CONSTRAINT RELAXATION WITH WEIGHTED FEATURE STRUCTURES

Frederik Fouvry
Department of Computational Linguistics
Saarland University
Saarbrücken, Germany
`fouvry@coli.uni-sb.de`

Abstract

In this paper, we present a definition of unification of weighted feature structures designed to deal with constraint relaxation. The application of phrase structure rules in a unification-based Natural Language Processing system is adapted such that inconsistent values do not lead to failure, but are penalised. These penalties are based on the signature and the shape of the feature structures, and thus realise an elegant and general approach to relaxation.

1 Introduction

The typed feature logic of Carpenter [2] has been the basis of Natural Language Processing (NLP) systems like the Linguistic Knowledge Base (LKB) [4, 5] and the Attribute Logic Engine (ALE) [3]. The grammar rules in his framework are applied by unification, which fails if values are inconsistent. Although large scale grammars have been successfully developed in such a framework (e.g. the LinGO project [12]), obtaining a sufficient coverage requires a considerable effort, both in grammatical and lexical description. We shall not talk about missing lexical entries in this paper.

There seem to be two approaches for ensuring sufficient coverage. One is extending the grammar through a careful linguistic analysis (as in the LinGO grammar), and the other is to work with an underspecified language description, the results of which may subsequently be filtered (e.g. [9] for Lexical-Functional Grammar (LFG)). In both cases however, inconsistent values do not lead to a result. Whatever the development strategy, it may be beneficial to be able to look beyond consistent values to detect what may be missing from the grammar.

On the other hand, unification failure is the only control strategy that is available in declarative grammar formalisms. If we relax constraints, failure points are postponed, and the search space increases.

Constraint relaxation is not new (e.g. the $\mathcal{TDL}$ formalism [10] has some devices for relaxation). Douglas for instance [6] uses PATR-II to model relaxation of grammar constraints. When for some input no analyses can be constructed, the constraints in the rules (e.g. agreement, subcategorisation) are relaxed one by one, until a solution is found. When the input is ungrammatical, this technique extends the search space in that for every rule it should be checked which constraints have to be ignored. This is not promising for scaling up.

In this paper, we present an approach that crucially relies on typed feature logic for relaxation, and that keeps track of the degree of ungrammaticality by exploiting the typed feature structures. That allows on the one hand to ignore relatively bad solutions, but keeps them as a fall-back option in case the more promising solutions lead to nothing.

The paper is structured as follows. First, we very briefly recapitulate Carpenter's definition of feature structures and their unification, and present the notion of information in feature structures that we shall use. Then we outline the intuitions of the proposal. That is followed by the presentation of weighted feature structures and of their unification. A discussion of this and of some other proposals comes at the end.

2 Outline

For the definition of feature structures we take as basis, we refer to Carpenter's book [2] and to the next section of this paper. Furthermore, let us for the rest — for reasons of the exposition — assume that a bottom-up parser is being used. In Carpenter's formalism then, rules rewrite a number of daughter feature structures as a mother feature structure.

In these typed feature structures there are three elements that define the information: features, types and re-entrancies. Unifications preserve this information. This is however a streamlined view of the content of typed feature structures. We make explicit three more sources of information:

(a) Type values can be provided several times, as in the following example:

(1)

**La*	*présidente*	*était*	*acclamé*
The-fem.sg	president-fem.sg	was	acclaimed-masc.sg

The president was acclaimed

There are two instances of feminine (*la*, *présidente*), only one of masculine (*acclamé*). All three of them are singular.

(b) The occurrence or use of a type also implies that its supertypes occur. For instance, the semantic relation for *dog* implies the semantic relations animate and non-human.

(c) Feature structures are nested into other feature structures, such that the value of a feature structure on a long path implies the occurrence of every containing feature structure. This is analogous to 2, but for feature nesting. In the following AVM for instance, the value of HEAD can only occur (in a linguistic object, which is of the type *sign*) when the values of SYNSEM, LOCAL and CAT are present.

