A Normal Form For Typed Feature Structures

Thilo W. Götz

April 9, 1994

Contents

0 Introduction **2**

1 Types and appropriateness **3**

2 Feature structures **8**

2.1 Syntax 8

2.2 Semantics 8

2.3 Models, Satisfiability And Well-Typing 9

2.4 Extended Feature Structures 15

3 Redundant arcs **17**

4 Normal Form Feature Structures **23**

4.1 Denotation 26

4.2 Redundant arcs 28

4.3 An example 32

4.4 Subsumption and unification 35

5 Conclusion **38**

5.1 Future research 38

0 Introduction

Since the advent of HPSG, typed feature structures have played an increasingly important role, both in linguistics proper as well as in computational linguistics. The approaches towards formalising HPSG and typed feature structures have been varied, in particular with respect to a formal semantics for these structures. One way has been to give an algebraic fixed point characterisation of models for grammars expressed with typed feature structures (cf. in particular Pollard, in press).

The other approach (exemplified by (Smolka, 1988), (King, 1989) and (Dörre and Eisele, 1991)), has been to define a formal language built out of so-called feature terms and standard logical connectives, and provide this formal language with a model-theoretic semantics that allows typed feature structures as models. Computationally, this approach is best seen in the logical programming language CUF (Dörre and Dorna, 1993), which provides a constraint language over feature terms and a full set of logical connectives (including negation). To check formulae in that language for satisfiability, they have to be converted into some sort of normal form which, not surprisingly, bears a strong similarity to typed feature structures.

This paper examines the logical notions underlying the NLP system TROLL currently under developement at the university of Tübingen, but abstracting away from computational issues insofar as no algorithms are given.[1] Still, some aspects of the later sections are fairly procedural.

TROLL provides a model-theoretic semantics, but it's formal language is not a standard term language with connectives, but simply typed feature structures. This is not a major difference, since one way to view feature structures is as normal forms for purely conjunctive formulae in a term language. For an account of how to convert feature structures into term language formulae and back, see (Carpenter, 1992).

But since feature structures are our formal language, we need to give them a formal semantics. This is actually quite useful, since we can directly talk about satisfiability of feature structures and the meaning of various operations we want to define on feature structures (e.g. subsumption and unification). The main concern of this paper is to investigate a normal form for feature structures that not only exhibits satisfiability, but is also in some sense minimal. We will also look at some properties of this normal form, in particular with respect to subsumption and unification.

TROLL, and consequently this paper, is built on Paul King's work on the satisfiability of type-resolved feature structures (King, 1993), and on Dale Gerdemann's work on removing redundant arcs from feature structures. This will be apparent throughout the following sections.

The paper is organised as follows:

- The first section examines the notions of type hierarchies and appropriateness specifications, which are viewed as specifying a logical signature, and interpretations for these signatures.
- In the second section, we define the syntax of feature structures and their denotation relative to an interpretation. We also look at a special interpretation defined by Paul King and describe some of his work on the satisfiability of feature structures. We then go on to define a certain sub-class of the satisfiable feature structures.
- In the third section we consider how an arc can be uninformative, i.e. have no effect on the denotation of a feature structure. We then examine how these arcs can be detected syntactically.
- In the fourht section, we define a normal form for typed feature structures and define some functions which map an arbitrary feature structure into its normal form. The structures used in the process are designed to make the inference functions transparent syntactic mappings that rather closely reflect the procedures actually employed in TROLL. We will then show that the standard definition of subsumption is not complete for normal form structures under its intended meaning, yet that it is possible to define a complete subsumption algorithm a lot easier then would be the case for feature structures not in our normal form. Finally, we'll have a quick look at unification.

Most of the results reported here come with proofs of varying degrees of formal rigour and readability. It is certainly possible (and hopefully useful) to read the paper without actually looking at the proofs.

[1]For more procedural aspects, see a forthcoming TROLL manual.

1 Types and appropriateness

In typed feature structures, all nodes are labelled by types. This is different from other feature structures as, e.g., in PATR-II (Shieber, 1986), which had atomic feature structures but no types in the sense we use them here. These types are arranged into a type hierarchy, a partial order among types. Type hierarchies are also sometimes called subsumption hierarchies or, if they carry additional information, inheritance hierarchies. We will now define type hierarchies and then discuss what they mean. Unfortunately, the definitions for type hierarchies given in the literature are not consistent, most notably *which way round* they are supposed to be interpreted, and nomenclature is very confusing. In TROLL, $\bot$ is the most general type, providing no information. We will try to avoid using *supertype* and *subtype*, since these concepts cause most of the confusion when type hierarchies are interpreted "bottom-up".

The initial definitions in this section are taken over or derived from Paul King's work.

1.1 **Definition (type hierarchy)**

A type hierarchy is a pair $\langle \mathsf{Type}, \sqsubseteq \rangle$ *such that:*

$\langle \mathsf{Type}, \sqsubseteq \rangle$ *is a finite meet semi-lattice.*

If $t, t' \in \mathsf{Type}$ *and* $t \sqsubseteq t'$, *then we say that* t *subsumes* t'.

We sometimes need to talk about the anti-reflexive part of $\sqsubseteq$.

1.2 **Definition (proper subsumption, $\sqsubset$)**

If $t, t' \in \mathsf{Type}$ *then* $t \sqsubset t'$ *iff* $t \sqsubseteq t'$ *and* $t \neq t'$

By virtue of the interpretation we want to give type hierarchies, types that subsume no type except themselves have a special status, and we give them a name so we can refer to them more easily.

1.3 **Definition (maximal types or varieties)**

If $\langle \mathsf{Type}, \sqsubseteq \rangle$ *is a type hierarchy and* $t \in \mathsf{Type}$, *then* t *is called* a maximal type *or* variety *iff there is no* $t' \in \mathsf{Type}$ *such that* $t \sqsubset t'$.

Additionally to ordering the types into a type hierarchy, we specify which features specific types deserve, and which types are appropriate on those features.

1.4 **Definition (appropriateness specification)**

If $\langle \mathsf{Type}, \sqsubseteq \rangle$ *is a type hierarchy, then an appropriateness specification over* $\langle \mathsf{Type}, \sqsubseteq \rangle$ *and a finite set* Feat *of feature names is a partial function* $Approp : \mathsf{Type} \times \mathsf{Feat} \rightharpoonup \mathsf{Type}$ *from type-feature pairs to types such that:*

if Approp(t,f) is defined and $t \sqsubseteq t'$
then Approp(t',f) is defined and $Approp(t, f) \sqsubseteq Approp(t', f)$

The intuitive meaning of a type hierarchy is to structure a domain, i.e. every type denotes a set of objects, and every object is of a certain type. This means that if type a subsumes type b, then the denotation of a contains the denotation of b. There are (at least) two different ways in which this intuitive meaning can be formalised, namely in a *closed world* or an *open world* interpretation. A closed world interpretation assumes that every object in the domain is of a type specified in the type hierarchy, and if an object is of a certain type and that type is not maximal, then it is also of some more specific type. In an open world interpretation, the second condition doesn't necessarily hold. We adopt a closed world interpretation, without further arguing this point here (but note that HPSG in its current incarnation seems to adopt the same interpretation; see also (Gerdemann and King, to appear) for how to encode complex feature cooccurence restriction in a type hierarchy under a closed world interpretation).

Features are interpreted as partial functions from objects to objects, and these functions have to obey the appropriateness specification.

1.5 **Definition (interpretation)**

An interpretation over a type hierarchy $\langle \mathsf{Type}, \sqsubseteq \rangle$, *a set of features* Feat *and appropriateness specification Approp is a triple* $\langle \mathcal{U}, \mathcal{V}, \mathcal{A} \rangle$ *s.t.:*

1. $\mathcal{U}$ *is a set of objects (the domain)*

2. $\mathcal{V} : \mathcal{U} \rightarrow \mathsf{Variety}$ *is a total function from objects to varieties.*

3. $\mathcal{A}$ *is an attribute interpretation function that maps features into functions s.t. for every* $f \in \mathsf{Feat}$, $\mathcal{A}(f) : \mathcal{U} \rightharpoonup \mathcal{U}$ *is a partial function from objects to objects such that:*

   (a) *if* $Approp(\mathcal{V}(\mathrm{u}), f)$ *is defined, then* $\mathcal{A}(f)$ *is defined on* u

   (b) *if* $\mathcal{A}(f)$ *is defined on* u *and* $\mathcal{V}(\mathrm{u}) = t$, *then* $Approp(t, f)$ *is defined and* $Approp(t, f) \sqsubseteq \mathcal{V}(\mathcal{A}(f)(\mathrm{u}))$

The closed world assumption also holds for features; all features must be declared in the appropriateness specification, and an object of a certain type must have all and only those features that the type deserves.

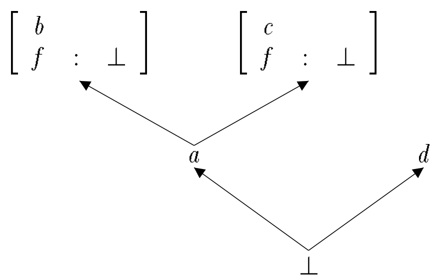
As another consequence of the closed world assumption, the definition of an interpretation only makes reference to maximal types. We can derive the denotation of other types as follows.

1.6 **Definition**

If $\mathcal{I}$ *is an interpretation, then for each* $t \in \mathsf{Type}$, *the denotation of a type is*

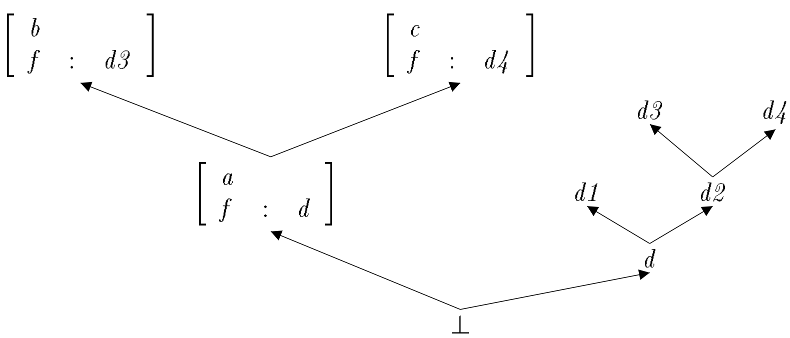
$$t^{\mathcal{I}} = \{\mathrm{u} \in \mathcal{U} \mid t \sqsubseteq \mathcal{V}(\mathrm{u})\}$$

We want to be able to depict type hierarchies and appropriateness conditions graphically. Suppose we have a type hierarchy with the set of types $\{\bot, a, b, c, d\}$, the (anti-reflexive part of the) subsumption relation $\{\langle\bot, a\rangle, \langle\bot, b\rangle, \langle\bot, c\rangle, \langle\bot, d\rangle, \langle a, b\rangle, \langle a, c\rangle\}$ and appropriateness specification $\{\langle b, f, \bot\rangle, \langle a, f, \bot\rangle\}$. We will picture this as follows:



From definition 1.5 one can see that the interpretation of features is solely dependent on the appropriateness specifications of the maximal types. Indeed, the appropriateness conditions for non-maximal types can be less informative then one might want them to be. To see this, consider the following examples. In the type hierarchy above, every object in the denotation of a must be either in the denotation of b or c. But both b and c require a feature f. Therefore we know that every object in the denotation of a must also have a feature f defined.

In the type hierarchy below, we have a similar situation as above. But this time, a is also defined for f. Yet by an argument parallel to the one above, it is clear that one could supply more information about $Approp(a, f)$, as the meet of *d3* and *d4* (the values of $Approp(b, f)$ and $Approp(c, f)$, respectively) is *d2*, which is more specific than d. But as every object in the denotation of a is either in the denotation of b or c, every object in the denotation of a will be mapped by $\mathcal{A}(f)$ into an object in the denotation of *d2*.



From these examples one can see that appropriateness specifications are not as specific as they could be for the semantic approach we have taken here. We will therefore define a function *ExtApp* that computes the most specific information that can be derived from the appropriateness specification.

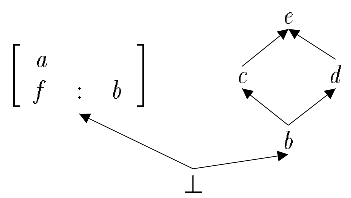
But this is not the only way in which a type hierarchy can provide us with more specific information than at first meets the eye. We can use the concept of *unary branches* for a more restrictive definition of *ExtApp*.

1.7 **Definition (unary branches)**

We say that there is a unary branch from t to t' iff

$t \sqsubset t'$ *and for all* $v \in \mathsf{Variety}$, $t \sqsubseteq v \iff t' \sqsubseteq v$

We can see from the definition that unary branches contribute nothing of semantic interest, since the interpretation of a type is the union of the interpretations of its maximal subtypes. To be able to use a representative for all types that are in the unary branch relation to each other, we define a function *Bump* that picks out the most specific type that necessarily has the same interpretation as a given type. Consider the following example.



Every object that's a b must either be a c or a d, but every c and every d is also an e. Consequently, any object is a b iff it's an e.

1.8 **Definition (*Bump*)**

Bump : $\mathsf{Type} \rightarrow \mathsf{Type}$ *is a total function from types to types such that*

$Bump(t) = \bigsqcap \{v \in \mathsf{Variety} \mid t \sqsubseteq v\}$

1.9 **Corollary**

For every $t \in \mathsf{Type}$, $t^{\mathcal{I}} = Bump(t)^{\mathcal{I}}$ *for all interpretations* $\mathcal{I}$

1.10 **Lemma**

For all $s, t \in \mathsf{Type}$

$Bump(s) \sqsubset Bump(t)$ *iff*
for all interpretations $\mathcal{I}$ *s.t.* $v^{\mathcal{I}} \neq \emptyset$ *for all* $v \in \mathsf{Variety}$, $t^{\mathcal{I}} \subset s^{\mathcal{I}}$

Proof

Let $\mathcal{I}$ be any interpretation s.t. $v^{\mathcal{I}} \neq \emptyset$ for all $v \in \mathsf{Variety}$

$Bump(s) \sqsubset Bump(t)$

$\iff \{v \in \mathsf{Variety} \mid t \sqsubseteq v\} \subset \{v' \in \mathsf{Variety} \mid s \sqsubseteq v'\}$ by definitions 1.1, 1.3 and 1.8

$\iff t^{\mathcal{I}} \subset s^{\mathcal{I}}$ by definition of $\mathcal{I}$ and definition 1.5

□

One might consider changing the appropriateness conditions in the first place, for instance allow appropriateness specifications only on maximal types. But that might be an inconvenient system to use, as we are used to the appropriateness specification as presented above. From the point of view of grammar developement, one wants to specify a type hierarchy and appropriateness conditions incrementally, adding further types and feature conditions as the grammar grows. Furthermore, the definition of *ExtApp* is neither mathematically nor computationally a problem.

1.11 **Definition (*ExtApp*)**

ExtApp : $Type \times Feat \rightarrow Type$ *is a partial function such that:*

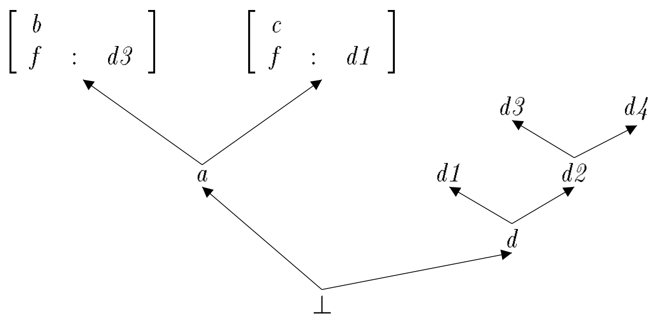
- *if t is a maximal type and $f \in Feat$ then ExtApp(t,f) is defined iff Approp(t,f) is defined and*

ExtApp(t,f) := Bump(Approp(t,f)).
- if t is a non-maximal type and $f \in Feat$ *then ExtApp(t,f) is defined iff ExtApp(v,f) is defined for every variety v properly subsumed by t and*
$ExtApp(t,f) := \bigsqcap \{ExtApp(v,f) \mid v \in \mathsf{Variety}\ \&\ t \sqsubset v\}$

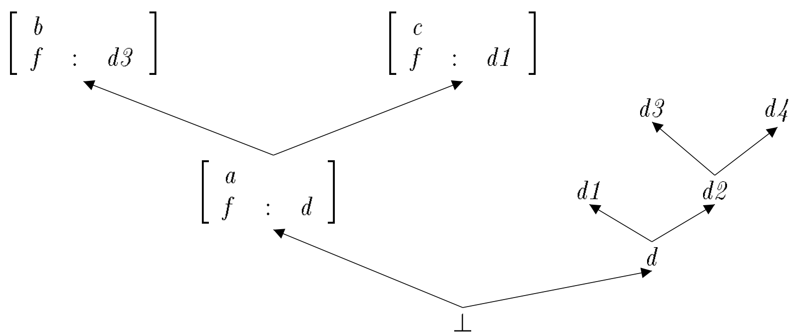
TROLL takes this idea even further. Instead of insisting on a single type as output of the *ExtApp* function, it will also use disjunctions of maximal types if a corresponding type is not defined in a type hierarchy. But since this is not important for our purposes, we'll leave this aside for the moment.

