Extracting Sense Trees from the Romanian Thesaurus by Sense Segmentation & Dependency Parsing

Neculai Curteanu
Institute for Computer Science,
Romanian Academy, Iaşi Branch
ncurteanu@yahoo.com

Alex Moruz
Institute for Computer Science,
Romanian Academy, Iaşi Branch
Faculty of Computer Science,
"Al. I. Cuza" University, Iaşi
mmoruz@info.uaic.ro

Diana Trandabăț
Institute for Computer Science,
Romanian Academy, Iaşi Branch
Faculty of Computer Science, "Al.
I. Cuza" University, Iaşi
dtrandabat@info.uaic.ro

Abstract

This paper aims to introduce a new parsing strategy for large dictionary (thesauri) parsing, called *Dictionary Sense Segmentation & Dependency* (DSSD), devoted to obtain the sense tree, *i.e.* the hierarchy of the defined meanings, for a dictionary entry. The real novelty of the proposed approach is that, contrary to dictionary 'standard' parsing, DSSD looks for and succeeds to separate the two essential processes within a dictionary entry parsing: sense tree construction and sense definition parsing. The key tools to accomplish the task of (autonomous) sense tree building consist in defining the dictionary sense marker classes, establishing a tree-like hierarchy of these classes, and using a proper searching procedure of sense markers within the DSSD parsing algorithm. A similar but more general approach, using the same techniques and data structures for (Romanian) free text parsing is SCD (Segmentation-Cohesion-Dependency) (Curteanu; 1988, 2006), which DSSD is inspired from. A DSSD-based parser is implemented in Java, building currently 91% correct sense trees from DTLR (Dicționarul Tezaur al Limbii Române – Romanian Language Thesaurus) entries, with significant resources to improve and enlarge the DTLR lexical semantics analysis.

1 Introduction

Since the last decade, researchers have proven the need for machine readable dictionaries. The idea behind parsing a dictionary entry is the creation of a lexical-semantic tree of senses corresponding to the meanings that define the dictionary lexical entry. The aim of this paper is to introduce a new parsing strategy for thesauri shallow parsing, called *Dictionary Sense Segmentation & Dependency* (DSSD), devoted to the task of extracting the *sense tree*, *i.e.* the hierarchy of the lexical-semantics defined meanings for a dictionary entry. The concrete task which DSSD algorithm was used for is to obtain the sense tree from an entry of the Romanian Language Thesaurus (DTLR – Dicționarul Tezaur al Limbii Române) within the eDTLR research project (Cristea et al., 2007) devised for DTLR electronic acquisition and processing (Curteanu et al., 2007).

In order to obtain the sense tree for a head word, the dictionary entry is divided into primary and secondary senses, respecting a sense hierarchy introduced by sense markers. For the DTLR dictionary, the sense markers hierarchy (presented in Section 3) includes 5 levels. Those are, from the topmost level: *capital letter* markers (**A.**, **B.**, etc.), *Roman numeral* markers (**I.**, **II.**, etc.), *Arabic numeral* markers (**1.**, **2.**, etc.), *filled diamond* ♦ and *empty diamond* ◊. Besides the

Coling 2008: Proceedings of the workshop on Cognitive Aspects of the Lexicon (COGALEX 2008), pages 55–63
Manchester, August 2008

five levels, there exists also a special marker category, the so-called *literal enumeration*, consisting of *lowercase letter* markers **a)**, **b)**, **c)**, etc. The literal enumeration can appear at any of the 5 levels, as presented in Section 3.

Thus, using the sense markers, any dictionary entry is represented as a tree of senses, the lower levels being more specific instances of the higher levels.

For example, for the dictionary entry *verb*, the sense tree contains 3 senses corresponding to level 3, one of them having a sub-sense corresponding to level 5. Each sense/sub-sense can have its own definition (gloss) or examples.

```
<entry>
  <hw>VERB</hw>
  <senses>
    <marker level="3">1.
      <definition>…</definition>
      <marker level="5">◊
      <definition>…</definition>
      </marker>
    </marker>
    <marker level="3">2.
      <definition>…</definition>
    </marker>
    <marker level="3">3.
      <definition>…</definition>
    </marker>
  </senses>
</entry>
```

The presented method can be applied to any dictionary, provided that a hierarchy of the sense markers of the dictionary is established.

The paper is organized as follows: Section 2 points out the characteristic features of DSSD strategy, discussing the special relationship between DSSD and SCD *parsing strategy for general text*, on one hand, and between DSSD and the standard *dictionary entry parsing* (DEP), on the other hand. Section 3 presents the main components of the DSSD strategy: DTLR sense marker classes, their dependency hyper-tree structure, and the DSSD parsing algorithm. The final Section 4 discusses the current stage implementation (in Java) of the DSSD algorithm, exposing several parsed examples. Possible sources of error and ambiguity in the DSSD parsing process are discussed, and further developments of DSSD analysis software are outlined.

