A TREATMENT OF NEGATIVE DESCRIPTIONS OF TYPED FEATURE STRUCTURES

KIYOSHI KOGURE
NTT Basic Research Laboratories
9-11, Midori-cho 3-chome, Musashino-shi, Tokyo, 180 Japan
kogure@atom.ntt.jp

Abstract

A formal treatment of typed feature structures (TFSs) is developed to augment TFSs, so that negative descriptions of them can be treated. Negative descriptions of TFSs can make linguistic descriptions compact and thus easy to understand. Negative descriptions can be classified into three primitive negative descriptions: (1) negations of type symbols, (2) negations of feature existences, and (3) negations of feature-address value agreements. The formalization proposed in this paper is based on Aït-Kaci's complex terms. The first description is treated by extending type symbol lattices to include complement type symbols. The second and third are treated by augmenting term structures with structures representing these negations. Algorithms for augmented-TFS unification have been developed using graph unification, and programs using these algorithms have been written in Common Lisp.

1 Introduction

In unification-based or information-based linguistic frameworks, the most important objects are structures called 'feature structures' (FSs), which are used to describe linguistic objects and phenomena. A feature structure is either atomic or complex: an atomic FS is denoted by an atomic symbol; a complex FS consists of a set of feature-value pairs each of which describes an aspect of an object. Partial information on an object is merged by applying the unification operation to FSs.

Research on unification-based linguistic theories has been accompanied by research on FSs themselves. Several extensions on FSs or on feature descriptions and formal treatments of the extensions have been proposed.

Disjunctive and negative descriptions on FSs help make the linguistic descriptions simple, compact, and thus easy to understand. For disjunctive feature descriptions, Kay[14] introduces them into FUG (Functional Unification Grammar) and gives the procedural semantics. Karttunen[11] also proposes procedural treatments of disjunctions in conjunction with relatively simple negations. Rounds and Kasper[19, 13] propose a logic-based formalism—feature logic—which uses automata to model FSs and can treat disjunctive feature descriptions, and they obtain important results.

For negative descriptions of FSs, one of the most fundamental properties of FSs, the partiality of information they carry, makes its insufficient to adopt relatively simple treatments. Classical interpretation of negation, for example, does not allow evaluation of negations to be freely interleaved with unification. Moshier and Rounds[17] propose a formal framework which treats negative feature descriptions on the basis of intuitionistic logic. However, their formalism has trouble treating double negations. Dawar[5] proposes a formal treatment based on three-valued logic.

In order to treat feature domains of complex FSs and to treat taxonomic hierarchies of symbolic feature values, type (or sort) hierarchies have been introduced, allowing definition of typed (or sorted) feature-structures (TFSs). A TFS consists of a type symbol from a lattice and a set of feature-value pairs. A TFS can be seen as a generalized concept of both atomic and complex FSs. Pollard and Sag[18] introduce sorts into HPSG (Head-driven Phrase Structure Grammar) and use sorted FSs to describe linguistic objects.

Aït-Kaci[1] proposes an algebraic framework using the ψ-types and ϵ-types, one of promising formalizations of TFSs, based on lattice theory. This formalization was originally aimed at formalizing and integrating various kinds of knowledge representation frameworks in AI. In this approach, types are defined as equivalence classes of complex term structures. A subsumption relation is defined on these term structures. The join and meet operations on them correspond to the generalization and unification operations on TFSs, respectively. This approach essentially adopts 'type-as-set' semantics. Subtype relationships on type correspond to subsumption relationships on denotations of types. Based on this framework, an extension to Prolog, LOGIN[2], has been developed.

Smolka[20] proposes a feature logic with subsorts. In this approach, negative descriptions can be decomposed into three kinds of primitive negations, namely, negations of sorts or complement sorts which denote the complements of sets that positive counterparts denote, negations of feature existences, and negations of feature-address agreement or feature-address disagreement. Smolka extends feature descriptions but a feature-structure interpretation of an extended description does not include negative information and corresponds to a simple TFS.

Some TFS-based natural language processing systems have been developed[7, 24, 12, 15, 8, 22]. Carpenter and Pollard[4] propose an interface to build type lattices.

Formalizations of extended FSs and of extended feature-descriptions, described above, are classified into two classes: (1) extensions of FSs themselves, and (2) extensions not of FSs themselves but of feature-descriptions. Previous attempts to introduce type hierarchies fall into the former class while previous treatments of disjunctive and negative descriptions mainly fall into the latter.

This paper proposes an extension to Aït-Kaci's ψ-type that incorporates three kinds of the primitive negative descriptions described below into the ψ-type. Aït-Kaci's ψ-type formalization uses term structures. In this paper, both these type structures and the type symbol lattice on which term structures are defined are extended to treat negative descriptions. Negations of type symbols are treated by extending type symbol lattices, and negations of feature existences and feature-address disagreements are treated by extending term structures. This extension can be seen as intuitionistic. The extension is classified into class (1) above.

Based on this paper's formalization, unification algorithms have been developed using graph unification techniques[23, 16]. Programs based on these algorithms have been implemented in Common Lisp.

