Computer Science Journal of Moldova, vol.**8**, no.3(24), 2000

A set-theoretic approach to linguistic feature structures and unification algorithms (II)

N. Curteanu P.-G. Holban

Abstract

The paper proposes formal inductive definitions for linguistic feature structures (FSs) taking values within a class of value types or sorts: single, disjunctive, (ordered) lists, multisets (or bags), **po**-multisets (multisets embedded into a partially ordered set), and indexed (re-entrance) values. The linguistic realization (semantics) of the considered sorts is proposed. The FSs having these multi-sort values are organized as (rooted) directed acyclic graphs. The concrete model of the FSs we had in mind for our set-theoretic definitions are the FSs used within the well-known HPSG linguistic theory. Set-theoretic general definitions for the proposed multi-sort FSs are defined. These constructive definitions start from atomic values and build recurrent multi-sorted values and structures, providing naturally a fixed-point semantics of the obtained FSs as a counterpart to the large class of logical semantics models on FSs. The linguistic unification algorithm based on tableau-subsumption is outlined. The Prolog code of the unification algorithm is provided and results of running it on some of the main multi-sort FSs is enclosed in the appendices. We consider the proposed formal approach to FSs definitions and unification as necessary steps to set-theoretical implementations of natural language processing systems.

Let us specify from the beginning that the present article represents the direct continuation (part II) of the paper with the same title appeared in the previous number of this journal [34, pp. 116–149]. Thus, all the relation, definition, remark, etc. numbers (having the first digit less than 5) refer to the part I of this paper, which the present part II makes a single unit with.

5 Indexed (sharing) multi-sort

Another extension of the FS concept is offered by the very useful possibility to share feature values in distinct structures, obtaining *indexed* FSs (IFSs). IFSs have as representation support the structure of a directed graph (DG) or directed acyclic graph (DAG), the indexing points in the IFSs being actually the nodes that share the same information for the arcs with distinct labels entering these same nodes. IFSs are formally defined in what follows. Examples of IFSs are in (**??**), (**??**), (**??**), (**??**) and Example **??**. We first introduce the notion of IFS on DGs and then constrain DGs not to include cycles, obtaining DAGs.

In order to define indexed FSs (see (S6) in section **??**) there is necessary to extend the five sorts considered in the definition **??** and relations (**??**–**??**) with the *indexed multi-sort* expressions belonging to the following *indexed* sort.

$$\begin{aligned} Sort^6 &= \{(v, \mathrm{m}), (\&, \mathrm{m}) | v \in V, \mathrm{m} \in \mathbf{N}\} \\ &= (V \cup \{\&\}) \times \mathbf{N}(, \text{ i.e. with } \& \notin V) \end{aligned} \tag{28}$$

The symbol "&" is used to denote the presence of the *sharing property* only for those nodes in the FSs having this quality, while "m" provides the value of *the node label*, also called *feature index*. *The sharing nodes* within a FS receive the same value of the node labels.

Definition 5.1 *The set IW of* indexed multi-sort *expressions defined on V for the sorts* $Sort^i$, $i = 1 \div 6$, *denoted* $IW = ims(V)$, *is introduced as being* the closure to sort composition *for the sorts considered in (**??**) together with the indexed sort in (28), i.e.* $V \subset IW$ *and* $\forall w_1, w_2, \ldots, w_n \in IW$, *then*

$$\begin{aligned} {}^{\vee}\{w_1, w_2, \ldots, w_n\}^{\vee} &\in IW, \\ \langle w_1, w_2, \ldots, w_n \rangle &\in IW, \\ {}^{+}\{w_1, w_2, \ldots, w_n\}^{+} &\in IW, \\ {}^{+\mathrm{p}}\{w_1, w_2, \ldots, w_n\}^{+\mathrm{p}} &\in IW, \end{aligned}$$

and $(w_1, \mathrm{m}) \in IW$, *or* $(\&, \mathrm{m}) \in IW$, *with* $m \in \mathbf{N}\backslash\{0\}$ *and "&" as a special (empty) character such that* $\& \notin V$. *The elements in IW are called* indexed multi-sort *expressions of any depth on* V.

The set $IW = ims(V)$ of *indexed multi-sort* expressions defined on V can also be seen as the following union:

$$IW = IMSort^1 \cup IMSort^2 \cup IMSort^3 \cup \\ \cup IMSort^4 \cup IMSort^5 \cup IMSort^6, \tag{29}$$

where:

$$\begin{array}{l} IMSort^1 = V; \\ IMSort^2 = {}^{\vee}\{v_1, v_2, \ldots, v_n\}^{\vee} | v_i \in ims(V), i = 1 \div n\}; \\ IMSort^3 = \langle v_1, v_2, \ldots, v_n\rangle | v_i \in ims(V), i = 1 \div n\}; \\ IMSort^4 = {}^{+}\{v_1, v_2, \ldots, v_n\}^{+} | v_i \in ims(V), i = 1 \div n\}; \\ IMSort^5 = {}^{+\mathrm{p}}\{v_1, v_2, \ldots, v_n\}^{+\mathrm{p}} | v_i \in ims(V), i = 1 \div n\}; \\ IMSort^6 = \{(v, \mathrm{m}), (\&, rmm) | v \in ims(V), \mathrm{m} \in \mathbf{N}, \& \in V\}. \end{array} \tag{30}$$

To notice that $IMSort^i, i = 1 \div 6$, are *pairwise disjoint*.

The new constructed set IW constitutes now the 0-level set of values for the *indexed (multi-sort)* FSs (IFSs).

Definition 5.2 *Similarly to the definition* **??**, *let* G *be a set of* IFS attributes defined on W, $IW = ims(V)$ *as in definition 5.1, i.e., there exists* $T \subseteq Pow(IW)$ *and* $T \leftrightarrow G$. *Following the definition* **??**, *we have the one-to-one functions* $AtVal : G \to T$, *and* $ValAt : G \to T$, *with* $ValAt = AtVal^{-1}$.

With the notations in (29), (30) and definition 5.1, the notion of *well-definedness* of IFSs with values in the considered sorts is as follows:

Definition 5.3 *An* IFS $D \subset G \times IW$, $IW = ims(V)$, *is* well-defined on V *iff:*

(5.3S) soundness: $(g, w) \in D$, *then*

$$w \in IMSort^i V \Rightarrow w \in^i_{sort} ValAt(g),\ i = 1 \div 6, \tag{31}$$

with the membership relation $\in^i_{sort}$ *specific to the sorts* $MSort^i$ *in (30). In detail:*

(sound$_1$) $w \in IMSort^1 \Rightarrow w \in ValAt(g)$;

(sound$_2$) $w \in IMSort^1 \Rightarrow w \subset ValAt(g)$;

(sound$_3$) $w \in IMSort^3 \Rightarrow w \in ValAt(g)$;

(sound$_4$) $w \in IMSort^4 \Rightarrow w \in ValAt(g)$;

(sound$_5$) $w \in IMSort^5 \Rightarrow w \in ValAt(g)$;

(sound$_6$) $(w, \mathrm{m}) \in IMSort^6 \Rightarrow w \in^i_{sort} ValAt(g)$ *or* $(w = \&$ *and* $\mathrm{m} > 0)$.