2 DSSD compared to Free Text Parsing and to Dictionary 'Standard' Parsing

This section outlines the origins of the DSSD idea, pointing out the connections between DSSD and free text parsing based on the SCD linguistic strategy (Curteanu 2006), on one hand, and between DSSD and dictionary *standard* parsing, *e.g.* (Neff, Boguraev; 1989), (Lemnitzer, Kunze; 2005), (Hauser, Storrer; 1993), on the other hand. The main difference (and positive feature) of the DSSD strategy compared to the standard approach to *dictionary entry parsing* (DEP), *e.g. LexParse* system in (Hauser, Storrer; 1993), (Kammerer; 2000), (Lemnitzer, Kunze; 2005), or JavaCC grammar-based parsing in (Curteanu, Amihaesei; 2004), is that DSSD *detached completely* the process of *sense tree building* from the process of *sense definition parsing*, within the DEP general task. This fact is clearly reflected in Fig. 2, which compares, at the macro-code level, the main four DEP operations for standard DEP and DSSD strategies.

2.1 SCD Marker Classes, Hierarchy, and Parsing Algorithms

DSSD parsing strategy involves a configuration of components that is similar (but less general) to the SCD (Segmentation-Cohesion-Dependency) parsing strategy, developed and applied to (Romanian) free text analysis (and generation) (Curteanu; 2006). The process of solving the parsing of DTLR entries have been inspired by the resemblance between the classes of DTLR sense markers and the SCD marker classes on one side, and between the *sense trees* of (DTLR) dictionary entries and the *discourse trees* of finite-clause dependency trees at sentence or paragraph levels on the other side. While discourse trees provide a formal similarity to the sense trees, nucleus–satellite rhetorical relations among discourse segments is quite different to the *subsumption relation* of *lexical semantics nature* among the sub-sense definitions (sub-senses) of a dictionary entry.

The *subsumption* relation is defined as follows: $sense_1$ *subsumes* $sense_2$ if (informally) $sense_1$ is *less informative* (or, more general) than $sense_2$, or if (formally) the sense tree of $sense_1$ is a (*proper*) *subtree* of $sense_2$. DSSD parsing of an

Parsing Strategy	SCD markers & DSSD markers	Semantics to be applied on the parsed textual spans	Resulted structures of the parsing process
SCD	**M4-class** (discourse) markers	rhetorical discourse semantics , *i.e.* RST discourse (high-level cohesion) dependencies	*discourse tree* (of RST-based discourse segments)
	M3-class (inter-clause) markers	inter-clause predicational semantics, *i.e.* Predicate-Argument (global-level cohesion) dependencies among finite clauses	clause-level dependency trees based on syntactic or semantic relations
	M2-class (clause) markers	single finite-clause predicational semantics, *i.e.* Predicate-Argument (local-level cohesion) dependencies among VG-NGs (Verbal Group – Noun Groups)	single finite clause(s)
	M1-class intra-clausal (phrase) markers	non-finite predicational semantics, *i.e.* (local-level cohesion) dependencies *inside* VG and NGs (Verbal Group – Noun Groups)	simple and complex VGs; simple and complex (predication-related) NGs
	M0-class flexionary markers of lexical categories	lexical semantics categories	lexical textual words = inflected words
SCD - DSSD	**M(–1)-class** of lemmatization markers for DTLR lexical entries	semantic description at the *lexicon* level	lexical lemmatized words = dictionary entries
DSSD	**sense and subsense definition markers** of a DTLR lexical entry	*subsumption* relations between the *subsenses* of a DTLR lexical entry (*cohesion-free* semantics)	*sense trees* and (XCES-TEI 2007 codification-based) *sense definitons* of DTLR entries

Fig. 1. DSSD *vs.* SCD marker classes, the corresponding semantics and textual structures

entry sense tree works in an akin *Breadth-First*, *Top-Down* manner as SCD does, for those classes of markers that produce only *segmentation* and *binary dependency* between discourse segments or finite clauses, ignoring the more complex "*cohesion*" relationship. Thus one can rightly say that DSSD approach is *derived* from the SCD parsing strategy (Fig. 1).

SCD parsing strategy is exposed at large in (Curteanu 2006). SCD-based discourse parsing presents a special interest for DSSD because of their (formal) algorithmic analogy. The method proposed by the SCD strategy includes building the discourse tree by the intensive use of discourse markers, while discourse segments are obtained by clause parsing. Employing the results of the SCD clausal parsing and a database which contains information about the discourse markers, one can obtain the discourse structure of a text. The outcome is represented as a *discourse tree* whose terminal nodes are clause-like structures, having specified on the arcs the name of the involved *rhetorical relations*.

