive Science Research Paper CSRP 280.

Copestake, Ann. (1993). "Defaults in Lexical Representation." In *Inheritance, Defaults and the Lexicon,* edited by Edward J. Briscoe, Ann Copestake and Valeria de Paiva. Cambridge University Press, Cambridge, England, 223–245.

Copestake, Ann, Daniel Flickinger, Rob Malouf, Susanne Riehemann, Ivan Sag. (1995). Translation using Minimal Recursion Semantics. *Proceedings of the 6th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-95)*, Leuven, Belgium: 15–32.

Daelemans, Walter. (1987). A Tool for the Automatic Creation, Extension and Updating of Lexical Knowledge Bases. *Proceedings of the 3rd Conference of the European Chapter of the Association for Computational Linguistics (EACL-87)*, Copenhagen: 70–74.

Daelemans, W., Koenraad de Smedt and Gerald Gazdar. (1992). Inheritance in Natural Language Processing. *Computational Linguistics* 18(2): 205–218.

Dörre, Jochen and Andreas Eisele. (1991). "A Comprehensive Unification-based Grammar Formalism." DYANA Technical Report, University of Edinburgh, Scotland.

Emele, Martin and Rémi Zajac. (1990). Typed Unification Grammars. *Proceedings of the 13th International Conference on Computational Linguistics (COLING-90)*, Helsinki: 293–298.

Evans, Roger and Gerald Gazdar. (1989a). Inference in DATR. *Proceedings of the 4th Conference of the European Chapter of the Association for Computational Linguistics (EACL-89)*, Manchester, England: 66–71.

Evans, Roger and Gerald Gazdar. (1989b). "The Semantics of DATR." In *Proceedings of the Seventh Conference of the Society for the Study of Artificial Intelligence and Simulation of Behavior (AISB-89),* edited by Anthony G. Cohn. Pitman/Morgan Kaufmann, London, 79–87.

Evans, Roger and Gerald Gazdar. (1996). DATR: A Language for Lexical Knowledge Representation. *Computational Linguistics* 22(2): 167–216.

Fillmore, Charles J.. (1986). Pragmatically Controlled Zero Anaphora. *BLS* 12: 95–107.

Flickinger, Daniel. (1987). Lexical Rules in the Hierarchical Lexicon. PhD dissertation, Stanford University, Stanford, CA.

Flickinger, Daniel, Carl Pollard and Tom Wasow. (1985). Structure Sharing in Lexical Representation. *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics (ACL-85)*, University of Chicago: 262–268.

Flickinger, Daniel and John Nerbonne. (1992). Inheritance and

Complementation: A Case Study of *easy* Adjectives and Related Nouns. *Computational Linguistics* 18(3): 269–310.

Flickinger, Daniel, Ivan Sag, Ann Copestake. (in preparation). “A Grammar of English in HPSG: Design and Implementation.” CSLI Publications, Stanford, CA.

Gazdar, Gerald. (1987). “Linguistic Applications of Default Inheritance Mechanisms.” In *Linguistic Theory and Computer Applications,* edited by Peter Whitelock, Harold Somers, Paul Bennett, Rod Johnson, and Mary McGee Wood. Academic Press, London, 37–68.

Gerdemann, Dale and Paul King. (1994). The correct and efficient implementation of appropriateness specifications for typed feature structures. *Proceedings of the 15th International Conference on Computational Linguistics (COLING-94)*, Kyoto, Japan.

Grover, Claire, Chris Brew, Suresh Manandhar and Marc Moens. (1994). Priority Union and Generalization in Discourse Grammars. *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL-94)*, Las Cruces: 17–24.

Kaplan, Ronald. (1987). “Three Seductions of Computational Psycholinguistics.” In *Linguistic Theory and Computer Applications,* edited by Peter Whitelock, Harold Somers, Paul Bennett, Rod Johnson, and Mary McGee Wood. Academic Press, London, 149–88.

Kilgarriff, Adam. (1993). Inheriting Verb Alternations. *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics (EACL-93)*, Utrecht, The Netherlands: 213–221.

Konolige, Kurt. (1988). “Hierarchic Autoepistemic Theories for Nonmonotonic Reasoning: Preliminary Report.” Technical Note No. 446, SRI International, Menlo Park, CA.

Kreiger, Hans-Ulrich and John Nerbonne. (1993). “Feature-based Inheritance Networks for Computational Lexicons.” In *Inheritance, Defaults and the Lexicon,* edited by Edward J. Briscoe, Ann Copestake and Valeria de Paiva. Cambridge University Press, Cambridge, England, 90–136.

