Towards a Comprehensive Framework for the Specification, Representation and Unification of Complex Feature Structures: the Lexical Objects Theory

José F. Quesada
CICA (Centro de Informática Científica de Andalucía), Sevilla
`josefran@cica.es`

Resumen

This paper describes the main characteristics of the *Lexical Objects Theory*: a framework that comprehends the formal, linguistic and computational aspects involved in the study of unification and features structures in Computational Linguistics. From a functional point of view, our proposal is based on a layered architecture which distinguish between three main levels. Specification level introduces new expressive tools: optionality and special values (null, complete and incomplete), and defines the inference rules that obtain the canonical form of any lexical object. Representation level permits three models: InRange, DataDefined and EndLess, and concentrates on computational techniques aimed to improve the management of lexical objects. Finally, Unification Level defines formally Low Level Logic-based, Weak and Constructive Unification strategies.

1 Introduction

The study of the formal properties of feature structures (FS henceforward) and unification has focused the attention of a lot of research works in Computational Linguistics (CL). These works can be organized in four main streams.

The first line has concentrated on the study of FS as a model for the *description of linguistic phenomena*. In this context, researchers have analyzed their expressive power and have proposed different extensions to the basic model: templates and lexical rules (Shieber, 1984), negation and disjunction (Karttunen, 1984), correference and non-local values (Kasper and Rounds, 1986), etc.

The second line is related to the use of typification strategies as part of FS–based formalisms, obtaining the so-called *typed (or sorted) feature structures* (Carpenter, 1992) and *order-sorted unification* (Meseguer et al., 1990).

Third, some authors have studied the problems associated with the *representation of feature structures*: denotational semantics of the model (Shieber, 1984; Pereira and Shieber, 1984), computationally efficient data structures (Karttunen and Kay, 1985; Pereira, 1985), techniques used for the storage and retrieval of very large FS–based knowledge bases (Quesada and Amores, 1995), and so on.

Finally, it is possible to delimitate a fourth stream of work concentrated on the design and implementation of *efficient unification algorithms*. Due to the critical role that unification plays in CL, this division is full of interesting and very famous proposal: binary trees (Karttunen and Kay, 1985), structure sharing (Pereira, 1985), non–destructive unification (Wroblewski, 1987), strategic lazy incremental unification (Kogure, 1990) or non–redundant lazy copy (Emele, 1991).

2 Lexical Objects Theory

This work presents a theoretical framework that takes into account the main result of the four research streams indicated. The model presented here has specially concentrated its attention on the logico-mathematical analysis of the formal properties of unification and FS. In this address, the main precedents we have considered are the works by Robert T. Kasper & William C. Rounds (Kasper and Rounds, 1986; Rounds and Kasper, 1986; Rounds, 1997), Stuart M. Shieber (Shieber, 1986) and Bob Carpenter (Carpenter, 1992).

The main contribution of our model, which justify a full *Lexical Objects Theory*, is to offer a uniform and completely integrated and formalized framework that accounts all the stages of the cycle of life of FS. Previous works have usually concentrated on partial aspects, and this is a source of problems:

- Inconsistency between the formal model that defines the properties of FS and their descriptions (Kasper and Rounds, 1986);
- Almost any study about FS incorporates a well-founded analysis of the problems generated during their representation. That is to say, the representation level isn't analysed, or, in the best case, it is considered as a direct translation of the logical model.
- The lack of a detailed analysis of the representation of FS and its link with the algorithms of unification increases the computational costs of the latter.

These problems arise as a consequence of very restrictive or partial formalization schemes. Our proposal aims to present a methodologically interdisciplinar framework that includes simultaneously computational, linguistic and formal requirements.

From a functional point of view, the study of lexical objects (LO henceforth) is divided in three main levels: Specification, Representation and Unification. In the next sections we describe their main characteristics.

3 Specification Level

3.1 New Expressive Tools: Optionality and Special Values

From the perspective of its informational domain, a unification–based grammatical formalism defines the notion of lexical object as a partial function λ from a set of attributes Σ to a set of values Υ: $\lambda : \Sigma^+ \cdots \rightarrow \Upsilon$ where $\Sigma^+ = \bigcup_{i\geq 1} \Sigma^i$.