The SCD segmentation / parsing algorithm in (Curteanu 2006) may have the same shape of a *Breadth-First* (or sequential-linear) processing form as DSSD does, using as input a morphologically tagged text, obtaining the finite clauses and sub-clausal phrase (XG-)structures. Data representation is in standard XML and the implementation of the SCD algorithm for free text parsing is made in Java. (Curteanu 2006) presents *recursive Breadth-First* (and *Depth-First*), or *parallel Breadth-First* shapes of the SCD segmentation-parsing algorithms.

The relationship between SCD and DSSD parsing strategies, the former devoted to the free text parsing and the latter to be used for DEP, could be summarized as follows: the two strategies work formally with the same technology, using very similar analysis tools and data structures, including the same *Breadth-First* search strategy. The clear *distinction* between SCD and DSSD consists in the quite different kind of texts to be analyzed (free text *vs.* dictionary entry text), and the two different (but complementary) semantics that *drive* the corresponding parsing structures: *predicational* and *rhetorical* (cohesion-proper) semantics for SCD, and *lexical semantics* (cohesion-free) for DSSD. The table in Fig. 1 gives a detailed comparison between the two parsing strategies. The SCD parsing technology, especially with its presently discovered DSSD sub-sort, evolves (at least) three features: *generality* (different text structures), *flexibility* (different underlying semantics), and *adequacy* (proper text markers and their corresponding hierarchies).

Dictionary Classical Parsing Strategy	DSSD Parsing Strategy
For *i* from 0 to *MarkerNumber* ① **Sense-*i* Marker Recognition;** ② **Sense-*i* Definition Parsing;** *If*(Success) ③ **Attach (Parsed) Sense-*i* Definition to Node-*i*;** ④ **Add Node-*i* to EntrySenseTree;** *Else* Fail and Stop. EndFor **Output: EntrySenseTree** *with* **Parsed Sense Definitions** (only if all sense definitions are parsed). **Notice:** *MarkerNumber* is the number of the input marker sequence.	For *i* from 0 to *MarkerNumber* ① **Sense-*i* Marker Recognition;** **Assign (Unparsed) Sense-*i* Definition** to **Node-*i*;** ④ **Add Node-*i* to EntrySenseTree;** **Standby** on **Sense-*i* Definition Parsing;** EndFor **Output: EntrySenseTree.** **Node-*k*** = Root(**EntrySenseTree**); *While* not all nodes in **EntrySenseTree** are visited ② **Sense-*k* Definition Parsing;** *If*(Success) ③ **Attach Sense-*k* Definition to Node-*k*;** *Else* **Attach Sense-*k* Parsing Result to Node-*k*;** **Node-*k*** = getNextDepthFirstNode(**EntrySenseTree**) Continue EndWhile. **Output: EntrySenseTree** *with* **Parsed** *or* **Unparsed Sense Definitions**

Fig. 2. A macro-code comparison of classical and DSSD parsing strategies

2.2 DSSD Approach *vs.* Standard DEP

Another perspective on DSSD is outlined in this section: the novelties of DSSD approach fetched to the standard DEP, *e.g.* (Neff, Boguraev; 1989), (Lemnitzer, Kunze; 2005), (Kammerer, 2000). DSSD applies the same "technology" as SCD strategy does, *i.e. marker classes*, *specific hierarchies*, and adequate *searching procedures* embedded and governing the parsing algorithms. Most important, DSSD parse and *construct* the *sense tree* of a (DTLR) dictionary entry, *independently of*, and possibly *lacking the*, DTLR sense definition parsing process.

In the standard DEP, including the Java-grammar based construction of parsers in the JavaCC environment (Curteanu, Amihaesei, 2004; Curteanu et al., 2007), building the *sense tree* for an entry is inherently embedded into the general process of parsing *all* the sense and sub-sense definitions enclosed into the dictionary entry. In the same typically (standard) DEP way works also the parser in (Neff, Boguraev; 1989) or *LexParse*, (Kammerer; 2000: 10-11) specifying that the *LexParse* recognition strategy is a *Depth-First*, *Top-Down* one.

The advantage of the proposed DSSD approach is that it "ignores", at least in the beginning, the "details" of sense definitions, concentrating only on the *sense marker* discovery and their dependency establishing. The result is that DSSD parsing concentrates on and obtains, in the first place, the *sense tree* of a DTLR entry. Of course, parsing of a dictionary entry does not means *only* its sense tree, but the entry sense tree represents the essential, indispensable structure for *any* kind of DEP.