(5.3S) uniqueness: $(g, v), (g, w) \in D$, *then* $v \in^i_{sort} w$, $i = 1 \div 6$, *with the equality relation specific to the sorts* $IMSort^i$ *in (30). More precisely:*

(unique$_1$) if $v \in IMSort^1$ *then* $w \in IMSort^1$, *and* $v = w$ *(equality as identity relation);*

(unique$_2$) if $v \in IMSort^2$ *then* $w \in IMSort^2$, *and* $v = w$ *(as set equality);*

(unique$_3$) if $v \in IMSort^3$ *then* $w \in IMSort^3$, *and* $v = w$ *(as list equality);*

(unique$_4$) if $v \in IMSort^4$ *then* $w \in IMSort^4$, *and* $v = w$ *(as multiset equality);*

(unique$_5$) if $v \in IMSort^5$ *then* $w \in IMSort^5$, *and* $v = w$ *(as* **po***-multiset equality);*

(unique$_6$) if $v = (v_1, \mathrm{m}) \in IMSort^6$ *then* $w = (w_1, \mathrm{n}) \in IMSort^6$, $v_1 =^i_{sort} w_1$, $i = 1 \div 6$ *(as multi-sort equality) and* $\mathrm{m} = \mathrm{n}$ *(as natural numbers).*

Consistent with the already established technique of FS definitions **??** and **??**, we can abbreviate the introduction of the *well-defined* indexed FSs (IFSs) as follows:

Definition 5.4 *Let* V_0 *be a non-empty set of* atomic *single* values, *and* $IW_0 = ims(V_0)$, *the set of all possible* atomic *indexed (multi-sort)* values *on* V_0 , *constructed similarly to the definition 5.1 and relations*

(29)–(30). Let $T_0 \subseteq Pow(IW_0)$ *be a non-empty set called the set of* effective values for the 0-level attributes, *while* $G_0 \leftrightarrow T_0 \subseteq Pow(IW_0)$ *(as in definition 29) is said to be the set of* indexed (multi-sort) *valued effective attributes defined on* IW_0*. Then:*

(I_0*)* *An* atomic *indexed (multi-sort)* valued FS *(*IFS*) is any* IFS_0 IFS_0 *well-defined on* V_0 *, i.e.* $IFS_0 \subset G_0 \times IW_0$ *satisfying (5.3S) and (5.3U). Accordingly,*

$s(IFS_0) = \{\mathrm{D} | \mathrm{D} \subset G_0 \times IW_0$*, and* D *is well-defined on* $V_0\}$*.*

(I_k*)* *An* indexed (multi-sort) valued IFS *of level* k*,* $k \geq 0$*, is any well-defined* IFS_k *on* V_k*, i.e.,* $IFS_k \subset G_k \times IW_k$ *with:*

(v_k*)* $V_k = V_{k-1} \cup s(IFS_{k-1})$ *the set of* all possible *single*-sort values *for* IFS_k*.*

Let $IW_k = ims(V_k)$*, defined similarly to the relation (29). Let* $T_k \subseteq Pow(IW_k)$ *be the set of* IFS effective values until level k*, including it, such that* $T_k \backslash T_{k-1} \neq \emptyset$*. Then:*

(g_k*)* G_k *with* $G_{k-1} \subseteq G_k \leftrightarrow T_k \subseteq Pow(IW_k)$ *is the set of* effective attributes until level k*, including it, defined on* V_k *(as in definition 5.2).*

(ifs_k*)* $s(IFS_k) = \{\mathrm{D} | \mathrm{D} \subset G_k \times IW_k$*,* $IW_k = ms(V_k)$*,* $V_k = V_{k-1} \cup s(IFS_{k-1})$*,* $s(IFS_{-1}) = G_{-1} = \emptyset$*, and* D *is well-defined on* $V_k\}$*,*

$\forall k \in \mathbf{N}$*, is called the set of all possible, well-defined* IFS*s until level* k*.*

Now the set of all IFSs is defined as:

$$\mathbf{IFS}(V_0) = \bigcup_{k \geq 0} s(IFS_k). \tag{32}$$

To define the *subsumption of indexed* FSs we shall adopt a somewhat different strategy compared to non-indexed MS-TFSs, viz. to work at the level of FS paths. This provides the possibility to make

the distinction between cyclic and acyclic FSs, namely between FSs defined on directed but general (including cyclic) graphs and FSs defined on directed acyclic graphs (DAGs). To attend this purpose it is necessary to define the *nodes,* the *graph,* and the *paths* associated to an IFS.

Definition 5.5 *If*

$$
\begin{aligned}
&v = {}^{\vee}\{v_1, v_2, \ldots, v_n\}^{\vee} \in IMSort^2, \text{ or}\\
&v = \langle v_1, v_2, \ldots, v_n \rangle \in IMSort^3, \text{ or}\\
&v = {}^{+}\{v_1, v_2, \ldots, v_n\}^{+} \in IMSort^4, \text{ or}\\
&v = {}^{+_{\mathrm{p}}}\{v_1, v_2, \ldots, v_n\}^{+_{\mathrm{p}}} \in IMSort^5,
\end{aligned}
$$

then $dim(v) = \mathrm{n}$ *is called the* dimension *of* v*, and* $v_i = pr(i, v)$ *is called the* i-projection *of* v *(or* projection of the i-component *of* v*).*

The graph associated to an IFS is defined as the set of 3-uples made up of paths in the IFS, their final nodes, and the indices of these final nodes.

Definition 5.6 *Let* V *be a set of atomic values and* $\mathrm{D} \in \mathbf{IFS}(V)$*. Then:*

$$
\begin{aligned}
Graph(\mathrm{D}) = \{(\langle \mathrm{Start}, 0\rangle, \mathrm{D}, 0)\}\cup\\
\cup \{(\mathrm{P}, v, 0) \mid\ & \exists(\mathrm{Q}, w, \mathrm{n}) \in Graph(\mathrm{D}),\ (g, v) \in w,\\
& v \notin IMSort^j,\ j = 2 \div 6,\\
& \mathrm{P} = append(\mathrm{Q}, \langle (g, 0)\rangle)\}\cup\\
\cup \{(\mathrm{P}, v, \mathrm{m}) \mid\ & \exists(\mathrm{Q}, w, \mathrm{n}) \in Graph(\mathrm{D}),\ (g, u) \in w,\\
& u \in IMSort^6,\ u = (v, \mathrm{m}),\\
& \mathrm{P} = append(\mathrm{Q}, \langle (g, \mathrm{m})\rangle)\}\cup\\
\cup \{(\mathrm{P}, v_i, 0) \mid\ & \exists(\mathrm{Q}, w, \mathrm{n}) \in Graph(\mathrm{D}),\ (g, v) \in w,\\
& w \in IMSort^j,\ j = 2 \div 5,\\
& v_i = pr(i, w) \notin IMSort^6,\\
& \mathrm{P} = append(Q, \langle (pr(i, w), 0)\rangle),\\
& i \in \{1 \div dim(w)\}\}\cup\\
\cup \{(\mathrm{P}, v_i, \mathrm{m}) \mid\ & \exists(Q, w, n) \in Graph(D),\ (g, v) \in w,\\
& w \in IMSort^j,\ j = 2 \div 5,\\
& u_i = pr(i, w) \in IMSort^6,\ u_i = (v_i, m),\\
& \mathrm{P} = append(\mathrm{Q}, \langle (pr(i, w), m)\rangle),\\
& i \in \{1 \div dim(w)\}\}
\end{aligned}
\tag{33}
$$

Definition 5.7 *The path* $\mathrm{P} = \langle (g_1, i_1), \ldots, (g_n, i_n) \rangle$ *in* $Graph(\mathrm{D})$ does not contain cycles *if* $\forall p, q \in \{1 \div n\}$ *with* $p \neq q$ *and* $i_p, i_q > 0 \Rightarrow i_p \neq i_q$.

Definition 5.8 *Let* $\mathrm{P} = \langle (g_1, i_1), \ldots, (g_n, i_n) \rangle$ *be a path. Then* $last(\mathrm{P}) = (g_n, i_n)$.

Definition 5.9 D does not contain cycles *if any path* P *in* $Graph(\mathrm{D})$ *does not.*

Definition 5.10 *The* $Graph(\mathrm{D})$ *of a* $\mathrm{D} \in \mathbf{IFS}(V)$ *is* well-defined *if:*

(1) D *does not contain cycles; and*

(2) $\forall(\mathrm{P}, v, \mathrm{m}), (Q, w, \mathrm{n}) \in Graph(\mathrm{D})$, *then* $\mathrm{n} = \mathrm{m} > 0$ *entails:*

(i) $v = w$, *or* $v = \&$, *or* $w = \&$, *and*

(ii) $v \neq \& \Rightarrow v \in^i_{sort} ValAt(last(\mathrm{P}))$, *and*
$\forall(\mathrm{R}, \&, n) \in Graph(\mathrm{D}) \Rightarrow v \in^i_{sort}, ValAt(last(\mathrm{R}))$,
$i \in \{1 \div 5\}$.

