ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΦΙΛΟΣΟΦΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΦΙΛΟΛΟΓΙΑΣ
ΤΟΜΕΑΣ ΓΛΩΣΣΟΛΟΓΙΑΣ

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Σύνδεση του προτύπου μορφολογικής επισημείωσης PAROLE με τον ενοποιητικό γραμματικό φορμαλισμό Lexical Functional Grammar (LFG). Εφαρμογή στα εργαλεία ILSP FBT Tagger και XLE/XEROX»

Τάσος Αλεξανδράκης

Α.Μ. 010191

Επιβλέποντες: Στέλλα Μαρκαντωνάτου, Γιάννης Μαίστρος, Χάρης Παπαγεωργίου

Τεχνογλωσσία

Αθήνα, Ιούνιος 2014

Περιεχόμενα

Εισαγωγή

Η εργασία αυτή τοποθετείται στα πλαίσια της Υπολογιστικής Γλωσσολογίας, ένα επιστημονικό πεδίο συνάντησης της Γλωσσολογίας και της Πληροφορικής. Εντάσσεται σε μια συλλογική προσπάθεια ανάπτυξης γραμματικών, για τη συντακτική ανάλυση προτάσεων στην ελληνική γλώσσα, στα πρότυπα του έργου ParGram (Parallel Grammar Project).

Το έργο ParGram ξεκίνησε το 1994 με τη συνεργασία του κέντρου ερευνών PARC, του πανεπιστημίου της Στουτγάρδης και του ευρωπαϊκού κέντρου ερευνών της XEROX. Έχει σκοπό την ανάπτυξη παράλληλων γραμματικών βασισμένων στον ενοποιητικό φορμαλισμό Lexical Functional Grammar (LFG), για τα αγγλικά, τα γερμανικά και τα γαλλικά. Ο όρος *παράλληλες γραμματικές* υποδηλώνει ότι βασίζονται σε κοινές, συμφωνημένες μεταξύ των ερευνητών, γλωσσολογικές συμβάσεις. Ο στόχος του έργου είναι διττός. Στο θεωρητικό επίπεδο ενδιαφέρει η εξέταση της καθολικότητας της θεωρίας της LFG, η διερεύνηση και η βελτιστοποίηση των αρχών της. Στο πρακτικό επίπεδο πάλι, ενδιαφέρει η δημιουργία γλωσσικών πόρων για την ανάπτυξη διάφορων εφαρμογών, όπως πχ η μηχανική μετάφραση. Από τότε έχουν προστεθεί περισσότεροι φορείς και πλέον αναπτύσσονται γραμματικές για γλώσσες όπως τα Νορβηγικά, τα Ιαπωνικά, τα Χίντι/Ουρντού, τα Τούρκικα κ.α. (Butt et al., 2002)

Το εργαλείο που χρησιμοποιείται για την ανάλυση, παραγωγή και εκσφαλμάτωση των γραμματικών αυτών είναι η πλατφόρμα XLE της XEROX. Η πλατφόρμα αυτή συνδυάζεται με το XFST, έναν ανεξάρτητο μορφολογικό αναλυτή πεπερασμένων αυτομάτων και μαζί αποτελούν ένα πολύ ισχυρό μηχανισμό συντακτικής ανάλυσης (Butt et al., 1999).

Το 2013 και στα πλαίσια του Π.Μ.Σ. Τεχνογλωσσία 6, ξεκίνησε μια προσπάθεια για την ανάπτυξη παράλληλης γραμματικής για τα ελληνικά. Η παρούσα διπλωματική εργασία αποτελεί αναπόσπαστο μέρος της προσπάθειας αυτής. Έρχεται συγκεκριμένα να καλύψει την ανάγκη δημιουργίας λεξικών, κατάλληλων για χρήση από το XLE, αποφεύγοντας την εκ του μηδενός δημιουργία μορφολογικού αναλυτή.

Αντικείμενο, λοιπόν, της εργασίας είναι η ανάπτυξη του λογισμικού *Αντιγόνη*, το οποίο αξιοποιεί το μορφολογικό αναλυτή ILSP FBT Tagger του Ινστιτούτου Επεξεργασίας Λόγου (ΙΕΛ), για την παραγωγή λεξικού το οποίο στην συνέχεια χρησιμοποιείται από την πλατφόρμα XLE. Το λογισμικό εκτελείται ασύγχρονα σε σχέση με τα άλλα δυο εργαλεία, ώστε να δίνεται η δυνατότητα επέμβασης του χρήστη στο παραγόμενο λεξικό. Από θεωρητική σκοπιά, αντικείμενο της παρούσας εργασίας είναι η σύνδεση του προτύπου μορφολογικής επισημείωσης PAROLE, που

χρησιμοποιεί το ΙΕΛ, με τον ενοποιητικό γραμματικό φορμαλισμό Lexical Functional Grammar (LFG) του XLE.

Η παρούσα διπλωματική εργασία αποτελείται από πέντε ενότητες, ως εξής:

Στην πρώτη ενότητα παρουσιάζεται ο μορφολογικός αναλυτής του ΙΕΛ και ιδιαίτερα η έξοδός του.

Στη δεύτερη ενότητα παρουσιάζεται η θεωρία της LFG και η πλατφόρμα XLE, με έμφαση στο λεξικό και στο μηχανισμό των υποδειγμάτων (templates) που παρέχει.

Στην τρίτη ενότητα αναλύονται λεπτομερώς οι προδιαγραφές του λογισμικού *Αντιγόνη* και δίνονται συγκεκριμένα παραδείγματα αξιοποίησής τους από το χρήστη /γλωσσολόγο.

Στην τέταρτη ενότητα γίνεται αναφορά στη γλώσσα προγραμματισμού Python με την οποία δημιουργήθηκε το λογισμικό *Αντιγόνη* και τη μονάδα PLY (Python Lex & Yacc) με την οποία γίνεται η λεκτική και συντακτική ανάλυση των υποδειγμάτων. Επίσης, δίνεται μια συνοπτική τεκμηρίωση του κώδικα του λογισμικού.

Στην πέμπτη και τελευταία ενότητα δίνονται παραδείγματα από την πρακτική εφαρμογή του εργαλείου, η αξιολόγησή του, συμπεράσματα από τη χρήση του καθώς και προτάσεις επέκτασής του.

1. Ο μορφολογικός αναλυτής του ΙΕΛ

Ο μορφολογικός αναλυτής FBT Tagger του Ινστιτούτου Επεξεργασίας του Λόγου (ΙΕΛ) έγινε διαθέσιμος διαδικτυακά στα πλαίσια του έργου PANACEA (ICT-248064), το οποίο "αποβλέπει στην ανάπτυξη υποδομών για τον συνδυασμό γλωσσικών τεχνολογιών με στόχο την αυτόματη παραγωγή των τεράστιων γλωσσικών πόρων που απαιτούν τα σύγχρονα συστήματα Μηχανικής Μετάφρασης και Επεξεργασίας Φυσικής Γλώσσας"[1].

Η λειτουργία του έγκειται στο διαχωρισμό ενός κειμένου εισόδου σε λεκτικές μονάδες και την απόδοση σε αυτές ετικετών μορφολογικής πληροφορίας καθώς και του λήμματος. Αποτελεί μετεξέλιξη του Brill Tagger κι είναι εκπαιδευμένος σε ελληνικά κείμενα. Κάνοντας χρήση ενός σετ μορφολογικών επισημειώσεων που αποτελείται από 584 διαφορετικούς συνδυασμούς ετικετών, καταφέρνει να καλύψει το σύνολο των μορφολογικών ιδιαιτεροτήτων της ελληνικής γλώσσας (Papageorgiou et al., 2000).

Το εργαλείο αποτελείται από τη μονάδα τεμαχισμού παραγράφων, προτάσεων και λεκτικών μονάδων (tokenizer), τον μορφολογικό επισημειωτή (pos tagger) και τον λημματοποιητή (lemmatizer).

1.1 Η μονάδα τεμαχισμού

Το πρώτο στάδιο της μορφολογικής ανάλυσης είναι ο τεμαχισμός του κειμένου εισόδου σε παραγράφους, προτάσεις και λεκτικές μονάδες (tokens). Ο τεμαχισμός γίνεται με μηχανή που βασίζεται στα Αυτόματα Πεπερασμένων Καταστάσεων (Finite State Automata).

Αρχικά οι παράγραφοι χωρίζονται είτε με βάση τις ετικέτες παραγράφων όταν η είσοδος είναι ιστοσελίδες, είτε ανιχνεύοντας τους χαρακτήρες αλλαγής γραμμής όταν πρόκειται για απλό κείμενο. Οι προτάσεις χωρίζονται μέσα στα όρια των παραγράφων αρχικά μέσω των διαχωριστικών συμβόλων των προτάσεων (.;! κλπ). Με τη χρήση κανονικών εκφράσεων, αναγνωρίζονται οι λεκτικές μονάδες που περιέχουν τα παραπάνω σύμβολα, όπως είναι οι ηλεκτρονικές διευθύνσεις, οι αριθμοί, οι ημερομηνίες, οι συντομογραφίες κ.α. Τέλος, με τη χρήση ευρετικών μεθόδων, αναγνωρίζονται τεμάχια που έχουν χωριστεί λανθασμένα κι ενώνονται σε προτάσεις. Η αναγνώριση των λεκτικών μονάδων γίνεται εντός των ορίων κάθε πρότασης. Όπως και στη διαδικασία χωρισμού προτάσεων, λαμβάνονται υπόψη καταρχήν τα προφανή σημάδια ορίων των λεκτικών μονάδων, δηλαδή τα κενά, τα

[1] http://www.ilsp.gr/el/infoprojects/meta?view=project&task=show&id=10

σημεία στίξης και άλλα σύμβολα. Σε ένα δεύτερο στάδιο επεξεργασίας, αναγνωρίζεται το είδος κάθε λεκτικής μονάδας και της αποδίδεται αντίστοιχη ετικέτα όπως DATE για τις ημερομηνίες, ABBR για τις συντομογραφίες, PTERM για τα τερματικά σύμβολα, TOK για τις λέξεις (Prokopidis et. al, 2011)

1.2 Ο επισημειωτής

Ο επισημειωτής μερών του λόγου (POS tagger) λαμβάνει ως είσοδο τις λεξικές μονάδες και σε κάθε μια από αυτές αποδίδει ετικέτες με α) το μέρος του λόγου στο οποίο ανήκει, β) μια σειρά μορφολογικών χαρακτηριστικών όπως αριθμό, πτώση, χρόνο κλπ και γ) το βασικό τύπο (λήμμα) στον οποίο αντιστοιχεί. Ο επισημειωτής φέρει την ονομασία FBT από τη μέθοδο μετασχηματισμού που χρησιμοποιεί η οποία βασίζεται σε χαρακτηριστικά (feature based transformation). Ο επισημειωτής του ΙΕΛ είναι εκπαιδευμένος σε ελληνικά σώματα κειμένων τα οποία έχουν υποσημειωθεί αρχικά αυτόματα και έπειτα διορθωθεί χειρωνακτικά, καταλήγοντας σε ένα λεξικό 455 χιλιάδων λεκτικών μονάδων.

Ο αριθμός των ετικετών που χρησιμοποιεί είναι αρκετά μεγαλύτερος από εκείνον που χρησιμοποιούν άλλες γλώσσες και φτάνει τους 584 συνδυασμούς βασικών ετικετών, καθώς η ελληνική είναι μια γλώσσα με ιδιαίτερα πλούσια μορφολογία. Οι συνδυασμοί αυτοί αποτελούν το σετ μορφολογικών ετικετών του ΙΕΛ (ILSP PAROLE Tagset), μια προσαρμογή του προτύπου PAROLE, για την επισημείωση ελληνικών σωμάτων κειμένων.