Based on different types of DTD standards for dictionary text representation, such as CONCEDE-TEI (Erjavec et al. 2000; Kilgarriff 1999, Tufis 2001) or (XCES-TEI; 2007), the parsing process may continue "in depth" for identifying the (other important) fields of sense and sub-sense definitions. DSSD strategy has the quality of being able to compute independently the entry sense tree, prior to the process of sense definition parsing. Subsequently, the process of parsing the sense definitions can be performed separately, one by one, avoiding the current situation when the general parsing of an entry may be stopped simply because of a single (even if the last one) unparsable sense definition.

The procedural *pseudo-code* in Fig. 2 shows clearly the important difference between *standard* DEP and DSSD *parsing*, with the essential advantage provided by DSSD: standard DEP is based on *Depth-First* search, while DSSD works with *Breadth-First* one. Specifically, the procedural running of the four operations that are compared for the standard DEP and DSSD strategies, labeled with ①, ②, ③, ④, are organized in quite different cycles: in the table left-side (standard DEP), there is a single, large running cycle, ① + ②, under ② being embedded (and strictly depending) the sub-cycle ③ + ④. The DSSD parsing exhibits two distinct (and in-

dependently) running cycles: ① + ④, for constructing the (DTLR) sense trees, and ② + ③, devoted to parse the sense definitions and to attach the parsed or unparsed sense definitions to their corresponding nodes in the sense tree(s).

We emphasize firstly, that the second procedural cycle is optional, and secondly, that the first cycle is working on the sense marker sequence of the entry (either correct or not), the DSSD output being an entry sense tree in any case (either correct or not). This is why the DSSD algorithm never returns on FAIL, regardless whether the obtained sense tree is correct or not.

3 DTLR Marker Classes, their Dependency Structure, and the DSSD Parsing Algorithm

As already pointed out, DSSD can be viewed as a simplified version of SCD, since only the *segmentation* and *dependency* aspects are involved, the (local) *cohesion* matters being without object for the (one-word) lexical semantics of DSSD. As in the case of SCD, the DSSD parsing strategy requires a set of *marker classes* (in our case, DTLR sense markers), arranged in a *hierarchy* illustrated in Fig. 3, and described below:

The *capital letter* marker class (**A.**, **B.**, etc.) is the topmost level on the sense hierarchy of DTLR markers (see Fig. 3) for any given dictionary entry. When it appears, this marker designates the (largest-grained meaning) *primary senses* of the lexical word defined. If the top level marker has only one element of this kind, then the marker is not explicitly represented.

The *Roman numeral* marker class (**I.**, **II.**, etc.) is the *second-level* of sense analysis for a given DTLR entry. It is subsumed by a capital letter marker if some exists for the head word; if a capital letter marker does not exist (it is not explicitly represented), the Roman numeral marker appears on the topmost level of the sense tree. If the lexical entry has only one sense value for this analysis level, the marker is not explicitly represented.

The *Arabic numeral* marker class (**1.**, **2.**, etc.) is the *third-level* of sense analysis for a DTLR entry. It is subsumed by a Roman numeral marker if there exists some for the entry; if a Roman numeral marker is not explicitly represented, it is subsumed by the first explicit marker on a higher level. If the entry has only one sense value for this level of sense analysis, the marker is not explicitly represented. These first *three levels* encode the *primary senses* of a DTLR lexical entry.

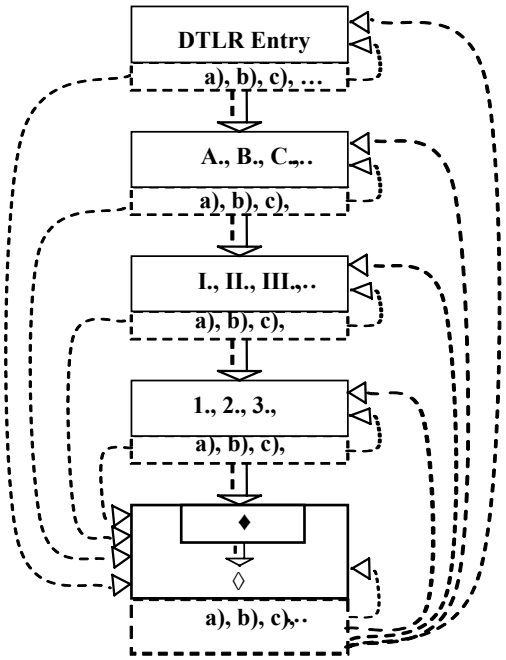

Fig. 3. The tree-like dependency structure for the classes of DTLR markers

The *filled diamond* marker class is the *fourth-level* of sense analysis and it is used for enumerating *secondary* (finer-grained) *senses* of a DTLR entry. It is generally subsumed by any explicit DTLR sense marker on a higher level, *i.e.* any of the primary sense markers.