Definition 5.11 $\mathrm{D} \in \mathbf{IFS}(V)$ *is* well-defined *iff its* $Graph(\mathrm{D})$ *is well-defined and does not contain cycles.*

Definition 5.12 *The set of* indexed (multi-sort) acyclic FS*s (*IAFS*s) defined on a set* V *of atomic values is introduced now as:*

$$\mathrm{IAFS}(V) = \{\mathrm{D} | \mathrm{D} \in \mathbf{IFS}(V) \text{ and } \mathrm{D} \text{ is well-defined}\} \tag{34}$$

In order to define the subsumption on *indexed multi-sort expressions* and their corresponding *acyclic feature structures* (IAFSs) it is necessary to introduce the *restrictions* of these indexed objects to their corresponding non-indexed ones.

Definition 5.13 *If* v *is an* indexed multi-sort expression *on* $V_0 \cup \cup \mathrm{IAFS}(V_0)$, *i.e.,* $v \in ims(V_0 \cup \mathrm{IAFS}(V_0))$, *then the* restriction *of* v *to its corresponding non-indexed multi-sort expression* $Res(v) \in ms(V_0 \cup \cup \text{MS-TFS}(V_0))$ *is inductively defined as follows:*

(res1) $v \in V_0 \Rightarrow Res(v) = v$;

(res1′) $v = \{(g_1, v_1), \ldots, (g_n, v_n)\} \in \mathrm{IAFS}(V_0) \Rightarrow$
$\Rightarrow Res(v) = \{(g_1, Res(v_1)), \ldots, (g_n, Res(v_n))\}$;
(we notice that in (res1) and (res1′), $v \in IMSort^1$*)*;

(res2) $v = {}^{\vee}\{v_1, v_2, \ldots, v_n\}^{\vee} \in IMSort^2 \Rightarrow$
$\Rightarrow Res(v) = {}^{\vee}\{Res(v_1), Res(v_2), \ldots, Res(v_n)\}^{\vee}$;

(res3) $v = \langle v_1, v_2, \ldots, v_n \rangle \in IMSort^3 \Rightarrow$
$\Rightarrow Res(v) = \langle Res(v_1), Res(v_2), \ldots, Res(v_n) \rangle$;

(res4) $v = {}^{+}\{v_1, v_2, \ldots, v_n\}^{+} \in IMSort^4 \Rightarrow$
$\Rightarrow Res(v) = {}^{+}\{Res(v_1), Res(v_2), \ldots, Res(v_n)\}^{+}$;

(res5) $v = {}^{+\mathrm{p}}\{v_1, v_2, \ldots, v_n\}^{+\mathrm{p}} \in IMSort^5 \Rightarrow$
$\Rightarrow Res(v) = {}^{+\mathrm{p}}\{Res(v_1), Res(v_2), \ldots, Res(v_n)\}^{+\mathrm{p}}$;

(res6) $v = (w, \mathrm{n}) \in IMSort^6 \Rightarrow$
$$\Rightarrow Res(v) = \begin{cases} Res(u), \textit{ if } \exists(\mathrm{P}, u, \mathrm{n}) \in Graph(\mathrm{D}), \textit{ with } u \neq \&; \\ [\,], \qquad \textit{otherwise.} \end{cases}$$

Definition 5.14 *We introduce now the* decomposition *of an* indexed multi-sort *as being a mapping* $Dec : ims(V_0 \cup \mathrm{IAFS}(V_0)) \to ims(V_0 \cup \mathrm{IAFS}(V_0))$ *defined inductively as follows:*

(dec1) $v \in V_0 \Rightarrow Dec(v) = v$;

(dec1′) $v = \{(g_1, v_1), \ldots, (g_n, v_n)\} \in \mathrm{IAFS}(V_0) \Rightarrow$
$\Rightarrow Dec(v) = \{(g_1, Dec(v_1)), \ldots, (g_n, Dec(v_n))\}$
(We notice that in (dec1) and (dec1′), $v \in IMSort^1$*)*;

(dec2) $v = {}^{\vee}\{v_1, v_2, \ldots, v_n\}^{\vee} \in IMSort^2 \Rightarrow$
$\Rightarrow Dec(v) = {}^{\vee}\{Dec(v_1), Dec(v_2), \ldots, Dec(v_n)\}^{\vee}$;

(dec3) $v = \langle v_1, v_2, \ldots, v_n \rangle \in IMSort^3 \Rightarrow$
$\Rightarrow Dec(v) = \langle Dec(v_1), Dec(v_2), \ldots, Dec(v_n) \rangle$;

(dec4) $v = {}^{+}\{v_1, v_2, \ldots, v_n\}^{+} \in IMSort^4 \Rightarrow$
$\Rightarrow Dec(v) = {}^{+}\{Dec(v_1), Dec(v_2), \ldots, Dec(v_n)\}^{+}$;

(dec5) $v = {}^{+\mathrm{p}}\{v_1, v_2, \ldots, v_n\}^{+\mathrm{p}} \in IMSort^5 \Rightarrow$
$\Rightarrow Dec(v) = {}^{+\mathrm{p}}\{Dec(v_1), Dec(v_2), \ldots, Dec(v_n)\}^{+\mathrm{p}}$;

(dec6) $v = (w, \mathrm{n}) \in IMSort^6 \Rightarrow Dec(v) = (Dec(w), \mathrm{n})$.

Remark 5.15 *According to the subsumption definition* **??** *we have (for the corresponding embedded sorts):* $\forall v \in ms(V_0 \cup \text{MS-TFS}(V_0))$, $v \approx_{\text{sub}} Dec(v)$. *The same relationship is intended to hold for* $v \in ims(V_0 \cup \text{IAFS}(V_0))$.

We can give now the *subsumption definition* for *indexed (multi-sort) values* and IAFSs.

Definition 5.16 *Let* v *and* w *be* indexed multi-sort expressions *on* $V_0 \cup \text{IAFS}(V_0)$, *i.e.,* $v, w \in ims(V_0 \cup \text{IAFS}(V_0))$. *Then* v subsumes w, *written* $v \geq_{\text{sub}} w$, *iff:*

(i) $Res(v) \geq_{\text{sub}} Res(w)$, *according to the subsumption definition* **??** *for (non-indexed) multi-sort expressions (including Table* **??***), and*

(ii) $\forall(\mathrm{P}, r, \mathrm{n}), (\mathrm{Q}, s, \mathrm{n}) \in Graph(Dec(v))$, *then* $\exists(\mathrm{P}', r', \mathrm{m}), (\mathrm{Q}', s', \mathrm{m}) \in Graph(Dec(w))$, *and* $n > 0 \Rightarrow m > 0$.

According to the above definition (i), $\forall(\mathrm{P}, v, \mathrm{m}) \in Graph(Dec(v))$, with $P = \langle(g_1, i_1), \ldots, (g_n, i_n)\rangle$, then $\exists(\mathrm{P}', v', \mathrm{m}') \in Graph(Dec(w))$ with $P' = \langle(g_1, j1), \ldots, (g_n, j_n)\rangle$.

Definition 5.17 *Let* $\mathrm{D}, \mathrm{D}_1, \mathrm{D}_2 \in \text{IAFS}(V)$. *Then* D *is an* unifier *for* D_1 *and* D_2 *iff* $\mathrm{D}_1 \geq_{\text{sub}} \mathrm{D}$ *and* $\mathrm{D}_2 \geq_{\text{sub}} \mathrm{D}$, *usually written* $\mathrm{D} \in uni(\mathrm{D}_1, \mathrm{D}_2)$, *the unifier set for* D_1 *and* D_2 .