Let's look at an example. Hierarchy a) is an ordinary inheritance hierarchy according to definitions 1.1 and 1.4, b) an equivalent hierarchy using *ExtApp*, and c) ditto for TROLL. In c), the appropriateness condition for f on a has been narrowed down to $\{d1, d3\}$, since these are (by a closed world interpretation) the only types that can possibly occur there. Since there is actually no type that represents the information $\{d1, d3\}$, one could insert a new type into the hierarchy, or use a disjunction. Inserting new types does not seem a nice move, particularly since these new types might appear on a feature structure after type inference.

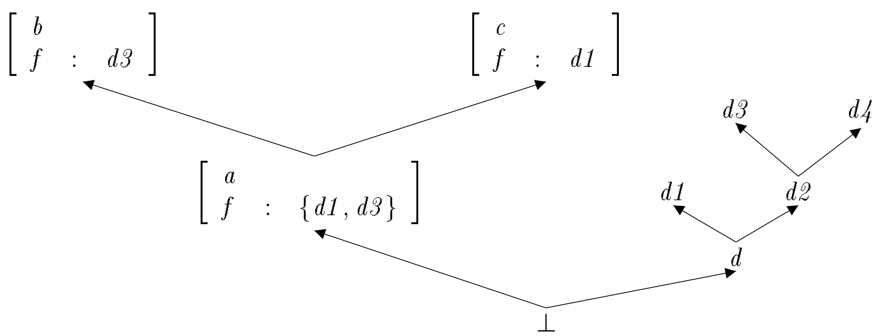
a)



b)



c)



To be able to use *ExtApp* instead of *Approp* for a certain type hierarchy, one needs to know that they have exactly the same semantic effects, i.e that they admit the same interpretations. But first we need to show that *ExtApp* is an appropriateness specification.

1.12 **Lemma (*ExtApp* is an appropriateness specification)**

If Approp is an appropriateness specification and ExtApp is defined over Approp, then ExtApp is also an appropriateness specification (definition 1.4).

Proof

We need only consider non-maximal types as input, since for maximal types as input, *Approp* and *ExtApp* are identical.

$ExtApp(t, f)$ is defined and $t \sqsubseteq t'$

$\Longrightarrow$ $ExtApp(v, f)$ is defined for all varieties v s.t. $t \sqsubseteq v$ and
$\{v \mid v \in \mathsf{Variety} \;\&\; t \sqsubseteq v\} \supseteq \{v' \mid v' \in \mathsf{Variety} \;\&\; t' \sqsubseteq v'\}$

$\Longrightarrow$ $ExtApp(t', f)$ is defined and
$\{ExtApp(v, f) \mid v \in \mathsf{Variety} \;\&\; t \sqsubseteq v\} \supseteq \{ExtApp(v', f) \mid v' \in \mathsf{Variety} \;\&\; t' \sqsubseteq v'\}$

$\Longrightarrow$ $ExtApp(t', f)$ is defined and
$\bigsqcap\{ExtApp(v, f) \mid v \in \mathsf{Variety} \;\&\; t \sqsubseteq v\} \sqsubseteq \bigsqcap\{ExtApp(v', f) \mid v' \in \mathsf{Variety} \;\&\; t' \sqsubseteq v'\}$

$\Longrightarrow$ $ExtApp(t', f)$ is defined and $ExtApp(t, f) \sqsubseteq ExtApp(t', f)$

□

1.13 **Lemma (*ExtApp* is equivalent to *Approp*)**

If $\langle \mathsf{Type}, \sqsubseteq \rangle$ is a type hierarchy and Approp is an appropriateness specification, then

$\langle \mathcal{U}, \mathcal{V}, \mathcal{A} \rangle$ is an interpretation over $\langle \mathsf{Type}, \sqsubseteq \rangle$ and Approp iff $\langle \mathcal{U}, \mathcal{V}, \mathcal{A} \rangle$ is an interpretation over $\langle \mathsf{Type}, \sqsubseteq \rangle$ and ExtApp.

Proof

Only condition 3 of Definition 1.5 is relevant. But condition 3 refers only to the appropriateness conditions of the varieties, and for the varieties, *ExtApp* is exactly the same as *Approp*. □

We have seen that *ExtApp* admits the same interpretations as *Approp* does, but also that *ExtApp* provides more information then *Approp*. Please note that *ExtApp* can only be used under a closed world semantics, under an open world semantics it admits different interpretations than *Approp*.

2 Feature structures

For reasons mentioned in the introduction, we will not only define a syntax, but also a semantics for feature structures.

2.1 Syntax of feature structures

We will use the standard definition of feature structures as finite state automata as given for instance in (Carpenter, 1992). This way of defining feature structures is cumbersome at times since two feature structures can be exactly identical as far as the type and feature information is concerned, but different in their state set. In other words, the class of feature structures forms a preorder under subsumption, and not a partial order as one would hope. One can get around this problem by defining an equivalence relation[2] $\sim$ on feature structures and then consider the class of feature structures modulo $\sim$. On the other hand, it's useful to be able to refer to nodes in a feature structure, and it is easy to picture feature structures in a graph-theoretical manner.

2.1 **Definition (syntax of feature structures)**

A feature structure is a quadruple $\langle Q, \overline{q}, \theta, \delta \rangle$ *such that*

Q *is a finite set of states,*

$\overline{q} \in Q$ *is a distinguished start state,*

$\theta : Q \rightarrow \mathsf{Type}$ *is a type assignment function,*

$\delta : Q \times \mathsf{Feat} \rightharpoonup Q$, *and*

if $q \in Q$ *then for some* $n \in \mathbb{N}$, *some* $q_0, \ldots, q_n \in Q$, *and some* $f_1, \ldots, f_n \in \mathsf{Feat}$

$q_0 = \overline{q}$,

$q_n = q$, *and*

for each $i < n$, $\delta(q_i, f_{i+1})$ *is defined and* $\delta(q_i, f_{i+1}) = q_{i+1}$.

δ is often called a *transition function*, and we call the last condition *reachability condition*. We write $\mathcal{F}$ for the class of feature structures and we can picture feature structures as finite rooted connected and directed graphs, where types label the nodes and features label the arcs.

2.2 Semantics of feature structures

2.2 **Definition (feature strings)**

We give feature strings the usual string definition.

ε *is the empty feature string and*
if π *is a feature string and* $f \in \mathsf{Feat}$, *then* $f\pi$ *is a feature string.*

We write Feat^* for the set of feature strings over a set Feat of features.

2.3 **Definition (feature string concatenation)**

Feature string concatenation $\bullet : \mathsf{Feat}^* \times \mathsf{Feat}^* \rightarrow \mathsf{Feat}^*$ *is a total function from pairs of feature strings to feature strings such that:*

$\varepsilon \bullet \pi = \pi$
$f\pi_1 \bullet \pi_2 = f(\pi_1 \bullet \pi_2)$

We write $\pi_1\pi_2$ for $\pi_1 \bullet \pi_2$.

2.4 **Definition (feature string denotation)**

The denotation of a feature string is the concatenation of the functions denoted by the individual features. If $\mathcal{I} = \langle \mathcal{U}, \mathcal{V}, \mathcal{A} \rangle$ *is an interpretation,* π *a string and* f *a feature, then*

[2] Two feature structures are equivalent iff they are alphabetical variants

$\varepsilon^{\mathcal{I}}(\mathrm{u}) = \mathrm{u}$ *for all* $\mathrm{u} \in \mathcal{U}$ *(the identity function on the domain)*
$(f\pi)^{\mathcal{I}} = \pi^{\mathcal{I}} \circ \mathcal{A}(f)$

2.5 **Definition (δ^*)**

If $F = \langle Q, \overline{q}, \theta, \delta \rangle$ *is in* $\mathcal{F}$*, then* $\delta^* : Q \times \mathsf{Feat}^* \rightharpoonup Q$ *computes the transitive and reflexive closure of* δ *as follows:*

$\delta^*(q, \varepsilon) = q$ *and*
$\delta^*(q, f\pi) = \delta^*(\delta(q, f), \pi)$

The following definition of paths (and later on of arcs) is simply for ease of notation.

2.6 **Definition (paths)**

If $F = \langle Q, \bar{q}, \delta, \theta \rangle$ *is a feature structure, then a path in* F *is a triple* $\langle q, \pi, q' \rangle$ *such that* $q, q' \in Q$, $\pi \in \mathsf{Feat}^*$ *and* $\delta^*(q, \pi)$ *is defined and* $\delta^*(q, \pi) = q'$.

We write P for the set of paths.

2.7 **Definition (direct path)**

If $F = \langle Q, \overline{q}, \theta, \delta \rangle$ *is in* $\mathcal{F}$*, then a path* $m = \langle q, \pi, q' \rangle$ *is direct iff it is not the case that* $\pi = \pi_1 \pi_2 \pi_3$ *for some feature strings* π_1, π_2, π_3 *such that:*

$\pi_2 \neq \varepsilon$ *and*
$\delta^*(\bar{q}, \pi_1) = \delta^*(\bar{q}, \pi_1 \pi_2)$

2.8 **Definition (prefixes and postfixes)**

If $F = \langle Q, \overline{q}, \theta, \delta \rangle$ *is in* $\mathcal{F}$ *and* $m_1 = \langle q, \pi, q' \rangle$, $m_2 = \langle q, \pi_1, q'' \rangle$ *are paths in* F*, then* m_2 *is a prefix of* m_1 *iff there is a* π_2 *such that* $\pi = \pi_1 \pi_2$. *If* $\pi_2 \neq \varepsilon$ *then* m_2 *is a proper prefix of* m_1. *Similarly,* $\langle q'', \pi_2, q' \rangle$ *is a postfix of* m_1.

2.9 **Definition (*Hooks*)**

Hooks $: \mathcal{F} \rightharpoonup Pow(\mathsf{P})$ *is a total function from feature structures to sets of paths s.t. if* $F = \langle Q, \overline{q}, \theta, \delta \rangle$*, then*

$Hooks(F) = \{ \langle \bar{q}, \pi, q \rangle \mid \langle \bar{q}, \pi, q \rangle$ *is a path in* F *and every of its proper prefixes is direct* $\}$

The idea of hooks originated with Drew Moshier's PhD thesis.[3] A hook is a path that may contain a loop only at the end, not in the middle. That's supposed to mean that if a hook comes across a node for the second time, it has to stop there.

2.10 **Definition (denotation of feature structures)**

If $F = \langle Q, \overline{q}, \theta, \delta \rangle$ *is in* $\mathcal{F}$ *and* $\mathcal{I}$ *is an interpretation, then the denotation of* F *is defined as:*

$$[\![F]\!]^{\mathcal{I}} = \{ \mathrm{u} \in \mathcal{U} \mid \langle \bar{q}, \pi_1, q \rangle, \langle \bar{q}, \pi_2, q \rangle \in Hooks(F) \Longrightarrow \\ \pi_1^{\mathcal{I}}(\mathrm{u}){\downarrow} \;\&\; \pi_2^{\mathcal{I}}(\mathrm{u}){\downarrow} \;\&\; \pi_1^{\mathcal{I}}(\mathrm{u}) = \pi_2^{\mathcal{I}}(\mathrm{u}) \;\&\; \theta(q) \sqsubseteq \mathcal{V}(\pi_1^{\mathcal{I}}(\mathrm{u})) \}$$

In effect what this definition does is to compute a conjunction of feature terms à la King and give their denotation. This illustrates the point I made earlier on about feature structures being used as some type of normal form for feature terms. But since we're determined to use feature structures as our formal language, we might as well give their semantics directly, without going via a term language introduced only for this purpose.

A last point on the use of hooks: we might have given the definition of the denotation of feature structures simply in terms of all possible paths instead of hooks. This has the slight conceptual disadvantage that cyclic feature structures have an infinite set of paths. One might want to be able to compute the denotation of a feature structure in finite time, and the correspondence between a feature structure and a conjunction of feature terms would also be less straightforward.

2.3 Models, Satisfiability And Well-Typing

In this section, we will give a standard definition of satisfiability, and look at satisfiable feature structures.

[3] Which I couldn't get hold of and therefore can't properly cite. But see (Carpenter, 1992, p.58)

2.11 **Definition (satisfaction)**

An interpretation $\mathcal{I}$ satisfies a feature structure F iff $[\![F]\!]^{\mathcal{I}} \neq \emptyset$

2.12 **Definition (satisfiable feature structure)**

A feature structure F is satisfiable iff there is an interpretation $\mathcal{I}$ such that $\mathcal{I}$ satisfies F.

The question we need to ask ourselves is: given a type hierarchy, how can we decide if a feature structure is satisfiable? We'll start by looking at the notion of well-typedness (Carpenter, 1992; King and Götz, 1993).

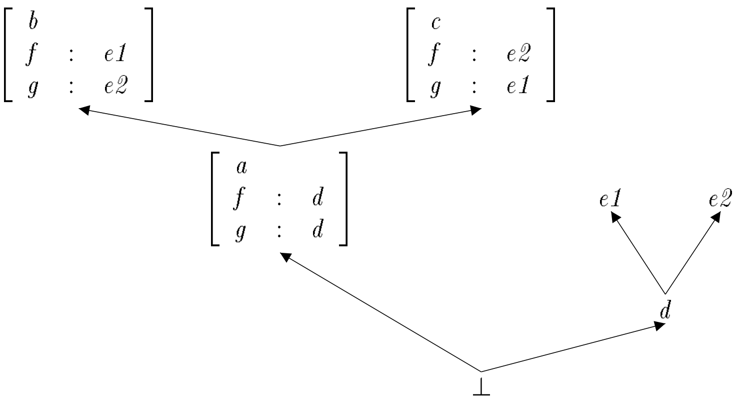
We see from definition 2.10 that the type of a node must subsume the variety of an object if that object is to be in the denotation of that node. But we know more about objects than simply their variety. The appropriateness conditions tell us which features are defined on each object, and they also tell us the range of which features an object may possibly be in. Looking at feature structures, we can determine which are well-behaved according to the appropriateness conditions, and which aren't.

2.13 **Definition (well-typing)**

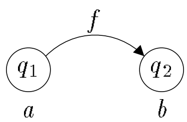
Let $F = \langle Q, \bar{q}, \theta, \delta \rangle \in \boldsymbol{\mathcal{F}}$.

F is well-typed iff for every $q \in Q$, if $\delta(q,f)\downarrow$ then $ExtApp(\theta(q), f)\downarrow$ and $ExtApp(\theta(q), f) \sqsubseteq \theta(\delta(q,f))$

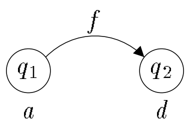
The stronger notion of total well-typing additionally requires that for each node, if a feature is appropriate for the type of that node, then this feature must be defined on that node. We'll briefly look at some examples using the following type hierarchy and appropriateness specification.



The following feature structure is clearly not well-typed, since the type on q_2 is not subsumed by $ExtApp(a, f)$.

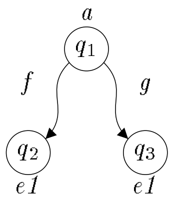


Furthermore, this feature structure denotes the empty set for any interpretation of the above type hierarchy. Let's look at a well-typed example.



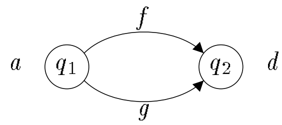
Not only is this feature structure well-typed, there are interpretations s.t. this structure denotes a non-empty set. Suppose for instance that the domain contains an object $\mathbf{o}$ of type c. Then $\mathbf{o}$ is in the denotation of our example feature structure.

These examples lead us to believe that well-typing has something to do with satisfiability. Unfortunately, the connection is not as straightforward as one might wish. Consider yet another example feature structure, still using the same type hierarchy.



Clearly, this feature structure (call it F) is (totally) well-typed, yet it's not satisfiable. To see this, recall that every object is of one and only one variety. Every object denoted by F must be either a b or a c. But the appropriateness conditions state that any object of type b or c is mapped by $\mathcal{A}(f)$ and $\mathcal{A}(g)$ onto objects of different varieties. We conclude that there can be no object in the denotation of a that is mapped onto two objects of type *e1* by both $\mathcal{A}(f)$ and $\mathcal{A}(g)$, thus F is not satisfiable.[4]

Another example using the same hierarchy shows a similar effect.



This time, we enforce identity for the objects under the f- and g-arcs. As we've seen this is not possible, since both b and c require the objects under the f- and g-arcs to be of different varieties, and therefore distinct.

We will now examine some work by Paul King (1993). King showed that well-typed and type-resolved feature structures are satisfiable, and that an arbitrary feature structure is satisfiable iff there is some type-resolved and well-typed feature structures that it subsumes. To this end, he introduced the concept of *Morphs*, which are useful for showing satisfiability. The basic idea is that for each type hierarchy, we can give an interpretation s.t. if a feature structure is satisfiable with respect to the type hierarchy, then it has a non-empty denotation in this interpretation. Such an interpretation is *canonical* in the sense of (Smolka, 1988) and comparable to a Herbrand universe in first order logic.

All the definitions and results of the rest of this section are taken directly from King's work, or follow from it in a straightforward way. We'll only be looking at a skeleton of the proof though to understand the major ideas, many subproofs have been left out.