No	Ουσιαστικό (Noun)
Aj	Επίθετο (Adjective)
Nm	Αριθμητικό (Numeral)
At	Άρθρο (Article)
Vb	Ρήμα (Verb)
Pn	Αντωνυμία (Pronoun)

Εικ.1 Οι ετικέτες για μερικά από τα κύρια μέρη του λόγου[2]

Η διαδικασία ξεκινά με την απόδοση αρχικής ετικέτας στη λεκτική μονάδα. Στην περίπτωση που η λέξη είναι καταχωρημένη στο λεξικό, αποδίδεται η ετικέτα με τη μεγαλύτερη συχνότητα για τη συγκεκριμένη λέξη. Το λεξικό έχει προκύψει από το σώμα εκπαίδευσης και έχει συμπληρωθεί με καταχωρήσεις από το Μορφολογικό Λεξικό του ΙΕΛ. Αν η λέξη δεν είναι καταχωρημένη, γίνεται χρήση λεξικού

[2] http://nlp.ilsp.gr/nlp/tagset_examples/tagset_en/index.html

καταλήξεων. Στη λέξη *συνοριακός* πχ, αποδίδεται η ετικέτα AjBaNeSgNm η οποία έχει αντιστοιχηθεί με την κατάληξη *–ιακός*. Μετά την αρχική επισημείωση, εφαρμόζονται μια σειρά από 800 περίπου κανόνες συμφραζομένων για τη διόρθωση της ετικέτας. Οι κανόνες αυτοί είναι αποτέλεσμα της χειρωνακτικής διόρθωσης των ετικετών που αποδόθηκαν κατά τη διαδικασία της αυτόματης επισημείωσης. Μπορούν δε να αλλάξουν την ετικέτα μόνο στην περίπτωση που αυτή συμπεριλαμβάνεται σε καταχώρηση του λεξικού.

```
AtDfNvNvNv  PnPeNvNvNvNvNv
NEXTTAG VbMnNvNvNvNvNvNvNvNv 462

NmCdNeNvNvNv  NmCdMaNvNvNv
NEXT1OR2TAG NoCmMaNvNv 37
```

Εικ.2 Παραδείγματα κανόνων συμφραζομένων

Το πρώτο από τα παραδείγματα κανόνων του σχ. 5 αντιστοιχεί στην αλλαγή της επισημείωσης μιας λεκτικής μονάδας από άρθρο σε αντωνυμία, όταν ακολουθεί ρήμα και το δεύτερο την αλλαγή του γένους ενός αριθμητικού από ουδέτερο σε αρσενικό όταν ακολουθεί αρσενικό ουσιαστικό. Η ακρίβεια του επισημειωτή αγγίζει το 97.49% όσον αφορά μόνο το μέρος του λόγου, και το 92.54% όταν λαμβάνονται υπόψη όλα τα μορφολογικά χαρακτηριστικά (Prokopidis et al., 2011).

1.3 Ο λημματοποιητής

Το τελικό στάδιο της επεξεργασίας αναλαμβάνει ο λημματοποιητής ο οποίος ανακτά τα λήμματα από το Μορφολογικό Λεξικό. Αυτό περιλαμβάνει 66 χιλιάδες λήμματα που στην κλιτή τους μορφή δίνουν ένα σύνολο δυο εκατομμυρίων καταχωρήσεων. Όταν υπάρχει αμφισημία στο λεξικό ο λημματοποιητής χρησιμοποιεί την πληροφορία του επισημειωτή. Έτσι στην περίπτωση της λέξης *γρήγορα* δίνει *γρήγορος* αν είναι επίθετο ή *γρήγορα* αν είναι επίρρημα (Prokopidis et al., 2011).

1.4 Η διαδικτυακή υπηρεσία

Ο μορφολογικός αναλυτής του ΙΕΛ είναι διαθέσιμος ως διαδικτυακή υπηρεσία στην ηλεκτρονική διεύθυνση http://nlp.ilsp.gr/soaplab2-axis/.

Εικ.3 Ο μορφολογικός αναλυτής του ΙΕΛ

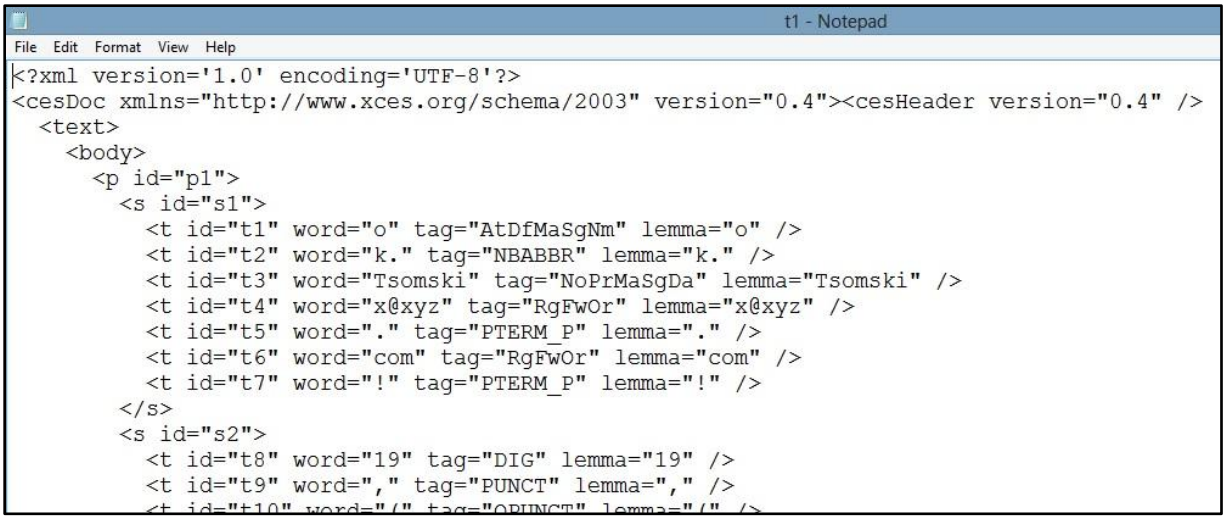
Δέχεται ως είσοδο είτε απλό κείμενο είτε αρχεία XCES, όπου το κείμενο είναι χωρισμένο σε παραγράφους. Η έξοδος της υπηρεσίας είναι ένα αρχείο σε μορφή XCES, ενώ υποστηρίζει και μορφές UIMA, GATE και GrAF[3].

Η έξοδος του μορφολογικού αναλυτή του ΙΕΛ αποτελεί την κύρια είσοδο του λογισμικού *Αντιγόνη*. Περιέχει καταχωρήσεις με τις λεκτικές μονάδες του κειμένου εισόδου του εργαλείου, επισημειωμένες με ετικέτες μορφολογικής πληροφορίας και με τα λήμματα στα οποία αντιστοιχούν. Η κωδικοποίηση του αρχείου γίνεται με το πρότυπο XCES, μια μετεξέλιξη του CES βασισμένη στο πρότυπο XML, ευρέως χρησιμοποιούμενη πλέον στις εφαρμογές επεξεργασίας φυσικής γλώσσας.

Σύμφωνα με αυτό, κάθε λεκτική μονάδα είναι καταχωρημένη με την ετικέτα <t… /> και περιέχει τις εξής πληροφορίες:

- Id Αύξων αριθμός
- word Η ίδια η λεκτική μονάδα
- tag Η μορφολογική επισημείωση
- lemma Το λήμμα στο οποίο αντιστοιχεί

[3] http://registry.elda.org/services/180



Εικ.4 Το αρχείο εξόδου του μορφολογικού αναλυτή

Η επισημείωση αποτελείται από ετικέτες δυο χαρακτήρων. Η πρώτη ετικέτα αναφέρεται στο μέρος του λόγου και οι επόμενες, στα μορφολογικά χαρακτηριστικά. Πχ στη λέξη *του* η επισημείωση AtDfMaSgGe το μέρος του λόγου είναι οριστικό άρθρο (At) και ακολουθούν τα μορφολογικά χαρακτηριστικά: γένος αρσενικό (Ma), αριθμός ενικός (Sg), πτώση γενική (Ge) ενώ το λήμμα είναι 'ο'.

Σε ορισμένες περιπτώσεις μπορεί να δίνεται κενή τιμή (Xx) σε κάποιο χαρακτηριστικό, όπως φαίνεται στα παρακάτω παραδείγματα:

```
<t id="t3" word="γράψω" tag="VbMnIdXx01SgXxPeAvXx" lemma="γράφω"/>
```

```
<t id="t1" word="όντας" tag="VbMnPpXxXxXxXxIpAvXx" lemma="ων"/>
```

Το ρήμα *(να) γράψω* δεν έχει χρόνο, γένος και πτώση ενώ η ενεργητική μετοχή *όντας* δεν έχει ούτε τα προηγούμενα ούτε πρόσωπο και αριθμό.

Στην περίπτωση των διαχωριστικών λεκτικών μονάδων (tokenizers), αυτές επισημειώνονται με μια μόνο ετικέτα η κάθε μία, χωρίς επιπλέον χαρακτηριστικά. Έτσι πχ τα σημεία στίξης έχουν τις ετικέτες PUNCT, P_TERM κλπ, οι συντομογραφίες ABBR και INIT, οι ημερομηνίες DATE, οι αριθμοί DIG και οι αριθμήσεις ENUM.

Παράδειγμα αρχείου εξόδου του εργαλείου παρατίθεται στο Παράρτημα Α.

2. Η γραμματική θεωρία LFG και η πλατφόρμα XLE

2.1 Η γραμματική θεωρία LFG

Η LFG (Lexical Functional Grammar) αποτελεί ένα φορμαλισμό για την αναπαράσταση της πολυεπίπεδης πληροφορίας που φέρουν τα εκφωνήματα της φυσικής γλώσσας. Ορίζει διαφορετικά επίπεδα για την κωδικοποίηση της φωνολογίας, της μορφολογίας, της σύνταξης, και της σημασιολογίας, κάθε ένα από τα οποία έχει δικούς του κανόνες και φόρμες. Όσον αφορά τη συντακτική ανάλυση, έχει δυο κύρια επίπεδα αναπαράστασης:

- τη δομή των συστατικών (*c-structure)*
- τη λειτουργική δομή (*f-structure)*

Η δομή των συστατικών (c-structure) αναπαριστά την επιφανειακή δομή μιας πρότασης σε δενδροειδή μορφή. Οι μη τερματικοί κόμβοι είναι τα φραστικά συστατικά, όπως οι ονοματικές και οι ρηματικές φράσεις (NP και VP αντίστοιχα), ενώ οι τερματικοί είναι οι λέξεις της πρότασης. Οι δομές αυτές προκύπτουν από την εφαρμογή φραστικών κανόνων, όπως θα δείξουμε και παρακάτω.

Η λειτουργική δομη (f-structure) παρέχει μια ακριβή αναπαράσταση των γραμματικών λειτουργιών που επιτελούν τα συστατικά. Η αναπαράσταση γίνεται συνήθως με πίνακες από ζευγάρια χαρακτηριστικών – τιμών. Οι πίνακες αυτοί αποκαλούνται συχνά δομές χαρακτηριστικών – τιμών (feature structures), όπου τα χαρακτηριστικά μπορεί να είναι τα μορφολογικά χαρακτηριστικά, όπως το γένος και η πτώση, ή οι γραμματικές συναρτήσεις όπως το *υποκείμενο* ή το *αντικείμενο* (Wescoat, 1987).

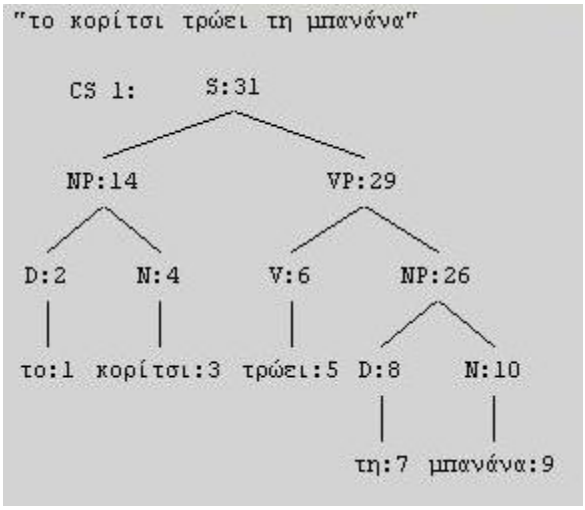
Θεωρούμε τους παρακάτω φραστικούς κανόνες (χρησιμοποιείται ο φορμαλισμός του XLE):

```
(1)   S ->   NP: (^SUBJ)=!
             VP: ^=!

      NP -> D
            N: ^=!


      VP -> V: ^=!
            NP: (^OBJ)=!
```

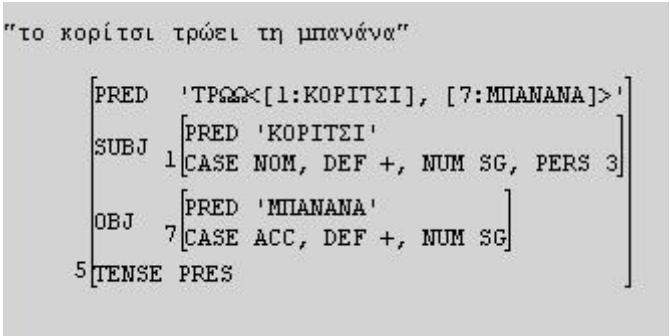
Για την πρόταση

(2) *το κορίτσι τρώει τη μπανάνα*

προκύπτουν οι παρακάτω δομές:



Εικ.5 Η δομή συστατικών (c-structure)



Εικ.6 Η δομή χαρακτηριστικών-τιμών (f-structure)

Οι δύο δομές συνδέονται μεταξύ τους με κατηγορηματικό τρόπο τον οποίο ορίζουν οι λειτουργικές εξισώσεις που συνοδεύουν τους φραστικούς κανόνες. Σε αυτές το '^' ('↑' στην LFG) αντιστοιχεί στο γονέα κάθε κόμβου και το '!' ('↓' στην LFG) στον ίδιο τον κόμβο. Έτσι πχ στο (1) το (^SUBJ)=! ορίζει ότι η πρόταση S έχει το χαρακτηριστικό SUBJ, η τιμή του οποίου είναι η ονοματική φράση NP. Η εξίσωση ^=! δηλώνει ότι η απεικόνιση της πρότασης S στη λειτουργική δομή ταυτίζεται με αυτήν τη ρηματικής φράσης VP. Η σύνδεση αναπαριστάται στο XLE με τους αριθμούς που εμφανίζονται στις δυο δομές. Στη λειτουργική δομή εμφανίζεται ο αριθμός του πρώτου τερματικού κόμβου που απεικονίζεται σε κάθε υποδομή (Fiotaki, 2012).