Definition 5.18 D^* *is the* most general unifier *of* $\mathrm{D}_1, \mathrm{D}_2 \in \text{IAFS}(V)$, *written* $\mathrm{D}^* = mgu(\mathrm{D}_1, \mathrm{D}_2)$, *iff:*

(i) $\exists \mathrm{D}^* \in uni(\mathrm{D}_1, \mathrm{D}_2)$;

(ii) $\forall D \in \text{IAFS}$, *and* $D \in uni(D_1, D_2)$, *then* $D^* \geq_{\text{sub}} D$, *i.e.*, $D^* = glb_{\text{sub}}(D_1, D_2)$.

Remark 5.19 *As in the case of* MS-TFS, *the subsumption relation for* IAFS *is a partial ordering,* modulo *a* renaming relation *on the set of indices.*

The unification algorithm *for* FS*s in* IAFS(V) *reduces to mgu (thus glb) computing (definition 5.18) and is based on the* subsumption *of* indexed multi-sort FS*s (definition 5.16) which appeals* unification algorithm *of* FS*s in* MS-TFS(V), *together with acyclicity conditions for the resulted* FS. *(See Appendices A and B.)*

The acyclicity conditions on the final result are necessary since otherwise from the unification of two IAFS (thus cycle-free) FSs one can obtain a FS with cycles!

6 Logical vs. set-theoretical approaches to NL

6.1 Attribute-oriented vs. value-oriented semantics for FSs

The classical approach to FS semantics is the so-called logical semantics proposed by [24]. In logical semantics there is defined "a domain of logical formulas which describes feature structures", made up of labels (actually FS attributes) to which are assigned (new) formulas, (sets of) label paths, conjunctions and disjunctions of formulas. In the obtained FDL (Feature Description Logic) language, the emphasis falls clearly on the attribute interpretation as predicates, resulting a language whose formulas, corresponding to FSs (with disjunction), can be completely specified by a set of finite automata. "An interesting result is that *conjunction can be used to describe unification...* Unifying two structures requires finding a structure which has all features of both structures; the conjunction of two formulas describes the structures which satisfy all conditions of both formulas" [24, p. 4]. The semantics of the formulas in the FDL language is defined on the basis of deterministic finite automata, and the propositional version of the proposed

logic calculus considers formulas defined on DAGs, a DAG being seen as particular case of deterministic finite automaton.

Devoted to support the M. Kay's Functional Unification Grammar framework [26], or the PATR-II unification grammar formalisms [32], this kind of semantics for FSs points out mainly the predicational aspects of the FS attributes, the FS value coming as a less important aspect. This is influenced, perhaps, by the theorem proving theory and the logical unification algorithms but, at a serious, pragmatic analysis, one can discover that the goals of a classical theorem prover and that of a NL analyzer (parser or generator) are essentially divergent, despite their remarkable resemblance and various attempts to achieve uniform processing frameworks based on parameterized theorem provers [33].

For this logical semantics, "conjunction can be used to describe unification" [24, p. 4] of the formulas corresponding to FSs, while when we come into the structure of the FDL formulas, "the unification operator on trees is just set union..." [29, p. 4]. From the linguistic and pragmatic points of view, the structure of the formula and the values borne by it, illustrated by the relations between these values, are characteristic to the objects (linguistic or not) we want to analyze, whose semantics we introduced by means of the *linguistic realization* map.

The *set-theoretical semantics* that we proposed starts from values, from their shape and properties as data types, and constructs the FSs that describe the involved (linguistic) objects. In our approach, the conjunction is treated merely as a FS constructor that gather the multi-sort FS values describing the involved objects, preserving still its essential interpretation as FS unification. Within the various types of NL analyzes we need lists of words, lists of linguistic feature values, etc., trees, sets of trees, multisets, **po**-multisets, etc. All these multi-sort values for FSs result as objects of computational processes (e.g., parsing, linguistic unification and/or generalization) that work on formal grammars, automated lexicons, etc. The proposed approach to set-theoretical definitions of FSs and their multi-sort values shifts the accent from *attribute-oriented semantics of* FS*s* towards a *value-oriented semantics,* or from logical semantics to set-theoretical seman-

tics of FSs. It is also important to notice that all our FS inductive, set-theoretical definitions provide careful conditions to ensure the well-definedness of FSs: soundness and uniqueness of FS attribute-value definitions, and all the necessary conditions for FS interpretation as graphs: well-definedness, acyclicity, restriction, decomposition etc.

6.2 Parsing and linguistic unification

The continuous increasing of the importance of *lexical semantics* in the development of linguistic theories and NL processing systems is a real fact connected directly to the NL representation as linguistic FSs and their unification process. Describing the linguistic categories as complex data types such as FSs entailed at least two consequences: **(a)** substitution of a lot of phrase structure rules describing the lexical information by their corresponding FSs, and **(b)** development of the FS unification processes till the somehow exaggerated (but interesting) situation that the formal derivation of syntactic structures of NL, usually realized by the parsing process of linguistic categories described through phrase structure grammars, can be simulated (or even only) by unification of the corresponding FSs. Actually, the unification processes simulate the derivation of FSs through their formal deduction within logical axiomatic systems describing the syntactic and semantic behaviour of FSs. Such a system is provided, e.g., by FDL (Feature Description Logic) [24], [29], while a complete simulation of parsing through logic deduction is proposed by the parametric utilization of a theorem prover for both NL parsing and generation, such as in [33].

Perhaps the first balanced approach between *unification* of FSs represented as DAGs and *parsing* of the corresponding linguistic categories which these FSs are labeled with is the unification-based environment PATR-II [32] of *conditional grammars*. Another frequently met example is represented by the so-called *logic grammars*, usually embedded within a logic programming language (such as Prolog) which can simulate the parsing process through mechanical deduction written and run into the language, e.g., [15], [16], [30].

The result of all these approaches is that NL lexicons and/or linguis-

tic knowledge bases used to represent the entries to the word description became central elements for the NL parsing processes involved in the automated NL analysis, generation, interpretation, translation, summarization etc. For the concrete, real NL processing systems there is always a trade-off between how much unification of the FSs representing the NL categories and phrases *and how much parsing* of the corresponding "backbone" of phrase structure grammars used to describe their functioning, such that the system to have an optimal behaviour computationally.

6.3 Comparison to similar work

The present paper is a result of several influences and stimuli: (**1**) the proposed set-theoretical recurrent definitions stem in our first approach [10]; (**2**) the formalizations in [3] inspired us to obtain a set-theoretical semantics for multi-sort valued FSs, as an important step to a new parsing paradigm in the HPSG context, based on computing recursively the model-theoretic interpretations of HPSG grammar rules; (**3**) [22] provided an axiomatic approach to the Schonfinkel-Bernays decidable class of predicate formulas modelling the behaviour of set-type valued FSs; (**4**) finally, despite the fact that our paper was already written at that time (1999), we discovered [20] as being somehow the most appropriate to our approach, and this constitutes both a satisfaction and a confirmation for the validity of our options and results.

We outline briefly the main points representing the resemblance and the differences between [20] and our present approach: (**a**) [20] is based on [22] results concerning the decidable class of Schonfinkel-Bernays predicate logic modelling the FS language, and extends this language to the so-called "multi-collection" extended FSs to support the HPSG-style of *set* and *list* constructions on FSs. The main goal is similar to [3] and to ours: constrained-based parsing of HPSG. (**b**) As S. Hegner [20] observes himself, "the only specialized constructions which are considered are those based upon sets and lists... linear precedence constraints are not addressed." In our approach, **po**-multisets are endowed with the trace of the partial order provided on FSs by the subsumption

relation, while the linear precedence of linguistic objects is encoded within lists. Their closure to sort (type) composition, including the other sorts, provide a better expressive power than [20] for HPSG constructions. (**c**) In our approach, the linguistic realization (semantics) of the FS "inner" conjunction is the unification of FS values, while the "outer" conjunction of FSs has the meaning of set-intersection. The "inner", value-level disjunction in FSs is encoded by the corresponding mathematical set of linguistic objects, while the linguistic realization of the "outer", FS-level disjunction has the meaning of set-union. (**d**) We did not address here *negation,* but this is really not a problem because its meaning, i.e., its set-theoretical linguistic realization of the negated FSs can be represented adequately as the "complementary" set (related to the lexicon involved) to the set of words on which the FS does hold, viz. that set of words on which does hold the negation of the FS. (**e**) Finally, in section **??** we expose a (non-connected) graph representing the hierarchy of the introduced sorts. This constructive hierarchy is essential for the proposed linguistic realization, thus for the computation of linguistic object interpretation within the parsing process.