(2) $\begin{bmatrix} \textit{sign} & \\ \text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \mid \text{HEAD} & [\textit{prep}] \end{bmatrix}$

In the rest of this section we explain how they will be used.

In Example (1), it was shown how different values may be provided a different number of times. We assume that this has an effect on the strength or *weight* of the information a value contains: in the example the value feminine is stronger than masculine. Therefore, unifications have to combine the weights of values that are unified, otherwise this new information is not preserved.

The other information sources, viz. type subsumption and feature structure nesting, are already maintained by 'normal' unification, but they also affect the weight of the information,

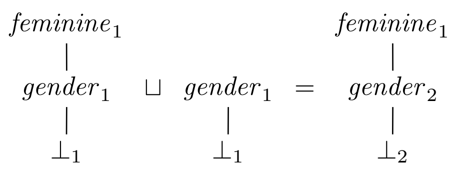

Figure 1: The type unification *feminine* $\sqcup$ *feminine* with weights

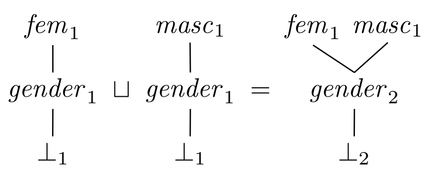

Figure 2: The type unification *fem* $\sqcup$ *masc* with weights

and this should be taken into account as well by the weight bookkeeping. That is demonstrated in Figure 1. The weight on a type is indicated with a subscript, e.g. $feminine_2$ is the type *feminine* with a weight of 2. For simplicity, we just assume that the initial weight of values is 1 (reflecting that the value occurs once), but other values can be used as well. In the unification *feminine* $\sqcup$ *gender*, not only the weights of *feminine* are added up, but also these of $\bot$ and of *gender*.

The rules, which define which linguistic descriptions will be unified, are responsible for adding the weights when type values are unified. One of the potential uses of these weights is that they can be used to choose a value for correcting mistakes (an idea also floated in [1]), e.g. the value with the greatest weight. In Example (1), this would mean that *acclamé* should be made feminine.

There is however a more direct use: they can keep track of inconsistencies. If a value is inconsistent with a grammar rule, unification can ignore the inconsistency and discard it. Because the weights have to be counted separately for each type, they all have to be explicitly present (as depicted in Figure 1; this relies on a type hierarchy which is completed to a distributive lattice as in [7]). Therefore, the value of a feature can be partly removed: all values more specific than the meet of the value in the rule and the value from the input are disposed of. That makes that only the generalisation of both values remains. This leads to a loss of weight, the value of which is the sum of weights on the removed type values. This is where constraint relaxation takes place. Because taking the meet is the relaxation operation, and in Carpenter's logic any two types are required to have precisely one meet (Type is a bounded complete partial order (BCPO)), this is a simple task.

Sometimes the values on a rule are very general, and do not impose any limitation at all, as would be the case for instance with a feature GENDER that had the value *gender*, where *gender* is also the most general satisfying value for the feature value. Then the rule cannot cause any information to disappear: all possible values are compatible with *gender*. The unification is defined (and is not replaced in this case by generalisation). An example is shown in Figure 2. The lower types and weights are the same as in the previous figure, but the most specific types, *fem* and *masc*, both are present, each with their own weight. Had this value been presented to a rule that requires that this value should be *masc*, then fem_1 would have been lost. Then the sum of the information before and after the unification would not have been the same, and this would have indicated an ungrammaticality. As it is, the ungrammaticality is recorded in the presence of two incompatible values, *fem* and *masc*.

We shall return to this difference in status of values in Section 4.1.

From the previous, it seems that robust unification of any two types is defined at all times, which clearly requires huge data structures. It is possible to split up the type hierarchy into different classes again which are disjoint from each other (not counting their supertypes) without harming the robust behaviour. This can be derived from the signature and the grammar. This division greatly reduces the data structures, since they do not need to reflect the entire signature anymore, but only a class.

In the next sections, we define weighted feature structures and recovering unification, which unifies them.