Οι λειτουργικές δομές οφείλουν να ικανοποιούν τρεις αρχές: της πληρότητας (completeness), της συνοχής (coherence) και της μοναδικότητας (uniqueness). Η πρώτη αποκλείει προτάσεις όπως *το κορίτσι τρώει*, αν έχει δοθεί στο λεξικό (μόνο) η καταχώρηση:

```
(3) τρώω V (^PRED) = 'τρώω (^SUBJ) (^OBJ)'
```

η οποία δηλώνει ότι το ρήμα *τρώω* έχει υποκείμενο και αντικείμενο.

Η δεύτερη, που είναι η αντιστροφή της πρώτης, αποκλείει προτάσεις όπως τη (2), αν έχει δοθεί στο λεξικό (μόνο) η καταχώρηση:

```
(4) τρώω V (^PRED) = 'τρώω (^SUBJ)'
```

η οποία δηλώνει ότι το ρήμα *τρώω* έχει μόνο υποκείμενο.

Η τρίτη ορίζει ότι σε μια λειτουργική δομή κάθε χαρακτηριστικό μπορεί να έχει μέχρι μια τιμή, δηλαδή οι λειτουργικές δομές είναι συναρτήσεις.

Τέλος, θεμελιώδης αρχή της LFG, η οποία στηρίζεται στο σαφή διαχωρισμό μεταξύ σύνταξης και μορφολογίας, είναι η Λεξική Ακεραιότητα (Lexical Integrity Principle). Σύμφωνα με αυτήν κανένας φραστικός κανόνας δε μπορεί να αναφέρεται σε στοιχεία σε επίπεδο κάτω από τη λέξη. Πρακτικά, η αρχή αυτή αποτρέπει τους φραστικούς κανόνες από το να αναφέρονται ή να επιδρούν στην εσωτερική δομή των λέξεων (Lapointe, 1980)

2.2 Η πλατφόρμα XLE[4]

Το XLE είναι μια πλατφόρμα ανάπτυξης γραμματικών LFG μεγάλης κλίμακας (Kaplan & Bresnan, 1982) η οποία δημιουργήθηκε το 1993 από το PARC (Palo Alto Research Center Incorporated), για λογαριασμό της Xerox. Παρέχει στους γλωσσολόγους ένα εύχρηστο περιβάλλον για τη συγγραφή συντακτικών κανόνων και λεξικών καταχωρήσεων καθώς και προαιρετικά τη δημιουργία μορφολογικού αναλυτή. Το σύστημα με βάση τα παραπάνω αναλύει φυσική γλώσσα σε επίπεδο πρότασης ή φράσης και παρέχει αναλυτικές πληροφορίες, όπως τη δομή των προτάσεων/φράσεων (c-structures), τις λειτουργικές εξαρτήσεις (f-structures) και τη μορφολογία κάθε λεξικής καταχώρησης.

Αποτελεί ένα ισχυρό εργαλείο, που παρέχει μεγάλη ακρίβεια ανάλυσης με μικρή επιβάρυνση χρόνου, συγκρινόμενο ακόμη και με έναν shallow parser (Kaplan, 2004, Denis & Kuhn, 2006)

Δίνει τη δυνατότητα στο γλωσσολόγο να δοκιμάσει, να συνδυάσει και να διορθώσει διάφορες γραμματικές και λεξικά, παρέχοντας μάλιστα βοηθητικά εργαλεία για την ανάλυση της αποδοτικότητας, την εκσφαλμάτωση των γραμματικών, τη δημιουργία διαγνωστικών τεστ και την αποθήκευση των αναλύσεων.

Είναι υλοποιημένος στη γλώσσα C και τρέχει σε συστήματα Solaris, Linux, MacOSX ενώ τελευταία έχει αναπτυχθεί και έκδοση για το λειτουργικό Windows.

[4] http://www2.parc.com/isl/groups/nltt/xle/doc/xle_toc.html

2.2.1 Το αρχείο γραμματικής

Μια γραμματική XLE μπορεί να περιέχεται σε ένα βασικό αλλά και σε περισσότερα αρχεία κειμένου. Οι πληροφορίες του βασικού αρχείου δομούνται σε ενότητες οι βασικότερες των οποίων παρουσιάζονται παρακάτω:

Η ενότητα ρυθμίσεων (CONFIG), όπου ορίζονται οι υπόλοιπες ενότητες, τα επιπλέον αρχεία που χρησιμοποιεί η γραμματική (λεξικά πχ) και έννοιες όπως οι κυβερνούμενες γραμματικές συναρτήσεις κλπ.

1. Οι φραστικοί κανόνες (RULES)
2. Τα υποδείγματα (TEMPLATES)
3. Το λεξικό (LEXICON)
4. Άλλες ενότητες όπως η μορφολογία και η δήλωση των χαρακτηριστικών

```
TOY    ENGLISH   CONFIG (1.0)
  ROOTCAT   S.
  FILES   .
  LEXENTRIES    (TOY ENGLISH).
  RULES    (TOY ENGLISH).
  TEMPLATES    (TOY ENGLISH).
  GOVERNABLERELATIONS     SUBJ OBJ OBJ2 OBL OBL-?+ COMP XCOMP.
  SEMANTICFUNCTIONS     ADJUNCT  TOPIC.
  NONDISTRIBUTIVES     NUM PERS.
  EPSILON   e.
  OPTIMALITYORDER      NOGOOD.

----
TOY   ENGLISH   RULES (1.0)


----
TOY   ENGLISH   TEMPLATES (1.0)

----
TOY   ENGLISH   LEXICON (1.0)

----
```

Εικ. 7 Το αρχείο γραμματικής XLE

Η μέχρι τώρα εμπειρία από το έργο ParGram έχει δείξει ότι η χρήση περισσότερων του ενός αρχείων είναι αναγκαία για γραμματικές μεγάλης κλίμακας.

Παράδειγμα αρχείου γραμματικής XLE παρατίθεται στο Παράρτημα Γ.

2.2.2 Ανάγκη για modularity

Οι γραμματικές μεγάλης κλίμακας έχουν πολλά κοινά με τα έργα λογισμικού ανάλογης κλίμακας, όσον αφορά τον τμηματικό τρόπο ανάπτυξής τους, δηλαδή τη χρήση των αρθρωμάτων (modules). Τα αρθρώματα αυτά είναι τμήματα κώδικα που αφενός αποτελούν λογικές μονάδες, με την έννοια ότι επιτελούν μια συγκεκριμένη λειτουργία, και αφετέρου νοούνται ως ‘’μαύρα κουτιά’’, ενδιαφέρει δηλ. ο τρόπος αλληλεπίδρασής τους με το υπόλοιπο λογισμικό αλλά όχι το πώς είναι

κατασκευασμένα εσωτερικά. Τα πλεονεκτήματα αυτής της προσέγγισης είναι θεωρητικά τεκμηριωμένα και πρακτικά αποδεδειγμένα και αφορούν στη διασφάλιση περισσότερης διαφάνειας και συνεκτικότητας, καθώς και καλύτερης συντηρησιμότητας και τεκμηρίωσης του κώδικα (Dipper, 2004).

Στην περίπτωση των γραμματικών, η προσέγγιση αυτή χρησιμεύει στην κωδικοποίηση γλωσσολογικών γενικεύσεων. Σημαντική συμβολή στο ρόλο αυτό έχουν στο XLE οι σύνθετες κατηγορίες (complex categories), οι λεξικοί κανόνες (lexical rules) και τα υποδείγματα (templates). Ειδικά τα τελευταία δίνουν τη δυνατότητα στο γλωσσολόγο να ενσωματώσει σε αυτά κώδικα από τις δομές συστατικών και τις λειτουργικές δομές. Επιπρόσθετα, η δυνατότητα ενσωμάτωσης που έχουν, η χρήση δηλαδή υποδειγμάτων σε άλλα υποδείγματα, τους επιτρέπει να μοντελοποιήσουν περιορισμούς όμοιους με τις ιεραρχίες τύπων, καθώς δημιουργούν δενδροειδείς ιεραρχικές δομές μεταξύ τους (Εικ. 8) (Dalrymple et al., 2004).

Η ανάπτυξη παράλληλων γραμματικών καθιστά την ανάγκη για κωδικοποίηση των γενικεύσεων ακόμα πιο επιτακτική. Αυτό συμβαίνει διότι γίνεται προσπάθεια τα κοινά, μεταξύ των διαφορετικών γλωσσών, γλωσσολογικά φαινόμενα, όπως η παθητική φωνή, η μεταβατικότητα των ρημάτων κ.α., να αντιμετωπίζονται με όσο το δυνατόν πιο ενιαίο τρόπο. Μια αποτελεσματική τακτική είναι η χρήση υποδειγμάτων με κοινή ονομασία (Butt et al., 1999).

2.2.3 Υποδείγματα

Η χρήση των υποδειγμάτων αποτελεί ένα από ιδιαίτερα γνωρίσματα της πλατφόρμας XLE. Μπορούν να νοηθούν ως ανεξάρτητες οντότητες οι οποίες υλοποιούν γενικεύσεις των χαρακτηριστικών (μορφολογικών, συντακτικών, σημασιολογικών) των λέξεων και που μπορούν να χρησιμοποιηθούν στο λεξικό, κυρίως, αλλά και στους φραστικούς κανόνες. Λειτουργούν καταρχήν ως μακροεντολές οι οποίες δίνουν στο γλωσσολόγο τη δυνατότητα να ομαδοποιήσει κοινά χαρακτηριστικά και να αναφέρεται σε αυτά με το όνομα του υποδείγματος. Για το ρήμα *τρέχει* πχ μια πιθανή καταχώρηση στο λεξικό θα μπορούσε να είναι:

```
(1)  τρέχει V * (^ SUBJ NUM)= SG
                (^ SUBJ PERS)=3
                (^ TENSE) = PRES
                (^ MOOD) = INDIC.
```

Ορίζοντας το υπόδειγμα:

```
(2)  V3SG =     (^ SUBJ NUM)= SG
                (^ SUBJ PERS) = 3
                (^ TENSE) = PRES
                (^ MOOD) = INDIC.
```

Η καταχώρηση στο λεξικό θα γινόταν:

```
(3)  τρέχει     V * @V3SG.
```

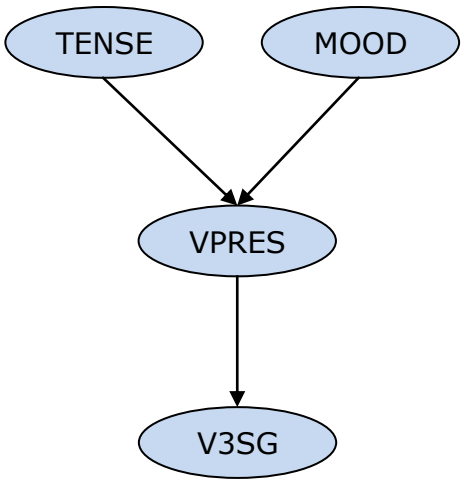
Οι λεξικές καταχωρήσεις (1) και (3) είναι ισοδύναμες, με την προϋπόθεση βέβαια ότι έχει οριστεί το (2).

2.2.3.1 Κλήσεις υποδειγμάτων

Τα υποδείγματα όμως, πηγαίνοντας ένα βήμα παραπέρα, δεν λειτουργούν απλά ως μακροεντολές, αλλά και ως συναρτήσεις. Αυτό σημαίνει ότι στον ορισμό τους, μπορούν να περιέχονται παράμετροι οι οποίες κατά την κλήση των υποδειγμάτων αυτών, να αντικαθιστούνται από τιμές ή μεταβλητές. Μάλιστα υπάρχει η δυνατότητα κλήσης ενός υποδείγματος από άλλα. Στην παρακάτω περίπτωση το υπόδειγμα V3SG καλεί το VPRES, το οποίο με τη σειρά του καλεί τα TENSE και MOOD με ορίσματα PRES και INDIC αντίστοιχα.

```
(4)  V3SG =     (^ SUBJ NUM)= SG
                (^ SUBJ PERS)=3
                @VPRES.
     VPRES =    @(TENSE PRES)
                @(MOOD INDIC).
     TENSE(T) =(^ TENSE) = T.
     MOOD(M) = (^ MOOD) = M.
```

Η ιεραρχία των παραπάνω υποδειγμάτων μπορεί να αναπαρασταθεί με την παρακάτω δενδροειδή δομή:



Εικ. 8 Η ιεραρχία των υποδειγμάτων

Για να γίνει η ενοποίηση θα πρέπει όλες οι κλήσεις των υποδειγμάτων να αντικατασταθούν με τα σώματά τους. Έτσι λοιπόν ο μηχανισμός του XLE θα μετατρέψει (εσωτερικά) πρώτα το VPRES σε:

```
(5)  VPRES =    (^ TENSE) = PRES
                (^ MOOD) = INDIC.
```

Τέλος θα κάνει την αντικατάσταση και στο V3SG για να καταλήξει στο (1).