This basic model has been enriched using different extensions, such as negation and disyunction of single and complex values. Our formalism for the specificacion of LO includes all these possibilities and also incorporates the following:

1. *Optionality*: `?`. This let us to indicate a default value for an attribute. There is an interesting linguistic motivation for this extension. With the definition of optionality we will incorporate a *weak unification mechanism*, that is, LO marked as optional will never fail when unifying.[1]

   $$\texttt{(gen:?masc)} \sqcup \texttt{(num:sing)} = \texttt{(gen:?masc,num:sing)}$$
   $$\texttt{(gen:?masc)} \sqcup \texttt{(gen:fem)} = \texttt{(gen:fem)}$$

2. *Lists*. The formalism includes lists as a basic data type.

3. *Special Values*.

   (a) *Null Values*: `#` and `(#)`. These let us to assign no value to an attribute. In conjunction with the multiple inheritance mechanism, these values allow us to delete (as an exception) an inherited value. Null values unify with any other:

   $$\texttt{(gen:\#)} \sqcup \texttt{(gen:fem)} = \texttt{(gen:fem)}$$
   $$\texttt{(agr:(\#))} \sqcup \texttt{(agr:(num:sing))} = \texttt{(agr:(num:sing))}$$

   (b) *Incomplete Values*: `_` and `(_)`. These values indicate that an attribute has no value, but it requires some value to be correct. Of course, these values unify with any other. Also, the *Unification Level* contains a function able to determine as an error of unification a LO in which there remain incomplete values:

   $$\texttt{(subj:(\_))} \sqcup \texttt{(subj:(head:Mary))} = \texttt{(subj:(head:Mary))} = \psi_1$$
   $$\texttt{(subj:(\_))} \sqcup \texttt{(obj:(head:Mary))} = \texttt{(subj:(\_),obj:(head:Mary))} = \psi_2$$
   $$\texttt{IncompleteValuesAnalysis}(\psi_1) = Success$$
   $$\texttt{IncompleteValuesAnalysis}(\psi_2) = Error$$

   (c) *Complete Values*. `+` and `(+)`. Its unification with any other value will always fail.

   $$\texttt{(head:+)} \sqcup \texttt{(head:Mary)} = Error$$
   $$\texttt{(subj:(+))} \sqcup \texttt{(subj:(head:Mary))} = Error$$

[1] $\sqcup$ represents the unification operation.

3.2 Definition of Lexical Object

1. *Atom*: Any identifier[2] is a LO. Special values `#`, `_` and `+` are also LO, and specifically atoms.
2. *List*: If Δ is a LO, then $[\Delta]$ is also a LO.
3. *Attribute–Value Pair*: If f is an identifier and Δ is a LO, then $f : \Delta$ is a LO. Special values `(#)`, `(_)` and `(+)` are also LO, and specifically pairs.
4. *Negation*: If Δ is a LO, then $-\Delta$ is a LO.
5. *Optionality*: If Δ is a LO, then $?\Delta$ is a LO.
6. *Conjunction*: If Δ and Ω are two LO, then Δ, Ω is a LO. We will use also the symbol $\wedge$ to represent conjunctions.
7. *Disjunction*: If Δ and Ω are two LO, then $\Delta; \Omega$ is a LO. We will use also the symbol $\vee$ to represent disjunctions.

3.2.1 Semantic Types and Constraints

Rules 1 to 3 introduce different semantic types of LO:

1. Atoms (SEM_{ATOM}). We will refer to them with lower case letters: $a, b, \ldots$
2. Lists (SEM_{LIST}): $[a], [b], \ldots$
3. Attribute–Value Pair (SEM_{PAIR}). We will use lower case Greek letters: $\alpha, \beta, \ldots$ to refer to LO of type pair. The notation $\alpha_{f \to \Delta}$ indicates that α is a pair, where the attribute is f and the value is Δ.

Over the previous definition of LO we will impose the following semantic constraints: lists will be generated only from SEM_{ATOM} objects, and both objects in a conjunction or disjunction will have the same semantic type.

We will refer to any special atomic value by means of a_S and to any special pair value by α_S. Capital Greek letters: $\Delta, \Omega, \ldots$, will refer to LO of any type.