2.14 **Definition (resolved feature structure)**

A resolved feature structure is a feature structure $\langle Q, \bar{q}, \rho, \delta \rangle$, *where*

$\rho : Q \rightarrow \mathsf{Variety}$, *and*

for each $f \in \mathsf{Feat}$ *and each* $q \in Q$,

if $\delta(q, f)\downarrow$ *then* $Approp(\rho(q), f)\downarrow$ *and* $Approp(\rho(q), f) \sqsubseteq \rho(\delta(q, f))$.

We normally use ρ instead of θ in resolved feature structures just to be able to distinguish them better. We write $\mathcal{RF}$ for the class of resolved feature structures.

As one can see from the definition, resolved feature structures are well-typed. Additionally, every node is labelled with a maximal type. It seems intuitively plausible that these feature structures should be satisfiable.

Next, we define a function that maps an arbitrary feature structure into a set of resolved feature structures.

2.15 **Definition** (R)

$\mathsf{R} : \mathcal{F} \rightarrow Pow(\mathcal{RF})$, *where*

$\mathsf{R}(\langle Q, \overline{q}, \theta, \delta \rangle) = \{\langle Q, \bar{q}, \rho, \delta \rangle \in \mathcal{RF} \mid$ *for each* $q \in Q$, $\theta(q) \sqsubseteq \rho(q)\}$.

[4] This and the following example are due to Paul King (p.c.)

2.16 **Lemma**

For each feature structure F,

$\mathsf{R}(F)$ *is finite.*

The following lemma says that the denotation of a feature structure F is identical to the union of the denotations in the set of resolved feature structures derived from F.

2.17 **Lemma**

For each feature structure F and each interpretation $\mathcal{I}$,

$[\![F]\!]^{\mathcal{I}} = \bigcup\{[\![F']\!]^{\mathcal{I}} \mid F' \in \mathsf{R}(F)\}$.

2.18 **Lemma**

For each feature structure F and each interpretation $\mathcal{I}$,

for each $F_1, F_2 \in \mathsf{R}(F)$, *if* $[\![F_1]\!]^{\mathcal{I}} \cap [\![F_2]\!]^{\mathcal{I}} \neq \emptyset$ *then* $F_1 = F_2$.

Lemma 2.18 follows from our definition of interpretations. Since every object is of exactly one variety, feature structures that have the same shape but where some nodes are labelled with distinct maximal types can't have the same denotation.

We now give the definition of a morph. A morph is the same as totally well-typed type-resolved abstract feature structure. The difference to an ordinary feature structure is that morphs are not necessarily finite, and there are no nodes. Instead of nodes, morphs have equivalence classes of feature paths. We'll make more use of morphs in later sections.

2.19 **Definition (morph)**

A morph is a triple $\langle \Delta, \Gamma, \Lambda \rangle$, *where*

$\Delta \subseteq \mathsf{Feat}^*$,

$\varepsilon \in \Delta$,

for each $\pi \in \mathsf{Feat}^*$ *and each* $f \in \mathsf{Feat}$, *if* $\pi f \in \Delta$ *then* $\pi \in \Delta$,

$\Gamma \subseteq \Delta \times \Delta$,

for each $\pi \in \mathsf{Feat}^*$, *if* $\pi \in \Delta$ *then* $\langle \pi, \pi \rangle \in \Gamma$,

for each π_1 *and* $\pi_2 \in \mathsf{Feat}^*$, *if* $\langle \pi_1, \pi_2 \rangle \in \Gamma$ *then* $\langle \pi_2, \pi_1 \rangle \in \Gamma$,

for each π_1, π_2 *and* $\pi_3 \in \mathsf{Feat}^*$, *if* $\langle \pi_1, \pi_2 \rangle \in \Gamma$ *and* $\langle \pi_2, \pi_3 \rangle \in \Gamma$ *then* $\langle \pi_1, \pi_3 \rangle \in \Gamma$,

for each π_1 *and* $\pi_2 \in \mathsf{Feat}^*$ *and each* $f \in \mathsf{Feat}$,

if $\langle \pi_1, \pi_2 \rangle \in \Gamma$, $\pi_1 f \in \Delta$ *and* $\pi_2 f \in \Delta$ *then* $\langle \pi_1 f, \pi_2 f \rangle \in \Gamma$,

$\Lambda : \Delta \to \mathsf{Variety}$,

for each π_1 *and* $\pi_2 \in \mathsf{Feat}^*$, *if* $\langle \pi_1, \pi_2 \rangle \in \Gamma$ *then* $\Lambda(\pi_1) = \Lambda(\pi_2)$,

for each $\pi \in \mathsf{Feat}^*$ *and each* $f \in \mathsf{Feat}$,

if $\pi f \in \Delta$ *then* $Approp(f, \Lambda(\pi))\downarrow$ *and* $Approp(f, \Lambda(\pi)) \sqsubseteq \Lambda(\pi f)$, *and*

for each $\pi \in \mathsf{Feat}^*$ *and each* $f \in \mathsf{Feat}$,

if $\pi \in \Delta$ *and* $Approp(f, \Lambda(\pi))\downarrow$ *then* $\pi f \in \Delta$.

We write $\mathcal{M}$ for the set of morphs.

Next, King gives a subsumption relation between feature structures and morphs and shows that a morph is subsummed by a feature structure iff it is subsumed by some type resolvant of that feature structure.

2.20 **Definition ($\preceq$)**

For each feature structure $F = \langle Q, \overline{q}, \theta, \delta \rangle$ *and each morph* $\langle \Delta, \Gamma, \Lambda \rangle$,

$\langle Q, \overline{q}, \theta, \delta \rangle \preceq \langle \Delta, \Gamma, \Lambda \rangle$

iff for each π_1 *and* $\pi_2 \in \mathsf{Feat}^*$,

if $\delta^*(\bar{q}, \pi_1)\downarrow$, $\delta^*(\bar{q}, \pi_2)\downarrow$ *and* $\delta^*(\bar{q}, \pi_1) = \delta^*(\bar{q}, \pi_2)$

then $\pi_1 \in \Delta$, $\pi_2 \in \Delta$ *and* $\langle \pi_1, \pi_2 \rangle \in \Gamma$, *and*

for each $\pi \in \mathsf{Feat}^*$ *and each* $t \in \mathsf{Type}$,
if $\delta^*(\bar{q}, \pi)\downarrow$ *and* $t = \theta(\delta^*(\bar{q}, \pi))$
then $\pi \in \Delta$ *and* $t \sqsubseteq \Lambda(\pi)$.

2.21 **Lemma**

For each feature structure F *and each morph* M,
$F \preceq M$ *iff for some* $F' \in \mathsf{R}(F)$, $F' \preceq M$.

2.22 **Lemma**

For each resolved feature structure F,
for some morph M, $F \preceq M$.

This lemma gets us halfway to the satisfiability of resolved feature structures. We just need to show how to get an interpretation out of morphs. The next definition defines a function that yields a morph for every object in an interpretation.

2.23 **Definition** (M)

If $\mathcal{I} = \langle \mathcal{U}, \mathcal{V}, \mathcal{A} \rangle$ *is an interpretation then* $\mathsf{M}_{\mathcal{I}} : \mathcal{U} \to \mathcal{M}$, *where*
$\mathsf{M}_{\mathcal{I}}(u) = \langle \Delta, \Gamma, \Lambda \rangle$
iff $\Delta = \{\pi \mid \pi^{\mathcal{I}}(u)\downarrow\}$,
$\Gamma = \{\langle \pi_1, \pi_2 \rangle \mid \pi_1^{\mathcal{I}}(u)\downarrow, \pi_2^{\mathcal{I}}(u)\downarrow$ *and* $\pi_1^{\mathcal{I}}(u) = \pi_2^{\mathcal{I}}(u)\}$, *and*
$\Lambda = \{\langle \pi, v \rangle \mid \pi^{\mathcal{I}}(u)\downarrow$ *and* $v = \mathcal{V}(\pi^{\mathcal{I}}(u))\}$.

2.24 **Lemma**

For each feature structure F *and each interpretation* $\mathcal{I} = \langle \mathcal{U}, \mathcal{V}, \mathcal{A} \rangle$,
for each $u \in \mathcal{U}$,
$u \in [\![F]\!]^{\mathcal{I}}$ *iff* $F \preceq \mathsf{M}_{\mathcal{I}}(u)$.

Lemma 2.24 shows that for each interpretation, an object is in the denotation of a feature structure iff the feature structure subsumes the morph associated with that object. Now we have to make sure that there is an interpretation $\mathcal{I}$ such that whenever a feature structure subsumes a morph, then there is also an object such that $\mathsf{M}_{\mathcal{I}}$ maps that object into the morph. We do this by making an interpretation out of morphs. King didn't use morphs as objects directly for reasons I don't want to go into here.[5] Instead, he used morphs with an additional pointer that picks out a "node" in that morph. Features are then interpreted as moving that pointer along that feature name, and lemma 2.26 shows that for each morph, there is an object in the interpretation.

2.25 **Definition** ($\widetilde{\mathcal{U}}$, $\widetilde{\mathcal{V}}$ **and** $\widetilde{\mathcal{A}}$)

$\widetilde{\mathcal{U}} = \{\langle \Delta, \Gamma, \Lambda, |\pi|_\Gamma \rangle \mid \langle \Delta, \Gamma, \Lambda \rangle$ *is a morph and* $\pi \in \Delta\}$.

$\widetilde{\mathcal{V}} : \widetilde{\mathcal{U}} \to \mathsf{Variety}$, *where* $\widetilde{\mathcal{V}}(\langle \Delta, \Gamma, \Lambda, |\pi|_\Gamma \rangle) = \Lambda(\pi)$.

$\widetilde{\mathcal{A}} : \mathsf{Feat} \to \{\widetilde{\mathcal{U}} \rightharpoonup \widetilde{\mathcal{U}}\}$, *where*
$\widetilde{\mathcal{A}}(f)\langle \Delta, \Gamma, \Lambda, |\pi|_\Gamma \rangle\downarrow$ *iff* $\pi f \in \Delta$, *and*
if $\widetilde{\mathcal{A}}(f)\langle \Delta, \Gamma, \Lambda, |\pi|_\Gamma \rangle\downarrow$ *then* $\widetilde{\mathcal{A}}(f)\langle \Delta, \Gamma, \Lambda, |\pi|_\Gamma \rangle = \langle \Delta, \Gamma, \Lambda, |\pi f|_\Gamma \rangle$.

2.26 **Lemma**

$\langle \widetilde{\mathcal{U}}, \widetilde{\mathcal{V}}, \widetilde{\mathcal{A}} \rangle$ *is an interpretation.*

Write $\widetilde{\mathcal{I}}$ for $\langle \widetilde{\mathcal{U}}, \widetilde{\mathcal{V}}, \widetilde{\mathcal{A}} \rangle$.

[5] This has to do with the fact that for certain type hierarchies, some feature structures would have the same denotations, although they would have different denotations in other interpretations. In $\widetilde{\mathcal{I}}$ (defined shortly), this is not the case (see lemma 2.31).

2.27 **Lemma**

For each morph $\langle\Delta, \Gamma, \Lambda\rangle$,

$$\mathsf{M}_{\widetilde{\mathcal{I}}}\langle\Delta, \Gamma, \Lambda, |\varepsilon|_\Gamma\rangle = \langle\Delta, \Gamma, \Lambda\rangle.$$

By adding up all the lemmas, we arrive at the main result showing that a feature structure is satisfiable iff it has a non-empty set of type-resolvands.

2.28 **Lemma**

For each feature structure F,

F is satisfiable iff $\mathsf{R}(F) \neq \emptyset$.

Proof

F is satisfiable

$\Longrightarrow$ for some interpretation $\mathcal{I} = \langle\mathcal{U}, \mathcal{V}, \mathcal{A}\rangle$ and some $u \in \mathcal{U}$, $u \in [\![F]\!]^{\mathcal{I}}$

$\Longrightarrow$ for some morph M, $F \preceq M$ — by lemma 2.24

$\Longrightarrow$ $\mathsf{R}(F) \neq \emptyset$, — by lemma 2.21

and

$\mathsf{R}(F) \neq \emptyset$

$\Longrightarrow$ for some morph M and some $F' \in \mathsf{R}(F)$, $F' \preceq M$ — by lemma 2.22

$\Longrightarrow$ for some morph M, $F \preceq M$ — by lemma 2.21

$\Longrightarrow$ for some morph $\langle\Delta, \Gamma, \Lambda\rangle$, $F \preceq \mathsf{M}_{\widetilde{\mathcal{I}}}\langle\Delta, \Gamma, \Lambda, |\varepsilon|_\Gamma\rangle$ — by lemma 2.27

$\Longrightarrow$ F is satisfiable. — by lemma 2.24

□

There are some useful lemmata that follow straight from King's work. We will use them later in section 3. The important one is lemma 2.31 which states that if two feature structures are equivalent in $\widetilde{\mathcal{I}}$, then they are equivalent in any interpretation.

2.29 **Lemma**

For all feature structures F_1, F_2,

$\{m \in \mathcal{M} \mid F_1 \preceq m\} = \{m' \in \mathcal{M} \mid F_2 \preceq m\}$

$\Longleftrightarrow$ *for all interpretations* $\mathcal{I}$, $[\![F_1]\!]^{\mathcal{I}} = [\![F_2]\!]^{\mathcal{I}}$

Proof

$\{m \in \mathcal{M} \mid F_1 \preceq m\} = \{m' \in \mathcal{M} \mid F_2 \preceq m\}$

$\Longleftrightarrow$ $\{u \in \mathcal{U} \mid \mathsf{M}_{\mathcal{I}}(u) \preceq F_1\} = \{u' \in \mathcal{U} \mid \mathsf{M}_{\mathcal{I}}(u) \preceq F_2\}$ for all interpretations $\mathcal{I} = \langle\mathcal{U}, \mathcal{V}, \mathcal{A}\rangle$

$\Longleftrightarrow$ $[\![F_1]\!]^{\mathcal{I}} = [\![F_2]\!]^{\mathcal{I}}$ for all interpretations $\mathcal{I}$ — by lemma 2.24

□

2.30 **Lemma**

For all $m \in \mathcal{M}$ *and all* $F \in \mathcal{F}$

$F \preceq m$ *iff* $\exists \mathrm{u} \in \widetilde{\mathcal{U}}.\ m = \mathsf{M}_{\widetilde{\mathcal{I}}}(\mathrm{u})$ *and* $\mathrm{u} \in [\![F]\!]^{\widetilde{\mathcal{I}}}$

Proof

"$\Longrightarrow$" Let $m = \langle\Delta, \Gamma, \Lambda\rangle$

$F \preceq \langle\Delta, \Gamma, \Lambda\rangle$

$\Longrightarrow$ $F \preceq \mathcal{M}_{\widetilde{\mathcal{I}}}(\langle\Delta, \Gamma, \Lambda, |\varepsilon|_\Gamma\rangle)$ — by lemma 2.21

$\Longrightarrow$ $F \preceq \mathcal{M}_{\widetilde{\mathcal{I}}}(\langle\Delta, \Gamma, \Lambda, |\varepsilon|_\Gamma\rangle)$ and $\langle\Delta, \Gamma, \Lambda, |\varepsilon|_\Gamma\rangle \in [\![F]\!]^{\widetilde{\mathcal{I}}}$ — by lemma 2.24

"$\Longleftarrow$"

$\exists \mathrm{u} \in \widetilde{\mathcal{U}}.\ m = \mathsf{M}_{\widetilde{\mathcal{I}}}(\mathrm{u})$

$\Longrightarrow F \preceq m$ by lemma 2.24

□

2.31 **Lemma**

For all interpretations $\mathcal{I}$ and all feature structures F_1, F_2,

$$[\![F_1]\!]^{\widetilde{\mathcal{I}}} = [\![F_2]\!]^{\widetilde{\mathcal{I}}} \Longrightarrow [\![F_1]\!]^{\mathcal{I}} = [\![F_2]\!]^{\mathcal{I}}$$

Proof

$[\![F_1]\!]^{\widetilde{\mathcal{I}}} = [\![F_2]\!]^{\widetilde{\mathcal{I}}}$

$\Longrightarrow \{m \in \mathcal{M} \mid m \preceq F_1\} = \{m' \in \mathcal{M} \mid m \preceq F_2\}$ by lemma 2.30

$\Longrightarrow [\![F_1]\!]^{\mathcal{I}} = [\![F_2]\!]^{\mathcal{I}}$ for all interpretations $\mathcal{I}$ by lemma 2.29

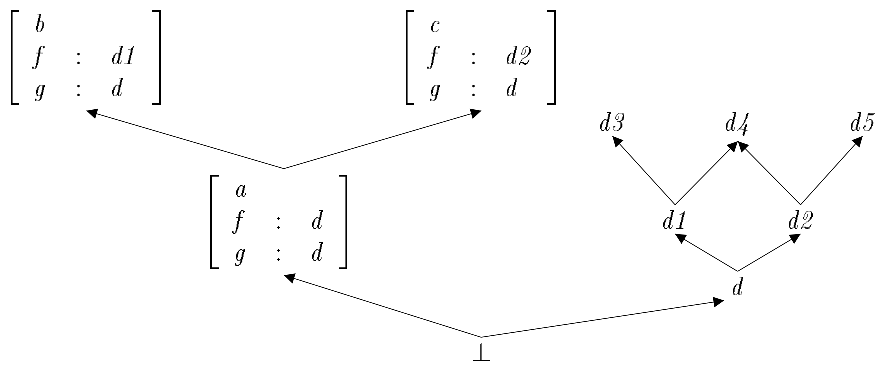
□

In this section we've seen that resolved feature structures are satisfiable, and that we can map any feature structure into a semantically equivalent set of resolved feature structures. So for any feature structure F, $\mathsf{R}(F)$ is a normal form that exhibits satisfiability. Unfortunately, $\mathsf{R}(F)$ will be very big in the general case. This is not important for theoretical considerations, but might make quite a difference in a practical system. But as it turns out, sets of resolved feature structures are also lacking in a theoretical apsect: they don't allow for easy recognition of redundant arcs. We will come back to this issue in section 4.