2 Requirements of Negative Descriptions of TFSs

In describing linguistic information using (typed) feature structures, negative descriptions make the description compact, intuitive, and hence easy to understand. For example, we want to describe the grammatical agreement for an English verb, say "eat", naturally as follows.

$$\mathbf{syn}\left[\textit{agreement} \quad \neg\left(\mathbf{agr}\begin{bmatrix}\textit{person} & \mathrm{3rd}\\ \textit{number} & \mathrm{sg}\end{bmatrix}\right)\right] \tag{1}$$

This description specifies compactly and directly that it is not the case that the person attribute is third and that the number attribute is singular. If we could not use such complex negative descriptions, we would write it using disjunctive descriptions with simple complement types as follows.

$$\left\{\begin{array}{l}\mathbf{syn}[\textit{agreement} \quad \mathbf{agr}[\textit{person} \quad \neg\mathrm{3rd}]]\\ \mathbf{syn}[\textit{agreement} \quad \mathbf{agr}[\textit{number} \quad \neg\mathrm{sg}]]\end{array}\right\} \tag{2}$$

or

$$\left\{\begin{array}{l}\mathbf{syn}[\textit{agreement} \quad \mathbf{agr}[\textit{person} \quad \mathbf{1st}]]\\ \mathbf{syn}[\textit{agreement} \quad \mathbf{agr}[\textit{person} \quad \mathbf{2nd}]]\\ \mathbf{syn}[\textit{agreement} \quad \mathbf{agr}[\textit{number} \quad \mathbf{pl}]]\end{array}\right\} \tag{3}$$

In this case, (1) is easier to understand than (2) or (3).

In the above case, we can describe the information because the complex negative descriptions can be transformed into the disjunction of simple negative descriptions (with an almost same intended meaning) and because both ***person*** and ***number*** features take their values from $\{\mathbf{1st}, \mathbf{2nd}, \mathbf{3rd}\}$ and $\{\mathbf{sg}, \mathbf{pl}\}$. However, it is not always the case that such transformations are possible and that feature takes its value from a finite set.

Let us consider more complicated cases using difference lists expressed using feature structures.[1] The empty list of categories is represented as follows.

$$\mathbf{dlist}\begin{bmatrix}\textit{in} & \mathsf{X1} : \mathbf{list}\\ \textit{out} & \mathsf{X1}\end{bmatrix} \tag{4}$$

In the above example, the tag symbol, X1 shows that features ***in*** and ***out*** must take the same value.

[1]In HPSG and JPSG (Japanese Phrase Structure Grammar), a difference list is very convenient for expressing ***subcat*** and ***slash*** feature values.

How can only non-emptiness be expressed? This is impossible using complement type symbols or disjunctions because we can consider the set of all finite length lists whose elements can be taken from infinite sets. Direct or indirect extension of feature structures is required.

So far, we have discussed the requirement of negative descriptions of type symbols and of feature-value agreements from the viewpoint of capability of describing linguistic information. There are other advantages of allowing negative descriptions. Consider, for example, debugging processes of grammatical descriptions by parsing sample sentences. We may obtain unexpected results such as a TFS with an unexpected type symbol, a TFS with an unexpected feature value agreement and so on. In such situations, negative descriptions can be useful tools for detecting their reasons.

To make linguistic descriptions compact and thus easy to understand, to treat natural language efficiently, and to detect error reasons rapidly, it is necessary to develop formalizations and methods of treating negative descriptions.

3 Formal Treatment of Negative Descriptions of TFSs

As stated earlier, a typed feature structure (TFS) consists of a type symbol and a set of feature-value pairs. Thus, descriptions of TFSs are classified into descriptions of TFSs having:

(1) a certain type symbol (or having a subtype symbol of a certain type symbol),

(2) a feature, and

(3) two feature-address values that agree.

A TFS can be described by using conjunctions and disjunctions of such kinds of descriptions. A conjunctive and disjunctive TFS can be formalized as Aït-Kaci's ψ-type and ϵ-type, respectively. That is, a ψ-type, which has a complex term structure called a ψ-term as its syntax, represents a conjunction of such kinds of descriptions or a conjunctive typed feature structure, and an ϵ-type is a maximal set of ψ-types representing the disjunction of them.

Negative counterparts of these descriptions are classified into descriptions of TFSs:

(1') not having a certain type symbol (or having a type symbol which is not subsumed by a certain type symbol),

(2') not having a certain feature, and

(3') having two feature-address values that do not agree.

By incorporating structures representing such negative descriptions into a ψ-term, a TFS with the negative descriptions can be formalized. Such a term is called an augmented ψ-term and a type with an augmented ψ-term as its syntax is called an augmented ψ-type. From augmented ψ-terms, an augmented ϵ-term can be constructed in the same manner that an ϵ-term is constructed from ψ-terms.

Next, augmented ψ-terms and ψ-types are defined. Term structures are first augmented with structures representing inhibited features and disagreement of feature address values. Then, type symbol lattices are extended to include complement type symbols as suggested in [1].

3.1 Typed Feature Structures as Augmented ψ-Types

In order to define complex term structures, a signature is used to specify their vocabulary. It serves as the interface between their syntax and semantics. A signature is formally defined as follows.

Definition 1 A signature is a quadruple $\langle \mathcal{T}, \leq_{\mathcal{T}}, \mathcal{F}, \mathcal{V}\rangle$ consisting of:

1. a set $\mathcal{T}$ of type symbols containing $\top$ and $\bot$,
2. a partial order $\leq_{\mathcal{T}}$ on $\mathcal{T}$ such that
   (a) $\bot$ is the least and $\top$ is the greatest element, and
   (b) every pair of type symbols $\mathbf{a}, \mathbf{b} \in \mathcal{T}$ have a least upper bound or join, which is denoted by $\mathbf{a} \vee_{\mathcal{T}} \mathbf{b}$ and a greatest lower bound or meet, which is denoted by $\mathbf{a} \wedge_{\mathcal{T}} \mathbf{b}$,
3. a set $\mathcal{F}$ of feature symbols, and
4. a set $\mathcal{V}$ of tag symbols

where $\mathcal{T}$, $\mathcal{F}$ and $\mathcal{V}$ are pairwise disjoint.