3 Weighted feature structures

The figures above have shown that the single type values are replaced by more complex values, both for weights and for the unification. That is also what is going to happen for feature structures.

In the following definitions, Type is a set of types ordered on $\sqsubseteq$, and Feat is a set of feature labels.

For reference, we reproduce here the feature structure definition from [2].

Definition 1 (Feature structure) *A* typed feature structure *F is a tuple $\langle Q, \overline{q}, \theta, \delta \rangle$ where:*

- *Q is a set of nodes, containing $\overline{q}$, the root node;*
- *$\theta : Q \to \mathsf{Type}$ is a total node typing function;*
- *$\delta : \mathsf{Path} \times Q \to Q$ is a partial feature value function.*

□

A feature structure is a set of nodes which are linked by a function δ, and which are assigned a type value through the function θ. The links in δ are labelled by elements of Feat.

In the following definition of weighted feature structures, δ is redefined and θ, the value assignment function, has been removed.

Definition 2 (Weighted feature structure) *A* weighted feature structure *F is a tuple $\langle Q, \overline{q}, \delta, w, \tau, S \rangle$ where:*

- *Q is a set of nodes, containing $\overline{q}$, the root node;*
- *S is a set of type value nodes;*
- *$w : Q \to (\langle \mathsf{Type}, \sqsubseteq \rangle \to \langle \mathbb{N}_0, \leq \rangle)$ is a total weight assignment function;*
- *$\tau : Q \to (\langle \mathsf{Type}, \sqsubseteq \rangle \to \langle S, \sqsubseteq \rangle)$ is a total type value assignment function;*
- *$\delta : \mathsf{Feat} \times S \to Q$ is a partial feature value function.*

□

The set of weighted feature structures is $\mathcal{F}_w$.

Like in Carpenter's book [2] feature structures consist of nodes that are linked by functions, and to which certain values are attached. θ's function is now taken over by τ (and w). τ and the

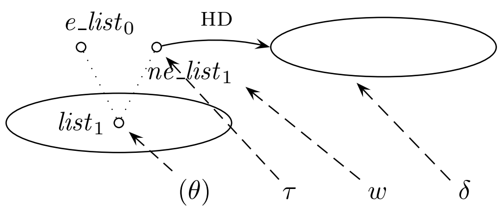

Figure 3: The anatomy of a weighted feature structure, and the results of the feature structure functions. The ellipses are elements of Q, the small circles are elements of S.

new δ together perform the task of δ from Definition 1: given a feature structure node q, they describe which feature structure nodes can be reached from q. For instance $\delta(f, \tau(q_1)(\sigma)) = q_2$ expresses that a feature f leads from q_1 to q_2 over the type σ. The difference is that there is one intermediate step over a type. A picture for the feature structure in (3) is shown in Figure 3.

(3) $\begin{bmatrix} \textit{ne\_list} \\ \text{HD} \;\; [\;\;] \end{bmatrix}$

The function w links feature structure nodes via types to a weight. Given a node, w returns a function that takes a type and then gives the weight for the type on that node. We have this defined here to be a natural number or 0, but real numbers for instance are possible as well. What is important for this definition is that the result of $w(q)$ for a node $q \in Q$ is an order-preserving function, i.e. $s \sqsubseteq t \Rightarrow w(q)(s) \geq w(q)(t)$. The constraint on the weight is relative to the other types in the hierarchy: a more general type should have less weight. When that condition is satisfied, the weight that is lost will be greater to the extent that the meet (the relaxation) is more general.

τ links a feature structure node with a type value node, via a type. This function runs in fact parallel to w: whereas w ultimately returned a weight, τ returns a type value node (elements of S). The task of this function is to make sure that the appropriateness values are maintained and to make loss of information possible. The appropriateness requirements in Chapter 6 of [2] define what feature a type can have, and what values these features can carry. This function separates out the nodes for the different type values, and thus prevents features appearing on types that should not carry them.