2.4 Extended Feature Structures

We will now look at a subclass of the satisfiable feature structures that is bigger than $\mathcal{RF}$, but still has some of its nice properties. An extended feature structure is a well-typed (according to *ExtApp*) feature structure such that replacing every type on a node with a maximal type, that is subsumed by the original type yields a resolved feature structure. This implies immediately that every extended feature structure is satisfiable.

For the examples in section 2.4, we will use the following type hierarchy.



2.32 **Definition (extended feature structures)**

A feature structure $F = \langle Q, \overline{q}, \theta, \delta\rangle$ is extended iff for each $q \in Q$, $\theta(q) = Bump(\theta(q))$ and for all arcs $\langle q, f, q'\rangle$ in F and for all maximal types t

$$\theta(q) \sqsubseteq t \implies ExtApp(t, f)\!\downarrow \text{ and } ExtApp(t, f) \sqsubseteq \theta(q')$$

2.33 **Corollary**

Every extended feature structure is well-typed according to ExtApp.

2.34 **Lemma**

If $F = \langle Q, \overline{q}, \theta, \delta\rangle$ is an extended feature structure, then every $F' = \langle Q, \overline{q}, \rho, \delta\rangle$ s.t. for each $q \in Q$, $\rho(q)$ is maximal and $\theta(q) \sqsubseteq \rho(q)$ is a member of $\mathsf{R}(F)$

Proof

Let $F' = \langle Q, \overline{q}, \rho, \delta \rangle$ for any ρ s.t. for each $q \in Q$, $\rho(q)$ is maximal and $\theta(q) \sqsubseteq \rho(q)$.

F is extended and F' is type-resolved

$\Longrightarrow$ F' is well-typed and type-resolved (by construction of F' and definition 2.32)

$\Longrightarrow$ $F' \in \mathsf{R}(F)$

□

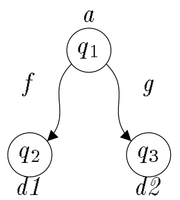
2.35 **Corollary (extended feature structures are satisfiable)**

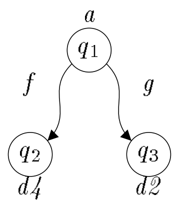
If F is an extended feature structure, then F is satisfiable.

Proof

By definition 2.32, $\mathsf{R}(F) \neq \emptyset$ for every extended feature structure F. Thus, by lemma 2.28, F is satisfiable.
□

Let's consider an example.

a) 

b) 

Feature structure a), although well-typed and satisfiable, is not extended. Furthermore, there is in fact no extended feature structure with the same denotation as a). This shows that even when we consider feature structures modulo semantical equivalence, the class of satisfiable feature structures is properly greater than the class of extended ones.

b) on the other hand is extended. We can make the types on the nodes more specific in whichever way we like, and still b) will remain well-typed.

Yet another proper subclass are the eq-extended feature structures. Those are extended feature structures with the additional requirement that when a feature is defined on a node, the types on the same node in the set of type-resolvands have the same appropriateness conditions for that feature.

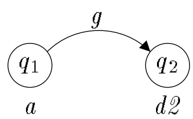
2.36 **Definition (eq-extended feature structures)**

A feature structure $F = \langle Q, \bar{q}, \theta, \delta \rangle \in \boldsymbol{\mathcal{F}}$ is eq-extended iff F is extended and for each node q, for all maximal types v, v'

if $\theta(q) \sqsubseteq v$ and $\theta(q) \sqsubseteq v'$

then for each $f \in \mathsf{Feat}$ s.t. $\delta(q, f)\downarrow$, $ExtApp(v, f) = ExtApp(v', f)$

This may seem a little obscure, so let's look at an example.

c) 

Clearly, the feature structure in c) is extended. Furthermore,

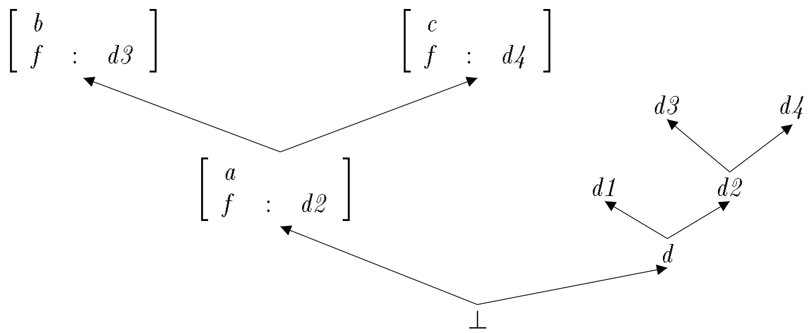
$ExtApp(a, f) = ExtApp(b, f) = d$,

so c) is eq-extended. Note that there is no notion of totality in extended feature structures.

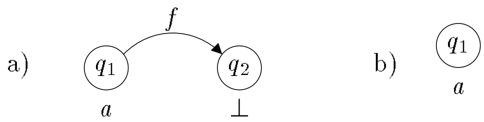
Eq-extended feature structures will play an important role from here on. We will examine their properties in section 3 and finally use them to build a our normal form in section 4.

3 Redundant arcs

A feature structure can contain arcs that are not really informative, since the types on the arcs subsume what is actually appropriate for those arcs. Consider the following type hierarchy

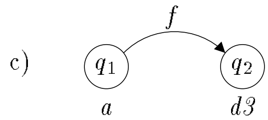


and feature structures



The information that something of type a has a feature f with value of type $\perp$ is clearly redundant, since we can already infer from the appropriateness conditions that every object of type a has feature f defined for it and that the value must be of type d, which is more specific information than the feature structure provides. According to the definition of interpretations, this must be true of any interpretation for the above type hierarchy. We could therefore remove the arc from the feature structure in a) to yield the one in b) without changing the denotation.

The next feature structure, on the other hand, provides more specific information than the appropriateness conditions.



We can not remove the arc without possibly changing the denotation of the feature structure, since there might be objects that are mapped by $\mathcal{A}(f)$ into something of type $d4$. These objects are not contained in the denotation of c), but they are contained in the denotation of b), the result of removing the f-arc from c).

In general we note that if we have a feature structure F, an arc a in F and another feature structure F' that is exactly like F except that it doesn't contain a, then for any type hierarchy and interpretation, the denotation of F is a subset of the denotation of F' (see lemma 3.8).

Before we look at removing arcs more closely, we define some auxilliary concepts.

3.1 **Definition (arcs)**

If $F = \langle Q, \bar{q}, \delta, \theta\rangle$ is a feature structure, then an arc in F is a triple $\langle q, f, q'\rangle$ such that $q, q' \in Q$, $f \in \mathsf{Feat}$, $\delta(q,f)\downarrow$ and $\delta(q,f) = q'$.

3.2 **Definition (terminal arcs)**

If $F = \langle Q, \bar{q}, \delta, \theta\rangle$ is a feature structure and $a = \langle q, f, q'\rangle$ is an arc in F, then a is terminal iff $\delta(q', f')$ is undefined for all $f' \in \mathsf{Feat}$

3.3 **Definition (reentrant arcs)**

If $F = \langle Q, \bar{q}, \delta, \theta\rangle$ is a feature structure and $a = \langle q, f, q'\rangle$ is an arc in F, then a is reentrant iff there is a direct path $m = \langle \bar{q}, \pi, q'\rangle$ in F s.t. a is not a postfix of m

We define a function that removes arcs from feature structures, and then give a more refined version that removes an arc only if the potential result is actually a feature structure.

3.4 **Definition (unconditional remove)**

UR is a partial function from pairs of feature structures and arcs to feature structures that don't necessarily meet the reachability condition. $UR(F, a)$ is defined iff a is an arc in F. Let $F = \langle Q, \bar{q}, \delta, \theta\rangle$ be a feature structure and $a = \langle q_1, f, q_2\rangle$ an arc in F. Then $UR(F, a) = \langle Q', \bar{q}, \delta', \theta'\rangle$ where

- *$\delta'(q, f')$ is defined iff $\delta(q_1, f)$ is defined and $q_1 \neq q$ or $f' \neq f$, and when defined, $\delta'(q, f') = \delta(q, f')$*
- *$Q' = \{q \in Q \mid \delta'(q, f')\downarrow$ or $q = \delta'(q', f')$ for some $q', f'\} \cup \{\bar{q}\}$*
- *$\theta' : Q' \to \mathsf{Type}$ is a total function from Q' to Type such that $\theta'(q) = \theta(q)$ for all $q \in Q'$*

3.5 **Definition (*Remove*)**

$Remove : \mathcal{F} \times \mathsf{Arc} \rightharpoonup \mathcal{F}$ is a partial function from pairs of feature structures and arcs to feature structures s.t.

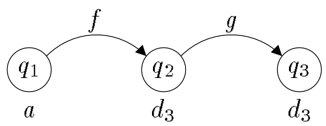
$$Remove(F, a) = \begin{cases} UR(F, a) & \text{if } UR(F, a)\downarrow \text{ and } UR(F, a) \in \mathcal{F} \\ \text{undefined} & \text{otherwise} \end{cases}$$

We will need the following corollary later on.

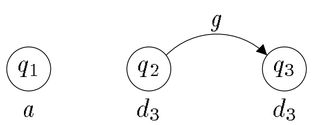
3.6 **Corollary**

If F is an eq-extended feature structure and a is an arc in F s.t. $Remove(F, a)\downarrow$, then $Remove(F, a)$ is also eq-extended.

It is by no means clear that $UR(F, a)$ is always a feature structure. By arbitrarily removing one arc, it is possible to create a graph where not every node can be reached from the root node, i.e. the graph would not be connected any more.



If we take out the f arc, then the result is something which is not a feature structure.



So to be able to remove an arc $\langle q_1, f, q_2\rangle$, there must either be no structure under q_2 or q_2 must be reachable via a different arc as well. The following lemma makes this point precise.

3.7 **Lemma**

If $F \in \mathcal{F}$ and $a = \langle q, f, q'\rangle$ is an arc in F, then $Remove(F, a)$ is defined iff

1. *either a is a terminal arc*
2. *or a is reentrant*

N.B. that the two conditions are not mutually exclusive.

Proof

"$\Longrightarrow$"

$Remove(F, a)$ is defined and $Remove(F, a) = \langle Q', \bar{q}, \delta', \theta' \rangle$

$\Longrightarrow \forall q \in Q'.\ \exists \pi \in \mathsf{Feat}^*.\ \langle \bar{q}, \pi, q \rangle$ is a direct path in $Remove(F, a)$

$\Longrightarrow q' \notin Q'$ or $\exists \pi \in \mathsf{Feat}^*.\ \langle \bar{q}, \pi, q' \rangle$ is a direct path in $Remove(F, a)$ and $\langle q, f, q' \rangle$ is not a postfix of $\langle \bar{q}, \pi, q' \rangle$

$\Longrightarrow a$ is terminal or $\exists \pi \in \mathsf{Feat}^*.\ \langle \bar{q}, \pi, q' \rangle$ is a direct path in F and $\langle q, f, q' \rangle$ is not a postfix of $\langle \bar{q}, \pi, q' \rangle$

"$\Longleftarrow$"

1. a is a terminal arc

   $\Longrightarrow \forall f \in \mathsf{Feat}.\ \delta(q', f)$ is undefined

   $\Longrightarrow Remove(F, a)$ is defined, since removing a will not disconnect any parts of the feature structure

2. a is reentrant

   $\Longrightarrow$ there is a direct path $m = \langle \bar{q}, \pi_1, q' \rangle$ in F s.t. $\langle q, f, q' \rangle$ is not a postfix of m

   $\Longrightarrow m$ is also a path in $UR(F, a)$

   $\Longrightarrow UR(F, a)$ is a feature structure, since q' is still reachable from $\bar{q}$

   $\Longrightarrow Remove(F, a)$ is defined

□

3.8 **Lemma**

If $F = \langle Q, \bar{q}, \delta, \theta \rangle$ is a feature structure and $a = \langle q, \pi, q' \rangle$ is an arc in F and $Remove(F, a)$ is defined, then for all $\mathcal{I} = \langle \mathcal{U}, \mathcal{V}, \mathcal{A} \rangle$:

$[\![F]\!]^{\mathcal{I}} \subseteq [\![Remove(F, a)]\!]^{\mathcal{I}}$

Proof

Follows straight from definition 2.10. □

We now formally define what it means for an arc to be informative, namely if removing it would either result in something that is not a feature structure, or the resulting structure would have a different denotation than the original for some interpretation.

3.9 **Definition (informative arcs)**

If $F = \langle Q, \bar{q}, \delta, \theta \rangle$ is a feature structure and $a = \langle q, f, q' \rangle$ is an arc in F, then a is informative iff

$Remove(F, a)$ is undefined or

$Remove(F, a) = F'$ and there is an interpretation $\langle \mathcal{U}, \cdot^{\mathcal{I}}, \mathcal{V} \rangle$ s.t. $[\![F]\!]^{\mathcal{I}} \neq [\![F']\!]^{\mathcal{I}}$

We say that an arc is uninformative iff it is not informative.

By looking at the definition for the denotation of a feature structure, we can see that reentrant arcs are always informative. Although a removing a reentrant arc always results in a well-formed feature structure, it also always changes the denotation for some interpretation. Actually, we can show that removing a reentrant arc always changes the denotation of a feature structure with respect to $\widetilde{\mathcal{I}}$.

3.10 **Lemma (reentrant arcs are informative)**

If $F = \langle Q, \bar{q}, \delta, \theta \rangle$ is a satisfiable feature structure and $a = \langle q, f, q' \rangle$ is a reentrant arc in F, then a is informative.

Proof

Since a is an arc in F, there is a feature string $\pi_1 \in \mathsf{Feat}^*$ s.t. $\langle \bar{q}, \pi_1 f, q' \rangle$ is a path in F.

a is reentrant

$\Longrightarrow Remove(F, a)\downarrow$ — by lemma 3.7

and there is a direct path $m = \langle \bar{q}, \pi_2, q' \rangle$ in F such that a is not a postfix of m — by definition 3.3

$\Longrightarrow$ there is a morph $M = \langle \Delta, \Gamma, \Lambda \rangle$ s.t. $\mathit{Remove}(F, a) \preceq M$, $\langle \pi_1 f, \pi_2 \rangle \notin \Gamma$ and $F \not\preceq M$ by definition 2.20

$\Longrightarrow$ $[\![\mathit{Remove}(F, a)]\!]^{\widetilde{\mathcal{I}}} \neq [\![F]\!]^{\widetilde{\mathcal{I}}}$ by lemma 2.30

$\Longrightarrow$ a is informative

□

3.11 **Definition (unfilled feature structure)**

A feature structure F is unfilled iff F contains only informative arcs.

We write $\mathcal{UF}$ for the class of unfilled feature structures.

The notion of informative/uninformative arcs is a semantic one, and therefore not very handy. To define a function that transforms feature structures into semantically equivalent ones without uninformative arcs, we need a characterisation that doesn't rely on feature structure denotation, i.e. we need a syntactic characterisation of what it means for an arc to be informative. As it turns out, it is easier to give a characterisation of the opposite, namely informative arcs. We will call such arcs *essential*.

It is very important not to confuse the informative/uninformative distinction with the essential/redundant one. Essential arcs are a purely syntactic notion and correspond to informative arcs only for a proper subclass of the typed feature structures.

	constraining denotation	not constraining denotation
syntactic	essential	redundant
semantic	informative	uninformative

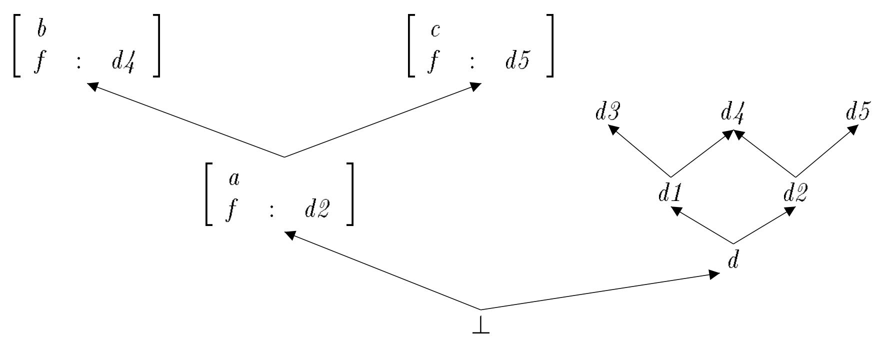
3.12 **Definition (essential arcs)**

If $F = \langle Q, \bar{q}, \delta, \theta \rangle$ is a feature structure and $a = \langle q, f, q' \rangle$ is an arc in F, then a is essential iff it meets one of the following conditions:

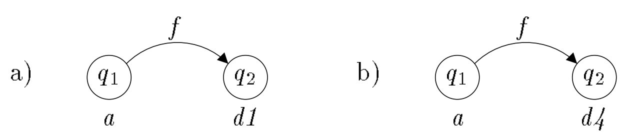
1. *a is not a terminal arc*
2. *a is a reentrant arc*
3. *$\mathit{ExtApp}(\theta(q), f) \sqsubset \theta(q')$*

We say that an arc is redundant iff it is not essential.