A simple 'type-as-set' semantics is adopted for these objects. That is, a type symbol in $\mathcal{T}$ denotes a set of objects in an interpretation. Here, $\top$ and $\bot$ denote the sets called the universe, written as U, and the empty set $\emptyset$, respectively. Another element $\mathbf{a}$ denotes a nonempty subset of U, written as $[\![\mathbf{a}]\!]$. The partial order $\leq_{\mathcal{T}}$ denotes the subsumption relation between these sets; for any type symbols $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$,

1. $\mathbf{a} \leq_{\mathcal{T}} \mathbf{b}$ if and only if $[\![\mathbf{a}]\!] \subseteq [\![\mathbf{b}]\!]$,
2. $\mathbf{a} \vee_{\mathcal{T}} \mathbf{b} = \mathbf{c}$ if and only if $[\![\mathbf{a}]\!] \cup [\![\mathbf{b}]\!] = [\![\mathbf{c}]\!]$, and
3. $\mathbf{a} \wedge_{\mathcal{T}} \mathbf{b} = \mathbf{c}$ if and only if $[\![\mathbf{a}]\!] \cap [\![\mathbf{b}]\!] = [\![\mathbf{c}]\!]$.

A feature symbol denotes a function from a subset of U to U. A feature path is a finite string of feature symbols and denotes the function obtained by the composition of the functions that the feature symbols denote.

A term is defined from a signature. First, a term domain is defined as a skeleton built from feature symbols.

Definition 2 A term domain Δ on $\mathcal{F}$ is a set of finite strings of feature symbols in $\mathcal{F}$ (including the empty string ϵ) such that

1. Δ is prefix-closed: $\forall p, q \in \mathcal{F}^*$, if $p \cdot q \in \Delta$, then $p \in \Delta$; and
2. Δ is finitely branching: if $p \in \Delta$, then $\{f \in \mathcal{F} \mid p \cdot f \in \Delta\}$ is finite

where '$\cdot$' is the string concatenation operator.

An element of a term domain is called a feature address or a feature path. By definition, the empty string ϵ must belong to all term domains and is called the root address. A term domain is represented by a rooted directed graph within which each arc has a feature symbol as its label.

A subdomain of a term domain, corresponding to a subgraph, is defined as follows.

Definition 3 Given a term domain Δ and a feature address $p \in \Delta$, the subdomain of Δ at p is defined to be the term domain $\Delta/p := \{p' \mid p \cdot p' \in \Delta\}$. The set of all subdomains of Δ is denoted by $\mathbf{Subdom}(\Delta)$.

Next, flesh is put on the term structure's skeleton as defined as a term domain by assigning several kinds of objects to each feature address. Aït-Kaci's term structure, the basis of the ψ-type, is defined by assigning a type symbol and a tag symbol to each feature address as follows.

Definition 4 A term is a triple $\langle \Delta, \tau, \upsilon\rangle$ where Δ is a term domain on $\mathcal{F}$, τ is a type symbol function from $\mathcal{F}^*$ to $\mathcal{T}$ such that $\tau(\mathcal{F}^* - \Delta) = \{\top\}$, and υ is a tag symbol function from Δ to $\mathcal{V}$.

Given a tag symbol function υ, $\mathbf{Addr}_\upsilon$ denotes the function from a tag symbol to the set of addresses:

$$\mathbf{Addr}_\upsilon(\mathsf{X}) \;:=\; \{p \in \Delta \mid \upsilon(p) = \mathsf{X}\}. \tag{5}$$

In order to treat negations of feature existences and feature-address value disagreement, the term structure defined above is augmented by assigning additional objects, a set of inhibited features and a set of disagreement tag symbols, to each feature address.

Definition 5 An augmented term is a quintuple $\langle \Delta, \tau, \upsilon, \phi, \chi\rangle$ where Δ is a term domain on $\mathcal{F}$, τ is a type symbol function from $\mathcal{F}^*$ to $\mathcal{T}$ such that $\tau(\mathcal{F}^* - \Delta) = \{\top\}$, υ is a tag symbol function from Δ to $\mathcal{V}$, ϕ is an inhibited feature function from $\mathcal{F}^*$ to $\mathbf{2}^{\mathcal{F}}$ such that $\phi(p)$ is finite for any $p \in \Delta$ and $\phi(\mathcal{F}^* - \Delta) = \{\emptyset\}$, and χ is a disagreement tag symbol function from $\mathcal{F}^*$ to $\mathbf{2}^{\mathcal{V}}$ such that $\chi(p)$ is finite for any $p \in \Delta$ and $\chi(\mathcal{F}^* - \Delta) = \{\emptyset\}$.[2]

The inhibited feature function ϕ specifies which features cannot exist at a given address. There is thus inconsistency if there is an address p in Δ such that

$$\phi(p) \cap \{f \in \mathcal{F} \mid p \cdot f \in \Delta\} \;\neq\; \emptyset. \tag{6}$$

The disagreement tag symbol function χ specifies, for a given address, substructures with which its argument disagrees. There is thus inconsistency if there is an address p in Δ such that

$$\upsilon(p) \;\in\; \chi(p). \tag{7}$$

The disagreement address function $\mathbf{Disagr}_{\upsilon,\chi}$ from Δ to $\mathbf{2}^{\mathcal{F}^*}$, based on υ and χ, takes an address as its argument, and gives the set of addresses with which the argument address must disagree, called the disagreement address set and defined as:

$$\mathbf{Disagr}_{\upsilon,\chi}(p) \;:=\; \bigcup_{\mathsf{X} \in \chi(p)} \mathbf{Addr}_\upsilon(\mathsf{X}). \tag{8}$$

Augmented terms are hereafter referred to simply as terms unless stated otherwise.