That makes it possible to realise the loss of information as a join operation (although loss of information represents a generalisation) by ignoring the type nodes that are inconsistent. A direct consequence is that all values and weights deeper in the feature structure are discarded as well.

The function δ, which in [2] links feature structure nodes with other feature structure nodes over a feature label, here links type value nodes with feature structure nodes.

S consists of a number of equivalence classes, whereby two nodes are equivalent iff they can be reached from the same feature structure node. It can be considered as a collection of copies of various subtrees from the signature Type.

4 Unification

Before we can define the unification of weighted feature structures, a distinction in the status of values in a grammar needs to be discussed. We already mentioned these differences in Section 2.

4.1 Two kinds of values

In the feature structures of phrase structure rules in a unification-based formalism, we need to distinguish two kinds of values. Let us demonstrate this with an example.

(4) $\begin{bmatrix}\textsc{head} & \boxed{1}\,[\textit{verb}] \\ \textsc{index} & \boxed{2}\end{bmatrix} \longrightarrow \begin{bmatrix}\textsc{head} & \boxed{1} \\ \textsc{index} & \boxed{2}\end{bmatrix} \begin{bmatrix}\textsc{head} & [\textit{noun}]\end{bmatrix}$

This is a version of a VP → V NP rule. In this rule, the HEAD value is shared between the left hand side and the first daughter, but limited to *verb*. The INDEX value is shared as well, but does not carry any further constraints. A last value is the HEAD value of the second daughter, which is "hard-wired" into the rule to be *noun*.

With the assumed bottom-up application of the rules, we unify the first daughter with (5), which succeeds: the HEAD and INDEX values are consistent.

(5) $\begin{bmatrix}\textsc{head} & [\textit{verb}] \\ \textsc{index} & [\textit{3sing}]\end{bmatrix}$

(6) $\begin{bmatrix}\textsc{head} & [\textit{noun}] \\ \textsc{index} & [\textit{3sing}]\end{bmatrix}$

The INDEX value can in fact never cause an inconsistency, because it is maximally underspecified in the rule. The HEAD value is different: the re-entrant *verb* imposes a restriction on the first daughter's HEAD value. As such it can lead to the removal of inconsistent information, for instance, when the feature structure in (6) would be the leftmost daughter. The intention is that in that case the weight of all nodes that are more specific than $\textit{noun} \sqcap \textit{verb}$ and that are incompatible with *verb*, are removed. The relation between the two values is not commutative (as it is in normal unification). The single value *noun* in the second daughter performs the same function, but does not pass on its information to the mother feature structure. If there is an inconsistency, only the loss of information is preserved, not the value (since it is not passed on). It is very important that information loss is carefully and completely recorded: consistency is maintained throughout the analysis tree; inconsistent values are removed (they only stay in the tree as long as they are consistent with the requirements of the rules).

The value that is given by the rule, and therefore "expected" is called the *reference value.* The value that comes from the input is called the *ground value.*

4.2 Definition

Definition 3 (Recovering unification) *Let* $F, F' \in \mathcal{F}_w$ *and* $F = \langle Q, \overline{q}, \delta, w, \tau, S\rangle$, $F' = \langle Q', \overline{q}', \delta', w', \tau', S'\rangle$. *It is required that* $Q \cap Q' = \varnothing$ *and* $S \cap S' = \varnothing$. *A least equivalence relation is defined on* $Q \cup Q'$ *and* $S \cup S'$ *such that*

- $\overline{q} \bowtie \overline{q}'$;
- $\tau(q)(t) \bowtie \tau'(q')(t')$ *if* $q \bowtie q'$ *and* $t = t'$;
- $\delta(f, s) \bowtie \delta(f, s')$ *if* $s \bowtie s'$ *and both are defined.*