Krifka, Manfred. (1987). Nominal Reference and Temporal Constitution: Towards a Semantics of Quantity. *Proceedings of the 6th Amsterdam Colloquium*, University of Amsterdam: 153–173.

Krieger, Hans-Ulrich, and Ulrich Schäfer. (1994). “TDL — A type description language for HPSG.” DFKI, Saarbrücken, Germany.

Lascarides, Alex and Ann Copestake. (in press). The Pragmatics of Word Meaning. *Journal of Linguistics.*

Lascarides, Alex and Nicholas Asher. (1993). Temporal Interpretation, Discourse Relations and Common Sense Entailment. *Linguistics and Philosophy* 16: 437–493.

Lascarides, Alex, Edward J. Briscoe, Nicholas Asher and Ann Copestake. (1996). Order Independent and Persistent Typed Default Unification. *Linguistics and Philosophy* 19: 1–89.

de Paiva, Valeria. (1993). “Types and Constraints in the LKB.” In *Inheritance, Defaults and the Lexicon,* edited by Edward J. Briscoe, Ann Copestake and Valeria de Paiva. Cambridge University Press, Cambridge, England, 164–189.

Pollard, Carl and Ivan Sag. (1994). *Head-driven Phrase Structure Grammar.* The University of Chicago Press, Chicago and CSLI, Stanford.

Quirk, Randolph, Sidney Greenbaum, Geoffrey Leech and Jan Svartvik. (1985). *A Comprehensive Grammar of the English Language.* Longman, London.

Reiter, Raymond. (1980). A Logic for Default Reasoning. *Artificial Intelligence* 13(1&2): 81–132.

Russell, Graham, John Carroll and Susan Warwick-Armstrong. (1991). Multiple Default Inheritance in a Unification-Based Lexicon. *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL-91)*, Berkeley, CA: 215–221.

Russell, Graham, Afzal Ballim, John Carroll and Susan Warwick-Armstrong. (1993). “A Practical Approach to Multiple Default Inheritance for Unification-Based Lexicons.” In *Inheritance, Defaults and the Lexicon,* edited by Edward J. Briscoe, Ann Copestake and Valeria de Paiva. Cambridge University Press, Cambridge, England, 137–147.

Sag, Ivan. (1997). English Relative Clause Constructions. *Journal of Linguistics* 33(2): 431–484.

Sanfilippo, Antonio. (1993). “LKB Encoding of Lexical Knowledge.” In *Inheritance, Defaults and the Lexicon,* edited by Edward J. Briscoe, Ann Copestake and Valeria de Paiva. Cambridge University Press, Cambridge, England, 190–222.

Shieber, Stuart. (1986a). *An Introduction to Unification-Based Approaches to Grammar*. CSLI Lecture Notes 4, CSLI, Stanford CA.

Shieber, Stuart. (1986b). A Simple Reconstruction of GPSG. *Proceedings of the 11th International Conference on Computational Linguistics (COLING-86)*, Bonn, Germany: 211–215.

Smolka, Gert. (1989). "Feature Constraint Logic for Unification Grammars." IWBS Report 93, IWBS-IBM, Stuttgart, Germany.

Uszkoreit, Hans, Rolf Backofen, Stephan Busemann, Abdel Kader Diagne, Elizabeth A. Hinkelman, Walter Kasper, Bernd Kiefer, Hans-Ulrich Krieger, Klaus Netter, Günter Neumann, Stephan Oepen and Stephen P. Spackman. (1994). DISCO — An HPSG-based NLP System and its Application for Appointment Scheduling. *Proceedings of the 15th International Conference on Computational Linguistics (COLING-94)*, Kyoto, Japan.

Vossen, Piek and Ann Copestake. (1993). "Untangling Definition Structure into Knowledge Representation." In *Inheritance, Defaults and the Lexicon,* edited by Edward J. Briscoe, Ann Copestake and Valeria de Paiva. Cambridge University Press, Cambridge, England, 246–274.

Young, Mark and Bill Rounds. (1993). A Logical Semantics for Nonmonotonic Sorts. *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics (ACL-93)*, Columbus, Ohio: 209–215.