Definition 6 Given a term $t = \langle \Delta, \tau, \upsilon, \phi, \chi\rangle$ and a feature address p in Δ, the subterm of t at the address p is the term $t/p = \langle \Delta/p, \tau/p, \upsilon/p, \phi/p, \chi/p\rangle$ where $\tau/p : \mathcal{F}^* \to \mathcal{T}$, $\upsilon/p : \Delta/p \to \mathcal{V}$, $\phi/p : \mathcal{F}^* \to \mathbf{2}^{\mathcal{F}}$, and $\chi/p : \mathcal{F}^* \to \mathbf{2}^{\mathcal{V}}$ are defined by

$$(\tau/p)(q) \;:=\; \tau(p \cdot q), \tag{9a}$$
$$(\upsilon/p)(q) \;:=\; \upsilon(p \cdot q), \tag{9b}$$
$$(\phi/p)(q) \;:=\; \phi(p \cdot q), \tag{9c}$$
$$(\chi/p)(q) \;:=\; \chi(p \cdot q). \tag{9d}$$

For a term $t = \langle \Delta, \tau, \upsilon, \phi, \chi\rangle$, a type symbol $\mathbf{a}$ (similarly, a tag symbol or a term t') is said to occur in t if there is a feature address p in Δ such that $\tau(p) = \mathbf{a}$ (similarly, $\upsilon(p) = \mathsf{X}$ or $\mathsf{X} \in \chi(p)$, or $t/p = t'$).

A term $t = \langle \Delta, \tau, \upsilon, \phi, \chi\rangle$ is said to be regular if the set of all subterms of t, $\mathbf{Subterm}(t) := \{t/p \mid p \in \Delta\}$, is finite. Hereafter, we will consider only regular terms. In a regular term, only finite numbers of type symbols and tag symbols occur.

[2] For any set S, $\mathbf{2}^S$ denotes the set of subsets of S.

$$t_{empty} = \begin{array}{l} \mathsf{X1} : \{\} : \{\} : \mathbf{dlist} \\ \begin{bmatrix} in & \mathsf{X2} : \{\mathit{first}\} : \{\} : \top \\ out & \mathsf{X2} \end{bmatrix} \end{array}$$

$$t_{nonempty} = \begin{array}{l} \mathsf{X3} : \{\} : \{\} : \mathbf{dlist} \\ \begin{bmatrix} in & \begin{array}{l} \mathsf{X4} : \{\} : \{\mathsf{X6}\} : \mathbf{list} \\ [\mathit{first} \ \ \mathsf{X5} : \{\} : \{\} : \top\ ] \end{array} \\ out & \mathsf{X6} : \{\} : \{\mathsf{X4}\} : \mathbf{list} \end{bmatrix} \end{array}$$

Figure 1: Examples of Augmented Terms in Matrix Notation

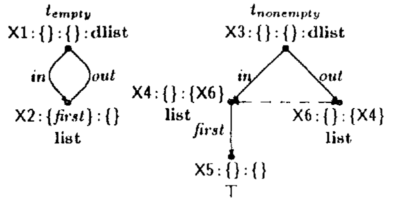

Figure 2: Examples of Augmented Terms in Directed Graph Notation

In a term, any two feature addresses bearing the same symbol are said to corefer. Thus, the coreference relation κ of a term is a relation defined on Δ as the kernel of the tag function v; i.e., $\kappa := \mathbf{Ker}(v) = v^{-1} \circ v$. Here, κ is an equivalence relation and a κ-class is called a coreference class.

Definition 7 A term t is referentially consistent if the same subterm occurs at all feature addresses in a coreference class.

If a term is referentially consistent, then by definition, for any p_1, $p_2 \in \Delta$, if $v(p_1) = v(p_2)$ then, for all p such that $p_1 \cdot p \in \Delta$, it follows that $p_2 \cdot p \in \Delta$ and $v(p_1 \cdot p) = v(p_2 \cdot p)$. Therefore, if a term is referentially consistent, κ is a right-invariant equivalence or right-congruence on Δ. That is, for any p_1, $p_2 \in \Delta$, if $p_1 \kappa p_2$ then $(p_1 \cdot p)\kappa(p_2 \cdot p)$ for any p such that $p_1 \cdot p \in \Delta$.

Definition 8 A well-formed term (wft) is a referentially-consistent regular term. The set of all well-formed terms is denoted by $\mathcal{WFT}$.

A term can be represented in matrix notation. Examples of terms are shown in Figure 1. In this figure, $\top$, **dlist** and **list** are type symbols, *in*, *out* and *first* are feature symbols, and X1, X2, ... are tag symbols. A matrix represents a set of feature-value pairs preceded by a tag symbol, followed by a set of inhibited features and followed by a set of disagreement tag symbols. In the term t_{empty}, its subterms at *in* and at *out* corefer while $t_{nonempty}$ is a term in which its subterms at *in* and at *out* should not corefer. The term t_{empty} should not have the feature address $in \cdot first$ while $t_{nonempty}$ has that address.

A term can also be represented by directed graphs (DGs). t_{empty} and $t_{nonempty}$ in Figure 1 are shown as DGs in Figure 2.

The set $\mathcal{WFT}$ of well-formed terms includes many terms that have the same type symbol function, the same coreference relations, the same inhibited feature function, and the same disagreement address function but different tag symbol functions. These terms have the same information and can describe the same linguistic object or the same linguistic phenomena. These terms construct equivalence classes by renaming tag symbols in a certain manner.

Definition 9 Two terms $t_1 = \langle \Delta_1, \tau_1, v_1, \phi_1, \chi_1 \rangle$ and $t_2 = \langle \Delta_2, \tau_2, v_2, \phi_2, \chi_2 \rangle$ are alphabetical variants of each other if and only if

1. $\Delta_1 = \Delta_2$,
2. $\mathbf{Ker}(v_1) = \mathbf{Ker}(v_2)$,
3. $\tau_1 = \tau_2$,
4. $\phi_1 = \phi_2$, and
5. $\mathbf{Disagr}_{v_1, \chi_1} = \mathbf{Disagr}_{v_2, \chi_2}$.