3.2.2 Parentheses and Labels.

The formalism includes the use of parentheses as a neutral operator that may improve the legibility of LO and change the precedence[3] and order of application[4] of operators:

[2]An identifier is a sequence of alphanumeric symbols. When needed, we permit the use of non alphanumeric symbols using standard techniques: quotes ("), inverted commas (') and escape sequences.

[3]By default, from right to left.

[4]By default negation and optionality, lists, pairs, disjunction, conjunction.

- *Parenthesis*: If Δ is a LO, then (Δ) is a LO too.

Rule 3 permits the creation of a LO from an attribute f and a LO Δ. The specification language permits the addition of a label to the attribute. The label is an identifier between two ˜ symbols:

$$feature\tilde{\ }label\tilde{\ } : \Delta$$

Labels allow for the introduction of correference constraints. Besides, labels allow for the extension of the basic scheme of LO specification with special constraints introduced by certain grammatical formalisms and theories.

3.3 Inference Rules

3.3.1 Stage 1: Negation and Optionality.

At the end of this stage, the negation and optionality operators will be applied on atoms exclusively, except optionality that may be applied over negation.

$$\begin{array}{rcll}
-\texttt{\#} & \vdash & \texttt{\#} & \text{(S.1.1)} \\
-\texttt{+} & \vdash & \texttt{\_} & \text{(S.1.2)} \\
-\texttt{\_} & \vdash & \texttt{+} & \text{(S.1.3)} \\
-(\texttt{\#}) & \vdash & (\texttt{\#}) & \text{(S.1.4)} \\
-(\texttt{+}) & \vdash & (\texttt{\_}) & \text{(S.1.5)} \\
-(\texttt{\_}) & \vdash & (\texttt{+}) & \text{(S.1.6)} \\
-[a] & \vdash & [-a] & \text{(S.1.7)} \\
-\alpha_{f\rightarrow\Delta} & \vdash & \alpha_{f\rightarrow-\Delta} & \text{(S.1.8)} \\
--\Delta & \vdash & \Delta & \text{(S.1.9)} \\
-?\Delta & \vdash & ?-\Delta & \text{(S.1.10)} \\
-(\Delta\wedge\Omega) & \vdash & (-\Delta)\vee(-\Omega) & \text{(S.1.11)} \\
-(\Delta\vee\Omega) & \vdash & (-\Delta)\wedge(-\Omega) & \text{(S.1.12)}
\end{array}$$

$$\begin{array}{rcll}
?a_T & \vdash & a_T & \text{(S.1.13)} \\
?\alpha_T & \vdash & \alpha_T & \text{(S.1.14)} \\
?[a] & \vdash & [?a] & \text{(S.1.15)} \\
?\alpha_{f\rightarrow\Delta} & \vdash & \alpha_{f\rightarrow?\Delta} & \text{(S.1.16)} \\
??\Delta & \vdash & ?\Delta & \text{(S.1.17)} \\
?(\Delta\wedge\Omega) & \vdash & (?\Delta)\wedge(?\Omega) & \text{(S.1.18)} \\
?(\Delta\vee\Omega) & \vdash & (?\Delta)\vee(?\Omega) & \text{(S.1.19)}
\end{array}$$

3.3.2 Stage 2: Disjunctive Normal Form.

At this stage we aim to get the disjunctive normal form (DNF) of SEM_{PAIR} LO.

It is worth to note that the specification level doesn't fulfil the commutative property (due to the inheritance mechanism). This is the motivation of formula (S.2.3).

$$\alpha_{f \to (\delta \vee \zeta)} \vdash \beta_{f \to \delta} \vee \gamma_{f \to \zeta} \quad \text{(S.2.1)}$$

$$(\alpha \vee \beta) \wedge \gamma \vdash (\alpha \wedge \gamma) \vee (\beta \wedge \gamma) \quad \text{(S.2.2)}$$

$$\alpha \wedge (\beta \vee \gamma) \vdash (\alpha \wedge \beta) \vee (\alpha \wedge \gamma) \quad \text{(S.2.3)}$$