Καθώς τα υποδείγματα καλούν το ένα το άλλο, υπάρχει η πιθανότητα αναδρομικής κλήσης, ένα υπόδειγμα δηλαδή να καλέσει τελικά τον εαυτό του. Σε αυτή την περίπτωση ο μηχανισμός του XLE εμφανίζει μήνυμα λάθους και αντικαθιστά την κλήση του υποδείγματος με τη σταθερά FALSE.

2.2.3.2 Τα υποδείγματα ως λογικές εκφράσεις

Στα παραπάνω παραδείγματα το κάθε υπόδειγμα λειτουργεί ως ομαδοποίηση χαρακτηριστικών. Ορίζει δηλαδή ότι η λεκτική μονάδα στην οποία αντιστοιχίζεται, θα έχει όλα τα χαρακτηριστικά με τις τιμές οι οποίες αναγράφονται. Παρόλα αυτά ο φορμαλισμός του XLE επιτρέπει τη λειτουργία των υποδειγμάτων ως πλήρεις λογικές εκφράσεις, ορίζοντας συζεύξεις, διαζεύξεις και αρνήσεις.

Στο (6) ορίζονται τρία υποδείγματα που καθορίζουν τη μεταβατικότητα των ρημάτων. Το υπόδειγμα OPT-TRANS λειτουργεί ως διάζευξη, η οποία έχει ως αποτέλεσμα διπλή καταχώρηση για κάθε λεκτική μονάδα στην οποία αντιστοιχίζεται.

```
(6)  TRANS(P) = (^ PRED) = 'P<SUBJ,OBJ>'.
     INTRANS(P) = (^ PRED) = 'P<SUBJ>'.
```

```
OPT-TRANS(P) = {@(TRANS P) | @(INTRANS P)}.
```

Έτσι η δήλωση:

```
(7)  τρώει V * @(OPT-TRANS τρώω).
```

Ισοδυναμεί με τις καταχωρήσεις:

```
(8)  τρώει V * (^ PRED) = 'τρώω <SUBJ,OBJ>'.
     τρώει V * (^ PRED) = 'τρώω <SUBJ >'.
```

Η άρνηση δηλώνεται με το σύμβολο '~' είτε μπροστά από μια έκφραση είτε μπροστά από το σύμβολο της ισότητας. Έτσι πχ τα υποδείγματα (9), τα οποία είναι ισοδύναμα[5], δηλώνουν ότι το χαρακτηριστικό TENSE δε μπορεί να έχει την τιμή PRES.

```
(9)  α. NOT_PRES1 = ~ (^ TENSE) = PRES.
     β. NOT_PRES2 = (^ TENSE) ~= PRES.
```

Τέλος, υπάρχει και η δυνατότητα ομαδοποίησης όρων σύζευξης με τη χρήση των αγκυλών. Αυτό είναι ιδιαίτερα χρήσιμο στην περίπτωση της άρνησης:

```
(10) VNOTPRES = ~[(^ TENSE) = PRES
                  (^ MOOD) = INDIC].
```

Η δομή των υποδειγμάτων αναλύεται λεπτομερώς στο 3.3.1.

2.2.3.3 Ο υπαρξιακός περιορισμός (existential constraint)

Υπάρχουν περιπτώσεις όπου είναι απαραίτητο να ελέγχεται η ύπαρξη ενός χαρακτηριστικού, ανεξάρτητα από την τιμή του. Αυτό επιτυγχάνεται με τη δήλωση μόνο του χαρακτηριστικού[6], πχ:

```
(11) HAS_TENSE = (^ TENSE).
```

Το (11) αληθεύει για οποιαδήποτε τιμή του TENSE.

2.2.4 Το λεξικό

Όπως αναφέρθηκε και παραπάνω η γραμματική του XLE χρησιμοποιεί λεξικές καταχωρήσεις που μπορούν να προέρχονται από διαφορετικές πηγές. Αν και αυτές θα μπορούσαν να περιέχονται σε ένα και μόνο αρχείο, είναι καλύτερη πρακτική να

[5] http://www2.parc.com/isl/groups/nltt/xle/doc/notations.html#N4.3.6
[6] http://www2.parc.com/isl/groups/nltt/xle/doc/notations.html#N4.2.4

μοιράζεται το λεξικό σε περισσότερα του ενός αρχεία. Στο ParGram πχ γίνεται χρήση των παρακάτω λεξικών:

- Λεξικό λειτουργικών λέξεων (σύνδεσμοι, άρθρα, αντωνυμίες, προθέσεις, μόρια, επιρρήματα)
- Λεξικό τεχνικών ή επιστημονικών όρων
- Λεξικό πλήρων λέξεων (ρήματα, ουσιαστικά, επίθετα)
- Μορφολογικό λεξικό

Μια τέτοια πρακτική έχει πλεονεκτήματα όπως την ευκολότερη συντήρηση και επιλογή λεξικών ανάλογα με τα κείμενα εισόδου, κάτι που οδηγεί σε μικρότερο φόρτο δεδομένων και μεγαλύτερη ταχύτητα ανάλυσης (Butt et al., 1999).

2.2.4.1 Η δομή μιας λεξικής καταχώρησης

Οι καταχωρήσεις στο λεξικό αποτελούνται από τέσσερα μέρη:

- Τη λέξη
- Το μέρος του λόγου της λέξης. Στο (12) η λέξη grigora μπορεί να αντιστοιχεί σε επίρρημα
- Το μορφοκώδικα, ο οποίος έχει δυο πιθανές τιμές:
  - *XLE* όταν η μορφολογία της λέξης αναλύεται από μορφολογικό αναλυτή και
  - '*' όταν δεν γίνεται μορφολογική ανάλυση. Σε αυτή την περίπτωση, η οποία αφορά και την παρούσα εργασία, πρέπει όλοι οι λεκτικοί τύποι ενός λεξήματος να περιέχονται στο λεξικό, μαζί με όλη την πληροφορία που απαιτείται από τη γραμματική
- Τη λίστα υποδειγμάτων ή χαρακτηριστικών που αντιστοιχούν στη λέξη

```
(12) grigora   ADV * @(ADVERB grigora).
```

2.2.4.2 Αλληλεπίδραση των λεξικών καταχωρήσεων

Το XLE επιτρέπει την εμφάνιση μιας λέξης περισσότερες από μια φορές στα διάφορα λεξικά. Εξ ορισμού, κάθε νέα καταχώρηση αντικαθιστά την προηγούμενη.

```
(13) grigora ADJ * @(ADJECTIVE grigoros).
```

Μια τέτοια δήλωση ακυρώνει την (12) καθιστώντας την ίδια τη μόνη ενεργή. Συνήθως όμως το ζητούμενο δεν είναι η ακύρωση της προηγούμενης πληροφορίας αλλά η επαύξησή της. Γι' αυτό και το XLE διαθέτει συγκεκριμένο φορμαλισμό, δίνοντας τη δυνατότητα στο γλωσσολόγο να μπορεί χειριστεί κατάλληλα τις καταχωρήσεις αυτές.

Αν η δήλωση περιέχει την ιδιότητα ETC τότε η πληροφορία που δηλώνεται, επαυξάνει την προηγούμενη.

```
(14) grigora    ADJ * @(ADJECTIVE grigoros);
                ETC.
```

Το αποτέλεσμα της αλληλεπίδρασης του (12) με το (14) είναι το (15):

```
(15) grigora    ADV * @(ADVERB grigora);
                ADJ * @(ADJECTIVE grigoros).
```

Αντίθετα, η ιδιότητα ONLY καθιστά μια καταχώρηση τη μόνη ενεργή, υποχρεώνοντας το XLE να απορρίψει οποιαδήποτε άλλη καταχώρηση.

Επιπλέον των χαρακτηριστικών ETC και ONLY, το XLE παρέχει τους τελεστές '!', '+', '-' και '=' για τον χειρισμό των υποκαταχωρήσεων: το '!' αντικαθιστά μια προηγούμενη υποκαταχώρηση, το '-' την αφαιρεί, το '=' την διατηρεί ενώ το '+' προσθέτει μια νέα. Οι τελεστές αυτοί συνδυάζονται με τα παραπάνω χαρακτηριστικά, δίνοντας έτσι ένα πολύ λειτουργικό τρόπο χειρισμού των λεξικών (Butt et al., 1999).

```
(16) α. grigora     !ADV * @(ADVERB2 grigora);
                    -ADJ
                    ETC.
     β. grigora     =ADV;
                    +ADJ * @(ADJECTIVE2 grigoros).
                    ONLY.
```

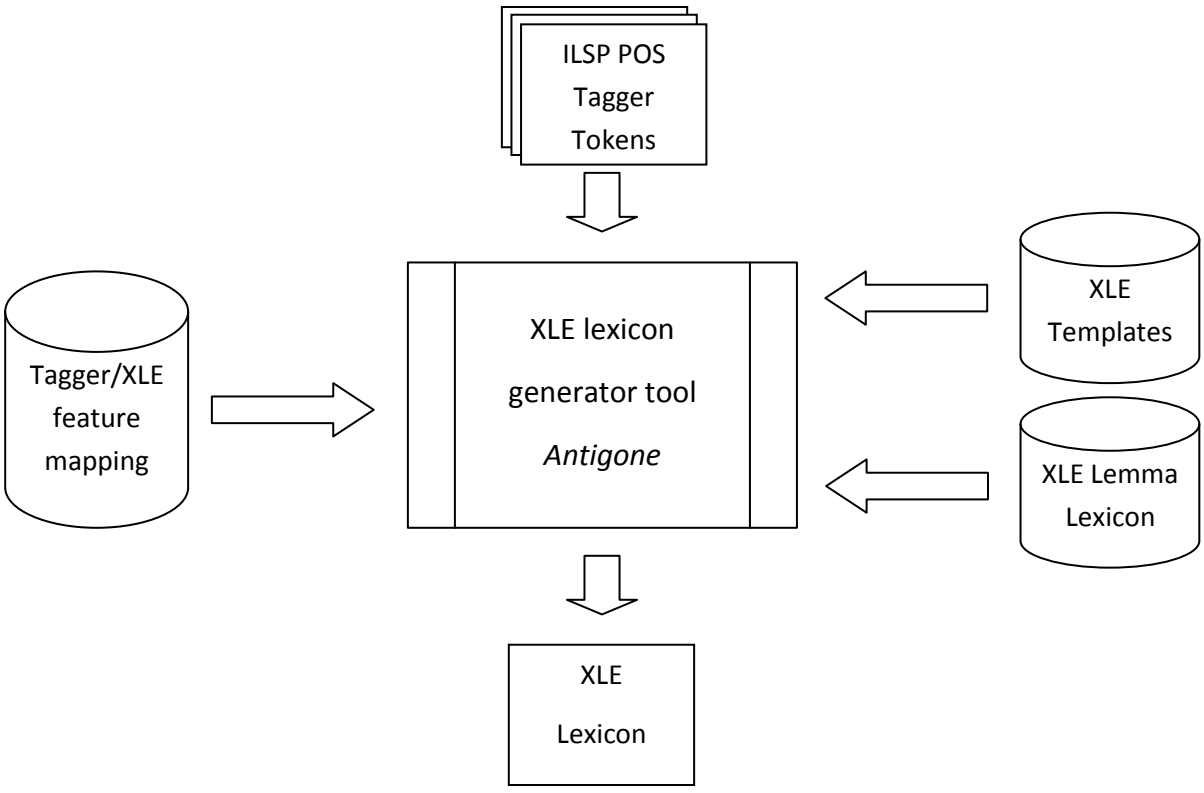
Αν θεωρήσουμε ενεργή δήλωση τη (15), η (16α) αντικαθιστά την υποκαταχώρηση του επιρρήματος και αφαιρεί αυτή του επιθέτου καταλήγοντας στο (17α). Η (16β) λόγω του χαρακτηριστικού ONLY ακυρώνει τη (15), διατηρώντας όμως την υποκαταχώρηση του επιρρήματος και προσθέτοντας αυτή του επιθέτου, καταλήγοντας στο (17β):

```
(17) α. grigora     ADV * @(ADVERB2 grigora).
     β. grigora     ADV * @(ADVERB grigora);
                    ADJ * @(ADJECTIVE2 grigoros).
```

Τα παραπάνω εργαλεία είναι ιδιαίτερα χρήσιμα όταν γίνεται χρήση πολλών λεξικών, καθώς κάθε νέο λεξικό μπορεί να τροποποιήσει τις ενεργές καταχωρήσεις, διατηρώντας το αρχικό λεξικό ακέραιο. Αν και στο λογισμικό *Αντιγόνη*, από τα παραπάνω χρησιμοποιείται μόνο το χαρακτηριστικό ETC, αποτελούν ένα πολύ χρήσιμο εργαλείο για την κατασκευή λεξικών.