This is written as $t_1 \,\alpha\, t_2$.

According to 'type-as-set' semantics, the symbols $\top$ and $\bot$ denote, respectively, the least informative type—the whole universe U—and the overdefined or inconsistency type—the empty set $\emptyset$. Therefore, a term containing $\bot$ should be interpreted as inconsistent. Such an inconsistency is called a type inconsistency. To treat such inconsistency, a relation $\Downarrow_1$ on $\mathcal{WFT}$ is defined as follows.

Definition 10 For any two terms t_1, $t_2 \in \mathcal{WFT}$, $t_1 \Downarrow_1 t_2$ if and only if $\bot$ occurs in both t_1 and t_2.

There are other kinds of inconsistency as mentioned earlier. If a term contains an address p such that $\phi(p) \cap \{f \in \mathcal{F} \mid p \cdot f \in \Delta\} \neq \emptyset$, it is inconsistent because it means that there are features that should not exist at the address. Such an inconsistency is called a feature inconsistency.

In addition, if a term contains an address p such that $v(p) \in \chi(p)$, it is inconsistent because it means that the subterm at p does not agree with itself. Such an inconsistency is called a tag inconsistency.

Hence, the three kinds of inconsistency are treated integratedly by a relation $\Downarrow$ on $\mathcal{WFT}$ defined as follows.

Definition 11 For any two terms t_1, $t_2 \in \mathcal{WFT}$, $t_1 \Downarrow t_2$ if and only if each of them contains at least one address p such that

1. $\tau(p) = \bot$,
2. $\phi(p) \cap \{f \in \mathcal{F} \mid p \cdot f \in \Delta\} \neq \emptyset$, or
3. $v(p) \in \chi(p)$.

Clearly, if $\bot$ occurs in a term, it also occurs in all terms in its α-class. This is also true for feature inconsistency and tag inconsistency. Hence, the relations α and $\Downarrow$ are such that their union $\simeq$ becomes an equivalence relation. Thus, we can defined the augmented ψ-types as follows.

Definition 12 An augmented ψ-type (or ψ-type for short) $[t]$ is an element of the quotient set $\Psi := \mathcal{WFT}/\simeq$.

Syntactic structures of augmented ψ-types will be called augmented ψ-terms. An augmented typed-feature-structure can be formalized as an augmented ψ-type.

The set of type symbols $\mathcal{T}$ has the partial order $\leq_{\mathcal{T}}$ which denotes a subsumption relation between the set denoted by type symbols. The partial ordering on $\mathcal{T}$ can be extended to augmented ψ-terms and ψ-types. The subsumption orders on $\mathcal{WFT}$ and on Ψ are defined as follows.

Definition 13 Let $t_1 = \langle \Delta_1, \tau_1, \upsilon_1, \phi_1, \chi_1 \rangle$ and $t_2 = \langle \Delta_2, \tau_2, \upsilon_2, \phi_2, \chi_2 \rangle$ be WFTs. t_1 is said to be subsumed by t_2, written $t_1 \leq t_2$, if and only if either $t_1 \simeq \bot$ or

1. $\Delta_2 \subseteq \Delta_1$,
2. $\mathbf{Ker}(\upsilon_2) \subseteq \mathbf{Ker}(\upsilon_1)$,
3. $\forall p \in \mathcal{F}^*$, $\tau_1(p) \leq_{\mathcal{T}} \tau_2(p)$,
4. $\forall p \in \mathcal{F}^*$, $\phi_2(p) \subseteq \phi_1(p)$, and
5. $\forall p \in \mathcal{F}^*$, $\mathbf{Disagr}_{\upsilon_2,\chi_2}(p) \subseteq \mathbf{Disagr}_{\upsilon_1,\chi_1}(p)$.

The subsumption order on $\boldsymbol{\Psi}$ are defined by $[t_1] \leq [t_2]$ if $t_1 \leq t_2$ is well-defined.

Lattice operations on $\boldsymbol{\Psi}$ can be defined to be compatible with the above subsumption order relation as follows.

Theorem 1 If $\langle \mathcal{T}; \leq_{\mathcal{T}} \rangle$ is a lattice, then so is $\boldsymbol{\Psi}$.

Proof. This theorem can be proved in a very similar manner to the counterpart for Aït-Kaci's ψ-terms. Therefore, instead of providing the proof in detail, only the definitions of the least upper bounds—or joins—and greatest lower bounds—or meets—are provided below. Let $t_1 = \langle \Delta_1, \tau_1, \upsilon_1, \phi_1, \chi_1 \rangle$ and $t_2 = \langle \Delta_2, \tau_2, \upsilon_2, \phi_2, \chi_2 \rangle$ be WFTs.

First, the join of t_1 and t_2, $t_3 = t_1 \vee t_2 = \langle \Delta_3, \tau_3, \upsilon_3, \phi_3, \chi_3 \rangle$, is defined as follows:

$$\Delta_3 = \Delta_1 \cap \Delta_2 \qquad (10a)$$

$\upsilon_3 : \Delta_3 \to \mathcal{V}$ such that

$$\mathbf{Ker}(\upsilon_3) = \kappa_1 \cap \kappa_2, \qquad (10b)$$

and $\forall p \in \mathcal{F}^*$

$$\begin{aligned}
\tau_3(p) &= \tau_1(p) \vee_{\mathcal{T}} \tau_2(p), && (10c)\\
\phi_3(p) &= \phi_1(p) \cap \phi_2(p), \quad \text{and} && (10d)\\
\chi_3(p) &= \{\upsilon_3(q) \mid q \in (\mathbf{Disagr}_{\upsilon_1,\chi_1}(p) \\
&\qquad \cap \mathbf{Disagr}_{\upsilon_2,\chi_2}(p))\}. && (10e)
\end{aligned}$$