3.3.3 Stage 3: Fusion, Simplification and Smoothing.

The goal of this stage is to merge the multiple definitions of the same attribute. To simplify this operation and obtain a very-well defined logical representation this stage includes a set of smoothing (simplification) rules.[5]

$$\alpha_{f \to a \wedge b} \vdash \alpha_{f \to b} \quad \text{(S.3.1)}$$

$$[a] \wedge [b] \vdash [b] \quad \text{(S.3.2)}$$

$$[a] \vee [b] \vdash [a \vee b] \quad \text{(S.3.3)}$$

$$\alpha_S \wedge \Delta \vdash \Delta \quad \text{(S.3.4)}$$

$$\alpha \wedge (\beta_S \wedge \Delta) \vdash \Delta \quad \text{(S.3.5)}$$

$$\alpha \wedge \beta_S \vdash \beta_S \quad \text{(S.3.6)}$$

$$\alpha_{f \to \Delta} \wedge \beta_{f \to \Omega} \vdash \gamma_{f \to (\Delta \wedge \Omega)} \quad \text{(S.3.7)}$$

$$\alpha_{f \to \Delta} \wedge (\beta_{f \to \Omega} \wedge \Phi) \vdash \delta_{f \to (\Delta \wedge \Omega)} \wedge \Phi \quad \text{(S.3.8)}$$

$$(\alpha \wedge \beta) \wedge \Delta \vdash \alpha \wedge (\beta \wedge \Delta) \quad \text{(S.3.9)}$$

$$(\alpha \vee \beta) \vee \Delta \vdash \alpha \vee (\beta \vee \Delta) \quad \text{(S.3.10)}$$

$$\alpha_{f \to \Delta} \wedge \beta_{g \to \Omega} \vdash \beta_{g \to \Omega} \wedge \alpha_{f \to \Delta} \ \texttt{[iff g << f]} \quad \text{(S.3.11)}$$

$$\alpha_{f \to \Delta} \wedge (\beta_{g \to \Omega} \wedge \Phi) \vdash \beta_{g \to \Omega} \wedge (\alpha_{f \to \Delta} \wedge \Phi) \ \texttt{[iff g << f]} \quad \text{(S.3.12)}$$

4 Representation Level

This level includes a typification layer, which main original contribution is the distinction between three main domain typification models:

1. *InRange*. The possible values of an attribute are previously specified.
2. *DataDefined*. The possible values of an attribute are also a very-well defined set, but the user doesn't define that set using a pre-declaration like with the InRange model. In this case, the system has to obtain the set of values from the LOs where the attribute appears.

[5]By $g << f$ we mean that attribute g is alphabetically smaller than attribute f.

3. $EndLess$. From a theoretical point of view, the domain of values is endless, because always it is possible to incorporate new values.

From an operational perspective, DataDefined objects may be thought as an user-friendly technique based on a post-typification mechanism. At the end of the analysis process, DataDefined objects will be considered as InRange.

As a result of the typification layer, the system will know for each attribute its semantic type (Atom, List or Pair) and its typification model (InRange or EndLess).

Next, the Representation Level concentrated on the relations between the logical description obtained from the previous level (Specification) and their use by the unification algorithms in the next level (Unification).

4.1 Representation of InRange Atomic Lexical Objects

Let us consider that the domain of an InRange Atomic LO (for instance, a) has n different values:

$$D_a = \{v_1, v_2, \ldots, v_n\}$$

We can represent each value of this domain as a sequence of n bits:

$$\begin{aligned} v_1 &= 100\ldots0 \\ v_2 &= 010\ldots0 \\ &\vdots \\ v_n &= 000\ldots1 \end{aligned}$$

The negation will be represented as the one's complement:

$$-v_1 = 011\ldots1$$

The disjunction of two values is their bitwise inclusive OR:

$$v_1 \vee v_2 = 110\ldots0$$

To store optionality we will use an additional bit:

$$\begin{aligned} v_1 &= 100\ldots0 : 0 \\ ?v_1 &= 100\ldots0 : 1 \\ ?-v_1 &= 011\ldots1 : 1 \end{aligned}$$

InRange special atomic LO will be stored as [6]:

$$\begin{aligned} \texttt{\#} &= 111\ldots1 : 0 = FULL : 0 \\ \texttt{\_} &= 000\ldots0 : 1 = NULL : 1 \\ \texttt{+} &= 000\ldots0 : 0 = NULL : 0 \end{aligned}$$

[6]We will use $NULL$ or $FULL$ to indicate that a field has 0 or 1, respectively, in all its bits.