A. Appendix

Proof of Lemma 1

First we prove that $Bot^{12} = Bot^{21}$. Note that by order independence of $\sqcap$, $I_{12} = I_{21}$. And by order independence of set union $T^1 \cup T^2 = T^2 \cup T^1$. So

$$\begin{array}{rcl} Bot^{12} & =_{def} & \{\langle F,t\rangle \in T^1 \cup T^2 \text{ such that } I_{12} \sqcap F = \bot\} \\ & = & \{\langle F,t\rangle \in T^2 \cup T^1 \text{ such that } I_{21} \sqcap F = \bot\} \\ & =_{def} & Bot^{21}. \end{array}$$

So:

$$\begin{array}{rcl} T^{12} & =_{def} & (T^1 \cup T^2) \setminus Bot^{12} \\ & = & (T^2 \cup T^1) \setminus Bot^{21} \\ & =_{def} & T^{21} \end{array}$$

□

Proof of Lemma 2

$$\begin{array}{rcl} T^{(12)3} & =_{def} & (T^{12} \cup T^3) \setminus Bot^{(12)3} \\ Bot^{(12)3} & =_{def} & \{\langle F,t\rangle \in T^{12} \cup T^3 \text{ such that } I_{(12)3} \sqcap F = \bot\} \\ & = & \{\langle F,t\rangle \in T^{12} \text{ such that } I_{(12)3} \sqcap F = \bot\} \cup \\ & & \{\langle F,t\rangle \in T^{3} \text{ such that } I_{(12)3} \sqcap F = \bot\} \end{array}$$

Let $\langle F,t\rangle \in T^{(12)3}$. Then:

(a) $\langle F,t\rangle \in T^{12} \setminus Bot^{(12)3}$; or

(b) $\langle F,t\rangle \in T^{3} \setminus Bot^{(12)3}$

Suppose (b). Then $\langle F,t\rangle \in T^3$ and $I_{(12)3} \sqcap F \neq \bot$.

Therefore, $\langle F,t\rangle \in T^2 \cup T^3$. Furthermore, by the definition of typed unification, $I_{(12)3} = I_{1(23)} \sqsubseteq I_{23}$. Therefore, since $I_{1(23)} \sqcap F \neq \bot$, $I_{23} \sqcap F \neq \bot$. So $\langle F,t\rangle \in T^{23}$. Furthermore, $I_{1(23)} \sqcap F = I_{(12)3} \sqcap F \neq \bot$. So $\langle F,t\rangle \notin Bot^{1(23)}$, and therefore $\langle F,t\rangle \in T^{1(23)}$.

Now suppose (a) holds. Then $\langle F,t\rangle \in ((T^1 \cup T^2) \setminus Bot^{12}) \setminus Bot^{(12)3}$. But $Bot^{12} \subseteq Bot^{(12)3}$. So either:

(i) $\langle F,t\rangle \in T^{1} \setminus Bot^{(12)3}$; or

(ii) $\langle F,t\rangle \in T^{2} \setminus Bot^{(12)3}$

Suppose (i) holds. Then $\langle F,t\rangle \in T^1$ and $I_{(12)3} \sqcap F \neq \bot$. So $I_{1(23)} \sqcap F \neq \bot$ by the order independence of typed unification. So $\langle F,t\rangle \in T^1 \setminus Bot^{1(23)} \subseteq T^{1(23)}$.

Suppose (ii) holds. Then $\langle F,t\rangle \in T^2$ and $I_{(12)3} \sqcap F \neq \bot$. So $I_{1(23)} \sqcap F \neq \bot$, and therefore $I_{23} \sqcap F \neq \bot$. So $\langle F,t\rangle \notin Bot^{23}$, and so $\langle F,t\rangle \in T^{23}$. Furthermore, since $I_{1(23)} \sqcap F \neq \bot$, $\langle F,t\rangle \notin Bot^{1(23)}$. And so $\langle F,t\rangle \in T^{1(23)}$.

So $T^{(12)3} \subseteq T^{1(23)}$.

By symmetry, $T^{1(23)} \subseteq T^{(12)3}$.

Therefore $T^{1(23)} = T^{1(23)}$. □

Proof of Lemma 3

The indefeasible TFS of TDFS$_1 \overset{<>}{\sqcap}$ TDFS$_2$ is unique because $\sqcap$ is deterministic. The tail is unique because set union and $\sqcap$ are deterministic. □

Proof of Lemma 4

The specificity partition of a tail is unique because the type hierarchy is a complete partial order. Furthermore, $\sqcap$ and $\sqcup$ are deterministic. Therefore the result of *DefFS*(TDFS) is unique. □

Proof of Lemma 5

Let us consider the $\overset{<>}{\sqcap}$ of the TDFSs F_1 and F_2. The indefeasible part of $F_1 \overset{<>}{\sqcap} F_2$ is the same as the indefeasible part of $F_2 \overset{<>}{\sqcap} F_1$ because $\sqcap$ is commutative. The tails $T^{12} = T^{21}$ by lemma 1. So $F_{12} = F_{21}$. So $\overset{<>}{\sqcap}$ is commutative. □

Proof of Lemma 6

One needs to prove:

1.$I_{(12)3} = I_{1(23)}$

2.$T^{(12)3} = T^{(12)3}$

Case 1 follows immediately from the associativity of $\sqcap$. Case 2 holds by lemma 2. So $\overset{<>}{\sqcap}$ is associative. □

Proof of Theorem 1

Follows immediately from lemmas 3, 5 and 6. □

Proof of Theorem 2

$T^{12} = T^1 \cup T^2$, because $I_{12} \sqcap \wp_1(T^1) \neq \bot$, and $I_{12} \sqcap \wp_1(T^2) \neq \bot$.