Next, the meet of t_1 and t_2, $t_4 = t_1 \wedge t_2 = \langle \Delta_4, \tau_4, \upsilon_4, \phi_4, \chi_4 \rangle$, is defined as follows:

$$\Delta_4 = \Delta^{[*]}, \qquad (11a)$$

$\upsilon_4 : \Delta_4 \to \mathcal{V}$ such that

$$\mathbf{Ker}(\upsilon_4) = \kappa^{[*]}, \qquad (11b)$$

and $\forall p \in \mathcal{F}^*$

$$\begin{aligned}
\tau_4(p) &= \bigvee\nolimits_{\mathcal{T}} \{\tau_i(q) \mid p\kappa_p q,\ i = 1,2\}, && (11c)\\
\phi_4(p) &= \bigcup \{\phi_i(q) \mid p\kappa_p q,\ i = 1,2\}, && (11d)
\end{aligned}$$

and

$$\begin{aligned}
\chi_4(p) &= \bigcup \{\upsilon_4(q) \mid q\kappa_q r, \\
&\qquad r \in (\mathbf{Disagr}_{\upsilon_1,\chi_1}(p) \\
&\qquad \cup \mathbf{Disagr}_{\upsilon_2,\chi_2}(p))\} && (11e)
\end{aligned}$$

where

$$\Delta^{[*]} = \bigcup_{n=0}^{\infty} \Delta^{[n]},$$

$$\Delta^{[n]} = \begin{cases} \Delta_1 \cup \Delta_2 & \text{for } n = 0, \\ \Delta^{[n-1]} \cup \\ \quad \{p \in \mathcal{F}^* \mid p\kappa^{[*]}q,\ q \in \Delta^{[n-1]}\} & \text{for } n \geq 1, \end{cases}$$

$$\kappa^{[*]} = \bigcup_{n=0}^{\infty} \kappa^{[n]},$$

$$\begin{array}{l}
t_{empty} \vee t_{nonempty} \\
\qquad \mathsf{X7} : \{\} : \{\} : \mathbf{dlist} \\
\quad = \begin{bmatrix} \mathit{in} & \mathsf{X8} : \{\} : \{\} : \mathsf{list} \\ \mathit{out} & \mathsf{X9} : \{\} : \{\} : \mathsf{list} \end{bmatrix} \\
t_{empty} \wedge t_{nonempty} \\
\qquad \mathsf{X10} : \{\} : \{\} : \mathbf{dlist} \\
\quad = \begin{bmatrix} \mathit{in} & \begin{array}{l} \mathsf{X11} : \{\mathit{first}\} : \{\mathsf{X11}\} : \mathbf{list} \\ \left[\mathit{first} \quad \mathsf{X12} : \{\} : \{\} : \top \right] \end{array} \\ \mathit{out} & \mathsf{X11} \end{bmatrix} \\
\quad \simeq \bot
\end{array}$$

Figure 3: Examples of Join and Meet of Augmented ψ-Terms

$$\kappa^{[n]} = \begin{cases} \bigcup_{m \geq 0} (\kappa_1^{\Delta_1 \cup \Delta_2} \circ \kappa_2^{\Delta_1 \cup \Delta_2})^m, \\ \qquad\qquad \text{for } n = 0, \\ \kappa^{[n-1]} \cup \\ \quad \{(p_1 \cdot p, p_2 \cdot p) \mid p_1 \kappa^{[n-1]} p_2\}, \\ \qquad\qquad \text{for } n \geq 1 \end{cases}$$

and $\kappa_i^{\Delta_1 \cup \Delta_2}$ is the reflexive extension of κ_i from Δ_i to $\Delta_1 \cup \Delta_2$ for $i = 1, 2$.

The conditions (11a–11e) define a meet, that collapses to $\bot$ whenever conditions (11c–11e) produce some address p such that type inconsistency, feature inconsistency, or tag inconsistency occurs at p.

The $\vee$ is a join operation and $\wedge$ is a meet operation which are compatible with the subsumption order defined in Definition 13. □

Examples of join and meet operations on augmented ψ-terms are shown in Figure 3. The join and meet operations on augmented ψ-types correspond to the generalization and unification operations on TFSs.

Aït-Kaci defines an ϵ-type as a maximal set of ψ-types. It is also possible to define an augmented ϵ-type as a maximal set of augmented ψ-types in the same manner, making disjunctive and negative descriptions possible.

3.2 Type Symbol Lattice Extension to Include Complement Type Symbols

Treating a negative description of a given type symbol, say $\mathbf{a}$, requires a type symbol $\mathbf{b}$ such that $\mathbf{b}$ has only information that unification of it with $\mathbf{a}$ yields inconsistency, or such that $\mathbf{a} \vee_{\mathcal{T}} \mathbf{b} = \top$ and $\mathbf{a} \wedge_{\mathcal{T}} \mathbf{b} = \bot$. Such a symbol is called a complement type symbol of $\mathbf{a}$ and written as $\mathbf{a}'$. If a given type symbol lattice $\langle \mathcal{T}; \leq_{\mathcal{T}} \rangle$ is a Boolean lattice, that is, a complemented[3] distributive lattice, we do not need to do anything. Otherwise, we must extend the lattice to include the complements of the type symbols contained in the given lattice.