We will now look at how closely essential arcs correspond to informative ones. It is perfectly clear that essential arcs correspond to informative ones only for certain subclasses of feature structures. Consider the following type hierarchy



and feature structures



In feature structure a), the only existing arc is informative, yet not essential according to def. 3.12. But this feature structure is not even well-typed. The one in b) on the other hand is totally well-typed, yet its sole arc is essential, but not informative. One can see this by looking at the set of type resolvands of b). The only maximal type we can have on q_1 is actually b, and for f on b, $d4$ is the only possible choice. But please note that b) is not extended.

Before we can show that informative arcs are essential under certain circumstances, we need to prove an auxilliary lemma. This says that if F is an extended feature structure, a a terminal, non-reentrant arc in F and F' is exactly like F except that a is missing, then if there is a morph that is subsumed by F' but not by F, a is essential in F. Since for every morph subsumed by a feature structure there is an object in $\widetilde{\mathcal{I}}$ in the denotation of that feature structure, that means that the arc is also informative (by lemma 2.24).

3.13 **Lemma**

Let $F = \langle Q, \bar{q}, \delta, \theta \rangle$ be an extended feature structure, $a = \langle q, f, q' \rangle$ a terminal, non-reentrant arc in F and $\langle \bar{q}, \pi, q \rangle$ a direct path in F. If there is a morph $M = \langle \Delta, \Gamma, \Lambda \rangle$ s.t. $Remove(F, a) \preceq M$ and $F \not\preceq M$ then $ExtApp(\theta(q), f) \sqsubset \theta(q')$.

Proof

Since F is extended, we know by corollary 2.33 that $ExtApp(\theta(q), f) \sqsubseteq \theta(q')$. We derive a contradiction from the assumption that $ExtApp(\theta(q), f) = \theta(q')$

$[\![F]\!]^{\mathcal{I}} = [\![Remove(F, a)]\!]^{\mathcal{I}} \cap \{\mathrm{u} \in \mathcal{U} \mid \pi f^{\mathcal{I}}(\mathrm{u})\downarrow \;\&\; \pi f^{\mathcal{I}}(\mathrm{u}) \in \theta(q')^{\mathcal{I}}\}$ for all interpretations $\mathcal{I}$, since a is terminal and non-reentrant

$\Longrightarrow [\![F]\!]^{\mathcal{I}} = [\![Remove(F, a)]\!]^{\mathcal{I}} \cap \{\mathrm{u} \in \mathcal{U} \mid \pi f^{\mathcal{I}}(\mathrm{u})\downarrow \;\&\; \theta(q') \sqsubseteq \mathcal{V}(\pi f^{\mathcal{I}}(\mathrm{u}))\}$ for all $\mathcal{I}$

$\Longrightarrow [\![F]\!]^{\mathcal{I}} = [\![Remove(F, a)]\!]^{\mathcal{I}}$ for all $\mathcal{I}$, since $ExtApp(\theta(q), f) = \theta(q')$ by assumption

$\Longrightarrow \{m \in \mathcal{M} \mid F \preceq m\} = \{m' \in \mathcal{M} \mid Remove(F, a) \preceq m\}$ by lemma 2.29

$\Longrightarrow$ contradiction

Therefore, $ExtApp(\theta(q), f) \sqsubset \theta(q')$. $\square$

Now we can show that every informative arc on an extended feature structure is essential and vice versa, i.e. that at least for extended feature structures, we've come up with a good syntactic characterisation of the semantic notion of informative arcs.

3.14 **Lemma (informative arcs are essential)**

If $F = \langle Q, \bar{q}, \delta, \theta \rangle$ is an extended feature structure and $a = \langle q, f, q' \rangle$ is an arc in F, then

a is informative $\Longrightarrow$ a is essential

Proof

It is useful to distinguish two cases in the proof. But first we note that since a is an arc in F, there is a $\pi \in \mathsf{Feat}^*$ s.t. $\langle \bar{q}, \pi, q \rangle$ is a direct path in F.

1. Suppose $Remove(F, a)\uparrow$

a is informative and $Remove(F, a)\uparrow$

$\Longrightarrow$ a is non-terminal and non-reentrant — by lemma 3.7

$\Longrightarrow$ a is non-terminal

$\Longrightarrow$ a is essential — by definition 3.12

2. Suppose $Remove(F, a)\downarrow$. Then we know by lemma 3.7 that a is reentrant or terminal. If a is reentrant then it's essential. It remains to be shown that a is also essential when it is terminal and non-reentrant.

a is informative, terminal and non-reentrant.

$\Longrightarrow [\![F]\!]^{\mathcal{I}} \neq [\![Remove(F, a)]\!]^{\mathcal{I}}$ for some $\mathcal{I}$

$\Longrightarrow$ there is a morph $M = \langle \Delta, \Gamma, \Lambda \rangle$ s.t. $Remove(F, a) \preceq M$ and $F \not\preceq M$ — by lemmas 2.29 and 3.8

$\Longrightarrow ExtApp(\theta(q), f) \sqsubset \theta(q')$ — by lemma 3.13

$\Longrightarrow a$ is essential by definition 3.12

□

3.15 **Lemma (essential arcs are informative)**

If $F = \langle Q, \bar{q}, \delta, \theta \rangle$ is an extended feature structure and $a = \langle q, f, q' \rangle$ is an arc in F, then

a is essential $\Longrightarrow a$ is informative

Proof

We will consider three cases.

1. Suppose a is reentrant. Then a is informative by lemma 3.10.
2. Suppose a is a non-terminal and non-reentrant arc. Then by lemma 3.7, $Remove(F, a)$ is undefined. Thus, a is informative.
3. Suppose a is a terminal, non-reentrant arc s.t. $ExtApp(\theta(q), f) \sqsubset \theta(q')$.

   $ExtApp(\theta(q), f) \sqsubset \theta(q')$

   $\Longrightarrow Bump(ExtApp(\theta(q), f)) \sqsubset \theta(q')$ by definition 1.11

   $\Longrightarrow \theta(q')^{\mathcal{I}} \subset ExtApp(\theta(q), f)^{\mathcal{I}}$ for all interpretations $\mathcal{I}$ s.t $v^{\mathcal{I}} \neq \emptyset$ for all varieties v by lemma 1.10

   $\Longrightarrow [\![F]\!]^{\mathcal{I}} \neq [\![Remove(F, a)]\!]^{\mathcal{I}}$ for some interpretation $\mathcal{I}$

   $\Longrightarrow a$ is informative

□

With these results, it is clear that for each extended feature structure we can find a semantically equivalent unfilled extended feature structure, by simply applying *Remove* to redundant arcs in the structure until there are no more left. The order in which redundant arcs are removed is arbitrary. Indeed, we conjecture that any two semantically equivalent unfilled eq-extended feature structures are alphabetic variants.

4 Normal Form Feature Structures

This section describes the normal form and how to get there. There is an extended example in section 4.3 that it might be useful to refer to now and then.

We begin by extending our notion of an ordinary feature structure to feature structures whose nodes are labelled by sets of maximal types. This is merely an auxilliary notion which we will not need later on. Note that allowing disjunctions of maximal types on nodes slightly increases the expressive power for feature structures. Yet all disjunctions of maximal types could be introduced into the type hierarchy as types without changing any interpretations, since an interpretation depends only on the maximal types.

4.1 **Definition**

$\langle Q, \bar{q}, \psi, \delta \rangle$ is a feature structure in case

- *Q, $\bar{q}$ and δ are as in definition 2.1*
- *$\psi : Q \to Pow^{+}(\mathsf{Variety})$ is a total function from states to non-empty sets of varieties*[6]

We need to extend *ExtApp* to be able to talk about well-typing conditions for these feature structures.

4.2 **Definition**

If S is a set of varieties and f is a feature, then

$ExtApp(S,f)\downarrow$ iff $ExtApp(s,f)\downarrow$ for all $s \in S$ and if $ExtApp(S,f)\downarrow$ then
$ExtApp(S,f) = \{v \in \mathsf{Variety} \mid s \in S$ and $ExtApp(s,f) \sqsubseteq v\}$

To get subsumption and denotation right, we need to know what subsumption means for sets of varieties. We will overload the subsumption operator a bit more to use it also for sets of varieties.

4.3 **Definition**

If S_1 and S_2 are sets of varieties and t is a type, then

$S_1 \sqsubseteq S_2$ iff $S_1 \supseteq S_2$ and
$S_1 \sqsubseteq t$ iff $S_1 \supseteq \{v \in \mathsf{Variety} \mid t \sqsubseteq v\}$

4.4 **Definition (normal form feature structures)**

A normal form feature structure F is a quadruple $\langle Q, \bar{q}, \Theta, \delta \rangle$ where

1. *Q, $\bar{q}$ and δ are as in definition 2.1*
2. *Θ is a non-empty set $\{\theta_1, \ldots, \theta_n\}$ such that each $\theta_i : Q \to Pow^{+}(\mathsf{Variety})$ is a total function from states to non-empty sets of varieties*
3. *for each $\theta \in \Theta$, $\langle Q, \bar{q}, \theta, \delta \rangle$ is eq-extended*
4. *for each $q \in Q$, for each $v \in \mathsf{Variety}$*
   there is a $\theta \in \Theta$ s.t. $v \in \theta(q)$
   $\Longrightarrow$ for each $\theta' \in \Theta$,
   if $ExtApp(\theta'(q), f) = ExtApp(v, f)$ for each $f \in \mathsf{Feat}$ s.t. $\delta(q,f)\downarrow$ and $ExtApp(\theta'(q'), f) \sqsubseteq v$ for each $f \in \mathsf{Feat}$ and $q' \in Q$ s.t. $\delta(q',f)\downarrow$ and $\delta(q',f) = q$, then $v \in \theta'(q)$

We write $\mathcal{NF}$ for the class of normal form feature structures.

Basically a $\mathcal{NF}$-structure is a set of eq-extended feature structures that all have the same shape, but condition 4 severely restricts the possible sets that we can have. Intuitively, condition 4 says that if a variety occurs on a node on one of the eq-extended feature structures in a normal form set, and the appropriateness conditions would allow it to also appear on the same node in a different member of the set, then the variety has to occur there. The net effect of this is that the information in a normal form set has to be represented as compactly and redundancy free as possible. The compactness is particularly important for the recognition

[6] $Pow^{+}(X) = Pow(X) \setminus \{\emptyset\}$

of redundant arcs, and compactness and redundancy freeness together are essential to guarantee that the normal forms will be unique.

We will now look at how we can get from any ordinary feature structure to a semantically equivalent normal form set. The normalisation procedure will consist of the composition of two partial functions. The functions *ExtTypDom* and *ExtTypRan* are based on ideas in (Carpenter, 1992)[7] and explorations of these ideas in (King and Götz, 1993).

As the range of *ExtTypDom* and the domain of *ExtTypRan* we need the notion of an *intermediate feature structure* (def. 4.5). An intermediate strucure filters out all those types that do not deserve a feature defined on the node they're attached to. Furthermore, the varieties are put into equivalence classes according to their appropriateness conditions for the features defined on their nodes.

4.5 **Definition (intermediate feature structure)**

$F = \langle Q, \bar{q}, \Psi, \delta \rangle$ *is an intermediate feature structure iff*

1. Q, $\bar{q}$ *and* δ *are as in def. 2.1*
2. $\Psi : Q \rightarrow Pow^{+}(Pow^{+}(Variety))$ *is a total function from states to non-emtpy sets of non-empty sets of varieties.*
3. *for each* $q \in Q$, *for each* $f \in \mathsf{Feat}$, *for each* $S \in \Psi(q)$
   if $\delta(q,f)\downarrow$ *then* $ExtApp(S,f)\downarrow$
   and for each $v, v' \in S$
   if $\delta(q,f)\downarrow$ *then* $ExtApp(v,f) = ExtApp(v',f)$
4. *for each* $q \in Q$, *for each* $S_1, S_2 \in \Psi(q)$, *for each* $v \in S_1$, $v' \in S_2$
   if $ExtApp(v,f) = ExtApp(v',f)$ *for each* $f \in \mathsf{Feat}$ *s.t.* $\delta(q,f)\downarrow$
   then $S_1 = S_2$

We write $\mathcal{IF}$ for the class of intermediate feature structures.

ExtTypDom looks at all the nodes in a feature structure individually and classifies the varieties subsumed by a type on a node according to their appropriateness conditions for the features defined on that node (for a terminal node, all varieties go into the same class). If there is a node in the feature structure whose type does not subsume a variety that deserves all the features defined on that node, then *ExtTypDom* is undefined for that feature structure.

4.6 **Definition (*ExtTypDom*)**

ExtTypDom : $\mathcal{F} \rightharpoonup \mathcal{IF}$ *is a partial function from feature structures to intermediate feature structures s.t. if* $F = \langle Q, \bar{q}, \theta, \delta \rangle$, *then* $ExtTypDom(F) = \langle Q, \bar{q}, \Psi, \delta \rangle$, *where for each* $q \in Q$,

1. $s \in \{v \in \mathsf{Variety} \mid \theta(q) \sqsubseteq v$ *and* $ExtApp(v,f)\downarrow$ *for each* f *s.t.* $\delta(q,f)\downarrow\}$
   iff $s \in S$ *for some* $S \in \Psi(q)$
2. *for each* $S \in \Psi(q)$, *for each* $s_1, s_2 \in S$, *for each* $f \in \mathsf{Feat}$ *s.t.* $\delta(q,f)\downarrow$,
   $ExtApp(s_1,f) = ExtApp(s_2,f)$
3. *for each* $S_1, S_2 \in \Psi(q)$, *for each* $s_1 \in S_1, s_2 \in S_2$,
   if $ExtApp(s_1,f) = ExtApp(s_2,f)$ *for all* $f \in \mathsf{Feat}$ *s.t.* $\delta(q,f)\downarrow$,
   then $S_1 = S_2$

ExtTypDom$(F)\downarrow$ *iff for each* $q \in Q$, $\Psi(q)$ *is a non-empty set of non-empty sets of varieties.*

---
[7] pp. 89 ff.

4.7 **Corollary**

ExtTypDom is well defined, i.e. for each $F \in \mathcal{F}$ *such that* $ExtTypDom(F)\downarrow$,

$ExtTypDom(F) \in \mathcal{IF}$

Proof

Follows directly from definitions 4.5 and 4.6 $\Box$

Intermediate feature structures are only half way of getting to normal form feature structures. We have that for every node q, if f is defined on q, then $ExtApp(\theta(q), f)$ is defined.[8] What's missing is the part that also $ExtApp(\theta(q), f) \sqsubseteq \theta(\delta(q, f))$.

To get an extended feature structure, we pick an element of $\Psi(q)$ for each q (a selection, def. 4.8) and intersect it with the appropriateness conditions for the incoming arcs (def. 4.9). We do this for all possible selections. It seems fairly plausible that this yields a set of eq-extended feature structures, but it is also important that the compactness condition holds, and we give a formal proof to this effect in lemma 4.10.

4.8 **Definition (selection)**

If $F = \langle Q, \bar{q}, \Psi, \delta\rangle \in \mathcal{IF}$ *then a selection in* Ψ *is a total function* $\psi : Q \rightarrow Pow^{+}(\mathsf{Variety})$ *where for each* $q \in Q$, $\psi(q) \in \Psi(q)$*. We write* $Sel(\Psi)$ *for the set of all selections in* Ψ.

4.9 **Definition (***ExtTypRan***)**

$ExtTypRan : \mathcal{IF} \rightarrow \mathcal{NF}$ *is a partial function from intermediate feature structures to semi-normal form feature structures s.t. if* $F = \langle Q, \bar{q}, \Psi, \delta\rangle \in \mathcal{IF}$*, then* $ExtTypRan(F) = \langle Q, \bar{q}, \Theta, \delta\rangle$*, where*

$$\Theta = \Big\{\theta : Q \rightarrow Pow^{+}(\mathsf{Variety}) \mid \psi \in Sel(\Psi) \text{ and for each } q \in Q,\\ \theta(q) = \psi(q) \cap \bigcap\{ExtApp(\psi(q'), f) \mid \langle q', f, q\rangle \text{ is an arc in } F\} \neq \emptyset\Big\}$$

$ExtTypRan(F)$ *is defined iff* Θ *is non-empty.*

4.10 **Lemma**

ExtTypRan is well defined, i.e. for each $F \in \mathcal{IF}$ *such that* $ExtTypRan(F)\downarrow$,

$ExtTypRan(F) \in \mathcal{NF}$

Proof

We go through the definition of $\mathcal{NF}$ point by point. Let $F = \langle Q, \bar{q}, \Psi, \delta\rangle \in \mathcal{IF}$ s.t. $ExtTypRan(F)\downarrow$ and $ExtTypRan(F) = \langle Q, \bar{q}, \Theta, \delta\rangle$.