7 Conclusions

Which is the reason behind the proposed set-theoretic approach to linguistic unification, why would be valuable this trend within the current context of computational linguistics and in which conditions or restrictions? We consider these questions to be answered and a critical and constructive view on the set-theoretic approach to be most important. The results contained here can represent a good step, but we already have in mind improved alternatives to the problems such as indexed multi-sort definitions, better settings for the set-theoretical semantics on FSs and NL representation, FS basic definitions through recurrent sets or comprised paths, subsumption, unification, generalization, etc.

The logical approaches to linguistic FS semantics are clearly dominant in the literature, starting with [29], [24], and this is natural because representing the morphologic, syntactic and semantic properties

of the NL components can be done by assigning and calling them with the labels that correspond to their linguistic function names within the FS. Thus the various levels of encoding the linguistic information within FSs are labeled by linguistically suggestive names and these labels are used level-free within their FS representation during unification processes.

Defining FSs as recurrently embedded sets is not something new. The major difficulty with set-theoretic definitions of FSs is their linguistic relevance and handling. If this problem can be solved conveniently, there are a lot of many other advantages, to mention only the uniformity of representation (particularly on the lexical and morphological levels), simplicity of algorithms, computational performance, clear theoretical semantics [3], an uniform approach to parsing and unification (and, in general, to linguistic operations) as *set-theoretical model checking.* Of course, the logical operations involved by the syntactic and/or semantic representation and functioning of FS NL should be simulated by simple and computationally efficient set-theoretical operations.

As the strong points of the present paper we consider to be the following ones:

1. Definitions of the *set-theoretical sorts* involved within HPSG theory [27], [28], their *linguistic realization* (semantics) and their complete hierarchy. A special attention deserves the sort of **po**-multiset, corresponding to HPSG "set descriptions" of objects, its linguistic realization (computation formula), and its rank.

2. *Set-theoretical recurrent definitions* of non-indexed and indexed *multi-sorted values* for FSs, being carried out on basic data types such as trees, (rooted) directed graphs, and (rooted) directed acyclic graphs.

3. A *tableau-based subsumption definition* for the non-indexed multi-sort FS values, extended subsequently to indexed ones. For indexed FSs we could not avoid the graph-based approach to subsumption definition and unification.

4. The *Prolog implementation* of general unification algorithms for

the considered FSs, together with running them on some simple examples.

We do not believe in the existence of a general solution to the exposed problems but only in partial and local solutions, depending on the linguistic application. As a general strategy to an efficient set-theoretical approach to FS definitions we think on the strong use of linguistic *data typing,* as well as specific definitions of structures and operations depending on the types involved. The background priority of the set-theoretical operations over the logical ones inside FSs is expressed even by the first important approach to logical semantics of FSs, namely the *feature description logic* FDL in [29]: *"The unification operator on trees is just set union, except that if a tree is obtained which is inconsistent, then unification is undefined".* In other words, unification is actually just a set-conditional union. Thus the set-theoretical approaches to FS semantics and NL computational processing, as well as their periodic recurrence in time, e.g., [3], are natural phenomena. But while the set-theoretical semantics approaches to FSs are already known to be more simple and clear than the corresponding logical ones, the present paper tries to meet the computationally increasing importance of the same trend of set-theoretical approaches to the forthcoming implementations of NL processing systems.

References

[1] Anne Abeillé, Y. Schabes, A. Joshi: *Using Lexicalized TAGs for Machine Translation*, Proceedings of COLING'90, Helsinki, Vol. 3, pp. 1–6, 1990.

[2] Joan Bresnan (Ed.): *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, Mass, 1980.

[3] H. Bunt, Ko van der Sloot: *Parsing as Dynamic Interpretation of Feature Structures*, in: *Recent Advances of Parsing Technologies* (Bunt & Tomita, Eds.), Kluwer Academic Publishers, Dordrecht, pp. 91.

[4] Noam Chomsky: *Lectures on Government and Binding.* The Pisa Lectures, Foris Publications, Dordrecht, Holland, 1982.

[5] Noam Chomsky: *Barriers*, The MIT Press, Cambridge, Massachusetts, 1986.

[6] Noam Chomsky: *Knowledge of Language. Its Nature, Origin, and Use*. Holt, Rinehart & Winston, New York, 1984.

[7] Noam Chomsky: *The Minimalist Program*, The MIT Press, Cambridge, Massachusetts, 1996.

[8] Alain Colmerauer: *Metamorphosis Grammars*, in: *Natural Language Communication with Computers*, L. Bolc (Ed.), pp. 133–189, Springer-Verlag, Berlin, Heidelberg, 1978.

[9] Alain Colmerauer: *Prolog and Infinite Trees*, in: *Logic Programming*, K.L. Clark, S.-A. Tarnlund (Eds.), pp. 45–66, Academic Press, New York, London, 1982.

[10] N. Curteanu, G. Holban, A. Cărăuşu: *On the Linguistic Feature Structure Unification*, Proceedings of the 2nd National Colloquium on Languages, Logic, and Mathematical Linguistics, Braşov (Romania), june 1988, pp. 93–110 (in Romanian).

[11] N. Curteanu: *A Marker Hierarchy-based Approach Supporting the S–C–D Parsing Strategy*, Research Report, Institute of Technical Cybernetics, Bratislava, 30 pp, 1990.

[12] N. Curteanu: *From Morphology to Discourse Through Marker Structures in the S–C–D Parsing Strategy. A Marker Hierarchy-based Approach*, Language and Cybernetics, Akademia Libroservo, Prague, pp. 61.

[13] N. Curteanu et al.: *A Linguistic Knowledge Base for Romanian*, Language and Technology (D. Tufiş, Ed.), The Editing House of Romanian Academy, pp. 101–108, 1996 (in Romanian).

[14] N. Curteanu, A. Todiraşcu: *Syntactic and Semantic Theories of Natural Language. Denotational Semantics*. Research Report, Res. Inst. Comp. Sci., Romanian Academy, Iaşi Branch, 96 pp., Dec. 1998 (in Romanian).

[15] Verónica Dahl, Patrick Saint-Dizier (Eds.): *Natural Language Understanding and Logic Programming*, North-Holland, Amsterdam & New York, XI + 243 pp., 1985.

[16] Verónica Dahl, Patrick Saint-Dizier: *Constrained Discontinuous Grammars – A Linguistically Motivated Tool for Processing Language*, INRIA, France, Research Report No. 573.

[17] Gerald Gazdar: *Phrase Structure Grammars and Natural Languages*, Proceedings of the Intern. Joint Conf. on Artif. Intell., A. Bundy (Ed.), IJCAI Public., pp. 556–565, 1983.

[18] Gerald Gazdar, G.K. Pullum: *Generalized Phrase Structure Grammar: A Theoretical Synopsis*, Cognitive Science Res. Paper CSRP 007, University of Sussex, 51 pp., 1982.

[19] G. Gazdar, E. Klein, G. Pullum, I. Sag: *Generalized Phrase Structure Grammar*, Harvard University Press, Cambridge, Massachusetts, 1985.

[20] Stephen J. Hegner: *A Family of Decidable Feature Logics which Support HPSG-Style Set and List Constructions*, in: (C. Retore; Ed.) *Logical Aspects of Computational Linguistics*, LNCS 1328, Springer.

[21] Alexander Herold, Jörg Siekmann: *Unification in Abelian Semigroups*, Journal of Automatic Reasoning Vol. 3, pp. 247–283, 1987.

[22] Mark Johnson: *Computing with Features as Formulae*, Computational Linguistics, Vol. 20, No. 1, March 1994.