For a finite type symbol lattice $\mathcal{T}$, for example, a Boolean lattice $\mathcal{T}'$ can be constructed as follows. Let $\mathcal{A} := \{\mathbf{a}_1, \ldots, \mathbf{a}_N\}$ be the set of atoms of $\mathcal{T}$, that is, type symbols which cover $\bot$.[4] If there are non-atomic type symbols which cover only one symbol, for each such symbol $\mathbf{a}$, a new atom is added

[3] A lattice is called complemented if its all elements have complements.[3]

[4] $\mathbf{a}$ is said to cover $\mathbf{b}$ if $\mathbf{b} <_{\mathcal{T}} \mathbf{a}$ and $\mathbf{b} \leq_{\mathcal{T}} \mathbf{c} <_{\mathcal{T}} \mathbf{a}$ implies $\mathbf{c} = \mathbf{b}$.

node structure	
tsymbol:	⟨a type symbol⟩
arcs:	⟨a set of *arc* structures⟩
ifeatures:	⟨a set of feature symbols⟩
dnodes:	⟨a set of *node* structures⟩
forward:	⟨a *node* structure⟩ | *NIL*
arc structure	
feature:	⟨a feature symbol⟩
value:	⟨a *node* structure⟩

Figure 4: Data Structures

```
Function Unify(node1, node2)
  begin
    node1 := Dereference(node1);
    node2 := Dereference(node2);
    if node1 = node2 then
      return(node1);
    node1.forward := node2;
    node2.tsymbol := node1.tsymbol $\wedge_{\mathcal{T}}$ node2.tsymbol;
    if node2.tsymbol = $\bot$ then
      return($\bot$)
    node2.ifeatures := node1.ifeatures $\cup$ node2.ifeatures;
    if node2.ifeatures $\cap$
        {arc.feature | arc $\in$ node1.arcs $\cup$ node2.arcs}
          $\neq \emptyset$ then
      return($\bot$);
    node2.dnodes := node1.dnodes $\cup$ node2.dnodes;
    if {node1, node2} $\cap$ node2.dnodes $\neq \emptyset$ then
      return($\bot$);
    arcpairs := Shared_Arc_Pairs(node1, node2);
    for (arc1, arc2) in arcpairs do
    begin
      value := Unify(arc1.value, arc2.value);
      if value = $\bot$ then
        return($\bot$);
    end;
  arcs := Complement_Arcs(node1, node2);
  node2.arcs := arcs $\cup$ node2.arcs;
  return(node2);
end
```

Figure 5: A Destructive Graph Unification Function

so that a covers an additional type symbol. The extended lattice $\mathcal{T}'$ is the set of subsets of $\mathcal{A}$ with set inclusion ordering. An element $\{\mathbf{a}_i\}_{i\in I} \in \mathcal{T}'$ denotes $\bigcup_{i\in I}[\mathbf{a}_i]$. The join and meet operations on $\mathcal{T}'$ are the set-union and set-intersection operations, respectively. The complement of an element $\{\mathbf{a}_i\}_{i\in I}$ in $\mathcal{T}'$ is the set-complement of it with respect to $\mathcal{A}$, that is, $\{\mathbf{a} \in \mathcal{A} \mid \mathbf{a} \notin \{\mathbf{a}_i\}_{i\in I}\}$.

4 Implementation of Augmented TFS Unification

The unification operation for augmented ψ-terms or augmented TFSs has been implemented using graph unification techniques. A term structure is represented as a directed graph by assigning a graph node to each κ-class as in Figure 2. The unification operation for such DGs corresponds to a graph merging operation. This takes two DGs and merges κ-classes of the same feature-address into a κ-class.

In a destructive graph unification method, which is very simple, such a graph is represented by the data structures in Figure 4. A *node* structure consists of five fields: *tsymbol* for a type symbol, *arcs* for a set of feature-value pairs, *ifeatures* for a set of inhibited features, *dnodes* for a set of disagreement nodes—i.e., disagreement κ-classes, and *forward*. The field *forward* is used for the Union-Find algorithm[9] to calculate unions of κ-classes in the same manner as Huet's algorithm[10]. By traversing two DGs' nodes with the same feature-address simultaneously, calculating the union of their κ-classes, and copying arcs, their unification can be calculated as in Figure 5.

The function *Unify* takes two input nodes and puts them in a κ-class by letting one input be the *forward* field values. The function then examines three kinds of inconsistency; namely, type inconsistency, feature inconsistency, and tag inconsistency. The function finally treats arcs in order to make the result graph right-congruent. For treating arcs, the function *Unify* assumes two functions, *Shared_Arc_Pairs* and *Complement_Arcs*. The function *Shared_Arc_Pairs* takes two nodes as its inputs and gives a set of arc pairs each consisting of both inputs' arcs with a shared feature. The function *Complement_Arcs* also takes two nodes and gives a set of arcs whose features exist in the first node but not in the second.

An inhibited feature function is implemented using the *ifeatures* field of nodes. When unification of two nodes results in a node with an arc with a feature in *ifeatures*, it yields $\bot$ because of feature inconsistency. A disagreement tag symbol function is implemented using *dnodes*. Unification of two nodes which have each other in their *dnodes* yields $\bot$ because of tag inconsistency. These computations require negligible additional computation.

To simplify the explanation, the destructive version of graph unification is used above. Other versions based on more efficient graph unification methods such as Wroblewski's and Kogure's method[23, 16] have also been developed. Furthermore, it is easy to modify other graph unification methods[21, 6] to allow augmented TFSs.

5 Conclusion

This paper has proposed an augmentation of feature structures (FSs) which introduces negative information into FSs in unification-based formalisms. Unification-based linguistic formalisms use FSs to describe linguistic objects and phenomena. Because linguistic information can be described compactly using disjunctive and negative descriptions, FSs and feature descriptions are required to treat such descriptions. In this paper, FSs have been augmented, using a promising method of formalization, Aït-Kaci's ψ-type, to allow three kinds of negative descriptions of them to be treated.