4.2 Representation of EndLess Atomic Lexical Objects

Our goal is to store only once every different value. This way, two objects are in fact the same if they have been stored in the same memory address.

Therefore, the representation of an object of this type will require a memory address (a pointer[7]) and two additional bits: the flags of optionality and negation. To store disjunction, we will use a pointer to the same data structure, obtaining a recursive list of elements.

Let be the domain of an EndLess Atomic LO (for instance, e), the following set:

$$D_e = \{w_1, w_2, \ldots\}$$

We will use the next representation strategies [8]:

$$\begin{aligned}
w_1 &= \boxed{w_1^*} : 0_{optn} : 0_{negt} : \boxed{\text{NULL}} \\
?w_1 &= \boxed{w_1^*} : 1_{optn} : 0_{negt} : \boxed{\text{NULL}} \\
-w_1 &= \boxed{w_1^*} : 0_{optn} : 1_{negt} : \boxed{\text{NULL}} \\
?-w_1 &= \boxed{w_1^*} : 1_{optn} : 1_{negt} : \boxed{\text{NULL}} \\
w_1 \vee w_2 &= \boxed{w_1^*} : 0 : 0 : \boxed{} \longrightarrow \boxed{w_2^*} : 0 : 0 : \boxed{\text{NULL}}
\end{aligned}$$

EndLess special atomic LO will be stored as:

$$\begin{aligned}
\texttt{\#} &= \boxed{FULL} : 0 : 0 : \boxed{\text{NULL}} \\
\texttt{\_} &= \boxed{NULL} : 1 : 0 : \boxed{\text{NULL}} \\
\texttt{+} &= \boxed{NULL} : 0 : 0 : \boxed{\text{NULL}}
\end{aligned}$$

4.3 Representation of Lists

Lists will be represented as strings of the corresponding types of atoms.

4.4 Representation of Pairs

To simplify the management of this kind of structures, the Representation and Unification Levels consider all the pairs as being of EndLess type. Also, each component of the DNF will be stored separately. The representation scheme in this case is based on three data structures:

1. *PairRoot*. This structure contains the following fields:
   (a) *roottype*. Normal (*pairlink*) or derreferenced (*derreference*).
   (b) *derreference*. A *PairRoot* pointer.
   (c) *pairlink*. A *PairLink* pointer.

[7] $FULL$ and $NULL$ are not allowed to be correct memory positions.

[8] $\boxed{w_1^*}$ represents the memory address where w_1 has been stored.

   (d) *postcopy*. An integer flag.

2. *PairLink*.
   (a) *pair*. A *PairVal* pointer.
   (b) *next*. A *PairLink* pointer.

3. *PairVal*.
   (a) *attribute*.
   (b) *strsharing*. An integer flag: structure sharing.
   (c) *valtype*. Controls the type of information pointed by *value*.
   (d) *value*. A pointer to a LO; it may be an Atom (InRange or EndLess), a List or a Pair (*PairRoot*).

We will use the following graphics for their representation:

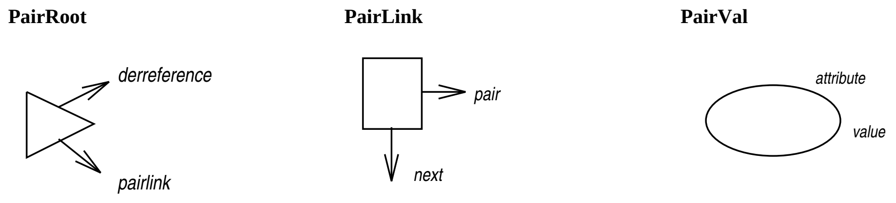

Fig. 1.

5 Unification Level

5.1 Unification of Atoms (A): Low-Level Logic Unification and Weak Unification

5.1.1 Unification of InRange Atomic (IA) Lexical Objects.

The next three unification rules control this case distinguishing between double, single and no optionality.[9]

$$a:1 \;\sqcup_{IA}\; b:1 \;\vdash\; (a \;\texttt{B-OR}\; b):1 \tag{U.1}$$

$$a:p \;\sqcup_{IA}\; b:(1-p) \;\vdash\; \begin{cases} a:0 & \texttt{if } (p \texttt{ == } 0), \\ b:0 & \texttt{otherwise.} \end{cases} \tag{U.2}$$

$$a:0 \;\sqcup_{IA}\; b:0 \;\vdash\; (a \;\texttt{B-AND}\; b):0 \tag{U.3}$$

[9]**B-OR** and **B-AND** are the bitwise OR and AND operations, respectively. In the next sections, m, n and p are binary variables.