The *empty diamond* marker class is the *fifth-level* of sense analysis and it is used for enumerating expressions for a given, *secondary sub-sense*. It is generally subsumed by a filled diamond marker or by any primary sense marker.

The *lowercase letter* markers **a)**, **b)**, **c)**, etc. are not an actual class of sense markers, but rather a *procedure* used to refine, through *literal enumeration*, a semantic paradigm of a DTLR entry sense or sub-sense. A lowercase letter marker does not have a specific level on the marker class tree-like hierarchy since it belongs to the sense marker level (of either primary or secondary sense) that is its parent. The important rules of the *literal enumeration* procedure in DTLR are: **(a)** it associates with the hierarchy level of the sense marker class to which is assigned (in Fig. 3), and **(b)** it can embed lower (than its parent level) senses, provided that each literal enumeration is closed finally on the sense level to which it belongs.

Fig. 3 is a *hyper-tree hierarchy* of the DTLR sense marker classes since (at least) the lowest hyper-node contains recursively embedded dia-

mond-marker nodes. The *dashed arrows* point to the upper or lower levels of DTLR sense marker hierarchy, from the *literal enumeration* layer-embedded level. The *continuous-dashed arrows* in Fig. 3 point downwards from the higher to the lower priority levels of DTLR marker class hyper-tree. Because of its special representation characteristics, the literal enumeration is illustrated on a layer *attached* to the hierarchy level to which it belongs, on each of the sense levels. Some examples supporting the marker hierarchy in Fig. 3, including the literal enumeration that can appear at any DTLR sense level, are presented below:

I. Literal enumeration under a filled diamond (secondary sense):

```
<entry>
  <hw>VÍȚĂ[2]</hw>
  <pos>s. f.</pos>
  <senses>
    <marker>I.
      <marker>1.
      <definition> (De obicei determinat prin „de vie") Arbust din familia vitaceelor, cu rădăcina puternică, cu tulpina scurtă, …</definition>
        <marker>♦
        <definition> C o m p u s e: viță-albă = </definition>
          <marker>a)
          <definition> arbust agățător din familia ranunculaceelor, cu tulpina subțire, cu frunze penate...; </definition>
          </marker>
          <marker>b)
          <definition>(regional) luminoasă (Clematis recta). Cf. CONV. LIT. XXIII, 571, BORZA, D. 49, 301; </definition>
          </marker>
          <marker>c)
          <definition>(învechit) împărătească (Bryonia alba).....</definition>
          </marker>
        </marker>
      </marker>
    </marker>
  </senses>
</entry>
```

II. Literal enumeration under an Arabic numeral (primary sense):

```
<entry>
  <hw>VERIGÚȚĂ</hw>
  <pos>s. f.</pos>
  <senses>
    <definition>Diminutiv al lui v e r i g ă. Cf. LB, POLIZU, DDRF, BARCIANU, ALEXI, W., TDRG, CADE, SCRIBAN, D., DL, DM, DEX.</definition>
    <marker>1.
      <marker>a)
      <definition> (Prin Transilv. şi prin sudul Mold.) Cf. v e r i g ă (2 c). Cf. ALR II 6 653/95, 192, 605.
</definition>
      </marker>
      <marker> b)
      <definition>Cf. v e r i g ă (2 b). Şi am dat cercel în narea ta şi veriguţe în urechile tale. BIBLIA (1688), 543[1]/25. La ferestre spînzurau nişte perdeluţe de adamască, aninate în nişte veriguţe ce se înşirau pe o vargă de fier. GANE, N. II, 160. </definition>
      </marker>
    </marker>
    <marker>2.
    <definition> (Popular) Verighetă. Cf. SCRIBAN, D., ȚIPLEA, P. P., BUD, P. P. Mi-o dat o veriguță Şi-ntr-on an i-am fost drăguță. BÎRLEA, C. P. 143. </definition>
    </marker>
  </senses>
</entry>
```

III. Literal enumeration directly under the entry root:

```
<entry>
  <hw>VENTRICÉA</hw>
  <pos>s. f.</pos>
  <senses>
    <definition> Numele mai multor specii de plante erbacee (folosite în medicină): </definition>
    <marker>a)
    <definition> ventrilică (c) (Veronica persica). Cf. GRECESCU, FL. 442, PANȚU, PL., CADE. Un gorun negru şi singuratic… e năpădit la poale de ventricele cu spicuri albăstrii....; </definition>
    </marker>
    <marker>b)
    <definition> ventrilică (a) (Veronica officinalis). Cf. TDRG, BORZA, D. 179, 300; </definition>
    </marker>
    <marker>c)
    <definition>bobornic (Veronica prostrata). Cf. BORZA, D. 179, 300. </definition>
    </marker>
  </senses>
</entry>
```