[23] Ronald Kaplan, Joan Bresnan: *Lexical-Functional Grammar: A Formal System for Grammatical Representation*, in: [2], 1982.

[24] R. Kasper, W. Rounds: *A Logical Semantics for Feature Structures*, Reprint from the Proceedings of the 24th ACL Meeting, Columbia University, New York, 10 pp., June 1986.

[25] Martin Kay: *Unification Grammar*, Xerox Palo Alto Res. Center, Palo Alto, California, 1983.

[26] Martin Kay: *Parsing in Functional Unification Grammar*. In (D. Dowty et al.; Eds) it Natural Language Parsing, Cambridge Univ. Press, Cambridge, England, 1985.

[27] C. Pollard, I. Sag: *Information-based Syntax and Semantics*, CSLI, Stanford, California, 1987.

[28] C. Pollard, I. Sag: *Head-Driven Phrase Structure Grammar*, The University of Chicago Press, Chicago & London, 1994.

[29] W.C. Rounds, R. Kasper: *A Complete Logical Calculus for Record Structures Representing Linguistic Information*, Res. Report, University of Michigan, Ann Arbor, 13 pp., March 1986.

[30] Patrick Saint-Dizier: *Context-Dependent Determiners in Logic Programming: Semantic Representation and Properties*, IRISA, Rapport No. 2977, Rennes, France, 1986.

[31] Peter Sells: *Lectures on Contemporary Syntactic Theories*, CSLI Lecture Notes No. 3, Stanford University, California, 1985.

[32] Stuart Shieber: *An Introduction to Unification-Based Approaches to Grammar*, CLSI Lecture Notes No. 4, Stanford University, California, 1986.

[33] Stuart Shieber: *A Uniform Architecture for Parsing and Generation*, Proceedings of COLING'88, Budapest, Vol. 2, pp. 614–619, 1988.

[34] N. Curteanu, P.-G. Holban: *A set-theoretic approach to linguistic feature structures and unification algorithms (I)*, Computer Science Journal of Moldova, Vol. 8, No. 2(23), 2000, pp. 116–149.

Neculai Curteanu, Paul-Gabriel Holban,
Research Institute of Computer Science,
Romanian Academy, Iaşi Branch
B-dul Carol I, No. 22A, 6600, Iaşi, România
e-mail: curteanu@iit.tuiasi.ro