5.1.2 Unification of EndLess Atomic (EA) Lexical Objects.

In this case, we have to control the flags of negation and optionality, as well as disjunctions:

$$a^* : 0 : m \;\sqcup_{EA}\; b^* : 0 : m \;\vdash\; \begin{cases} a^* : 0 : m & \texttt{if } (a^* \texttt{ == } b^*), \\ NULL : 0 : 0 & \texttt{otherwise.} \end{cases} \tag{U.4}$$

$$a^* : 0 : m \;\sqcup_{EA}\; b^* : 0 : (1-m) \;\vdash\; \begin{cases} a^* : 0 : 0 & \texttt{if } (m \texttt{ == } 0) \texttt{ and } (a^* \texttt{ != } b^*), \\ b^* : 0 : 0 & \texttt{if } (m \texttt{ == } 1) \texttt{ and } (a^* \texttt{ != } b^*), \\ NULL : 0 : 0 & \texttt{otherwise.} \end{cases} \tag{U.5}$$

$$a^* : p : m \;\sqcup_{EA}\; b^* : (1-p) : n \;\vdash\; \begin{cases} a^* : p : m & \texttt{if } (p \texttt{ == } 0) \\ b^* : (1-p) : n & \texttt{otherwise.} \end{cases} \tag{U.6}$$

$$a^* : 1 : m \;\sqcup_{EA}\; b^* : 1 : n \;\vdash\; \begin{cases} a^* : 1 : m & \texttt{if } \begin{array}{l} (a^* \texttt{ == } b^*) \texttt{ and} \\ (m \texttt{ == } n), \end{array} \\ a^* : 1 : m \vee b^* : 1 : n & \texttt{otherwise.} \end{cases} \tag{U.7}$$

$$(\Delta \vee \Omega) \;\sqcup_{EA}\; \Phi \;\vdash\; \nabla((\Delta \;\sqcup_{EA}\; \Phi) \vee (\Omega \;\sqcup_{EA}\; \Phi)) \tag{U.8}$$

$$\Delta \;\sqcup_{EA}\; (\Omega \vee \Phi) \;\vdash\; \nabla((\Delta \;\sqcup_{EA}\; \Omega) \vee (\Delta \;\sqcup_{EA}\; \Phi)) \tag{U.9}$$

The operator ∇ eliminates duplied components in conjunctions and disjunctios of EndLess Atomic objects and duplied components in conjunctions of InRange Atomic objects:

$$\nabla(\Delta \vee a^* : p : m \vee \Omega \vee a^* : p : m \vee \Psi) \;\vdash\; \Delta \vee a^* : p : m \vee \Omega \vee \Psi \tag{U.10}$$

$$\nabla(\Delta \wedge a^* : p : m \wedge \Omega \wedge a^* : p : m \wedge \Psi) \;\vdash\; \Delta \wedge a^* : p : m \wedge \Omega \wedge \Psi \tag{U.11}$$

$$\nabla(\Delta \wedge a : p \wedge \Omega \wedge a : p \wedge \Psi) \;\vdash\; \Delta \wedge a : p \wedge \Omega \wedge \Psi \tag{U.12}$$

5.2 Unification of Lists (L)

List Unification is defined using the previous definitions of Atomic Unification join with the next rules:

$$[\Delta] \;\sqcup_L\; [\Omega] \;\vdash\; [\Delta \;\sqcup_A\; \Omega] \tag{U.13}$$

$$a \wedge b \;\sqcup_A\; c \;\vdash\; \nabla((a \;\sqcup_A\; c) \wedge (b \;\sqcup_A\; c)) \tag{U.14}$$

$$a \;\sqcup_A\; b \wedge c \;\vdash\; \nabla((a \;\sqcup_A\; b) \wedge (a \;\sqcup_A\; c)) \tag{U.15}$$