The DSSD algorithm for the construction of the DTLR sense tree, according to the marker hierarchy described in Fig. 3, is the following:

```
Stack S
Tree T
S.push(root)
while article has more markers
  crt = get_next_marker()
  while crt > S.top() – get to the
first higher rank marker in the
stack
    S.pop()
  if(crt = lowercaseLetter)
    S.top.addPart(crt) – add a low-
ercase marker as a subset of the
higher level sense value
```

```
      crt.level=S.top.level+1 - the
  lowercase letter maker is given a
  level in accordance to the level of
  its parent
      S.push(crt)
    else
      S.top.add_son(crt) - add the
  son to the higher level marker in
  the stack
      S.push(crt) - add the current
  marker to the stack
```

The DSSD parsing algorithm was implemented in Java and running examples of its application on DTLR entries are presented in Section 4. While the DTLR sense marker recognition in DSSD is achieved with a *Breadth-First* search, the marker sequence analysis for sense tree construction is based on a *Depth-First* parsing of the sense marker sequence input, as it uses a stack to keep track of previous unfinished (in terms of attaching subsenses) sense markers.

4 DTLR Parsing with DSSD Algorithm: Examples and Developments

4.1 DSSD Parser Applied on DTLR Entries

The enclosed Fig. 4 shows the result of applying the DSSD Java parser described in Section 3 on a DTLR entry. We notice that the presented input example (*VENIT*[2]) represents just sequences of DTLR sense markers. The entry for which the parsing was conducted is given only as tags, in part below (the entire entry spans for more than two dictionary pages):

```
<entry>
  <hw><VENÍT², -Ă </hw>
  <pos>adj. </pos>
  <senses>
    <definition>...</definition>
    <marker>1.
    <definition>...</definition>
    <marker>2.
    <definition>...</definition>
      <marker>◊
        <marker> a)
        <definition>...</definition>
        </marker>
        <marker> b)
        <definition>...</definition>
        </marker>
        <marker> c)
        <definition>...</definition>
        </marker>
      </marker>
      <marker>◊
        <marker> a)
        <definition>...</definition>
        </marker>
        <marker> b)
        <definition>...</definition>
        </marker>
      </marker>
    </marker>
  </senses>
</entry>
```

```
− <entry>
    <list>VENIT2, -Ă 1. 2. ◊ a) b) c) ◊ a) b) n-11</list>
  − <node value="VENIT2, -Ă" class="0">
      <node value="1." class="6"> </node>
    − <node value="2." class="6">
      − <node value="◊" class="10">
        − <parts>
            <node value="a)" class="11"> </node>
            <node value="b)" class="11"> </node>
            <node value="c)" class="11"> </node>
          </parts>
        </node>
      − <node value="◊" class="10">
        − <parts>
            <node value="a)" class="11"> </node>
            <node value="b)" class="11"> </node>
          </parts>
        </node>
      </node>
    </node>
  </entry>
− <entry>
```

Fig. 4. DSSD parsing for the sense tree building of DTLR entry **VENIT²**

As one can see, the input of the sense tree parser is the DSSD marker sequence of the considered DTLR entry (the <list> tag in Figure 4). The output of the parsing is much less verbose than the original dictionary entry, since the sense definitions and the entire example text is not depicted, in order to better observe the sense tree of the entry. Also, this representation proves that the understanding of the sense definitions is not strictly necessary for building the sense tree, a task for which the marker hierarchy discussed in Section 3 is sufficient.

Fig. 5 presents the sense tree for the dictionary entry "VIÉRME" (En: *worm*). It can be seen that this particular entry is quite large, with the original dictionary text spanning for more than *six* pages of DTLR thesaurus.

After its completion, the DSSD parser was tested on more than 500 dictionary entries (of medium and large sizes), the only ones already in electronic format to which we had access to at the moment (the vast majority of dictionary volumes is only available in printed form). The success rate was determined to be 91.18%, being