Received April 15, 2000

Appendices

A The Prolog Unification Algorithm

```
/*Unification algorithms for multi-sort expressions of FSs. When the */
/* lists, Ú-sets (disjunction), multisets (bags), po-multisets are made up of a */
/* single element, they are represented as the single-sort of the element itself. */
?- op( 500, xfx, ':'), /* the feature structure (FS) as the list [attribute : value] */
op( 450, xfy, '^'), /* index ^value means the indexed FS : (n) FS */
op( 400, xfy, '/'), /* Ú-set (disjunction of values) FS1 / FS2 /.../ FSn */
op( 350, xfy, '#'), /* po-multiset as a Ú-set (disjunction) of multisets */
op( 300, xfy, '*'), /* multisets (bags) of FSs as Ú-set (disjunction) of lists */
op( 250, xfy, '+'). /* (usual, ordered) list of FSs (list of FS values) */
/* Unification of multi-sort expressions of non-indexed FSs */
unify( Expr, Expr, Expr) :- !. /* identity of multi-sorted FS expressions */
unify( bottom, _, bottom) :- !. /* all the inconsistent multi-sorted FSs */
unify( _, bottom, bottom) :- !. /* as the special FS BOTTOM */
unify( [], Expr, Expr) :- !. /* the universal multi-sorted FS as the special */
unify( Expr, [], Expr) :- !. /* FS TOP (also the neutral FS to unification) */
/* Disjunction of FS values with ...a disjunction of FS values */
unify( Val1 / RVal1, Val2 / RVal2, LResult) :- !,
unify( Val1, Val2 / RVal2, Val), unify( RVal1, Val2 / RVal2, RVal),
( Val = bottom, Result = RVal, !
; RVal = bottom, Result = Val, !
```

```
; Result = Val / RVal ), !, l, iniarDisj(Result, LResult).
/* ..... with any other FSs */
unify( Val1 / Val2, Expr, Result) :- !, unify( Expr, Val1 / Val2, Result).
/* FS (multi-sorted) expression with ... a disjunction of FS values */
unify( Expr, Val1 / Val2, LResult) :- !,
unify( Expr, Val1, RVal1), unify( Expr, Val2, RVal2),
( RVal1 = bottom, Result = RVal2
; RVal2 = bottom, Result = RVal1
; Result = RVal1 / RVal2 ), !, l, iniarDisj(Result, LResult).
/* Proper FS with ..... a proper FS */
unify( [Atr : Val1 | Rest1], [AtrVal2 | Rest2], Result) :-
select( Atr : Val2, [AtrVal2 | Rest2], Rest3), unify( Val1, Val2, Val),
( Val = bottom, Result = bottom
; unify( Rest1, Rest3, Rest),
( Rest = bottom, Result = bottom
; Result = [Atr : Val | Rest])), !.
unify( [Atr : Val | Rest1], [AtrVal | Rest2], Result) :-
unify( Rest1, [AtrVal | Rest2], Rest),
( Rest = bottom, Result = bottom
; Result = [Atr : Val | Rest]), !.
/*. ... a po-multiset of FS values */
unify( [Atr : Val | Rest], POmultiset # RestPOmultiset, Result) :- !,
unify( [Atr : Val | Rest], POmultiset, NewFS), unify( NewFS, RestPOmultiset, Result).
/* A list with ..... a list of FS values */
unify( List1 + RList1, List2 + RList2, Result) :- !, unify( List1, List2, List),
( List = bottom, Result = bottom, !
; unify( RList1, RList2, RList),
( RList = bottom, Result = bottom, !
; Result = (List + RList))), !.
/* ... a multiset of FS values */
unify( List1 + RList1, Multiset2 * RMultiset2, Result) :- !,
permutMultiset( Multiset2 * RMultiset2, DList3), !,
unify( List1 + RList1, DList3, Result).
/* ... a po-multiset of FS values */
unify( List + Rest, POmultiset # RestPOmultiset, Result) :- !,
partMset( POmultiset # RestPOmultiset, DMultiset), !,
unify( List + Rest, DMultiset, Result), !.
/* A multiset of FS values with ..... a list of FS values */
unify( Multiset * URest, List + ORest, Result) :- !,
unify( List + ORest, Multiset * URest, Result).
/* ... a multiset of FS values */
unify( Multiset1 * Rest1, Multiset2 * Rest2, Result) :- !,
permutMultiset( Multiset1 * Rest1, DList1), !,
permutMultiset( Multiset2 * Rest2, DList2), !,
unify( DList1, DList2, Result).
/* ... a po-multiset of FS values */
unify( Multiset * Rest, POmultiset # RestPOmultiset, Result) :- !,
partMset( POmultiset # RestPOmultiset, DMultiset), !,
unify( Multiset * Rest, DMultiset, Result), !.
/* A po-multiset with ..... a po-multiset of FS values */
unify( POmultiset1 # Rest1, POmultiset2 # Rest2, Result) :- !,
partMset( POmultiset1 # Rest1, DMultiset1), !,
partMset( POmultiset2 # Rest2, DMultiset2), !,
unify( DMultiset1, DMultiset2, Result), !.
/* ... any other multi-sort expression oFS FS values */
unify( POmultiset # RestPOmultiset, Expr, Result) :- !,
unify( Expr, POmultiset # RestPOmultiset, Result).
/* Final clause */
unify( _, _, bottom).
/* Utility functions for non-indexed multi-sort expressions of FSs */
liniarDisj( (A / B) / (C / D), LA / LB / LC / LD) :- !,
liniarDisj( A, LA), l, iniarDisj( B, LB), l, iniarDisj( C, LC), l, iniarDisj( D, LD).
liniarDisj( (A / B) / C, LA / LB / LC) :- !,
liniarDisj( A, LA), l, iniarDisj( B, LB), l, iniarDisj( C, LC).
liniarDisj( A, A).
select( Atr : Val, [], _) :- !, fail.
select( Atr : Val, [Atr : Val | Rest], Rest) :- !.
```

```
select( Atr : Val, [Atr1 : Val1 | Rest1], [ Atr1 : Val1 | Rest]) :- select( Atr : Val, Rest1, Rest).
permutation( [], []).
permutation( [X | Y], Z) :- permutation( Y, W), insert( X, W, Z).
insert( Y, Xz, Xyz) :- append( X, Z, Xz), append( X, [Y | Z], Xyz).
append( [], X, X).
append( [X | Y], Z, [X | W]) :- append( Y, Z, W).
permutMultiset( Multiset, DList) :- multisetList( Multiset, List),
bagof( PList, permutation( List, PList), LPList), listDList(LPList, DList).
multisetList( A * B, [ A | LB]) :- !, multisetList( B, LB).
multisetList( A, [A]).
listList( [A], A) :- !.
listList( [A | B], A + OB) :- listList( B, OB).
listDList( [A], OA) :- !, listList( A, OA).
listDList( [A | B], OA / LOB) :- listList( A, OA), listDList( B, LOB).
init( _, 1, [0]) :- !.
init( [A | B], K, [A | C]) :- K > 1, K1 is K - 1, init( B, K1, C).
max( _, 1, Max, Max) :- !.
max( [A | B], N, OldM, NewM) :- N > 1, N1 is N - 1,
( A > OldM, max( B, N1, A, NewM)
; max( B, N1, OldM, NewM)), !.
item( [Item | _], 1, Item) :- !.
item( [_ | List], N, Item) :- N > 1, N1 is N - 1, item( List, N1, Item).
copySt( [_ | List], 1, ItemK, [ItemK | List]) :- !.
copySt( [Item | List], N, ItemK, [Item | NewList]) :-
N > 1, N1 is N - 1, copySt( List, N1, ItemK, NewList).
succesor( St, K, St1) :- max( St, K, 1, Max), Max1 is Max + 1, item( St, K, ItemK),
ItemK < Max1, ItemK < K, ItemK1 is ItemK + 1, copySt( St, K, ItemK1, St1).
part( _, 0, _, []).
part( St, K, N, Part) :- K > 0,
( succesor( St, K, St1),
( K = N, Part = [St1 | NewPart], part( St1, K, N, NewPart)
; K < N, K1 is K + 1, init( St1, K1, St2), part( St2, K1, N, Part))
; K1 is K - 1, part( St, K1, N, Part)).
sizeof( [], 0).
sizeof( [A | B], M) :- sizeof( B, N), M is N + 1.
poMultiset_list( A # B, [ A | LB]) :- !, poMultiset_list( B, LB).
poMultiset_list( A, [A]).
listMultiset( [A], Multiset, [B], NewMultiset) :- !, item( Multiset, A, ItemU),
( var( ItemU), ItemU = B, NewMultiset = Multiset
; unify( ItemU, B, Result),
( Result = bottom, NewMultiset = bottom
; copySt( Multiset, A, Result, NewMultiset))), !.
listMultiset( [A | Rest1], Multiset, [B | Rest2], NewMultiset) :- item( Multiset, A, ItemU),
( var( ItemU), ItemU = B, listMultiset( Rest1, Multiset, Rest2, NewMultiset)
; unify( ItemU, B, Result),
( Result = bottom, NewMultiset = bottom
; copySt( Multiset, A, Result, Multiset1),
listMultiset( Rest1, Multiset1, Rest2, NewMultiset))), !.
listDlist( [A], UA, List) :- !, listMultiset( A, _, List, UA).
listDlist( [A | Rest], UA / URest, List) :-
listMultiset( A, _, List, UA), listDlist( Rest, URest, List).
listMultiset( [A | []], A) :- !.
listMultiset( [A | B], A * C) :- listMultiset( B, C).
listDMultiset( bottom / A / Rest, DMultiset) :- !, listDMultiset( A / Rest, DMultiset).
listDMultiset( A / bottom / Rest, DMultiset) :- !, listDMultiset( A / Rest, DMultiset).
listDMultiset( A / B / Rest, UA / URest) :- !,
listMultiset( A, UA), listDMultiset( B / Rest, URest).
listDMultiset( bottom / bottom, bottom) :- !.
listDMultiset( bottom / A, DMultiset) :- !, listMultiset( A, DMultiset).
listDMultiset( A / bottom, DMultiset) :- !, listMultiset( A, DMultiset).
listDMultiset( A / B, UA / UB) :- listMultiset( A, UA), listMultiset( B, UB).
partMset( POmultiset, DMultiset) :- poMultiset_list( POmultiset, List), sizeof( List, N),
init( [], 1, St), part( St, 1, N, LPart), listDlist( LPart, DList, List),
listDMultiset( DList, DMultiset).
/* Unification for multi-sort expressions of indexed FSs (Obs.: TOP is [ ] ) */
unify_i( Expr1, Expr2, Expr) :- reduce( Expr1, RedExpr1, Index1),
reduce( Expr2, RedExpr2, Index2), unify( RedExpr1, RedExpr2, RedExpr),
```

```
Expr = RedExpr.
/* RedExpr has to be updated with the index specific information */
/* Utility functions for multi-sort, including indexed, FS expressions */
reduce( Expr, RedExpr, CIndex) :-
colect_index( Expr, [], CIndex), reduce_index( Expr, C, index, RedExpr).
colect_index( [Atr : Val | Rest], OldIndex, NewIndex) :- !,
colect_index( Val, OldIndex, Index), colect_index( Rest, Index, NewIndex).
colect_index( List + ORest, OldIndex, NewIndex) :- !,
colect_index( List, OldIndex, Index), colect_index( ORest, Index, NewIndex).
colect_index( Multiset * URest, OldIndex, NewIndex) :- !,
colect_index( Multiset, OldIndex, Index), colect_index( URest, Index, NewIndex).
colect_index( Val / Rest, OldIndex, NewIndex) :- !,
colect_index( Val, OldIndex, Index), colect_index( Rest, Index, NewIndex).
colect_index( POmultiset # Rest, OldIndex, NewIndex) :- !,
colect_index( POmultiset, OldIndex, Index), colect_index( Rest, Index, NewIndex).
colect_index( I ^[], Index, Index) :- !.
colect_index( I ^Val, Index, [ I ^Val | NewINdex]) :- !,
colect_index( Val, Index, NewIndex).
colect_index( _, Index, Index).
reduce_index( [Atr : Val | Rest], Index, [Atr : RedVal | RedRest]) :- !,
reduce_index( Val, Index, RedVal), reduce_index( Rest, Index, RedRest).
reduce_index( List + Rest, Index, RedList + RedRest) :- !,
reduce_index( List, Index, RedList), reduce_index( Rest, Index, RedRest).
reduce_index( Multiset * Rest, Index, RedMultiset * RedRest) :- !,
reduce_index( Multiset, Index, RedMultiset), reduce_index( Rest, Index, RedRest).
reduce_index( POmultiset # Rest, Index, RedPOmultiset # RedRest) :- !,
reduce_index( POmultiset, Index, RedPOmultiset),
reduce_index( Rest, Index, RedRest).
reduce_index( I ^[], Index, Val) :- !, f, ind_index( I, Index, Index, Val).
reduce_index( I ^Val, Index, RedVal) :- !, reduce_index( Val, Index, RedVal).
reduce_index( Val, _, Val).
find_index( _, [], _, []) :- !.
find_index( I, [ I ^J ^Val | Rest], Index, FVal) :- !, f, ind_index( J, Index, Index, FVal).
find_index( I, [ I ^Val | _], _, Val) :- !.
find_index( I, [_ | Rest], Index, Val) :- find_index( I, Rest, Index, Val).
```