Then $F \sqcup F' = \langle (Q \cup Q')/_{\bowtie}, [\overline{q}]_{\bowtie}, \delta^{\bowtie}, w^{\bowtie}, \tau^{\bowtie}, (S \cup S')/_{\bowtie}\rangle$ *with*

$$\tau^{\bowtie}([q]_{\bowtie})(t) = \{(\tau \cup \tau')(q'')(t) \mid q \bowtie q''\}$$

$$\delta^{\bowtie}(f, [s]_{\bowtie}) = \begin{cases} [(\delta \cup \delta')(f,s)]_{\bowtie} & \textit{if } (\delta \cup \delta')(f,s) \textit{ is defined} \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

and for a feature structure node $q \in Q$*: if*

(1) for all $q \bowtie q' \in Q'$*:* $w'(q')(t) \geq 1$*, and*

(2) with $q_p \in Q$*, there is a* $\delta^{\bowtie}(f, \tau^{\bowtie}([q_p]_{\bowtie})(t)) = [q]_{\bowtie}$ *such that* $w^{\bowtie}([q_p]_{\bowtie})(t) \geq 1$*,*

then

$$w^{\bowtie}([q]_{\bowtie})(t) = \textstyle\sum_{q \bowtie q'' \in Q} w(q'')(t).$$

$w^{\bowtie}([q]_{\bowtie})(t) = 0$ *otherwise.*

Condition (1) $w'(q'')(t) \geq 1$ *designates* F' *as the reference value.* □

Two feature structures are unified by walking in parallel through the graphs starting from the feature structure root node. The graphs should be disjoint. Every time that feature structure node is chosen, the next node has to be a type value node (element of S) that is reached by using the same type for both feature structures. From the type value nodes, the graph is continued over a feature link to a feature structure node. The feature has to have the same name for both feature structures. The shape of the graph over links to type value nodes and over feature links is preserved in the result of the unification.

The actual work is done by $w^{\bowtie}$. The weight on a type value node in the result for a given type t ($w^{\bowtie}([q]_{\bowtie})(t)$) is set to 0 if it is 0 in the reference value (which means that the value did not occur, and hence was inconsistent), or if every path to the type value node contains a 0-weight on one of its type value nodes. Setting the weight of a type value node to 0 is the same as removing it. The last condition thus ensures that not just the root value of a feature structure is lost, but that the entire feature structure goes with it.

It remains to be shown that the unification of two feature structure is itself a feature structure. We follow the lines of the proof Carpenter [2, 47].

Proof Since Q and Q' are finite, $Q \cup Q'$ is. The set of type value nodes S is finite because Q and the number of types are finite. $S \cup S'$ is finite as well for the same reasons as $Q \cup Q'$ (*mutatis mutandis*). $S \cup S'$ is a quotient set: the classes are $\{\tau^{\bowtie}([q]_{\bowtie})(t) \mid t \in \theta^{\bowtie}([q]_{\bowtie})\}$ for every $[q]_{\bowtie} \in Q \cup Q'$ since the type value nodes are still related to a single feature structure node. $\tau^{\bowtie}$ is total (all nodes have type values) because $S \cup S'$ is always defined. $w^{\bowtie}$ is total as well: all nodes have a weight, because the weights of the unificands were taken over (either from one of the feature structures, or as a sum of both; see later). These operations also keep $w^{\bowtie}([q]_{\bowtie})$ order preserving: if $t \sqsubseteq t'$, then $w(q)(t) + w'(q')(t) \geq w(q)(t') + w'(q')(t')$. For $\delta^{\bowtie}$, the paths do not depend on the type value nodes that have been chosen, because if $t_1, t_2 \in \mathsf{Type}, q \in Q$ and $\text{Intro}(f) \sqsubseteq t_1$, $t_1 \sqsubseteq t_2$ implies $\delta^{\bowtie}(f, \tau^{\bowtie}([q]_{\bowtie})(t_1)) = \delta^{\bowtie}(f, \tau^{\bowtie}([q]_{\bowtie})(t_2))$ and type values nodes are equivalent if they are reached by the same type on equivalent feature structure nodes, nor do they depend on the chosen feature structure nodes since the latter are $\bowtie$-equivalent if they are on the same path. □

4.3 Remarks

We already have discussed rule applications to some extent, but there are a few more issues to be mentioned.