In a formalization of typed feature structures, negative descriptions can be decomposed into three kinds of negations: negations of type symbols, negations of feature existences, and negations of feature-address value agreements. It is shown that the second and third kinds can be treated by augmenting term structures to include structures representing such kinds of descriptions. Subsumption relations on augmented terms are defined. It is also shown that the first kind can be treated by extending type symbol lattices to include complement type symbols.

The proposed formalization can provide efficient algorithms for generalization and unification operations as well as treat primitive negations. The formalization can be integrated with logic-based frameworks such as [20] which can treat wider ranges of descriptions but which do not have such efficient algorithms for these operations. Logic-based frameworks can be used to obtain the data structures for this paper's formalization.

Unification algorithms for augmented terms or augmented TFSs have been developed using graph unification techniques. Unification programs based on these algorithms have been developed in Common Lisp.

The augmentation of TFSs makes linguistic descriptions compact and easy to understand. In an HPSG-based grammar, for example, non-emptiness of a *subcat* or *slash* feature value can be easily described by using feature-address value disagreement. Moreover, negative descriptions make debugging processes of grammatical descriptions easier.

Acknowledgments

This research was performed in the VCAT project of the Takeuchi Research Group in NTT Basic Research Laboratories. The author would like to thank Ikuo Takeuchi, Akira Shimazu, Shozo Naito, Masahito Kawamori, Mikio Nakano, and other colleagues of the group for their encouragement and thought-provoking discussions.

References

[1] Hassan Aït-Kaci. An algebraic semantics approach to the effective resolution of type equations. *Journal of Theoretical Computer Science*, 45:293–351, 1986.

[2] Hassan Aït-Kaci and Roger Nasr. Login: a logic programming language with built-in inheritance. *Journal of Logic Programming*, 3:185–215, 1986.

[3] Garrett Birkhoff. *Lattice Theory*. American Mathematical Society, Providence, Rhode Island, USA, 3rd edition, 1967.

[4] Bob Carpenter and Carl Pollard. Inclusion, disjointness and choice: the logic of linguistic classification. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 9–16, ACL, University of California, Berkeley, California, USA, 1991.

[5] Annuj Dawar and K. Vijay-Shanker. A three-valued interpretation of negation in feature structure descriptions. In *Proceedings of the 27th Annual Meeting of Association for Computational Linguistics*, pages 18–24, ACL, Vancouver, British Columbia, Canada, 1989.

[6] Martin Emele. Unification with lazy non-redundant copying. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 325–330, ACL, University of California, Berkeley, California, USA, 1991.

[7] Martin Emele and Rémi Zajac. *RETIF: A Rewriting System for Typed Feature Structures*. Technical Report TR-I-0071, ATR, Kyoto, Japan, 1989.

[8] Martin Emele and Rémi Zajac. Typed unification grammars. In *Proceedings of the 13th International Conference on Computational Linguistics, Vol. 3*, pages 293–298, 1990.

[9] J. E. Hopcroft and R. M. Karp. *An Algorithm for Testing the Equivalence of Finite Automata*. Technical Report TR-71-114, Dept. of Computer Science, Cornell University, Ithaca, New York, USA, 1971.

[10] Gérard Huet. *Résolution d'Equations dans des Langages d'Order 1, 2, ..., ω*. PhD thesis, Université de Paris VII, France, 1976.

[11] Lauri Karttunen. Features and values. In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 28–33, Stanford, California, USA, 1984.

[12] Robert T. Kasper. Unification and classification: an experiment in information-based parsing. In *Proceedings of the International Workshop on Parsing Technologies*, pages 1–7, Pittsburgh, Pennsylvania, USA, 1989.

[13] Robert T. Kasper and William C. Rounds. A logical semantics for feature structure. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, ACL, New York, New York, USA, 1986.

[14] Martin Kay. Parsing in functional unification grammar. In D. R. Dowty, editor, *Natural Language Parsing*, chapter 7, pages 251–278, Cambridge University Press, 1985.

[15] Kiyoshi Kogure. Parsing Japanese spoken sentences based on HPSG. In *Proceedings of the International Workshop on Parsing Technologies*, pages 132–141, Pittsburgh, Pennsylvania, USA, 1989.

[16] Kiyoshi Kogure. Strategic lazy incremental copy graph unification. In *Proceedings of the 13th International Conference on Computational Linguistics, Vol. 2*, pages 223–228, 1990.

[17] M. Drew Moshier and William C. Rounds. A logic for partially specified data structures. In *Proceedings of the 14th ACM Symposium on Principles of Programming Language*, pages 156–167, Munich, West Germany, 1987.

[18] Carl Pollard and Ivan Sag. *An Information-Based Syntax and Semantics—Volume 1: Fundamentals. CSLI Lecture Notes Number 13*, CSLI, 1987.

[19] William C. Rounds and Robert T. Kasper. A complete logical calculus for record structures representing linguistic information. In *Proceedings of Symposium on Logic in Computer Science*, IEEE Computer Society, 1986.

[20] Gert Smolka. *A Feature Logic with Subsorts*. Technical Report LILOG Report 33, IBM Deutschland, 7000 Stuttgart 80, West Germany, 1988.

[21] Hideto Tomabechi. Quasi-destructive graph unification. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 315–322, ACL, University of California, Berkeley, California, USA, 1991.

[22] Yoshihiro Ueda and Kiyoshi Kogure. Generation for dialogue translation using typed feature structure unification. In *Proceedings of the 13th International Conference on Computational Linguistics, Vol. 1*, pages 64–66, 1990.

[23] David A. Wroblewski. Nondestructive graph unification. In *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 582–587, AAAI, Seattle, Washington, USA, 1987.

[24] Rémi Zajac. A transfer model using a typed feature structure rewriting system with inheritance. In *Proceedings of the 27th Annual Meeting of Association for Computational Linguistics*, pages 1–6, ACL, Vancouver, British Columbia, Canada, 1989.