3. Προδιαγραφές του λογισμικού *Αντιγόνη*

Το παρόν λογισμικό υλοποιεί τη σύνδεση του μορφολογικού αναλυτή (επισημειωτή) του ΙΕΛ με την πλατφόρμα ανάπτυξης γραμματικών συντακτικής ανάλυσης XLE της XEROX, για την ελληνική γλώσσα. Συγκεκριμένα, το πρόγραμμα λαμβάνει ως είσοδο τις επισημειωμένες λεκτικές μονάδες από το αρχείο εξόδου του εργαλείου του ΙΕΛ και με βάση τα υποδείγματα (templates) που χρησιμοποιούνται από την γραμματική του XLE, δημιουργεί λεξικό κατάλληλο για χρήση από τη γραμματική αυτή. Η σύνδεση των λεκτικών μονάδων με τα υποδείγματα γίνεται μέσω αρχείου αντιστοίχισης των χαρακτηριστικών του επισημειωτή με αυτά του XLE. Επιπρόσθετη πληροφορία, που δεν παρέχει το εργαλείο του ΙΕΛ, μπορεί να αντληθεί από λεξικό βασικών τύπων (λημμάτων) που ορίζει προαιρετικά ο χρήστης και να ενσωματωθεί σε κάθε λεκτική μονάδα που αντιστοιχεί στον τύπο αυτό.



Εικ. 9 Το λογισμικό *Αντιγόνη*

Παρακάτω δίνονται λεπτομερώς οι προδιαγραφές, για κάθε μια από τις λογικές ενότητες του λογισμικού, δηλαδή τις τέσσερις διαδικασίες εισόδου και τη διαδικασία εξόδου.

3.1 Είσοδος 1: Οι λεκτικές μονάδες

Οι λεκτικές μονάδες αποτελούν την κύρια είσοδο για τη δημιουργία του λεξικού του XLE. Περιέχονται σε αρχεία κάθε ένα από τα οποία είναι το αποτέλεσμα της επεξεργασίας ενός κειμένου πηγή από τον επισημειωτή ILSP FBT του ΙΕΛ (βλ. Ενότητα 1). Τα αρχεία αυτά τα αποθηκεύει ο χρήστης στο φάκελο TokenFiles, ο οποίος πρέπει να βρίσκεται στον ίδιο φάκελο με το λογισμικό.

Το λογισμικό επιτελεί τις παρακάτω λειτουργίες:

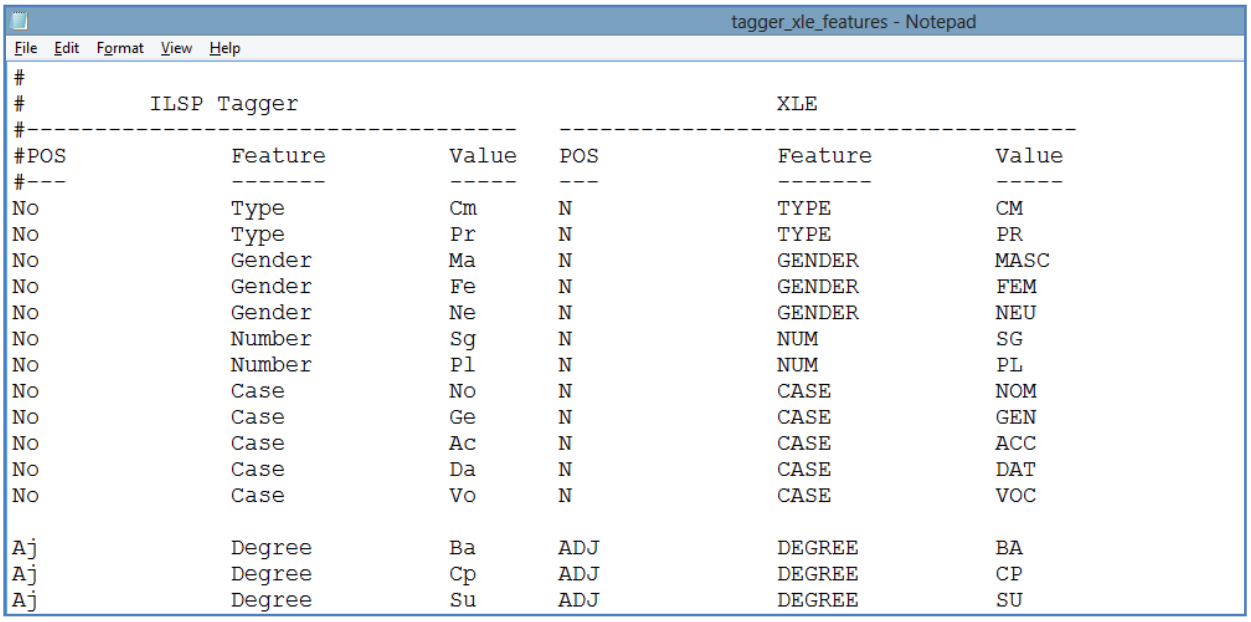
1. Άνοιγμα του φακέλου TokenFiles και διάβασμα των αρχείων εξόδου του επισημειωτή. Αν υπάρχει αδυναμία ανοίγματος του φακέλου ή δε βρεθούν αρχεία, το πρόγραμμα τερματίζεται.
2. Διάβασμα, για κάθε αρχείο, όλων των ετικετών με την ονομασία ‘’t’’ (token) και των χαρακτηριστικών τους: id, word, tag και lemma καθώς και αποθήκευσή τους σε κατάλληλη δομή δεδομένων.
3. Μεταγραφή των ελληνικών χαρακτήρων σε λατινικούς σύμφωνα με το πρότυπο ISO 843:1996 (βλ. Παράρτημα Η), καθώς η πρακτική εφαρμογή έδειξε αστάθεια κατά την εκτέλεση του XLE, όταν γίνεται χρήση της ελληνικής.
4. Έλεγχος κάθε λεκτικής μονάδας για το αν έχει ήδη αποθηκευτεί, οπότε είναι περιττή η εκ νέου καταχώρησή της.
5. Διασφάλιση της αποθήκευσης των λεκτικών μονάδων που αν και παρουσιάζουν την ίδια γραφή, έχουν διαφορετικά χαρακτηριστικά. Αυτό συμβαίνει τόσο στην περίπτωση διαφορετικών λημμάτων, όπως πχ το ‘’γρήγορα’’ ως επίθετο και ως επίρρημα, όσο και στην περίπτωση ίδιων λημμάτων με διαφορετικά χαρακτηριστικά, όπως πχ το άρθρο “το’’ που μπορεί να είναι στην ονομαστική ή στην αιτιατική. Και στις δυο περιπτώσεις θα πρέπει να καταχωρούνται και οι δυο λεκτικές μονάδες.

```
<t id="t2" word="γρήγορα" tag="AjBaNePlAc" lemma="γρήγορος"/>
<t id="t6" word="γρήγορα" tag="AdXxBa" lemma="γρήγορα"/>

<t id="t1" word="το" tag="AtDfNeSgNm" lemma="ο"/>
<t id="t2" word="το" tag="AtDfNeSgAc" lemma="ο"/>
```

3.2 Είσοδος 2: Το αρχείο αντιστοίχισης χαρακτηριστικών

Είναι ένα αρχείο κειμένου μορφής .txt. Σε αυτό ορίζεται η αντιστοίχιση των μερών του λόγου καθώς και των χαρακτηριστικών και των τιμών τους, όπως δίνονται από

τον επισημειωτή του ΙΕΛ, με αυτά της γραμματικής XLE του χρήστη. Το αρχείο αποτελείται από έξι στήλες, εκ των οποίων οι τρεις πρώτες αναφέρονται στον επισημειωτή και οι τρεις επόμενες στο XLE (βλ. Παράρτημα Β).


tagger_xle_features - Notepad

File Edit Format View Help

#	ILSP Tagger		XLE		
#POS	Feature	Value	POS	Feature	Value
#---	-------	-----	---	-------	-----
No	Type	Cm	N	TYPE	CM
No	Type	Pr	N	TYPE	PR
No	Gender	Ma	N	GENDER	MASC
No	Gender	Fe	N	GENDER	FEM
No	Gender	Ne	N	GENDER	NEU
No	Number	Sg	N	NUM	SG
No	Number	Pl	N	NUM	PL
No	Case	No	N	CASE	NOM
No	Case	Ge	N	CASE	GEN
No	Case	Ac	N	CASE	ACC
No	Case	Da	N	CASE	DAT
No	Case	Vo	N	CASE	VOC
Aj	Degree	Ba	ADJ	DEGREE	BA
Aj	Degree	Cp	ADJ	DEGREE	CP
Aj	Degree	Su	ADJ	DEGREE	SU

Εικ. 10 Το αρχείο αντιστοίχισης

Ο χρήστης έχει τη δυνατότητα να ορίσει τις τιμές στις στήλες που αντιστοιχούν στο XLE, ανάλογα με την προτίμησή του για τον τρόπο εμφάνισής τους στη γραμματική. Επίσης έχει τη δυνατότητα να μη δώσει τιμές σε κάποιες από αυτές, οπότε το πρόγραμμα αγνοεί την αντίστοιχη πληροφορία του επισημειωτή. Με αυτό τον τρόπο ο χρήστης μπορεί να αποκλείσει από τη δημιουργία του λεξικού συγκεκριμένα μέρη του λόγου, χαρακτηριστικά ή/και τιμές, αν αυτό εξυπηρετεί τις ανάγκες του. Οι τρεις πρώτες στήλες δεν πρέπει να αλλάζουν καθώς αυτό θα οδηγήσει σε αποτυχία του προγράμματος.

Η συγκεκριμένη μορφή του αρχείου επιλέχθηκε, κυρίως λόγω της ευκολίας και της απλότητάς της τόσο στον ορισμό, όσο και στον έλεγχο της αντιστοίχισης. Ο χρήστης μπορεί να κάνει την αντιστοίχιση της πληροφορίας που τον ενδιαφέρει, χρησιμοποιώντας ένα μόνο αρχείο και με τρόπο ενιαίο για όλες τις περιπτώσεις. Παράλληλα του δίνεται η ευχέρεια χειρισμού του κάθε χαρακτηριστικού με διαφορετικό τρόπο, αν αυτό απαιτείται. Για παράδειγμα το χαρακτηριστικό *πρόσωπο* του επισημειωτή αντιστοιχεί στο XLE σε διαφορετικό χαρακτηριστικό στην περίπτωση του ρήματος (SUBJ PERS) και σε διαφορετικό χαρακτηριστικό στην περίπτωση της αντωνυμίας (PERS). Όταν η γραμματική XLE χειρίζεται ένα χαρακτηριστικό (πχ η πτώση ουσιαστικών, επιθέτων κλπ) ανεξαρτήτως μέρους του λόγου και φραστικής κατηγορίας, είναι ευθύνη του χρήστη να δίνει ίδιες τιμές στα αντίστοιχα πεδία.

Το πρόγραμμα ανοίγει το αρχείο αντιστοίχισης, διαβάζει τα περιεχόμενά του και τα αποθηκεύει σε κατάλληλη δομή δεδομένων. Το όνομα (ή το πλήρες μονοπάτι) του αρχείου καθορίζει ο χρήστης στο αρχείο ρυθμίσεων config.ini ενώ αν υπάρχει αδυναμία ανοίγματος, το πρόγραμμα τερματίζεται.

3.3 Είσοδος 3: Τα υποδείγματα (templates) της γραμματικής XLE

Είναι ο κατάλογος με τα υποδείγματα τα οποία περιέχουν χαρακτηριστικά αντίστοιχα με αυτά του επισημειωτή όπως ο χρόνος, η πτώση, το πρόσωπο κλπ. Υποδείγματα που περιέχουν μη μορφολογική πληροφορία, όπως τα TRANS(P) και INTRANS(P) που ορίζουν τη μεταβατικότητα των ρημάτων για παράδειγμα, δεν πρέπει να βρίσκονται στην ίδια ενότητα, καθώς δε μπορούν να αξιοποιηθούν από το εργαλείο. Η ανάγκη για εκμετάλλευση και αυτής της πληροφορίας αντιμετωπίζεται με τη δημιουργία λεξικού λημμάτων (βλ. 3.4).