The first concerns the lexical entries. In this setup, lexical entries contain the original weights, with which parsing begins. Those are the values that feed into the rules, and may be lost. This is also the reason that we only use information loss: the size of the feature structures for different lexical entries can vary considerably, but that does not have any significance for the "grammaticality" of the lexical entries. On the contrary, they should all be equally good. When only the subtracted information is taken into account, that reflects much better what is happening than the information sum would have done. Bigger feature structures can of course lose more information, and can therefore become more ungrammatical than smaller ones, but that is in the spirit of the proposal, which is centered around the information feature structures contain.

In the rule in (4) there is one type of value missing: a "hard-wired" value in the left hand side. Clearly such a value cannot get anything from the daughters when it is not re-entrant with any value on the right hand side. It nevertheless contributes information: for instance it may trigger a unification failure, and thus prohibit the application of a rule. To keep the monotonicity of information loss (see later) and in order to be able to compare the obtained information weight with that of the lexical entries in the input, these values are "initialised" with a non-zero weight that satisfies all constraints, once the rule *has been* applied. This weight is not added to the original total weight of the input, but it can be lost nevertheless. Information loss is what counts.

A last item pertains to the weights on reference values. Because they should filter out all incompatible information, all compatible type values should get a weight greater than 0 in the feature structure with the reference values. Then condition (1) in Definition 3 will have the desired effect. This is especially relevant for joins of two incomparable types. Suppose $a \sqcup b = c$ is defined, and $a \neq c \neq b$. Then the unification of a_1 with b_1 would keep the weight 1 for a and b, but for c it would be $0 + 0$. c is clearly the desired result, and therefore, its weight should be set to a value greater than 0.

5 Discussion

5.1 Weights

The weights should not be confused with frequencies or probabilities. Their purpose is to measure how much of a feature structure is lost in a rule application. As such the weight of any feature structure indicates how much useful information it contains (which is: how information much can be lost).

We have pointed out that although in this paper the weights are defined as non-negative integers, there is no limitation to extend this to real numbers. The proposal only specifies how the numbers are to be used, and not what they should represent. This makes it possible to use any kind of value, where one may think of automatically collected weights, or still different values. For the paper, we used a weight of 1 for every type and every node, which reflects that the value is used once every time it is mentioned or used in a feature structure. This is only a choice of a working value based on what can be observed in the (mathematical) structures that can be found in a grammar, and can be done by a compiler. (It is not our intention in this paper to suggest any techniques or algorithms to obtain other values. We expect it to be a topic of further research.) It is for instance conceivable that the gender information on a noun should be weighted heavier than that of a determiner or an adjective.

The effect of the weights should nevertheless be clear: solutions can be ranked according to how much of their original information they lost. How much this amounts to depends on the location and the value of the generalisation. An example is where a head has an empty subcategorisation list, but is used in a head-complement rule that requires the list not to be empty. Let us assume that the HPSG Subcategorisation Principle applies, i.e. there is a re-entrancy between the subcategorisation of the mother and of the head-daughter, and between an element of that list of the head-daughter and the complement-daughter itself. The (weight of the) complement-daughter will be lost entirely since there is no re-entrancy to pass it on, and that is because the re-entrancy is only active on subcategorisation lists that are *not* empty. It generalises for other values. This loss is dramatic, and the result is therefore unlikely to be useful later on.

Initial tests on toy grammars show that the information loss in feature structures can spiral down into a very considerable loss even after just a few "bad" rule applications. More rule applications compound this effect. This makes that the information loss may well be able to effectively fulfil its function as a sign post in the search space.

5.2 Rule definition

In this framework, the way in which the rules are defined is important. The choice is between specifying many different specific values, or one (or few) general value(s). The choice of the former will lead to much loss of information and an increased ungrammaticality in the results. A general value needs less rules, but more variation in the values will be tolerated. We speculate that the best choice has to be determined empirically for a grammar and perhaps the text type. Since this choice does not affect the declarative meaning of the rule, it will not behave differently as far as the linguistic descriptions are concerned, but it will have a different effect on the degree of ungrammaticality of the result.