B Examples

PROLOG test queries	**Program answers**
?- tell('tt.txt').	
?- nl, write('**FS to FS**').	**FS to FS**
?- A = [nr : sg, agr : [cas : nom]],	A=[nr:sg,agr:[cas:nom]]
B = [agr : [cas : nom], gen : fem],	B=[agr:[cas:nom],gen:fem]
unify(A, B, U),	U=[nr:sg,agr:[cas:nom],gen:fem]
nl, write('A='), write(A),	
nl, write('B='), write(B),	
nl, write('U='), write(U).	
?- A = [nr : pl, agr : [cas : acc]],	A=[nr:pl,agr:[cas:acc]]
B = [agr : [cas : dat], gen : masc],	B=[agr:[cas:dat],gen:masc]
unify(A, B, U),	U=bottom
nl, write('A='), write(A),	
nl, write('B='), write(B),	
nl, write('U='), write(U).	
?- nl, write('**FS to (FS) disjunction**').	**FS to (FS) disjunction**
?- A = [nr : sg],	A=[nr:sg]
B = ([gen : fem] / [cas : nom]),	B=[gen:fem]/[cas:nom]
unify(A, B, U),	U=[nr:sg,gen:fem]/[nr:sg,cas:nom]
nl, write('A='), write(A),	
nl, write('B='), write(B),	
nl, write('U='), write(U).	
?- A = [nr : pl],	A=[nr:pl]
B = [gen : fem] / [nr : sg] / [cas : acc],	B=[gen:fem]/[nr:sg]/[cas:acc]
unify(A, B, U),	U=[nr:pl,gen:fem]/[nr:pl,cas:acc]

```
  nl, write( 'A='), write( A),
  nl, write( 'B='), write( B),
  nl, write( 'U='), write( U).
?- A = [ cas : nom],                          A=[cas:nom]
  B = [ cas : acc] / [ cas : dat],            B=[cas:acc]/[cas:dat]
  unify( A, B, U),                            U=bottom
  nl, write( 'A='), write( A),
  nl, write( 'B='), write( B),
  nl, write( 'U='), write( U).
?- nl, write( ' FS to po-multiset').          FS to po-multiset
?- A = [ nr : sg],                            A=[nr:sg]
  B = [ gen : fem] # [ cas : nom],            B=[gen:fem]#[cas:nom]
  unify( A, B, U),                            U=[nr:sg,gen:fem,cas:nom]
  nl, write( 'A='), write( A),
  nl, write( 'B='), write( B),
  nl, write( 'U='), write( U).
?- A = [ nr : pl],                            A=[nr:pl]
  B = [ gen : fem] # [ nr : sg] # [ cas : acc],  B=[gen:fem]#[nr:sg]#[cas:acc]
  unify( A, B, U),                            U=bottom
  nl, write( 'A='), write( A),
  nl, write( 'B='), write( B),
  nl, write( 'U='), write( U).
?- nl, write( 'list to list').                list to list
?- A = ([ nr : sg] + [ cas : nom]),           A=[nr:sg]+{cas:nom}
  B = ([ nr : sg] + []),                      B=[nr:sg]+{}
  unify( A, B, U),                            U=[nr:sg]+{cas:nom}
  nl, write( 'A='), write( A),
  nl, write( 'B='), write( B),
  nl, write( 'U='), write( U).
?- A = ([ nr : pl] + [gen : fem]),            A=[nr:pl]+{gen:fem}
  B = ([] + [ gen : masc]),                   B=[]+{gen:masc}
  unify( A, B, U),                            U=bottom
  nl, write( 'A='), write( A),
  nl, write( 'B='), write( B),
  nl, write( 'U='), write( U).
?- A = ([ nr : pl] + [gen : fem]),            A=[nr:pl]+{gen:fem}
  B = ([] + [ cas : nom]),                    B=[]+{cas:nom}
  unify( A, B, U),                            U=[nr:pl]+{gen:fem,cas:nom]
  nl, write( 'A='), write( A),
  nl, write( 'B='), write( B),
  nl, write( 'U='), write( U).
?- nl, write( 'list to multiset').            list to multiset
?- A = [ nr : pl] + [gen : fem],              A=[nr:pl]+{gen:fem}
  B = [] * [ cas : nom],                      B=[]*[cas:nom]
  unify( A, B, U),                            U=[nr:pl]+{gen:fem,cas:nom]/
  nl, write( 'A='), write( A),                  [nr:pl,cas:nom]+{gen:fem]
  nl, write( 'B='), write( B),
  nl, write( 'U='), write( U).
?- nl, write( 'multiset to multiset').        multiset to multiset
?- A = ([ nr : sg] * [ cas : nom]),           A=[nr:sg]*[cas:nom]
  B = ([] * [ nr : sg]),                      B=[]*[nr:sg]
  unify( A, B, U),                            U=[nr:sg]+{cas:nom,nr:sg]/
  nl, write( 'A='), write( A),                  [nr:sg]+{cas:nom]/
  nl, write( 'B='), write( B),                  [cas:nom]+{nr:sg]/
  nl, write( 'U='), write( U).                  [cas:nom,nr:sg]+{nr:sg]
?- A = ([ nr : pl] * [gen : fem]),            A=[nr:pl]*[gen:fem]
  B = ([] * [ cas : nom]),                    B=[]*[cas:nom]
  unify( A, B, U),                            U=[nr:pl]+{gen:fem,cas:nom]/
  nl, write( 'A='), write( A),                  [nr:pl,cas:nom]+{gen:fem]/
  nl, write( 'B='), write( B),                  [gen:fem]+{nr:pl,cas:nom]/
  nl, write( 'U='), write( U).                  [gen:fem,cas:nom]+{nr:pl]
?- A = ([ nr : sg] * [ cas : dat]),           A=[nr:sg]*[cas:dat]
  B = ([ cas : nom] * [ cas : acc]),          B=[cas:nom]*[cas:acc]
  unify( A, B, U),                            U=bottom
  nl, write( 'A='), write( A),
```

```
  nl, write( 'B='), write( B),
  nl, write( 'U='), write( U).
?- nl, write( 'po-multiset to po-multiset').
?- A = ([ m : t] # [m : d]),
  B = ([ c : r] # [c : w]),
  unify(A,B,U),
  nl, write('A='), write( A),
  nl, write('B='), write( B),
  nl, write('U='), write( U).
?- A=([ m : t] # [ m : d] # [ c : b]),
  B=([ c : r] # [ c : w]),
  nl, write( 'A='), write( A),
  nl, write( 'B='), write( B),
  unify( A, B, U),
  nl, write('U='), write(U).
?- A = ([ m : t] # [ m : d] # [ c : w]),
  B = ([ c : r]#[ c : w]),
  unify( A, B, U),
  nl, write( 'A='), write( A),
  nl, write( 'B='), write( B),
  nl, write( 'U='), write( U).
?- nl, write( 'Indexed FS ...').
?- A = [ agr : [ nr : sg, cas : nom],
  subj : 2^[agr : [ cas : nom,nr : sg]]],
  B = [ agr : 1^2^[], subj : [ agr : 3^1^[]]],
  unify_i( A, B, U),
  nl, write( 'A='), write( A),
  nl, write( 'B='), write( B),
  nl, write( 'U='), write( U).
?-told.
```

po-multiset to po-multiset

A=[m:t]#[m:d]
B=[c:r]#[c:w]
U=[m:t,c:r]$^+${m:d,c:w]/[m:t,c:w]$^+${m:d,c:r]/
[m:d,c:r]$^+${m:t,c:w]/[m:d,c:w]$^+${m:t,c:r]

A=[m:t]#[m:d]#[c:b]
B=[c:r]#[c:w]
U=bottom

A=[m:t]#[m:d]#[c:w]
B=[c:r]#[c:w]
U=[c:r,m:d]$^+${c:w,m:t]/[c:w,m:t]$^+${c:r,m:d]/
[c:r,m:t]$^+${c:w,m:d]/[c:w,m:d]$^+${c:r,m:t]

Indexed FS ...

A=[agr:[nr:sg,cas:nom],
subj:2^[agr:[cas:nom,nr:sg]]]
B=[agr:1^2^[],subj:[agr:3^1^[]]]
U=[agr:[nr:sg,cas:nom],
subj:[agr:[cas:nom,nr:sg]]]