Το λογισμικό επιτελεί τις παρακάτω λειτουργίες:

1. Άνοιγμα του αρχείου υποδειγμάτων, το όνομα (ή το πλήρες μονοπάτι) του οποίου καθορίζει ο χρήστης στο αρχείο ρυθμίσεων config.ini. Στο ίδιο αρχείο καθορίζεται και το όνομα της ενότητας των υποδειγμάτων. Αν υπάρχει αδυναμία ανοίγματος του αρχείου ή ευρέσεως της ενότητας, το πρόγραμμα τερματίζεται.
2. Διάβασμα των υποδειγμάτων και αποθήκευσή τους σε κατάλληλη δομή δεδομένων.
3. Λεξική και συντακτική ανάλυση των υποδειγμάτων σύμφωνα με τους κανόνες, όπως αναφέρονται στην ενότητα ‘Δομή των υποδειγμάτων’.
4. Σε περίπτωση λεκτικού ή συντακτικού λάθους, εμφανίζεται μήνυμα λάθους και το υπόδειγμα δεν αποθηκεύεται.
5. Σε περίπτωσης δήλωσης δυο υποδειγμάτων με το ίδιο όνομα, εμφανίζεται μήνυμα λάθους. Το πρόγραμμα συνεχίζεται αντικαθιστώντας το πρώτο υπόδειγμα με το δεύτερο.
6. Σε περίπτωση αναδρομικής κλήσης υποδείγματος εμφανίζεται μήνυμα λάθους και η κλήση αντικαθιστάται με την τιμή False.

3.3.1 Δομή των υποδειγμάτων[7]

Ο φορμαλισμός του XLE για τα υποδείγματα είναι ιδιαίτερα εκφραστικός και καλύπτει λειτουργίες που ξεπερνούν τις ανάγκες του εργαλείου. Αυτό συμβαίνει για δύο λόγους. Αφενός, στοιχεία του φορμαλισμού φαίνεται να αφορούν σε υποδείγματα που χρησιμοποιούνται στους λεκτικούς[8] και στους συντακτικούς κανόνες[9] κι όχι στις λεξικές καταχωρήσεις. Αφετέρου, το εργαλείο λειτουργεί αντίστροφα από το XLE. Στο δεύτερο ο γλωσσολόγος χρησιμοποιεί τα υποδείγματα στις λεξικές καταχωρήσεις για να τις αντιστοιχίσει τελικά σε ομάδες χαρακτηριστικών. Στο λογισμικό *Αντιγόνη* η πορεία της αντιστοίχισης αντιστρέφεται. Γίνεται έλεγχος των χαρακτηριστικών κάθε λεκτικής μονάδας ώστε να αποφασίσει με ποια υποδείγματα μπορεί αυτή να συνδεθεί. Το λογισμικό αφού διαβάσει το κάθε υπόδειγμα και το αναλύσει συντακτικά, δίνει τις τιμές που συνοδεύουν κάθε λεκτική μονάδα, στα χαρακτηριστικά που αυτό περιέχει και αποφασίζει αν υπάρχει ταίριασμα ή όχι.

Η ενσωμάτωση λοιπόν όλου του φορμαλισμού στο λογισμικό θα αύξανε κατά πολύ την πολυπλοκότητά του αλλά και το χρόνο εκτέλεσής του, χωρίς να συνεισφέρει στο σκοπό του. Για τους λόγους αυτούς κρίθηκε ότι είναι έξω από τα πλαίσια της παρούσης εργασίας. Η δομή των υποδειγμάτων που επεξεργάζεται το λογισμικό, παρουσιάζεται παρακάτω με τη μορφή συντακτικών κανόνων:

1. *Template:* *TemplateHead = TemplateBody.*

Ένα υπόδειγμα αποτελείται από την κεφαλή, το σύμβολο '=', το σώμα και την τελεία '.'.

2. *TemplateHead:* *TemplateName*
   TemplateName($Param_1$ $Param_2$… $Param_i$*)*

Η κεφαλή του υποδείγματος αποτελείται από το όνομά της ή το όνομα ακολουθούμενο από παραμέτρους που περικλείονται σε παρενθέσεις και είναι χωρισμένες με κενά.

3. *Param*

Μια παράμετρος είναι οποιοσδήποτε συνδυασμός αλφαριθμητικών χαρακτήρων

4. *TemplateBody: And_List*

Το σώμα του υποδείγματος είναι μια λίστα της οποίας τα στοιχεία συνδέονται με λογική σύζευξη.

---

[7] http://www2.parc.com/isl/groups/nltt/xle/doc/notations.html#N4.3
[8] http://www2.parc.com/isl/groups/nltt/xle/doc/notations.html#N5.2
[9] http://www2.parc.com/isl/groups/nltt/xle/doc/notations.html#N5.3

5. *Conjunction:* *[And_List]*
Η σύζευξη αποτελείται από τη λίστα των στοιχείων που περικλείονται σε αγκύλες (‘[‘ και ‘]’).

6. And_List: *$Item_1$ $Item_2$… $Item_i$*
Η λίστα της σύζευξης αποτελείται από στοιχεία χωρισμένα με κενό (ή το χαρακτήρα αλλαγής γραμμής).

7. Item: *Conjunction*
Disjunction
Negation
FeatureExpr
TemplateCall
Ένα στοιχείο μπορεί να είναι, σύζευξη, διάζευξη, άρνηση ή κλήση υποδείγματος.

8. FeatureExpr: *Equality*
Inequality
Existential
Ένα μπορεί να είναι ισότητα, ανισότητα ή υπαρξιακός περιορισμός (existential constraint).

9. Disjunction: *{ Or_List }*
Η διάζευξη αποτελείται από λίστα στοιχείων που περικλείονται σε άγκιστρα (‘{‘ και ‘}’).

10.Or_List: *And_List_1 | And_List_2 | … | And_List_i*
Η λίστα της διάζευξης αποτελείται από λίστες σύζευξης χωρισμένες με το σύμβολο ‘|’.

11.Negation: *~ Item*
Η άρνηση είναι ένα στοιχείο που άγεται από το σύμβολο ‘~’.

12.Equality: *Feature = Value*
Η ισότητα αποτελείται από ένα χαρακτηριστικό, το σύμβολο ‘=’ και την τιμή.

13.Inequality: *Feature ~= Value*
Η ανισότητα αποτελείται από ένα χαρακτηριστικό, τα σύμβολα ‘~’ και ‘=’ και την τιμή.

14.Existential: *Feature*

Ο υπαρξιακός περιορισμός δηλώνεται με το όνομα του χαρακτηριστικού

15.TemplateCall: *@TemplateName*

@(TemplateName Var_1 Var_2*…* Var_i*)*

Η κλήση υποδείγματος αποτελείται από το χαρακτήρα '@' ακολουθούμενο από το όνομα του υποδείγματος ή από το χαρακτήρα '@' ακολουθούμενο από παρενθέσεις που περικλείουν το όνομα του υποδείγματος και τις πραγματικές παραμέτρους, χωρισμένες με κενά.

16.Feature: *(^ NAME)*

Το χαρακτηριστικό δηλώνεται με το σύμβολο '^' και το όνομά του (όπως τα ορίζει ο χρήστης στην $5^η$ στήλη του αρχείου αντιστοίχισης) που περικλείονται από παρενθέσεις.

17.Value: *NAME*

Var

Οι τιμές των χαρακτηριστικών, όπως τις ορίζει ο χρήστης στο αρχείο αντιστοίχισης ($6^η$ στήλη) ή μια πραγματική παράμετρος

18.Var

Μια πραγματική παράμετρος είναι οποιοσδήποτε συνδυασμός αλφαριθμητικών χαρακτήρων

3.4 Είσοδος 4: Λεξικό λημμάτων της γραμματικής XLE

Πρόκειται για ξεχωριστό λεξικό το οποίο δημιουργεί ο γλωσσολόγος και περιέχει καταχωρήσεις με βασικούς τύπους λέξεων (λήμματα). Κάθε καταχώρηση συνοδεύεται από επιπλέον πληροφορία (υποδείγματα ή και χαρακτηριστικά) που είναι απαραίτητη στη γραμματική αλλά ο επισημειωτής δεν την παρέχει, όπως πχ τις ιδιότητες υποκατηγοριοποίησης των ρημάτων, που αναφέρθηκαν παραπάνω. Το εργαλείο διαβάζει την πληροφορία αυτή και την προσθέτει αυτούσια σε κάθε λεκτική μονάδα που αντιστοιχεί στο λήμμα αυτό.

Οι λειτουργίες που επιτελούνται είναι:

1. Άνοιγμα του λεξικού λημμάτων, το όνομα (ή το πλήρες μονοπάτι) του οποίου καθορίζει ο χρήστης στο αρχείο ρυθμίσεων config.ini. Στο ίδιο αρχείο καθορίζεται και το όνομα της ενότητας του λεξικού. Αν υπάρχει αδυναμία ανοίγματος του αρχείου ή ευρέσεως της ενότητας, το πρόγραμμα εμφανίζει προειδοποιητικό μήνυμα και συνεχίζει την εκτέλεσή του.

2. Διάβασμα του αρχείου και αποθήκευση των καταχωρήσεων σε κατάλληλη δομή δεδομένων

3.5 Έξοδος: Πλήρες λεξικό για το XLE

Το λεξικό περιέχει τις λεκτικές μονάδες του αρχείου του επισημειωτή με τα υποδείγματα που αντιστοιχούν στα χαρακτηριστικά τους (βλ. Ενότητα 1)

Το λογισμικό επιτελεί τις παρακάτω λειτουργίες:

1. Δημιουργία του αρχείου εξόδου το όνομα (ή το πλήρες μονοπάτι) του οποίου καθορίζει ο χρήστης στο αρχείο ρυθμίσεων config.ini. Στο ίδιο αρχείο καθορίζεται και το όνομα της ενότητας του λεξικού. Αν υπάρχει αδυναμία δημιουργίας του αρχείου, το πρόγραμμα τερματίζεται.
2. Καταχώρηση κάθε λεκτικής μονάδας στο λεξικό, συνοδευόμενη από το μέρος του λόγου της.
3. Έλεγχος για το αν περιέχονται σε άλλα υποδείγματα που είναι ήδη εν χρήσει τα υποδείγματα ή και τα χαρακτηριστικά που αντιστοιχούν σε μία λεκτική μονάδα που καταχωρείται στο λεξικό. Στην καταχώρηση της λεκτικής μονάδας θα συμπεριληφθούν μόνο όσα δεν περιέχονται αλλού (αποφυγή πλεονασμού). Για να γίνει κατανοητή η διαδικασία ας θεωρήσουμε τα παρακάτω υποδείγματα:

```
V3SG =        (^ SUBJ NUM)= SG
              (^ SUBJ PERS)=3
              @VPRES.

VPRES =       @(TENSE PRES)
              @(MOOD INDIC).

TENSE(T) =    (^ TENSE) = T.

MOOD(M) =     (^ MOOD) = M.
```

Για τη λέξη *τρέχει*, με τα χαρακτηριστικά SUBJ NUM (SG), SUBJ PERS (3), TENSE (PRES), MOOD (INDIC) και VOICE (ACT), η καταχώρηση στο λεξικό θα είναι:

```
Τρέχει    V *    @V3SG
                 (^ VOICE) = ACT.
```

Τα χαρακτηριστικά TENSE, MOOD και τα υποδείγματα TENSE, MOOD και VPRES, αν και συνδέονται με τη λεκτική μονάδα *τρέχει*, δεν εμφανίζονται στην καταχώρησή της, καθώς περιέχονται στο υπόδειγμα V3SG. Αντίθετα το χαρακτηριστικό VOICE εμφανίζεται αυτούσιο.

4. Έλεγχος για κάθε λεκτική μονάδα για το αν υπάρχει στο λεξικό λημμάτων το λήμμα στο οποίο αυτή αντιστοιχεί. Σε αυτή την περίπτωση η πληροφορία που συνοδεύει το λήμμα, αντιγράφεται αυτούσια στην καταχώρηση της λεκτικής μονάδας.
5. Έλεγχος κάθε καταχώρησης και
    a. Απόρριψή της στην περίπτωση που είναι διπλή
    b. Προσθήκη του χαρακτηριστικού ETC στην καταχώρηση, όταν αυτή αφορά σε ίδια λεκτική μονάδα αλλά διαφορετικό λήμμα, όπως πχ για τη λεκτική μονάδα το που μπορεί να αντιστοιχεί τόσο σε άρθρο όσο και σε αντωνυμία:

```
to PRON *  …
           …
           ETC.
to D *     …
           …
           ETC.
```

6. Εμφάνιση των καταχωρήσεων ομαδοποιημένες ανά μέρος του λόγου και ταξινομημένες σε αλφαβητική σειρά.