5.3 Evaluation

There is no concrete evaluation yet of the technique that is presented in this paper. A possible evaluation scenario is that a text containing errors is taken. The text is corrected, and parsed with the non-robust grammar interpretation. Next, both versions of the text are parsed robustly. The results of the robust parser should at least be as good (i.e. in coverage) as the non-robust one, and the text with errors should receive at least the same analyses as the corrected version (both should be guaranteed by the logical properties of the formalism). The analyses that are returned for ungrammatical sentences should be inspected (to what extent are they correct?), as well as the ranking of the candidate analyses.

There are two places where the ranking can be tuned: by modifying the weights (using other values than 1), or by changing the grammar (the rules (see Section 5.2) or the signature).

It is possible that the restriction on the ordering of the weights (order-preserving) is not acceptable. Since that constraint is based on fundamental properties of feature structures, it would seem that in that case other representations should take the place of feature structures.

5.4 Monotonicity

One of the most important arguments for declarative approaches to NLP is that it is monotonic. It is clear however that here — in the case of an inconsistency — information is removed. Can we still rely on that property? It is clear that monotonicity is maintained in the case of consistent values. The treatment of inconsistent value falls into two categories. If the reference value (grammar rule) is not part of the inconsistency, the values are kept, which is monotonic. If the grammar rule causes a value to be stripped off from a feature, and thus generalises its value, there is indeed the chance that values will be filled which are not compatible with the original value, and thus non-monotonicity appears. Thanks to the loss of weight, the inconsistency is clearly marked as such, which makes it possible to treat these results differently. Moreover, the loss of information is monotonically increasing. Information can be added by a rule application, but it is not registered, except when it is lost.

We have not investigated whether and how this setup interacts with default unification (e.g. the persistent associative default unification in [11]). It is interesting to note that Schöter [13] for instance uses signature assumptions that show similarities to ours to model defaults.

5.5 Other devices

Other formal devices like definite clauses or lexical rules have not been considered. Lexical rules are usually not very different from unary phrase structure rules, and apart from mentioning the potential problem of circular applications, we do not discuss them further. Definite clauses, as they exist for instance in ALE [3], are more different, and would need closer study. We want to offer two remarks on this topic. They may not be necessary for processing of large scale grammars, e.g. the LinGO English Resource Grammar [12] is written in a formalism without them (at least as far as parsing is concerned). The second is that they may be treated in a very similar way as phrase structure rules. After all, they have a single head and a body with several clauses. Bottom-up application is not an option, but if the weights are carefully traced, processing can stop when the loss of information stabilises.

6 Other approaches

Earlier descriptions of constraint relaxation have been named already. Most of them do not use types. Douglas [6] for instance works in the untyped unification formalism PATR-II. The rules with a context-free backbone are annotated with the constraints that have to be satisfied. The extension of Douglas consists of associating the constraints with relaxation control information, which the grammar writer has to encode explicitly.

A more general proposal comes from Krieger and Schäfer [10], but they do not use weights. They allow the grammar writer to define open-world types. Two of these types always unify, unless they were explicitly defined to be incompatible. That gives the same behaviour as the combination of type values as in Figure 2. Its main purpose is for aiding grammar development.

(Currently similar techniques are proposed for the treatment of coordination in unification-based frameworks. Since those approaches still need to distinguish between grammatical and ungrammatical values, it remains to be seen how similar they really are to what we have proposed here.)

Another example is Vogel and Cooper [15] who, although they work in a typed framework, do not exploit that advantage. The treatment of clashes is limited to atomic types. In that respect it is comparable to the first one. Kim [8] also describes a version of graded unification, but he reduces the unification to unification of atomic values, and focusses on the parsing strategies.