1. Q, $\bar{q}$ and δ are unchanged
2. $ExtTypRan(F)\downarrow$ only if $\Theta \neq \emptyset$ and for each $\theta \in \Theta$ and $q \in Q$, $\theta(q) \neq \emptyset$
3. from the fact that $F \in \mathcal{IF}$ and the definition of *ExtTypRan* we can directly conclude that for each $\theta \in \Theta$, for each $q \in Q$, for each $f \in \mathsf{Feat}$

   $\delta(q, f)\downarrow$ and $v, v' \in \theta(q)$

   $\Longrightarrow ExtApp(v, f)\downarrow$, $ExtApp(v', f)\downarrow$, $ExtApp(v, f) = ExtApp(v', f)$ and $ExtApp(\{v, v'\}, f) \sqsubseteq \theta(\delta(q, f))$

   $\Longrightarrow \langle Q, \bar{q}, \theta, \delta\rangle$ is eq-extended
4. for each $q \in Q$, for each $v \in \mathsf{Variety}$

   there is a $\theta \in \Theta$ s.t. $v \in \theta(q)$

   $\Longrightarrow v \in S$ for exactly one $S \in \Psi(q)$ by construction of Ψ

   $\Longrightarrow v \in \psi(q)$ for each $\psi \in Sel(\Psi)$ s.t. $\psi(q) = S$

   $\Longrightarrow$ for each $\theta' \in \Theta$,

   if $ExtApp(\theta'(q), f) = ExtApp(v, f)$ for each $f \in \mathsf{Feat}$ s.t. $\delta(q, f)\downarrow$,
   then $\theta'(q) \subseteq S$

[8] This is called semi-well-typed in King and Götz (1993)

$\Longrightarrow$ for each $\theta' \in \Theta$,

if $ExtApp(\theta'(q), f) = ExtApp(v, f)$ for each $f \in \mathsf{Feat}$ s.t. $\delta(q, f)\downarrow$ and $ExtApp(\theta'(q'), f) \sqsubseteq v$ for each $f \in \mathsf{Feat}$ and $q' \in Q$ s.t. $\delta(q', f)\downarrow$ and $\delta(q', f) = q$,
then $v \in \theta'(q')$

□

The concatenation of *ExtTypDom* and *ExtTypRan* we call *Extend* for ease of notation.

4.11 **Definition**

Extend : $\mathcal{F} \to \mathcal{NF}$ *is a partial function from feature structures to normal form feature structures s.t. for every feature structure* F,

$$Extend(F) = ExtTypRan(ExtTypDom(F))$$

4.1 Denotation of normal form feature structures

To be able to show that *Extend* does not alter the denotation of a feature structure, we first have to say what the denotation of a $\mathcal{NF}$-structure is. As mentioned before, $\mathcal{NF}$-structures can be thought of as sets of feature structures, and we interpret these sets disjunctively.

4.12 **Definition (denotation of normal form feature structures)**

If $\langle Q, \bar{q}, \Theta, \delta\rangle \in \mathcal{NF}$ *then*

$$[\![\langle Q, \bar{q}, \Theta, \delta\rangle]\!]^{\mathcal{I}} = \bigcup\{[\![\langle Q, \bar{q}, \theta, \delta\rangle]\!]^{\mathcal{I}} \mid \theta \in \Theta\} \text{ for all interpretations } \mathcal{I}$$

To show that for any given feature structure F, if $Extend(F)$ is defined, then the denotation of F equals the denotation of $Extend(F)$, we will show that their sets of type resolvands are identical. We start by defining R for $\mathcal{NF}$-structures.

4.13 **Definition**

If $F = \langle Q, \bar{q}, \Theta, \delta\rangle \in \mathcal{NF}$ *then*

$$\mathsf{R}(F) = \{\langle Q, \bar{q}, \rho, \delta\rangle \in \mathcal{RF} \mid \text{there is a } \theta \in \Theta \text{ s.t. for each } q \in Q,\ \rho(q) \in \theta(q)\}$$

After the discussion in section 2.4, we're not surprised to find that picking any variety out of a set assigned to a node will lead to a well-typed, type-resolved feature structure.

4.14 **Lemma**

If $F = \langle Q, \bar{q}, \Theta, \delta\rangle \in \mathcal{NF}$ *then*

$$\mathsf{R}(F) = \{\langle Q, \bar{q}, \rho, \delta\rangle \mid \text{there is a } \theta \in \Theta \text{ s.t. for each } q \in Q,\ \rho(q) \in \theta(q)\}$$

Proof

$F \in \mathcal{NF}$

$\Longrightarrow$ for each $q \in Q$, for each $\theta \in \Theta$, for each $v \in \theta(q)$,

for each $f \in \mathsf{Feat}$, if $\delta(q, f)\downarrow$ then $ExtApp(v, f)\downarrow$ and — by def. 4.6

if $\langle q', f, q\rangle$ is an arc in F then for every $v' \in \theta(q')$, $ExtApp(v', f) \sqsubseteq v$

by def. 4.6 and 4.9

$\Longrightarrow$ $\langle Q, \bar{q}, \rho, \delta\rangle$ is well-typed and type-resolved

□

4.15 **Corollary**

If $F \in \mathcal{NF}$ *then*

$$[\![F]\!]^{\mathcal{I}} = \bigcup\{[\![F']\!]^{\mathcal{I}} \mid F' \in \mathsf{R}(F)\}$$

We can now show that for any feature structure F, $\mathsf{R}(F) = \mathsf{R}(Extend(F))$.

4.16 **Lemma**

If $F = \langle Q, \bar{q}, \theta, \delta\rangle \in \mathcal{F}$, $Extend(F)\downarrow$, $Extend(F) = \langle Q, \bar{q}, \Theta, \delta\rangle$ *and* $F_1 = \langle Q, \bar{q}, \theta_1, \delta\rangle \in \mathcal{RF}$, *then*

$F_1 \in \mathsf{R}(F)$ *iff* $F_1 \in \mathsf{R}(Extend(F))$

Proof

"$\Longrightarrow$"

Let $ExtTypDom(F) = \langle Q, \bar{q}, \Psi, \delta\rangle$

$F_1 \in \mathsf{R}(F)$

$\Longrightarrow$ for each $q \in Q$, $\theta(q) \sqsubseteq \theta_1(q)$ and
for each $f \in \mathsf{Feat}$ s.t. $\delta(q, f)\downarrow$, $ExtApp(\theta_1(q), f)\downarrow$ — since $F_1 \in \mathcal{RF}$

$\Longrightarrow$ for each $q \in Q$, $\theta(q) \sqsubseteq \theta_1(q)$ and $\theta_1(q) \in S$ for some $S \in \Psi(q)$ — by def. 4.6

$\Longrightarrow$ for each $q \in Q$,
$\theta_1(q) \in (\ \{v \in \mathsf{Variety} \mid \theta(q) \sqsubseteq v\} \cap$
$\bigcap\{ExtApp(s, f) \mid \langle q', f, q\rangle$ is an arc in $ExtTypDom(F)$ and
$s \in S$ for some $S \in \Psi(q')\}\ )$ — since $F_1 \in \mathcal{RF}$

$\Longrightarrow$ there is a $\theta_2 \in \Theta$ s.t. for each $q \in Q$, $\theta_1(q) \sqsubseteq \theta_2(q)$ — by def. 4.9

$\Longrightarrow$ $F_1 \in \mathsf{R}(Extend(F))$

"$\Longleftarrow$"

$F_1 \in \mathsf{R}(Extend(F))$

$\Longrightarrow$ there is a $\theta_2 \in \Theta$ s.t. for each $q \in Q$, $\theta_1(q) \sqsubseteq \theta_2(q)$ — by def. 4.13

$\Longrightarrow$ for each $q \in Q$, for each $v \in \theta_2(q)$, $\theta(q) \sqsubseteq v$ — by def. 4.6 and 4.9

$\Longrightarrow$ $F_1 \in \mathsf{R}(F)$

□

This lemma allows us to proof that a feature structure F is satisfiable iff $Extend(F)$ is defined.

4.17 **Lemma**

For each $F = \langle Q, \bar{q}, \theta, \delta\rangle \in \mathcal{F}$,

$[\![F]\!]^{\mathcal{I}} \neq \emptyset$ *for some interpretation* $\mathcal{I}$ *iff* $Extend(F)$ *is defined*

Proof

We make use of the fact that $[\![F]\!]^{\mathcal{I}} \neq \emptyset$ for some $\mathcal{I}$ iff $\mathsf{R}(F) \neq \emptyset$.

"$\Longrightarrow$"

Let $F' = \langle Q, \bar{q}, \theta', \delta\rangle \in \mathsf{R}(F)$

$F' \in \mathsf{R}(F)$

$\Longrightarrow$ for each $q \in Q$, $\theta(q) \sqsubseteq \theta'(q)$ and
for each $f \in \mathsf{Feat}$ s.t. $\delta(q, f)\downarrow$, $ExtApp(\theta'(q), f)\downarrow$ — since $F' \in \mathcal{RF}$

$\Longrightarrow$ for each $q \in Q$, $\theta(q) \sqsubseteq \theta'(q)$ and $ExtTypDom(F)\downarrow$ — by def. 4.6

$\Longrightarrow$ if $ExtTypDom(F) = \langle Q, \bar{q}, \Psi, \delta\rangle$ then for each $q \in Q$ there is an $S \in \Psi(q)$ s.t.
$\theta'(q) \in (\ S \cap \bigcap\{ExtApp(s, f) \mid \langle q', f, q\rangle$ is an arc in $ExtTypDom(F)$ and
$s \in S$ for some $S \in \Psi(q')\}\ )$ — since $F_1 \in \mathcal{RF}$

$\Longrightarrow$ $ExtTypRan(ExtTypDom(F))\downarrow$

$\Longrightarrow$ $Extend(F)\downarrow$

"$\Longleftarrow$"

$Extend(F)\downarrow$

$\Longrightarrow$ $\mathsf{R}(Extend(F)) \neq \emptyset$ — by def. 4.13

$\Longrightarrow$ $\mathsf{R}(F) \neq \emptyset$ — by lemma 4.16

$\Longrightarrow [\![F]\!]^{\mathcal{I}} \neq \emptyset$ for some $\mathcal{I}$

□

Finally, the main result; normalisation doesn't change the denotation.

4.18 **Lemma**

For each $F \in \mathcal{F}$ and every interpretation $\mathcal{I}$, if $Extend(F)$ is defined, then

$$[\![F]\!]^{\mathcal{I}} = [\![Extend(F)]\!]^{\mathcal{I}}$$

Proof

By corollary 4.15 and lemma 4.16 □

In this section, we have seen how to transform feature structures into a normal form that shows satisfiability. It is not clear at this point why this normal form is more useful then, say, the set of resolvands of a feature structure. As we will see in the next section, though, $\mathcal{NF}$ feature structures have the useful property that redundant arcs can be recognised easily.

4.2 Redundant arcs for normal form feature structures

As with the ordinary feature structures, we'll investigate the possibilities to recognise redundant arcs. Removing those arcs, however, is more complicated for $\mathcal{NF}$-sets, since we have to preserve the compactness properties.

We begin by defining a function to remove terminal, non-reentrant arcs from normal form feature structures. As we've seen in section 3, only terminal, non-reentrant arcs can be redundant. Thus it is sufficient to define the removing operation for those arcs. Since the properties of $\mathcal{NF}$-structures are more complex than those of ordinary feature structures, the definition of this function is a bit involved; also the proof that this function is well-defined is somewhat lengthy.

Next, we look at a definition of *essential arcs* for $\mathcal{NF}$-structures and show that uninformative arcs are the same as redundant arcs.

4.19 **Definition** ($RemoveNonRe$)

$RemoveNonRe : \mathcal{NF} \times \mathsf{Arc} \rightharpoonup \mathcal{NF}$ is a partial function from pairs of $\mathcal{NF}$ feature structures and arcs to $\mathcal{NF}$ feature structures such that $RemoveNonRe(F, a)\downarrow$ iff a is a terminal, non-reentrant arc in F; if $F = \langle Q, \bar{q}, \Theta, \delta\rangle \in \mathcal{NF}$, $a = \langle q_1, f_1, q_2\rangle$ and $RemoveNonRe(F, a)\downarrow$ then $RemoveNonRe(F, a) = \langle Q', \bar{q}, \Theta', \delta'\rangle$, where

- $Q' = Q \setminus \{q_2\}$
- *for each $q \in Q'$, for each $f \in \mathsf{Feat}$,*

  $\delta'(q, f)\downarrow$ *iff* $\delta(q, f)\downarrow$ *and* $q \neq q_1$ *or* $f \neq f_1$
  and if $\delta'(q, f)\downarrow$ *then* $\delta'(q, f) = \delta(q, f)$
- *We define an equivalence relation $\sim_{q_1}$ on Θ as follows: for each $\theta, \theta' \in \Theta$, $\theta \sim_{q_1} \theta'$ iff for each $q \in Q' \backslash \{q_1\}$, $\theta(q) = \theta'(q)$ and for each $f \in \mathsf{Feat}$, if $\delta'(q_1, f)\downarrow$ then $ExtApp(\theta(q_1), f) = ExtApp(\theta'(q_1), f)$. Θ' then looks as follows:*

$$\Theta' := \Big\{\theta' : Q' \to Pow^{+}(\mathsf{Variety}) \mid [\theta]_{\sim_{q_1}} \in \Theta_{/\sim_{q_1}},$$
$$\theta'(q_1) = \bigcup\{\theta''(q_1) \mid \theta'' \in [\theta]_{\sim_{q_1}}\} \text{ and for each } q \in Q'\backslash\{q_1\},\ \theta'(q) = \theta(q)\Big\}$$

4.20 **Lemma**

$RemoveNonRe$ is well-defined.

Proof

Let $F = \langle Q, \bar{q}, \Theta, \delta\rangle \in \mathcal{NF}$ and $a = \langle q_1, f_1, q_2\rangle$ such that $RemoveNonRe(F, a)\downarrow$ and let $RemoveNonRe(F, a) = \langle Q', \bar{q}, \Theta', \delta'\rangle$.

1. Clearly, Q, $\bar{q}$ and δ are as in def. 2.1.
2. Θ' is a non-empty set of total functions from Q' to non-empty sets of varieties.
3. The only node we need to consider is q_1. By definition of *RemoveNonRe*, for each $\theta' \in \Theta'$, for each $v \in \theta'(q_1)$, if $\langle q, f, q_1 \rangle$ is an arc in $RemoveNonRe(F, a)$, then $ExtApp(\theta'(q), f) \sqsubseteq v$. Thus, for each $\theta' \in \Theta'$, $\langle Q', \bar{q}, \theta', \delta' \rangle$ is extended.

   Additionally, for each $\theta, \theta' \in \Theta$,

   if $\theta \sim_{q_1} \theta'$ then for each $q \in Q'$ and $f \in \mathsf{Feat}$ s.t. $\delta'(q,f)\downarrow$,
   $ExtApp(\theta(q), f) = ExtApp(\theta'(q), f)$

   and for each $\theta \in \Theta$, each $q \in Q'$ and each $f \in \mathsf{Feat}$ s.t. $\delta'(q,f)\downarrow$,

   if $v, v' \in \theta(q)$ then $ExtApp(v, f) = ExtApp(v', f)$

   $\Longrightarrow$ for each $\theta \in \Theta'$, each $q \in Q'$ and each $f \in \mathsf{Feat}$,

   if $\delta'(q,f)\downarrow$ and $v, v' \in \theta(q)$ then $ExtApp(v, f) = ExtApp(v', f)$

   $\Longrightarrow$ for each $\theta \in \Theta'$, $\langle Q, \bar{q}, \theta, \delta \rangle$ is eq-extended
4. for each $q \in Q$, for each $v \in \mathsf{Variety}$

   there is a $\theta \in \Theta'$ s.t. $v \in \theta(q)$

   $\Longrightarrow$ there is a $\theta_1 \in \Theta$ s.t. $v \in \theta_1(q)$

   $\Longrightarrow$ for each $\theta_2 \in \Theta$,

   if $ExtApp(\theta_2(q), f) = ExtApp(v, f)$ for each $f \in \mathsf{Feat}$ s.t. $\delta(q,f)\downarrow$ and $ExtApp(\theta_2(q'), f) \sqsubseteq v$ for each $f \in \mathsf{Feat}$ and $q' \in Q$ s.t. $\delta(q',f)\downarrow$ and $\delta(q', f) = q$,
   then $v \in \theta_2(q)$

   $\Longrightarrow$ for each $[\theta_3]_{\sim_{q_1}}$,

   if $\theta_4 \in [\theta_3]_{\sim_{q_1}}$ s.t. $ExtApp(\theta_4(q), f) = ExtApp(v, f)$ for each $f \in \mathsf{Feat}$ s.t. $\delta(q,f)\downarrow$ and $ExtApp(\theta_4(q'), f) \sqsubseteq v$ for each $f \in \mathsf{Feat}$ and $q' \in Q$ s.t. $\delta(q',f)\downarrow$ and $\delta(q', f) = q$,
   then $v \in \theta_4(q)$

   $\Longrightarrow$ for each $\theta' \in \Theta'$,

   if $ExtApp(\theta'(q), f) = ExtApp(v, f)$ for each $f \in \mathsf{Feat}$ s.t. $\delta'(q,f)\downarrow$ and $ExtApp(\theta'(q'), f) \sqsubseteq v$ for each $f \in \mathsf{Feat}$ and $q' \in Q$ s.t. $\delta'(q',f)\downarrow$ and $\delta'(q', f) = q$,
   then $v \in \theta'(q)$

$\square$

4.21 **Definition ($\mathcal{NF}$-essential)**

If $F = \langle Q, \bar{q}, \Theta, \delta \rangle \in \mathcal{NF}$ and $a = \langle q, f, q' \rangle$ is an arc in F, then

a is $\mathcal{NF}$-essential iff there is a $\theta \in \Theta$ s.t. a is essential in $\langle Q, \bar{q}, \theta, \delta \rangle$

An arc is essential in a $\mathcal{NF}$-set just in case it is essential in at least one of the elements of this set. This seems intuitively clear, but it's not quite straightforward to prove that it's semantically correct. In the following we will abbreviate $\mathcal{NF}$-essential to essential in unambiguous contexts.