3.5.1 Η χρήση της διάζευξης από το λογισμικό *Αντιγόνη*

Η διάζευξη στα υποδείγματα του XLE λειτουργεί παραγωγικά. Αυτό έχει την έννοια ότι όταν μια λεκτική μονάδα αντιστοιχίζεται σε ένα υπόδειγμα που περιέχει διάζευξη, το XLE –αφού πρώτα το μετατρέψει σε κανονική διαζευκτική μορφή- δημιουργεί

εσωτερικά για κάθε μέλος της διάζευξης μια λεξική καταχώρηση. Ας δούμε τα παρακάτω υποδείγματα:

```
(1) α. NOM_ACC = {@(CASE NOM) | @(CASE ACC)}.
    β. CASE(P) = (^ CASE) = P.
```

και τη χρήση του πρώτου στην λεξική καταχώρηση *παιδί*:

```
(2) παιδί N * @NOM_ACC
```

Η δήλωση (2) ισοδυναμεί με τις δυο παρακάτω:

```
(3) παιδί N * (^ CASE) = NOM
    παιδί N * (^ CASE) = ACC
```

Το (1α) είναι αρκετά γενικό και είναι ευθύνη του χρήστη να το αντιστοιχίσει με τις λεκτικές μονάδες που έχουν ίδια γραφή στην ονομαστική και την αιτιατική. Το λογισμικό *Αντιγόνη* λειτουργεί αντίστροφα, αφού αυτό θα πρέπει να αποφασίσει με ποιες λεκτικές μονάδες θα αντιστοιχίσει το (1α) κι όχι ο χρήστης. Θεωρητικά, λοιπόν θα έπρεπε να το κάνει για λεκτικές μονάδες όπου εμφανίζονται στα αρχεία εισόδου και στις δυο πτώσεις, συγχωνεύοντάς τις σε μια λεξική καταχώριση.

Αντί της παραπάνω αντιμετώπισης, το λογισμικό *Αντιγόνη,* αντιστοιχίζει ένα υπόδειγμα με διάζευξη με κάθε λεκτική μονάδα που ικανοποιεί έστω και ένα μέρος της διάζευξης. Η προσέγγιση αυτή μπορεί να οδηγήσει σε λάθη, για αυτό και είναι ευθύνη του γλωσσολόγου η σωστή δήλωση του υποδείγματος. Στο (1α) λοιπόν το υπόδειγμα μπορεί να περιλαμβάνει και το γένος, ώστε να διασφαλίζει την αντιστοίχιση μόνο με τα ουδέτερα:

```
(4) NOM_ACC = {@(CASE NOM) | @(CASE ACC)}
                @(GENDER NEU).
```

Με τον τρόπο αυτό από τη μια περιορίζεται η αντιστοίχιση του υποδείγματος μόνο στα ουδέτερα και από την άλλη επεκτείνεται και σε λεκτικές μονάδες οι οποίες εμφανίζονται μόνο με τη μια πτώση στο κείμενο εισόδου. Ο συγκεκριμένος τρόπος χειρισμού της διάζευξης από το λογισμικό, έχει λοιπόν τρία πλεονεκτήματα:

- ✓ Αντί να μεταφέρει απλά, παράγει λεξικό
- ✓ Δεσμεύει το χρήστη στη δημιουργία υποδειγμάτων με τρόπο που να αντικατοπτρίζει πραγματικά γραμματικά φαινόμενα
- ✓ Είναι απλούστερος στην υλοποίηση και λιγότερο χρονοβόρος

3.5.2 Η χρήση της άρνησης από το λογισμικό *Αντιγόνη*

Όπως αναφέρθηκε στο 2.2.3.2 η άρνηση '~' είτε μπει μπροστά από μια έκφραση είτε μπροστά στην ισότητα έχει το ίδιο αποτέλεσμα για το XLE.

```
(5)  α. NOT_PRES1 = ~(^ TENSE) = PRES.

     β. NOT_PRES2 = (^ TENSE) ~= PRES.
```

Οι δηλώσεις στο (5) είναι ισοδύναμες. Για το πρόγραμμα όμως αυτό δεν ισχύει, καθώς το (5α) αληθεύει σε δυο περιπτώσεις:

- όταν μια λεκτική μονάδα έχει άλλη τιμή στο TENSE (πχ PAST)
- όταν μια λεκτική μονάδα δεν έχει χαρακτηριστικό TENSE (όπως τα άρθρα, τα ουσιαστικά κλπ). Αυτό είναι προφανές αφού το (^ TENSE) = PRES είναι ψευδές στην περίπτωση απουσίας του χαρακτηριστικού TENSE.

Το (5β) αντίθετα αληθεύει μόνο στην πρώτη περίπτωση.

Ως εκ τούτου ο γλωσσολόγος θα πρέπει να είναι ιδιαίτερα προσεκτικός στη χρήση της άρνησης. Για να αποφύγει πιθανά λάθη προτείνεται η χρήση της άρνησης μαζί με την ισότητα, όπως στο (5β). Σε αντίθετη περίπτωση η άρνηση θα πρέπει να συνδυάζεται με τη χρήση του υπαρξιακού περιορισμού (βλ. 2.2.3.3). Αντί του (5α) λοιπόν, προτείνεται το (6):

```
(6)  NOT_PRES1 = ~(^ TENSE) = PRES
                    (^ TENSE).
```

4. Τεκμηρίωση του λογισμικού *Αντιγόνη*

4.1 Η γλώσσα προγραμματισμού Python

Η Python είναι μια καινούρια σχετικά γλώσσα προγραμματισμού καθώς αναπτύχθηκε το 1990 από τον Ολλανδό Guido van Rossum. Έχει δυο βασικά χαρακτηριστικά που την καθιστούν εξαιρετικά δημοφιλή: είναι απλή στη γραφή και στην εκμάθησή της ενώ ταυτόχρονα είναι ιδιαίτερα ισχυρή. Αποτελεί μια γλώσσα υψηλού επιπέδου, που υποστηρίζει διαδικασιακό και αντικειμενοστραφή προγραμματισμό, κάνοντας χρήση αποδοτικών δομών δεδομένων, όπως οι λίστες (lists) και τα λεξικά (dictionaries) που χρησιμοποιούνται κατά κόρον στο λογισμικό *Αντιγόνη*. Είναι διερμηνευόμενη κι έτσι δεν απαιτείται διαδικασία μεταγλώττισης του πηγαίου κώδικα σε γλώσσα μηχανής. Η εκτέλεση του προγράμματος γίνεται απευθείας από τον πηγαίο κώδικα, καθιστώντας τη χρήση της πιο εύκολη. Η εκτέλεση του λογισμικού σε έναν υπολογιστή προϋποθέτει απλά την εγκατάσταση της Python και την αντιγραφή του κώδικα σε αυτόν.

Ανήκει στις γλώσσες ΕΛΛΑΚ (Ελεύθερο Λογισμικό και Λογισμικό Ανοικτού Κώδικα) κι έτσι επιτρέπει την ελεύθερη διανομή αντιγράφων του λογισμικού, το διάβασμα και την τροποποίηση του πηγαίου κώδικα καθώς και τη χρήση του σε άλλα λογισμικά. Λόγω του ανοικτού της κώδικα, η Python έχει υλοποιηθεί σε πολλά λειτουργικά συστήματα όπως Windows, Unix, Macintosh, Solaris, OS/2, Windows CE κλπ. Το παραγόμενο λογισμικό μπορεί να εκτελεστεί σε οποιαδήποτε από αυτές τις πλατφόρμες χωρίς την απαίτηση αλλαγών ικανοποιώντας έτσι της απαίτηση για φορητότητα[10].

Αν και νέα σχετικά γλώσσα, διαθέτει ήδη μια τεράστια πρότυπη βιβλιοθήκη έτοιμου κώδικα, αποτέλεσμα της μεγάλης αποδοχής και υποστήριξής της από την προγραμματιστική κοινότητα, προσφέροντας έτσι μια μεγάλη γκάμα ευκολιών. Η βιβλιοθήκη περιέχει αρθρώματα (modules), γραμμένα τόσο σε Python που αντιμετωπίζουν τα συνηθέστερα προβλήματα με έναν πρότυπο τρόπο, όσο και σε C/C++ που προσφέρουν λειτουργικότητα που δεν είναι υλοποιήσιμη στην ίδια τη γλώσσα[11].

Στην πρώτη περίπτωση ανήκουν και τα αρθρώματα Regex.py, ElementTree.py και Lex.py/Yacc.py που χρησιμοποιήθηκαν στο λογισμικό *Αντιγόνη* και αφορούν στη χρήση κανονικών εκφράσεων, το διάβασμα αρχείων XML και στη λεκτική/συντακτική ανάλυση των υποδειγμάτων αντίστοιχα.

[10] http://files.swaroopch.com/python/byte_of_python.pdf
[11] http://docs.python.org/2/library/

4.2 Ο συντακτικός αναλυτής Python Lex & Yacc (PLY parser)[12]

Σημαντικό ρόλο στο λογισμικό *Αντιγόνη* παίζει ο συντακτικός αναλυτής PLY. Είναι υπεύθυνος για την αρχική επεξεργασία των υποδειγμάτων του XLE, δηλαδή για τον τεμαχισμό τους (σε λεκτικές μονάδες και σύμβολα) και την συντακτική τους ανάλυση.

Ο PLY αποτελεί υλοποίηση των εργαλείων κατασκευής μεταγλωττιστών lex και yacc στη γλώσσα προγραμματισμού Python. Τηρεί όλες τις προδιαγραφές των εργαλείων αυτών, κάτι που σημαίνει ότι αφενός βασίζεται στον αλγόριθμο LALR(1) και αφετέρου ότι παρέχει δυνατότητες όπως έλεγχο της εισόδου, αναφορά λαθών και διαγνωστικά εργαλεία. Ο αλγόριθμος LALR(1) (**L**ook **A**head, **L**eft to Right, **R**ightmost derivation) είναι ένας bottom-up αλγόριθμος με χρήση ενός token look-ahead. Παρότι είναι σημαντικά πιο απλός από τον αρχικό LR(1), είναι πολύ γρήγορος και αποδοτικός.

Η μονάδα PLY, δίνει τη δυνατότητα στον προγραμματιστή να δημιουργήσει γραμματικές χωρίς συμφραζόμενα, με τη χρήση κανονικών εκφράσεων και κανόνων μορφής BNF εύκολα και με πολύ λιγότερο κώδικα.

4.2.1 Η μονάδα LEX

Η μονάδα lex είναι υπεύθυνη για τον τεμαχισμό μιας ακολουθίας χαρακτήρων εισόδου σε λεκτικές μονάδες. Για το σκοπό αυτό χρησιμοποιεί λεκτικούς κανόνες που ορίζονται ως κανονικές εκφράσεις (regex) και δηλώνονται είτε ως μεταβλητές είτε ως συναρτήσεις, με όνομα που αρχίζει με το χαρακτηριστικό 't_'. Όταν ο λεκτικός κανόνας δηλώνεται ως αλφαριθμητική μεταβλητή, η τιμή της είναι η κανονική έκφραση και το ταίριασμα είναι η τιμή της λεκτικής μονάδας. Έτσι πχ η δήλωση

```
t_NAME = r'[a-zA-Z0-9]*'
```

ορίζει ότι οποιαδήποτε συμβολοσειρά που περιέχει τα σύμβολα 'a' έως 'z','A' έως 'Z' ή '0' έως '9' αναγνωρίζεται ως λεκτική μονάδα NAME.

Όταν ο προγραμματιστής θέλει να γίνουν κάποιες ενέργειες οι οποίες θα πυροδοτούνται από την αναγνώριση μιας συγκεκριμένης λεκτικής μονάδας, ορίζει τον λεκτικό κανόνα ως συνάρτηση (def) και η κανονική έκφραση βρίσκεται στη συμβολοσειρά τεκμηρίωσής της (docstring). Η συνάρτηση δίνει τη δυνατότητα στον προγραμματιστή να αλλάξει τη λεκτική μονάδα, να επιστρέψει διαφορετική μονάδα ή και να την αγνοήσει, χωρίς να επιστρέψει τίποτα. Έτσι πχ η δήλωση:

```
def t_newline(t):
    r'\n+'
```

[12] http://www.dabeaz.com/ply/ply.html

```
    t.lexer.lineno += t.value.count("\n")
```

ορίζει τη λεκτική μονάδα newline (εσωτερική μεταβλητή της μονάδας lex) ως μια η περισσότερες εμφανίσεις του χαρακτήρα αλλαγής γραμμής '\n'. Όταν υπάρχει ταίριασμα, αυξάνει την εσωτερική μεταβλητή lineno κατά τον αριθμό των εμφανίσεων του χαρακτήρα.

Οι λεκτικές μονάδες που ορίζει ο χρήστης πρέπει να περιλαμβάνονται στην *πλειάδα* (tuple) με όνομα tokens, η οποία είναι απαραίτητη για να γίνονται έλεγχοι ορθότητας και καθορισμός των τερματικών κόμβων από τη μονάδα yacc.