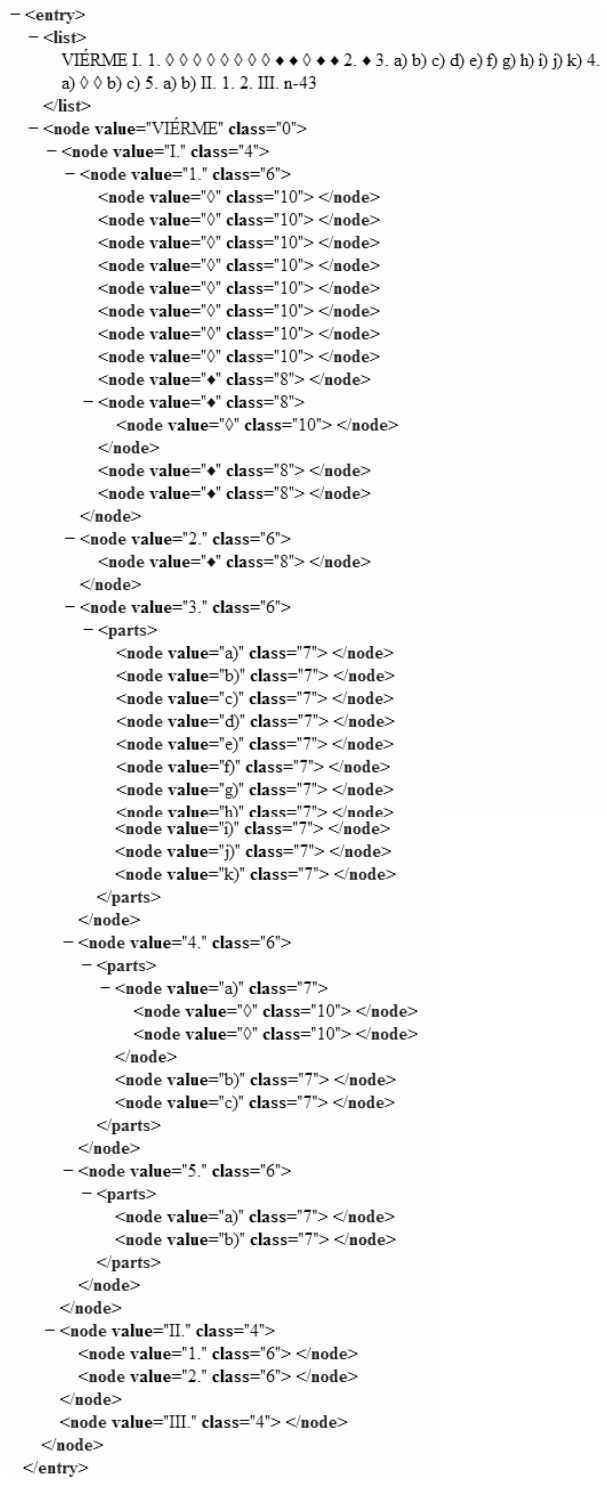

Fig. 5. Sense tree for the dictionary entry "VIÉRME"

computed as a perfect match between the output of the program and the gold standard.

Furthermore, it is worth noting that an article with only *one* incorrect parse (*i.e.* one node in the sense tree attached incorrectly) was considered to be erroneously parsed in its entirety, an approach which disregards all the other correctly attached nodes in that entry. This fact shows also significant improvement resources of the DSSD parsing process.

4.2 Error and Ambiguity Sources in DTLR Parsing

It is worth to mention some sources of error and ambiguity found in DSSD parsing for DTLR sense tree computing. We grouped the error sources in three main classes:

I. Inconsistencies in writing the original DTLR article

A first source of parsing errors is the non-monotony of the marker values on the same level of sense marker hierarchy (Fig. 3):

Ex.1. **A.** [**B.** missing] … **C.** etc.;

Ex.2. **2.** [instead of **1.**]... **2.** etc.;

Ex.3. **a)**… **b)** … **c)** … **b)** [instead of **d)**]etc.

The tree structure returned by the parser does not consider the consistency of each marker level. Thus, in **Ex.1**, it will place the two markers **A.** and **C.** as brother nodes in the sense tree. A (partial but feasible) solution for the parser is to check the *strict monotony* of the marker neighbors, an operation which is useful also when sense markers interfere with literal enumeration.

A validity of the marker succession at each level will be checked after the completion of the sense tree parsing.

II. Ambiguity in deciding which is the regent and which is the dependent (sub)sense

An inherently ambiguity was found for the following sequences of DTLR sense markers:

Ex.4. **1. a) b) c)** ◊ [◊]

The problem occurs since one can not discern between attaching the first (and / or second) "◊" as depending on **c)** or on the upper level marker (**1.)**. Solving these ambiguities is a problem related on syntactic and / or semantic contexts of the involved (multiple) pairs of markers. Namely, if "**c)**" is the last small letter in the literal enumeration, then "◊" is attached to the "**1.**" marker (and sense), while if "**c)**" in the literal enumeration, followed by "◊", has a continuation "**d)**" in the literal enumeration, then "◊" depends on its small letter "**c)**" regent sense.