Before we can go on to show that essential arcs coorespond to informative arcs for $\mathcal{NF}$-structures, we need some more auxilliary results.

4.22 **Lemma**

If $F = \langle Q, \bar{q}, \Theta, \delta \rangle \in \mathcal{NF}$, $F_1 = \langle Q, \bar{q}, \theta_1, \delta \rangle$, $F_2 = \langle Q, \bar{q}, \theta_2, \delta \rangle$ where $\theta_1, \theta_2 \in \Theta$ and $F_0 = \langle Q, \bar{q}, \theta_0, \delta \rangle \in \mathsf{R}(F)$ then

$F_0 \in \mathsf{R}(F_1)$ *and* $F_0 \in \mathsf{R}(F_2) \implies F_1 = F_2$

Proof

$F_0 \in \mathsf{R}(F_1)$ and $F_0 \in \mathsf{R}(F_2)$

$\Longrightarrow$ for each $q \in Q$, $\theta_0(q) \in \theta_1(q)$ and $\theta_0(q) \in \theta_2(q)$
$\Longrightarrow$ for each $q \in Q$, if $\delta(q,f)\downarrow$ then $ExtApp(\theta_1(q), f) = ExtApp(\theta_2(q), f)$ since $F \in \mathcal{NF}$
$\Longrightarrow$ $\theta_1 = \theta_2$ since $F \in \mathcal{NF}$
$\Longrightarrow$ $F_1 = F_2$
□

This lemma showed that every resolvand in the set of type resolvands of a $\mathcal{NF}$-set comes from only one member of that set. We can conlcude that the denotations of all members of a $\mathcal{NF}$-set are disjoint.

4.23 **Corollary**

If $F = \langle Q, \bar{q}, \Theta, \delta\rangle \in \mathcal{NF}$, $F_1 = \langle Q, \bar{q}, \theta_1, \delta\rangle$, $F_2 = \langle Q, \bar{q}, \theta_2, \delta\rangle$ *where* $\theta_1, \theta_2 \in \Theta$ *then*
$F_1 \neq F_2 \iff [\![F_1]\!]^{\mathcal{I}} \cap [\![F_2]\!]^{\mathcal{I}} = \emptyset$ *for all* $\mathcal{I}$

The next lemma is the result we were aiming at.

4.24 **Lemma**

If $F = \langle Q, \bar{q}, \Theta, \delta\rangle \in \mathcal{NF}$ *and* $a = \langle q, f, q'\rangle$ *is an arc in* F, *then*
a *is essential* $\iff$ a *is informative*

Proof

"$\Longrightarrow$"

Clearly, reentrant or non-terminal arcs are informative (see section 3). It remains to be shown that terminal and non-reentrant $\mathcal{NF}$-essential arcs are also informative. We derive a contradiction from the assumption that a is essential but not informative.

a is $\mathcal{NF}$-essential and not informative
$\Longrightarrow$ there is a $\theta \in \Theta$ s.t. a is essential in $\langle Q, \bar{q}, \theta, \delta\rangle$
and for all interpretations $\mathcal{I}$, $[\![F]\!]^{\mathcal{I}} = [\![RemoveNonRe(F, a)]\!]^{\mathcal{I}}$
$\Longrightarrow$ $[\![\langle Q, \bar{q}, \theta, \delta\rangle]\!]^{\mathcal{I}} \neq [\![Remove(\langle Q, \bar{q}, \theta, \delta\rangle, a)]\!]^{\mathcal{I}}$ for some $\mathcal{I}$,
for all interpretations $\mathcal{I}$, $[\![F]\!]^{\mathcal{I}} = [\![RemoveNonRe(F, a)]\!]^{\mathcal{I}}$
$\Longrightarrow$ for some $\theta' \in \Theta$, $\theta' \neq \theta$ and $[\![\langle Q, \bar{q}, \theta', \delta\rangle]\!]^{\mathcal{I}} \supseteq [\![Remove(\langle Q, \bar{q}, \theta, \delta\rangle, a)]\!]^{\mathcal{I}} \setminus [\![\langle Q, \bar{q}, \theta', \delta\rangle]\!]^{\mathcal{I}}$
for all interpretations $\mathcal{I}$
$\Longrightarrow$ $[\![Remove(\langle Q, \bar{q}, \theta, \delta\rangle, a)]\!]^{\mathcal{I}} \cap [\![Remove(\langle Q, \bar{q}, \theta', \delta\rangle, a)]\!]^{\mathcal{I}} \neq \emptyset$ for some interpretation $\mathcal{I}$
$\Longrightarrow$ $\theta \sim_{q_1} \theta'$, $\theta \neq \theta'$ by corollary 4.23
$\Longrightarrow$ for each $q \in Q \setminus \{q_1, q_2\}$, $\theta_1(q) = \theta_2(q)$ and for each $f' \neq f$ s.t. $\delta(q_1, f')\downarrow$, $ExtApp(\theta(q_1), f') = ExtApp(\theta'(q_1), f')$ by def.4.19
$\Longrightarrow$ $\theta(q_1) \cap \theta'(q_1) = \emptyset$, since $\theta \neq \theta'$
$\Longrightarrow$ $[\![Remove(\langle Q, \bar{q}, \theta, \delta\rangle, a)]\!]^{\mathcal{I}} \cap [\![Remove(\langle Q, \bar{q}, \theta', \delta\rangle, a)]\!]^{\mathcal{I}} = \emptyset$, contradiction

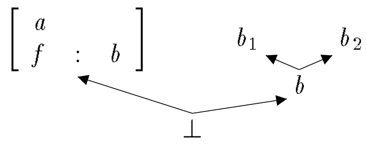
"$\Longleftarrow$"

a is not $\mathcal{NF}$-essential
$\Longrightarrow$ for each $F' = \langle Q, \bar{q}, \theta, \delta\rangle$ s.t. $\theta \in \Theta$, a is not essential in F'
$\Longrightarrow$ for each $F' = \langle Q, \bar{q}, \theta, \delta\rangle$ s.t. $\theta \in \Theta$, $[\![Remove(F', a)]\!]^{\mathcal{I}} = [\![F']\!]^{\mathcal{I}}$ for all $\mathcal{I}$
$\Longrightarrow$ $\{[\![Remove(F', a)]\!]^{\mathcal{I}} \mid F' = \langle Q, \bar{q}, \theta, \delta\rangle \text{ s.t. } \theta \in \Theta\}$
$= \{[\![F']\!]^{\mathcal{I}} \mid F' = \langle Q, \bar{q}, \theta, \delta\rangle \text{ s.t. } \theta \in \Theta\}$ for all $\mathcal{I}$
$\Longrightarrow$ a is not informative on F
□

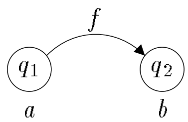
The reasoning for the left-to-right part is maybe not quite so obvious. If a is essential for some θ but not informative for the whole structure, then then there must be a θ' different from θ that already provides the part of the denotation that gets added when a is removed form θ. That means that θ and θ' are q_1-equivalent, since after removing a, there denotations are not disjoint. Yet from this we can conclude that after removing a, θ and θ' are still disjoint, and we have a contradiction.

This proof also shows why we have chosen this particular form of normal form structures (def. 4.4). The definition of essential is extremely simple, but the proof that the definition is the correct one requires all the properties of $\mathcal{NF}$. Other possible representations do not have that property. We could for instance simply take the set of type resolvants of a feature structure as a normal form. But this set would not have the nice property of an easy definition of essential. That the one that we give here wouldn't work can easily be seen from the following example (under a disjunctive interpretation of a set, as usual).

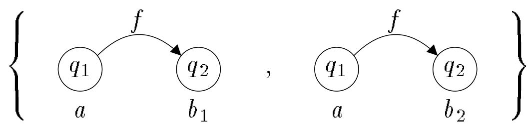
Let



be a type hierarchy and



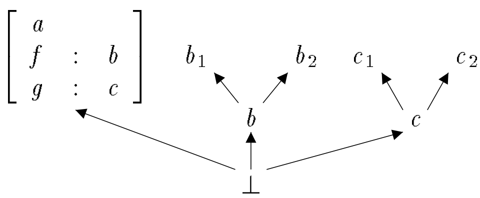
a feature structure F. Then



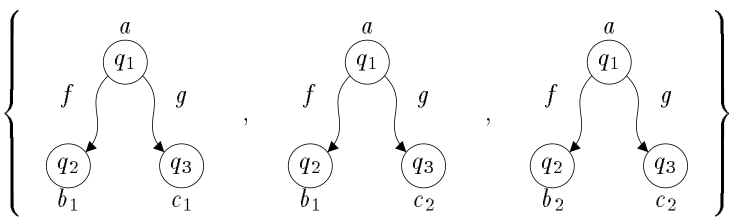
is the set $\mathsf{R}(F)$. Clearly, the arc $\langle q_1, f, q_2\rangle$ is essential on both type resolvands, yet it is not informative on the whole structure. Consequently, a much more sophisticated method for detecting uninformative arcs would have to be employed for this kind of representation. In effect, such a method would probably involve computing a $\mathcal{NF}$-set (if possible). Our next example shows that this is not always possible.

Suppose we had a way of detecting uninformative arcs on sets of type resolvands. Then we might consider dropping the restriction that all graphs have the same underlying shape, i.e. the same feature backbone, and reduce as many structures as far as we can. But there is not a unique way of doing that, as the following example shows.

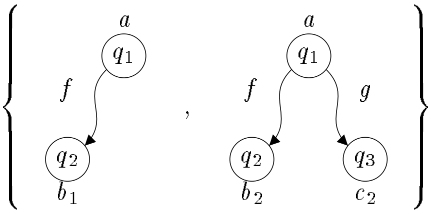
Let



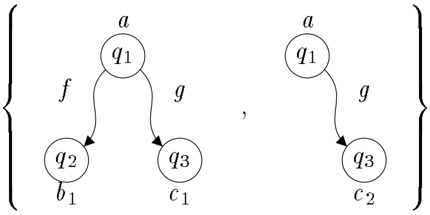
be a type hierarchy and



a set of type resolved feature structures. Now if we want to unfill these structures further, there are two possibilities. We could look at the first two structures and notice that they both have the value b_1 under the f-arc. If we merge these two structures, then the g-arc becomes uninformative and we can remove it to get the following result:



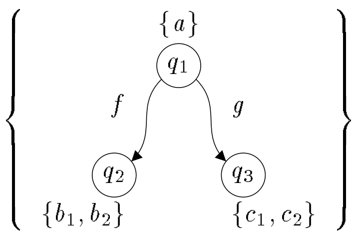
Arguing similarly, we can merge the second and third structure of our original set, and find that the g-arc is redundant. So with equal justification we could derive the following set:



There is no further merging or unfilling that can be done in either of the resulting sets, they're both as compact as possible. Yet these two sets are very different. They are semantically equivalent, but bear no syntactic similarity. A definition of subsumption that makes these two sets subsume each other does not easily spring to mind. So unfilling members of sets of resolvands individually does not seem to be a good idea either.

From the last example, we can also see that our definition of $\mathcal{NF}$ is quite restrictive in the sense that there are many sets of resolvands (with the same underlying graph) that could never be the set of resolved structures of a normal form structure, and by that token, of an ordinary feature structure. In that sense, the last example is an artificial one, since it could never "naturally" arise.

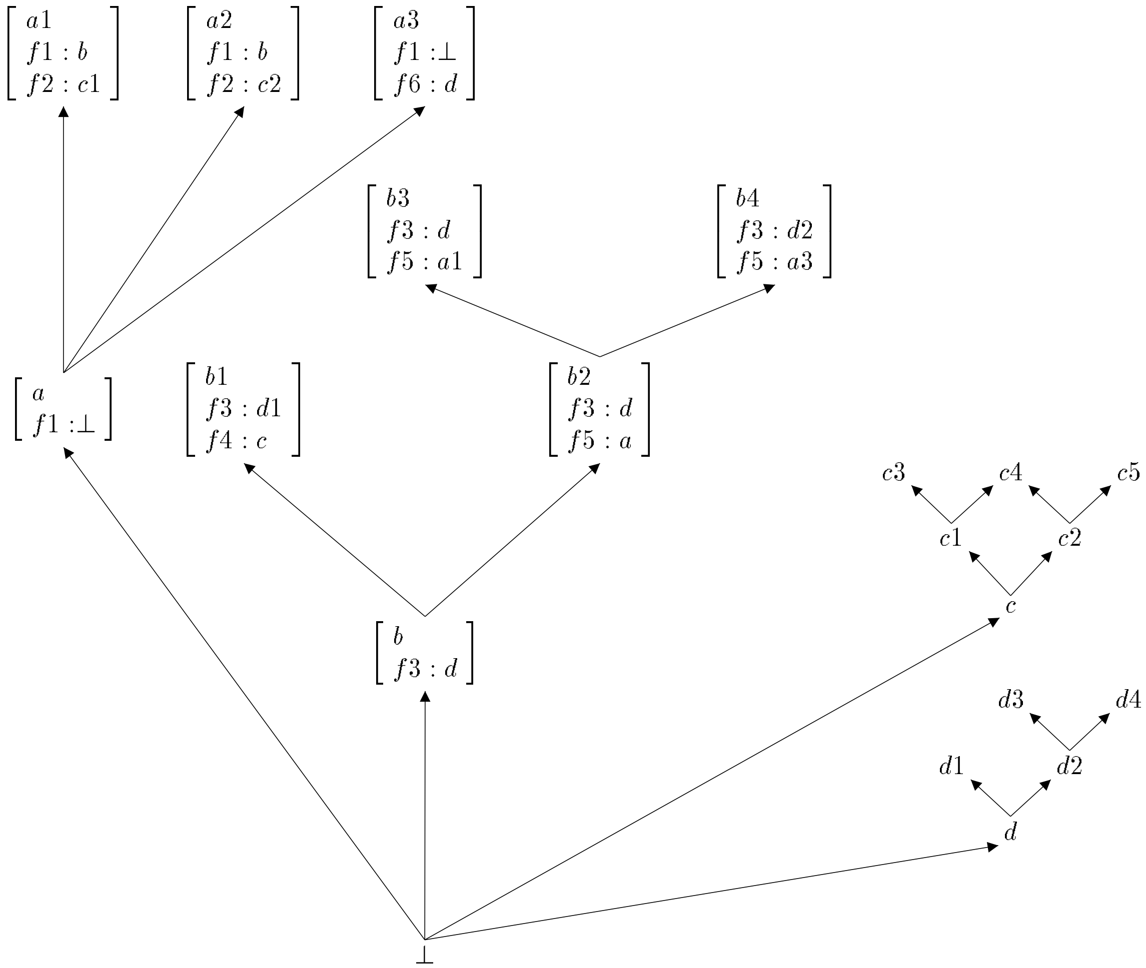
To see why the original set of the example is not a normal form set, consider condition 4 of definition 4.4. When looking at the first two elements of the set, we note that one has type c_1 under the g-arc, and the other c_2, yet both have type a on the root node. If we now apply condition 4 to node q_3, we find that to be in normal form, q_3 should be labelled with $\{c_1, c_2\}$ on both structures. Arguing similarly with elements two and three, we see that node q_2 should be labelled with $\{b_1, b_2\}$. So a possible $\mathcal{NF}$-structure (not unfilled) would be



But that's semantically different form the set we started from, and in fact, there is no $\mathcal{NF}$-structure equivalent to the set in the example.

4.3 An example

An example that uses all the functions desribed in this section sofar might be useful. We'll be using the following type hierarchy.



For space reasons, we'll use AVM notation for feature structures and leave out reference to nodes. The next figure (figure 1, p.34) displays all steps from ordinary feature structure to unfilled $\mathcal{NF}$-structure in one picture.

A first step (*ExtTypDom*, definition 4.6) converts the original feature structure into an intermediate feature structure ($\mathcal{IF}$, definition 4.5). The types on every node are replaced by sets of sets of maximal types. The varieties in the sets all deserve the features defined on the node. Also, all varieties in a set have the same appropriateness conditions for the defined features.

In a second step (*ExtTypRan*, definition 4.9), the intermediate feature structure is mapped onto a $\mathcal{NF}$-structure (definition 4.4). This is done by exploring all possible combinations of the sets on the nodes, to see if a well-typed feature structure can result.