5.3 Unification of Pairs (P): Constructive Unification

The formal properties of Pair Unification are summarized in the next rule:

$$(\alpha_{f\rightarrow\Delta} \wedge \Psi^{\star}) \ \sqcup_P \ (\beta_{g\rightarrow\Omega} \wedge \Phi^{\star}) \ \vdash \begin{cases} \alpha_{f\rightarrow\Delta} \wedge (\Psi^{\star} \ \sqcup_P \ (\beta_{g\rightarrow\Omega} \wedge \Phi^{\star})) & \texttt{if f << g,} \\ \beta_{g\rightarrow\Omega} \wedge ((\alpha_{f\rightarrow\Delta} \wedge \Psi^{\star}) \ \sqcup_P \ \Phi^{\star}) & \texttt{if g << f,} \\ \gamma_{f\rightarrow(\Delta \ \sqcup \ \Omega)} \wedge (\Psi^{\star} \ \sqcup_P \ \Phi^{\star}) & \texttt{if f == g.} \end{cases} \tag{U.16}$$

where $\Psi^{\star}$ and $\Phi^{\star}$ may be normal or empty pairs objects. If $\Psi^{\star}$ is empty ($\Psi^{\star} = \emptyset$), then $\alpha \wedge \Psi^{\star} = \alpha$. The empty set is a neutral element for unification:

$$\Delta \ \sqcup_P \ \emptyset \ \vdash \ \Delta \tag{U.17}$$

$$\emptyset \ \sqcup_P \ \Delta \ \vdash \ \Delta \tag{U.18}$$

From a computational perspective we introduce Constructive Unification. This algorithm mixes in a novel way different well-known strategies:

- Structure-sharing.
- Reversible Unification. The algorithm includes two working models: reversible unification with disunification and non-reversible unification with post-copy (with in its turn avoids pre-, over- and redundant copying).
- Strategic Unification. The algorithm uses ordered LO as input, and by the manipulation of the $PairLink$ data structures is able to obtain a ordered LO too. If the ordering criteria isn't the alphabetic order of attributes, but the descendent probability of unification fail previously obtained in a training process, we will easily obtain a strategic algorithm.
- Typed Unification. In any case, the algorithm eliminates the process of searching attributes, a expensive task both from the computational and the complexity perspectives.

As a single ilustration, Fig. 2 contains the representation of the LO $F1$ and $F2$. Dark lines in this figure represent the modifications introduced by the unification algorithm.

$F1$ = `(a:p,c:q,d:r,e:s)`
$F2$ = `(a:p,b:s,c:q)`

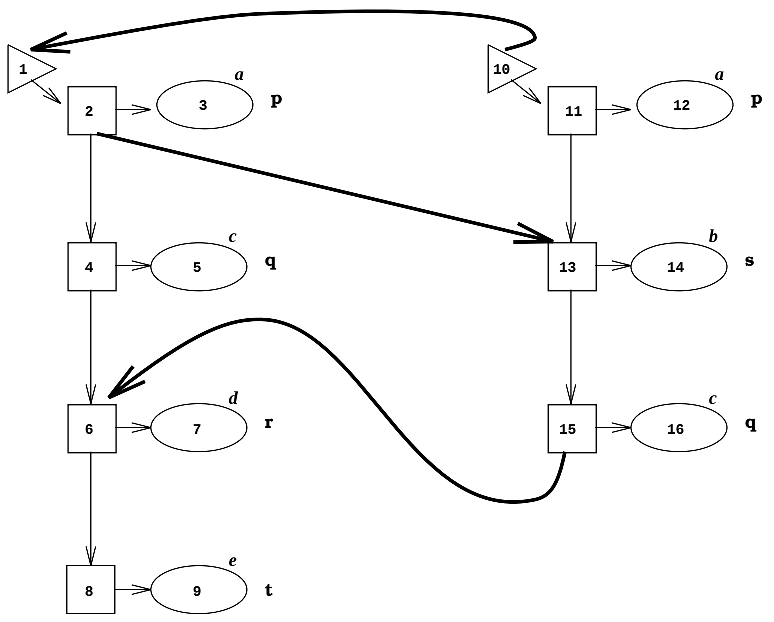

Fig. 2.: Constructive Unification of $F1$ and $F2$

As part of the unification algorithm we will have obtained a list of the

```
<data-structure, field, old-value>
```

changed (disunification information), that for Fig. 2 is

```
{<10,derreference,NULL>,<2,next,4>,<15,next,NULL>}
```