7 Summary and conclusion

We have presented a form of typed unification that can deal with inconsistent values. If the unified values are inconsistent, then they are so either because at least one is not consistent with the value that was expected by the rule, or because they are not consistent with each other. Because of the assumption that the grammar is correct, the values of the input are removed to the extent that they are inconsistent. This loss of information is penalised. That is obtained by attaching weights to the feature structures, and this property sets off this proposal from others. Because it relies on the underlying logic, it is a very general and elegant way of realising typed constraint relaxation.

Acknowledgements

This research was funded by a University of Essex studentship and by the Collaborative Research Centre SFB 378 “Resource-adaptive Cognitive Processes” (funded by the Deutsche Forschungsgemeinschaft). We thank Doug Arnold for discussions, and three anonymous reviewers for their helpful comments.

References

[1] Rens Bod and Ronald [M.] Kaplan. A probabilistic corpus-driven model for lexical-functional analysis. In *Proceedings of the 36th Conference of the ACL and the 17th International Conference on Computational Linguistics*, volume I, pages 145–151, Université de Montréal, Montreal, Quebec, Canada, August 10–14 1998. International Committee on Computational Linguistics.

[2] Bob Carpenter. *The logic of typed feature structures: With applications to unification grammars, logic programs and constraint resolution.* Number 32 in Cambridge Tracts in Computer Science. Cambridge–New York–Melbourne, 1992.

[3] Bob Carpenter and Gerald Penn. *ALE. The Attribute Logic Engine: User's Guide*, December 2001. Version 3.2.1. `http://www.cs.toronto.edu/~gpenn/ale/files/aleguideA4.ps.gz`.

[4] Ann Copestake. *Implementing Typed Feature Structure Grammars.* Stanford, CA, 2002.

[5] Ann Copestake and Dan Flickinger. An open-source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of LREC-2000*, Athens, Greece, 31 May–2 June 2000. European Language Resource Association.

[6] Shona Douglas. Robust PATR for error detection and correction. In Schöter and Vogel [14], chapter 4, pages 139–156.

[7] Frederik Fouvry. Robust unification for linguistics. In *Proceedings of ROMAND 2000*, pages [77–88], Lausanne, 19–20 October 2000.

[8] Albert Kim. Graded unification: A framework for interactive processing. In *Proceedings of the 32nd Annual Meeting of the ACL*, pages 313–315, Las Cruces, New Mexico, USA, 27–30 June 1994. Association for Computational Linguistics.

[9] Tracy Holloway King, Stefanie Dipper, Annette Frank, Jonas Kuhn, and John [T.] Maxwell, [III]. Ambiguity management in grammar writing. In Erhard Hinrichs, Detmar Meurers, and Shuly Wintner, editors, *Proceedings of the Workshop on Linguistic Theory and Grammar Implementation*, Birmingham, Great Britain, 14–18 August 2000. ESSLLI '00.

[10] Hans-Ulrich Krieger and Ulrich Schäfer. TDL: A type description language for HPSG. Part 1: Overview. Research report RR-94-37, Deutsches Forschungszentrum für künstliche Intelligenz (DFKI), Saarbrücken, Germany, November 1994.

[11] Alex Lascarides, Ted Briscoe, Nicholas Asher, and Ann Copestake. Order independent and persistent typed default unification. *Linguistics and Philosophy*, 19(1):1–90, February 1996. Revised version of ACQUILEX II WP 34 (August 1994/March 1995). Also chapter 3 in [14].

[12] LinGO. English Resource Grammar. Available on-line. `http://lingo.stanford.edu/ftp/erg.tgz` (26 July 2002), 2001. The version of 20 June 2002 was used.

[13] Andreas Schöter. Paraconsistent feature logic. In Schöter and Vogel [14], chapter 1, pages 3–38.

[14] Andreas Schöter and Carl Vogel, editors. *Nonclassical feature systems*, volume 10 of *Edinburgh Working Papers in Cognitive Science*. Centre for Cognitive Science, University of Edinburgh, 1995.

[15] Carl Vogel and Robin Cooper. Robust chart parsing with mildly inconsistent feature structures. In Schöter and Vogel [14], chapter 6, pages 197–216.