III. More refined subsense classification

A third source of errors when creating the sense tree is met within the following marker sequence **I. 1. ♦ a) b) c) a) b)**. Even if at a quick

look the problem with the inconsistent literal enumeration is similar to the problems presented in the first class, at a closer inspection we realized that under the full diamond ♦ there are three subsenses (three expressions), two of them having literal enumeration: (1) **viță-albă = a)... b)... c)**; (2) **viță-neagră = ...**; (3) **vița-evreilor = a)...b)**. To solution this problem makes necessary a more refined subsense classification within the sense definition and adding possible new markers to the hierarchy. Working to solve these problems is in good progress, as it concerns types of sense structure closely related to various sense definition parsing, the next step in the development of the DSSD dictionary parser.

We already identified *seven* definition types, encoded as follows, together with the most important *dependency conditions* among the definitions below, within DTLR senses and subsenses:

1. *MorfDef* (Morphological Definitions);
2. *SpecDef* (Specification-based Definitions);
3. *SpSpec* (Spaced-character Definitions);
4. *RegDef* (Regular-font Definitions);
5. *BoldDef* (Bold-font Definitions);
6. *ItalDef* (Italic-font Definitions);
7. *ExemDef* (Example-based Definitions),

The 4, 5, 6, definition types are possibly followed by the *literal enumeration* scheme of sense codification.

Further developments of DSSD analysis software are meant to be achieved: **(a)** The complete parsing of a DTLR entry entails the natural extension of DSSD approach towards sense definition parsing and representation within the XCES TEI P5 (2007) standard set of tags. **(b)** A specialized subset of TEI P5 tags for representing all the types of definitions met within the primary and secondary senses of a DTLR entry is necessary. **(c)** Resolution of all the references within a DTLR entry is necessary: references to the excerpt sources (sigles), reference to a sense within the same entry (internal reference), or to a (sub)sense within another entry (external reference). **(d)** Verification of the sense-tree correctness can be achieved by restoring the linear structure of a DTLR entry from its parsed sense-tree representation, and comparing it with the DTLR original entry.

Acknowledgement. The present research was financed within the **eDTLR** grant, PNCDI II Project No. 91_013/18.09.2007.

References

Cristea, D., Răschip, M., Forăscu, C., Haja, G., Florescu, C., Aldea, B., Dănilă, E. (2007): *The Digital Form of the Thesaurus Dictionary of the Romanian Language*. In Proceedings of the 4th International IEEE Conference SpeD 2007.

Curteanu, Neculai (1988): *Augmented X-bar Schemes*. COLING'88 Proceedings, Budapest, pp. 130-132.

Curteanu, N., E. Amihăesei (2004): *Grammar-based Java Parsers for DEX and DTLR Romanian Dictionaries*. ECIT-2004 Conference, Iasi, Romania.

Curteanu, N. (2006): *Local and Global Parsing with Functional* (F)X-*bar Theory and SCD Linguistic Strategy*. (I.+II.), Computer Science Journal of Moldova, Academy of Science of Moldova, Vol. 14 no. 1 (40):74-102 and no. 2 (41):155-182.

Curteanu, N., D. Trandabăț, G. Pavel, C. Vereştiuc, C. Bolea (2007): *eDTLR – Thesaurus Dictionary of the Romanian Language in electronic form.* Research Report at the PNCDI II Project No. 91_013/18.09.2007, Phase 2007, and (D. Cristea, D. Tufiș, Eds.) *eDTLR Parsing – The Current Stage, Problems, and Development Solutions*, Romanian Academy Editorial House (in Romanian – to appear).

DLR Revision Group (1952): Codification Rules for the Dictionary (Thesaurus) of the Romanian Language. Institute of Philology, Bucharest, Romanian Academy.

Erjavec, T, Evans, R., Ide, N., Kilgariff A., (2000): The CONCEDE Model for Lexical Databases. Research Report on TEI-CONCEDE LDB Project, Univ. of Ljubljana, Slovenia.

Hauser, R., Storrer, A. (1993). *Dictionary Entry Parsing Using the LexParse System*. Lexikographica 9 (1993), 174-219

Kammerer, M. (2000): *Wöterbuchparsing Grundsätzliche Überlegungen und ein Kurzbericht über praktische Erfahrungen*, http://www.matthias-kammerer.de/content/WBParsing.pdf

Lemnitzer, L., Kunze, C. (2005): *Dictionary Entry Parsing*, ESSLLI 2005

Neff, M., Boguraev, B. (1989) *Dictionaries, Dictionary Grammars and Dictionary Entry Parsing*, Proc. of the 27th annual meeting on Association for Computational Linguistics Vancouver, British Columbia, Canada Pages: 91 - 101

Tufiș, Dan (2001): From Machine Readable Dictionaries to Lexical Databases, RACAI, Romanian Academy, Bucharest, Romania.

XCES TEI Standard, Variant P5 (2007): http://www.tei-c.org/Guidelines/P5/