Let $\mu_1 \ldots \mu_n$ be a Specificity Partition Tail of T^{12}. Then by the definition of *DefFS*:

$$D_{12} = \sqcup(I_{12} \overset{<}{\sqcap}_{cs} \wp_{fs}(\mu_1) \overset{<}{\sqcap}_{cs} \ldots \overset{<}{\sqcap}_{cs} \wp_{fs}(\mu_n))$$

But by assumption $I_{12} \sqcap \wp_{fs}(\mu_1) \neq \bot$. So by the definition of $\overset{<}{\sqcap}_{cs}$:

$$I_{12} \overset{<}{\sqcap}_{cs} \wp_{fs}(\mu_1) = I_{12} \sqcap \wp_{fs}(\mu_1) \neq \bot$$

Similarly $(I_{12} \sqcap \wp_{fs}(\mu_1)) \sqcap \wp_{fs}(\mu_2) \neq \bot$. So by the definition of $\overset{<}{\sqcap}_{cs}$:

$$((I_{12} \overset{<}{\sqcap}_{cs} \wp_{fs}(\mu_1)) \overset{<}{\sqcap}_{cs} \wp_{fs}(\mu_2) = I_{12} \sqcap \wp_{fs}(\mu_1) \sqcap \wp_{fs}(\mu_2)$$

By similar arguments for $\mu_3, \ldots, \mu_n$:

$$\begin{aligned} D_{12} = & \sqcup((I_{12} \overset{<}{\sqcap}_{cs} \wp_{fs}(\mu_1)) \ldots \overset{<}{\sqcap}_{cs} \wp_{fs}(\mu_n)) \\ = & \sqcup(I_{12} \sqcap \wp_{fs}(\mu_1) \sqcap \ldots \sqcap \wp_{fs}(\mu_n)) \\ = & I_{12} \sqcap \wp_{fs}(T^{12}) \\ = & I_1 \sqcap \wp_{fs}(T^1) \sqcap I_2 \sqcap \wp_{fs}(T^2) \end{aligned}$$

By a similar argument to that above:

$$\begin{aligned} D_1 = & I_1 \sqcap \wp_1(T^1) \\ D_2 = & I_2 \sqcap \wp_1(T^2) \end{aligned}$$

So

$$D_{12} = D_1 \sqcap D_2$$

as required. □

Proof of Lemma 7

For any basic TDFS I/T:

1.T is its own specificity partition;

2.$\sqcap \wp_{fs}(T) \neq \bot$; and

3.$\forall F \in \wp_{fs}(T)$, $I \sqcap F \neq \bot$.

So by the definitions of $\sqcup$ and $\overset{<}{\sqcap}_{cs}$:

$$DefFS(I/T) = \sqcup(I \overset{<}{\sqcap}_{cs} \wp_{fs}(T)) = I \sqcap \wp_{fs}(T)$$

Thus we need to prove that

$$I \sqcap I_D = I \sqcap \wp_{fs}(T)$$

But this follows immediately by the definition of T for $BasicTDFS(I, I_D)$ (T is the set of atomic TFSs that are subsumed by $I \sqcap I_D$ but not subsumed by I). □

Proof of Corollary 1

Similarly to the proof given in the above lemma: $D_i = I_i \sqcap \wp_{fs}(T^i)$ for $i = 1, 2$. So, since $D_1 \sqcap D_2 \neq \perp$:

$$\begin{aligned} I_1 \sqcap \wp_{fs}(T^1) \sqcap I_2 \sqcap \wp_{fs}(T^2) = & \quad I_{12} \sqcap \wp_{fs}(T^1) \sqcap \wp_{fs}(T^2) \\ \neq & \quad \perp \end{aligned}$$

So by theorem 2:

$$\begin{aligned} D_{12} = & \quad D_1 \sqcap D_2 \\ = & \quad I_1 \sqcap I_{D_1} \sqcap I_2 \sqcap I_{D_2} \end{aligned}$$

□