We then look at each of the elements of the first $\mathcal{NF}$-structure and find that the $f3$-arc is redundant in all four of them. Thus we can unfill that arc and obtain a new $\mathcal{NF}$-structure. Now that the $f3$-arc is gone, $b3$ and $b4$ fall into the same equivalence class, and the two feature structures whose root node is labelled with $a1$ can be merged. Likewise the other two.

$$\begin{bmatrix} a & \\ f1: & \begin{bmatrix} b2 & \\ f3: & d \end{bmatrix} \\ f2: & c \end{bmatrix}$$

$$\begin{array}{c} || \\ \mathit{ExtTypDom} \\ \Downarrow \end{array}$$

$$\begin{bmatrix} \{\{a1\},\{a2\}\} & \\ f1: & \begin{bmatrix} \{\{b3\},\{b4\}\} & \\ f3: & \{\{d1,d3,d4\}\} \end{bmatrix} \\ f2: & \{\{c3,c4,c5\}\} \end{bmatrix}$$

$$\begin{array}{c} || \\ \mathit{ExtTypRan} \\ \Downarrow \end{array}$$

$$\left\{ \begin{array}{l} \begin{bmatrix} \{a1\} & \\ f1: & \begin{bmatrix} \{b3\} & \\ f3: & \{d1,d3,d4\} \end{bmatrix} \\ f2: & \{c3,c4\} \end{bmatrix}, \begin{bmatrix} \{a1\} & \\ f1: & \begin{bmatrix} \{b4\} & \\ f3: & \{d3,d4\} \end{bmatrix} \\ f2: & \{c3,c4\} \end{bmatrix}, \\ \begin{bmatrix} \{a2\} & \\ f1: & \begin{bmatrix} \{b3\} & \\ f3: & \{d1,d3,d4\} \end{bmatrix} \\ f2: & \{c4,c5\} \end{bmatrix}, \begin{bmatrix} \{a2\} & \\ f1: & \begin{bmatrix} \{b4\} & \\ f3: & \{d3,d4\} \end{bmatrix} \\ f2: & \{c4,c5\} \end{bmatrix} \end{array} \right\}$$

$$\begin{array}{c} || \\ \mathit{Unfilling}\ f3 \\ \Downarrow \end{array}$$

$$\left\{ \begin{bmatrix} \{a1\} & \\ f1: & \{b3,b4\} \\ f2: & \{c3,c4\} \end{bmatrix}, \begin{bmatrix} \{a2\} & \\ f1: & \{b3,b4\} \\ f2: & \{c4,c5\} \end{bmatrix} \right\}$$

$$\begin{array}{c} || \\ \mathit{Unfilling}\ f2 \\ \Downarrow \end{array}$$

$$\left\{ \begin{bmatrix} \{a1,a2\} & \\ f1: & \{b3,b4\} \end{bmatrix} \right\}$$

Figure 1: an example for type inference and unfilling

In the last step, we remove the $f2$-arc from the two remaining feature structures and merge them into one. But notice that the order in which the arcs are removed is completely arbitrary.

$$\left\{ \begin{bmatrix} \{a1\} & \\ f1: & \begin{bmatrix} \{b3\} & \\ f3: & \{d1,d3,d4\} \end{bmatrix} \\ f2: & \{c3,c4\} \end{bmatrix}, \begin{bmatrix} \{a1\} & \\ f1: & \begin{bmatrix} \{b4\} & \\ f3: & \{d3,d4\} \end{bmatrix} \\ f2: & \{c3,c4\} \end{bmatrix}, \right.$$

$$\left. \begin{bmatrix} \{a2\} & \\ f1: & \begin{bmatrix} \{b3\} & \\ f3: & \{d1,d3,d4\} \end{bmatrix} \\ f2: & \{c4,c5\} \end{bmatrix}, \begin{bmatrix} \{a2\} & \\ f1: & \begin{bmatrix} \{b4\} & \\ f3: & \{d3,d4\} \end{bmatrix} \\ f2: & \{c4,c5\} \end{bmatrix} \right\}$$

$$\begin{array}{c} || \\ \textit{Unfilling } f2 \\ \Downarrow \end{array}$$

$$\left\{ \begin{bmatrix} \{a1,a2\} & \\ f1: & \begin{bmatrix} \{b3\} \\ f3:\{d3,d4\} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \{a1,a2\} & \\ f1: & \begin{bmatrix} \{b4\} \\ f3:\{d1,d3,d4\} \end{bmatrix} \end{bmatrix} \right\}$$

$$\begin{array}{c} || \\ \textit{Unfilling } f3 \\ \Downarrow \end{array}$$

$$\left\{ \begin{bmatrix} \{a1,a2\} & \\ f1: & \{b3,b4\} \end{bmatrix} \right\}$$

Although the intermediate result looks totally different from what we had before, the final outcome is the same.

4.4 Subsumption and unification for $\mathcal{NF}$-structures

This section is of a somewhat speculative nature. It is not yet clear what the best approach to subsumption or a subsumption like operation would be. We'll give a standard definition of subsumption and see what the problems are with that.

The definition of feature structure morphisms is due to (Carpenter, 1992).

4.25 **Definition (feature structure morphism)**

A feature structure morphism[9] *from* $FS_1 = \langle Q_1, \bar{q}_1, \theta_1, \delta_1 \rangle$ *to* $FS_2 = \langle Q_2, \bar{q}_2, \theta_2, \delta_2 \rangle$ *is a total function* $h : Q_1 \to Q_2$ *such that*

- $h(\bar{q}_1) = \bar{q}_2$
- *for every* $q_1 \in Q_1$, $\theta(q_1) \sqsubseteq Bump(\theta(h(q_1)))$
- *for every* $q_1 \in Q_1$ *and every* $f \in \mathsf{Feat}$
  if $\delta_1(q_1, f)\downarrow$ *then* $\delta_2(h(q_1), f)\downarrow$ *and* $h(\delta_1(q_1, f)) = \delta_2(h(q_1), f)$

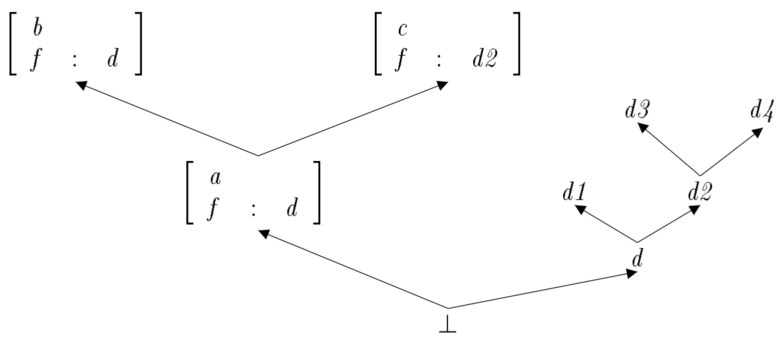
4.26 **Definition (subsumption)**

For each $F_1 = \langle Q_1, \bar{q}_1, \Theta_1, \delta_1 \rangle \in \mathcal{NF}$ *and* $F_2 = \langle Q_2, \bar{q}_2, \Theta_2, \delta_2 \rangle \in \mathcal{NF}$

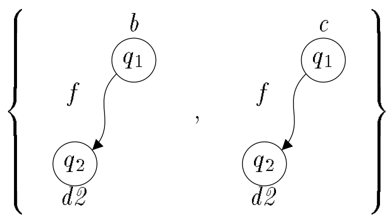
[9] We sometimes write *FSM* for feature structure morphism.

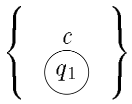
$F_1 \sqsubseteq F_2$ iff
there is a total function $h : Q_1 \to Q_2$ s.t. for each $\theta_2 \in \Theta_2$ there is a $\theta_1 \in \Theta_1$ s.t. h is a feature structure morphism from $\langle Q_1, \bar{q}_1, \theta_1, \delta_1 \rangle$ to $\langle Q_2, \bar{q}_2, \theta_2, \delta_2 \rangle$

Since we're talking about disjunctive sets of feature structures, every member of F_2 must be subsumed by some member of F_1. Unfortunately, subsumption is not complete for $\mathcal{NF}$-structures. Let's look at an example that illustrates this. Consider the following type hierarchy



and feature structures

a) 

b) 

a) is an unfilled $\mathcal{NF}$-set, since the f-arc in the first element of a) is essential. b) is obviusly also an unfilled $\mathcal{NF}$-set. Clearly a) does not subsume b), yet the denotation of a) contains the denotation of b) for all interpretations, since the f-arc on the second structure in a) is redundant, which means that the denotation of that second structure is identical to the denotation of b) for all interpretations. This means that to get a complete subsumption algorithm for unfilled $\mathcal{NF}$-structures, we still have to use a more complicated algorithm, something along the lines of King and Carpenter (forthcoming).

But as it turns out, we've still made quite some progress towards complete subsumption. First of all, by unfilling we cover a lot more cases than without. There are also cases were we lose subsumption by unfilling, but most of the time, we gain by unfilling. Moreover, there's a fairly straightforward subsumption algorithm that is complete. The following definition is not very formal, but it should still be sufficiently clear.

4.27 **Definition (informal)**

Suppose we want to check if $F \sqsubseteq F'$, for $F, F' \in \mathcal{UNF}$. For each F_{eq} in F', we fill in all arcs that are in F, but not in F'. Each newly introduced node, we label with $\perp$. Then we apply Extend to each element of F' to get a new set F''. If Extend is undefined for any element of F', subsumption doesn't hold. Now we can use the old subsumption algorithm to check if $F \sqsubseteq F''$.

Since F'' has the same denotation as F', and F'' has as a minimum all the arcs that F has, we speculate with reasonable confidence that this subsumption check is complete. A more thorough investigation of the formal properties of this algorithm would still be necessary, though.[10]

Note that this algorithm has a serious advantage over the (King and Carpenter, forthcoming) approach: it needs to manipulate and fill in only one of the input structures, (King and Carpenter, forthcoming) needs to do that for both.

The question then arises wether we couldn't unfill individual elements of these sets after all. As we've seen, completely unconstrained sets of (resolved or extended) feature structures are just that: too unconstrained.

[10] Notice for instance that $|F'| \le |F''|$

In particular, completely unfilled sets of resolved structures are not unique under semantic equivalence. One would have to come up with a good characterisation of what constitutes a possible set of unfilled structures and what doesn't. The characterisation would certainly have to have the property of being unique for semantically equivalent structures, since the ordinary subsumption algorithm will only be complete for any class of structures if they have this uniqueness property. Unfortunately, a characterisation of these structures is not so easy. They would have to have a compactness property similar to the one of $\mathcal{NF}$-structures, and they can't just have arbitrarily different arcs.

From a computational point of view, one might actually prefer to stay with unfilled $\mathcal{NF}$-structures as a reasonable compromise. They have the advantage of being easy to represent in a computational system, since all elements of a $\mathcal{NF}$-set have the same shape, and only the type information is different. This would not be so straightforward with sets where the elements possibly have different shapes. There's certainly some research still to be done here.

We'll now look at unification a bit in an informal manner. Unification is not really a main topic of this paper, but since it is the single most important operation in a logical system like TROLL, we should look at some of the properties that the different constructs we've looked at have under unification. Let's consider first which properties a unification algorithm needs to have.

1. The unification of two feature structures F_1, F_2 in a certain class $\mathcal{C}$ must be a feature structure that's also in $\mathcal{C}$. This may seem obvious, but it's an important point for any typed system.

2. If F_1, F_2 are feature structures, then the unification of F_1 and F_2 is defined iff the intersection of the denotations of F_1 and F_2 is non-empty for some interpretation, and the result of the unification must have exactly the same denotation as the intersection of those of F_1 and F_2 for all interpretations.

In other words, unification defines the "conjunction" of two feature structures. Let's consider the standard unification algorithm[11] for typed feature structures (Carpenter 1992, p. 46) and see how well it serves for different classes of feature structures.

If we wanted to work under an open world semantics, we might need a unification algorithm for well-typed feature structures. In that case, the standard algorithm is too weak, since the output is not necessarily well-typed. This means that in a computational system, some run-time type inferencing has to be performed.[12] Under a closed world semantics as we've described it in this paper, the standard algorithm is useless since it doesn't detect unsatisfiability.

For extended feature structures, the standard algorithm is exactly what we need. The extended structures are closed under this operation, and the result is also correct. The picture is slightly different for unfilled extended feature structures. Unifying those will yield an extended, but not necessarily unfilled feature structure.

When we finally consider $\mathcal{NF}$-structures, we get a similar picture as with the extended feature structures. But since we're dealing with sets, we have to unify each member of one set with each of the other. When we use this on $\mathcal{NF}$-structures, we have exactly the algorithm we need, but again, for unfilled $\mathcal{NF}$-structures, we fall out of the class. This is not a problem, however, since the result is still correct (in the sense of 2. above). Knowing this allows us to use a lazy evaluation strategy and to unfill only when it is convenient or necessary for some reason.[13] Notice that we can only use this algorithm because we can do completely static type inference.

[11] The unification of two types is their l.u.b. (if it exists), and the feature part is as for untyped structures.
[12] As for instance in CUF and in ALE.
[13] Practical experiance with TROLL-grammars has shown that hardly any unfilling needs to be done on-line.

5 Conclusion

We have looked at typed feature structures from a closed world semantics angle. Further departing from most current work on feature logics and their implementations, we've directly interpreted feature structures, instead of viewing them as objects in a domain. This has given us the possibility to compare feature structures under a semantic ordering relation, namely set-containment of denotations.

We've seen that a satisfiability check (type inference) for typed feature structures under these circumstances involves looking at sets of satisfiable feature structures, and we've looked at possible candidates out of which to build these sets. The well-typed and type-resolved feature structures were a good point to start from, since we knew that they where satisfiable. Looking for a larger class of satisfiable structures, we hit on the extended feature structures.

Then we looked more closely at feature structures under semantical equivalence, and noticed that for extended feature structures, we could syntactically tell wether a given arc on a feature structure contributed to the denotation or not. We called structures that had no redundant arcs *unfilled*.

In the next section, we looked at subsumption, and realised that the ordinary subsumption algorithm is complete only with respect to a subclass of the extended feature structures, namely the unfilled eq-extended feature structures.

In the last section, we examined how we can get from any given feature structure to a set of satisfiable feature structures, and which conditions these sets obey. These conditions needed to be fairly strong so that a straightforward synatctic definition of a semantically uninformative arc could be given. This allowed us to syntactically define the class of unfilled $\mathcal{NF}$-structures. We conjectured that for any two semantically equivalent feature structures, there is a unique (up to renaming) $\mathcal{NF}$-structure. Unfortunately, what we didn't quite get was completeness of the subsumption algorithm as defined in that section. Finally, we saw that a very simple unification algorithm can be used for $\mathcal{NF}$-structures.

5.1 Future research

It would certainly be interesting to see if a good characterisation of sets of completely unfilled satisfiable feature structures could be given, sets unique for semantical equivalence classes and complete for the (or a) simple subsumption algorithm. This would have to go hand in hand with an investigation of how to represent these structures efficiently in a computational system.

Hooking up ideas presented here with a feature structure treatment of disjunction as in (Griffith, forthcoming) and also negation would be very interesting. Since we now know that type feature logics with full negation are decidable (under a closed world interpretation, see Kepser, forthcoming), the formal ground work has already been laid.

References

Carpenter, B. (1992). *The logic of typed feature structures*, volume 32 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

Dörre, J. and Dorna, M. (1993). CUF - a formalism for linguistic knowledge representation. In Dörre, J., editor, *Computational aspects of constraint based linguistic descriptions I*, pages 1–22. DYANA-2 Deliverable R1.2.A, Universität Stuttgart.

Dörre, J. and Eisele, A. (1991). A comprehensive unification based formalism. DYANA Deliverable R3.1.B, Universität Stuttgart.

Gerdemann, D. (forthcoming). Troll manual. Technical report, Universität Tübingen.

Gerdemann, D. and King, P. J. (1993). Typed feature structures for expressing and computationally implementing feature cooccurence restrictions. In *Proceedings of 4. Fachtagung der Sektion Computerlinguistik der Deutschen Gesellschaft für Sprachwissenschaft*, pages 33–39.

Griffith, J. (forthcoming). *DUSt: a disjunctive approach to unification*. PhD thesis, Universität Tübingen.

Kepser, S. (forthcoming). Master's thesis, Universität Tübingen.

King, P. J. (1989). *A logical formalism for head-driven phrase structure grammar*. PhD thesis, University of Manchester.

King, P. J. (1993). Typed feature structures as descriptions. Unpublished ms., Universität Tübingen.

King, P. J. and Götz, T. W. (1993). Eliminating the feature introduction condition by modifying type inference. Arbeitspapiere des SFB 340 Nr.31, Universität Tübingen.

Pollard, C. J. (in press). Sorts in unification based grammar and what they mean. In Pinkal, M. and Gregor, B., editors, *Unification in natural language analysis*. MIT Press, Cambridge MA.

Shieber, S. (1986). *An Introduction to Unification-Based Approaches to Grammar*. CSLI Lecture Notes No. 4. Chicago University Press, Chicago.

Smolka, G. (1988). A feature logic with subsorts. LILOG-Report 33, IBM Deutschland GmbH, Stuttgart.