Now we can select between:

1. Post-copy the unified object. In this case, we have to copy only the $PairRoot$ (1) and $PairLink$ (2, 13, 15, 6 and 8) structures, marking the $ParVal$ (3, 14, 16, 7 and 9) as structure-shared. Once post-copied, we will apply the disunification algorithm obtaining the input structures.
2. Store the disunification information in the $PairRoot$ structure and continue. Later, we can apply or not the disunification algorithm.

When post-copying structures, we will use the $postcopy$ mark to avoid redundant copying.

6 Bibliography

Bob Carpenter. 1992. *The logic of typed feature structures*. Cambridge Tracts in Theoretical Computer Science. Cambridge: Cambridge University Press.

Martin C. Emele. 1991. Unification with lazy non-redundant copying. In *29th Annual Meeting of the Association for Computational Linguistics*, pages 323–330. Association for Computational Linguistics.

Lauri Karttunen. 1984. Features and Values. In *Proceedings of the Tenth International Conference on Computational Linguistics COLING–84.* Stanford, California. Also in (Shieber et al., 1986), Part I, 17–36.

Lauri Karttunen and Martin Kay. 1985. Structure sharing with binary trees. In *23rd Annual Meeting of the Association for Computational Linguistics,* pages 133–136. Also in (Shieber et al., 1986), Part II, 5–16.

Robert Kasper and William C. Rounds. 1986. A Logical Semantics for Feature Structures. In *24th Annual Meeting of the Association for Computational Linguistics,* pages 257–266.

Claude Kirchner. ed. 1990. *Unification*. San Diego, California: Academic Press Inc.

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

José Meseger, Joseph A. Goguen and Gert Smolka. 1990. Order–Sorted Unification. In (Kirchner, 1990), pages 457–487.

Fernando C. N. Pereira and Stuart M. Shieber. 1984. The Semantics of Grammar Formalisms seen as Computer Languages. In *Proceedings of the Tenth International Conference on Computational Linguistics.* Stanford, California. Also in (Shieber et al., 1986), Part I, 37–58.

Fernando C. N. Pereira. 1985. A Structure–Sharing Representation for Unification–Based Grammar Formalisms. In *23rd Annual Meeting of the Association for Computational Linguistics,* pages 137–144. Also in (Shieber et al., 1986), Part II, 17–35.

José F. Quesada and J. Gabriel Amores. 1995. A Computational Model for the Efficient Retrieval of Very Large Structure–Based Knowledge Bases. In *Proceedings of the Knowledge Representation, Use and Storage for Efficiency (KRUSE'95) Symposium,* pages 86–96. Santa Cruz, California.

William C. Rounds and Robert Kasper. 1986. A complete logical calculus for record structures representing linguistic information. In *Proceedings of the 1st Symposium on Logic in Computer Science.*

William C. Rounds. 1997. Feature logics. In J. van Benthem and A. ter Meulen. (eds.) 1997. *Handbook of Logic and Language*. North–Holland.

Stuart M. Shieber. 1984. The Design of a Computer Language for Linguistic Information. In *Proceedings fo the Tenth International Conference on Computational Linguistics.* Stanford, California. Also in (Shieber et al., 1986), Part I, 4–16.

Stuart M. Shieber. 1986. *An Introduction to Unification–based Approaches to Grammar.* CSLI Lecture Notes 4. Stanford, California: Center for the Study of Language and Information.

Stuart M. Shieber, Fernando C. N. Pereira, Lauri Karttunen and Martin Kay. (eds.) 1986. *A Compilation of Papers on Unification–Based Grammar Formalisms. Parts I and II.* Report No. CSLI-86-48. Stanford, California: Center for the Study of Language and Information.

Gert Smolka and Hassan Aït–Kaci. 1990. Inheritance Hierarchies: Semantics and Unification. In (Kirchner, 1990), pages 489–516.

David Wroblewski. 1987. Non–destructive Graph Unification. In *Proceedings of the 6th National Conference on Artificial Intelligence, AAAI*, pages 582–587, Seatle, Washington.