Τέλος, όταν γίνεται χρήση αυτούσιων συμβόλων (literals) αυτά δηλώνονται ως στοιχεία *λίστας* στη μεταβλητή literals:

```
literals = ['=', '~', '@', '(', ')', '.']
```

4.2.2 Η μονάδα YACC

Η μονάδα YACC (Yet Another Compiler Compiler) είναι υπεύθυνη για τη συντακτική ανάλυση των λεκτικών μονάδων. Δέχεται ως είσοδο λεκτικές μονάδες (αυτές που έχουν προκύψει από τη μονάδα lex) και με κανόνες γραμματικής BNF που ορίζει ο προγραμματιστής, αναλύει τον τρόπο με τον οποίο αυτές συνδέονται. Η γενική μορφή ενός κανόνα είναι:

```
<αναγνωριστικό> : _έκφραση_
```

Όπου:

- το *<αναγνωριστικό>* αποτελεί έναν μη τερματικό κόμβο,
- η *_έκφραση_* αποτελεί ένα συνδυασμό άλλων αναγνωριστικών
- το ':' ορίζει ότι όταν αναγνωρίζεται μια *_έκφραση_*, αυτή αντικαθίσταται από το *<αναγνωριστικό>*.

Τα αναγνωριστικά που δεν εμφανίζονται ποτέ στην αριστερή μεριά του κανόνα ονομάζονται τερματικοί κόμβοι.

Οι συντακτικοί κανόνες ορίζονται στο yacc ως συναρτήσεις, των οποίων το όνομα ξεκινάει με το χαρακτηριστικό 'p_' ακολουθούμενο από το όνομα του μη τερματικού συμβόλου. Ο κανόνας γράφεται στη συμβολοσειρά τεκμηρίωσης (docstring) της συνάρτησης και οι ενέργειες που πρέπει να γίνουν, γράφονται στο σώμα της ως εντολές Python.

Για όλα τα μη τερματικά σύμβολα πρέπει να οριστούν αντίστοιχες συναρτήσεις ενώ τα τερματικά πρέπει να έχουν δηλωθεί στη μονάδα lex.

```
def p_template(p):
    'template : temp_head "=" conjunction "."'
    <κώδικας>
def p_temp_head(p):
    '''temp_head : temp_name
                 | temp_name "(" params ")"
    '''
    <κώδικας>
```

Στο παραπάνω παράδειγμα ορίζονται οι συντακτικοί κανόνες για τα μη τερματικά σύμβολα *template* και *temp_head*. Μη τερματικά είναι επίσης και τα *conjunction*, *temp_name* και *params* ενώ τερματικά είναι τα '=', '.', '(' και ')'. Το σύμβολο '|' δηλώνει τη λογική διάζευξη, οπότε στην παραπάνω περίπτωση το *temp_head* μπορεί να αντιστοιχεί σε ένα *temp_name* ή στην ακολουθία *temp _name "(" params ")".*

Κάθε συνάρτηση δέχεται μια λίστα *p* ως όρισμα, η οποία περιέχει τα σύμβολα του κανόνα στον οποίο αντιστοιχεί, με τη σειρά που αυτά εμφανίζονται. Έτσι στον πρώτο κανόνα το *template* αντιστοιχεί στο *p[0],* το *temp_head* στο p[1], το '=' στο p[2], το *conjunction* στο p[3] και το '.' στο p[4]. Το p[0] είναι και η τιμή που επιστρέφει η συνάρτηση.

4.2.2.1 Εντοπισμός λαθών

Το yacc έχει τη δυνατότητα εντοπισμού λαθών στη γραμματική και εμφάνισης αντίστοιχων μηνυμάτων, όπως:

- Διπλά ονόματα συναρτήσεων
- Shift/reduce και reduce/reduce conflicts
- Εσφαλμένοι γραμματικοί κανόνες
- Ατέρμονη αναδρομή
- Κανόνες και λεκτικές μονάδες που δεν χρησιμοποιούνται
- Κανόνες και λεκτικές μονάδες που δεν έχουν οριστεί

4.3 Ο κώδικας του λογισμικού

Ο κώδικας του λογισμικού *Αντιγόνη* αποτελείται από το αρχείο Antigone.py και το βοηθητικό αρχείο GRtoEng.py, ενώ χρησιμοποιεί, όπως φάνηκε και στις προδιαγραφές, και το αρχείο ρυθμίσεων Config.ini. Στην ενότητα αυτή γίνεται

αρχικά η περιγραφή του αρχείου Config.ini και ακολουθεί μια σύντομη τεκμηρίωση του κώδικα. Για μια πιο εμπεριστατωμένη τεκμηρίωση ο αναγνώστης χρειάζεται να ανατρέξει τόσο στις προδιαγραφές του λογισμικού όσο και στον ίδιο τον κώδικα που συνοδεύεται από λεπτομερή σχόλια (βλ. Παράρτημα Z).

4.3.1 Το αρχείο ρυθμίσεων Config.ini

Είναι ένα αρχείο μορφής .txt το οποίο δίνει στο χρήστη τη δυνατότητα να εκτελεί το πρόγραμμα ορίζοντας κάποιες βασικές παραμέτρους, όπως το αρχεία χαρακτηριστικών, το αρχείο εξόδου κλπ. Οι παράμετροι αυτές δίνονται με τη μορφή

<παράμετρος> = <τιμή>

και είναι οι εξής:

- TagSetFile

Το όνομα, αν είναι στον ίδιο φάκελο, ή το πλήρες path του αρχείου αντιστοίχισης των χαρακτηριστικών μεταξύ FBT tagger και XLE

- XleGrammarFile

Το όνομα, αν είναι στον ίδιο φάκελο, ή το πλήρες path του αρχείου γραμματικής του XLE.

- OutputFile

Το όνομα, αν είναι στον ίδιο φάκελο, ή το πλήρες path του αρχείου εξόδου του προγράμματος,

- TemplateSection

Το όνομα της ενότητας (section) του αρχείου γραμματικής του XLE όπου βρίσκονται τα υποδείγματα τα οποία επεξεργάζεται το πρόγραμμα

- LexiconSection

Το όνομα της ενότητας (section) του αρχείου γραμματικής του XLE όπου βρίσκονται οι βασικοί τύποι, με την επιπλέον πληροφορία (συντακτική, σημασιολογική κλπ)

- LogFile

Το όνομα, αν είναι στον ίδιο φάκελο, ή το πλήρες path του αρχείου καταγραφής συμβάντων της εκτέλεσης του προγράμματος.

4.3.2 Το αρχείο GrtoEng.py

Περιέχει τη συνάρτηση ConvertGrToEng η οποία μεταγράφει σε αγγλικούς τους ελληνικούς χαρακτήρες του αρχείου εξόδου του επισημειωτή. Για το σκοπό αυτό κάνει η συνάρτηση χρήση του λεξικού dGRtoENG, το οποίο αποτελεί τη δομή αντιστοίχισης των χαρακτήρων στις δυο γλώσσες. Επίσης, περιέχει τη λίστα lTokenizerTags η οποία έχει τις ετικέτες με τις οποίες επισημειώνονται οι διαχωριστικές λεκτικές μονάδες (tokenizers) του μορφολογικού επισημειωτή του ΙΕΛ και οι οποίες απαιτούν διαφορετικό χειρισμό από το πρόγραμμα.

4.3.3 Το αρχείο Antigone.py

Αποτελεί το βασικό αρχείο του λογισμικού. Ο κώδικας είναι χωρισμένος σε εννέα αριθμημένα τμήματα κάθε ένα από τα οποία αποτελεί μια λογική μονάδα. Παρακάτω δίνεται η περιγραφή του κάθε τμήματος.

Τμήμα 1: Αρχικοποίηση

Είναι ο κώδικας αρχικοποίησης, όπου στο πρώτο μέρος βρίσκονται οι δηλώσεις ενσωμάτωσης των απαραίτητων για το πρόγραμμα αρθρωμάτων ενώ στο δεύτερο διαβάζεται το αρχείο ρυθμίσεων config.ini

Τμήμα 2: Ο αναλυτής PLY

Περιέχει τον κώδικα που είναι απαραίτητος για τη συντακτική ανάλυση των υποδειγμάτων και περιλαμβάνει τις δηλώσεις των κλάσεων των κόμβων των δέντρων AST (S1.1), τις απαραίτητες δηλώσεις του tokenizer (S1.2) και τους συντακτικούς κανόνες (S1.3).

Τμήμα 3: Οι συναρτήσεις

Στο τμήμα αυτό περιέχονται όλες οι συναρτήσεις που χρησιμοποιούνται στο κυρίως πρόγραμμα. Η περιγραφή τους δίνεται, για όσες κρίνεται απαραίτητο, στα τμήματα από τα οποία καλούνται.

Τμήμα 4: Εισαγωγή των χαρακτηριστικών

Είναι το τμήμα του κώδικα το οποίο διαβάζει τα χαρακτηριστικά του επισημειωτή και του XLE, όπως τα έχει δηλώσει ο χρήστης στο αρχείο αντιστοίχισης sTagSetFile. Τα χαρακτηριστικά αυτά φορτώνονται στη λίστα lFeatures ως αντικείμενα FeatureMap, εξάδες δηλαδή μεταβλητών που αντιστοιχούν στο μέρος του λόγου, στο χαρακτηριστικό και στην τιμή που παίρνει αυτό στον tagger και στο XLE. Επίσης τα

μέρη του λόγου αντιστοιχίζονται στο λεξικό dXLETagPOSMap για μεγαλύτερη ταχύτητα.

Τμήμα 5: Εισαγωγή των υποδειγμάτων

Είναι το τμήμα του κώδικα το οποίο διαβάζει τα υποδείγματα από την ενότητα sTemplateSection του αρχείου γραμματικής του XLE. Τα υποδείγματα διαβάζονται γραμμή προς γραμμή, καθαρίζονται από βοηθητικούς χαρακτήρες και σχόλια και φορτώνονται στο λεξικό dTemplates ως αντικείμενα Template.

Τμήμα 6: Εισαγωγή των λημμάτων

Είναι το τμήμα του κώδικα το οποίο διαβάζει το λεξικό λημμάτων από την ενότητα sLexiconSection του αρχείου γραμματικής του XLE, εφόσον βέβαια την έχει δηλώσει ο χρήστης. Τα λήμματα φορτώνονται στο λεξικό dLexEntries.

Τμήμα 7: Επεξεργασία των υποδειγμάτων

Είναι το τμήμα επεξεργασίας των υποδειγμάτων. Αρχικά, δημιουργούνται ο λεκτικός και ο συντακτικός αναλυτής (S7.1). Ακολουθεί η λεκτική και συντακτική ανάλυση των υποδειγμάτων που είναι αποθηκευμένα στο λεξικό dTemplates (S7.2). Η ανάλυση του κάθε υποδείγματος επιστρέφει ένα αντικείμενο Template το οποίο αποτελεί ένα αφηρημένο συντακτικό δέντρο (Abstact Syntax Tree - AST). Έπειτα γίνεται η τελική επεξεργασία των υποδειγμάτων με τη συνάρτηση FixTemplateCallsOf (S7.3). Αυτή αντικαθιστά κάθε κλήση υποδείγματος που βρίσκεται εντός του σώματος του αρχικού υποδείγματος από το σώμα του πρώτου. Τέλος, γίνεται εγγραφή του υποδείγματος, στην πλήρη του πλέον μορφή, στο αρχείο καταγραφής logFile για έλεγχο από τον χρήστη.

Τμήμα 8: Εισαγωγή των λεκτικών μονάδων

Στο τμήμα αυτό γίνεται το άνοιγμα του φακέλου TokenFiles και η αποθήκευση των ονομάτων των αρχείων που περιέχει στη λίστα lTokenFiles. Για κάθε ένα από αυτά γίνεται η μετεγγραφή των ελληνικών χαρακτήρων σε λατινικούς με τη συνάρτηση ConvertGrToEng, διαβάζονται οι λεκτικές μονάδες και αποθηκεύονται στη λίστα lTagTokens. Όσες είναι διπλές απορρίπτονται. Για το διάβασμα του αρχείου χρησιμοποιείται η διεπαφή (API) xml.etree.ElementTree.

Τμήμα 9: Εγγραφή των λεξικών καταχωρήσεων

Για κάθε λεκτική μονάδα που έχει καταχωρηθεί στη λίστα lTagTokens:

- ελέγχονται τα υποδείγματα με τα οποία μπορεί αυτή να αντιστοιχηθεί, με τη συνάρτηση CheckTokenTemplates

- δημιουργείται η καταχώρηση του λεξικού με όλες τις πληροφορίες που χρειάζεται, με τη συνάρτηση CreateLexiconEntry

- ελέγχεται η ύπαρξη διπλής καταχώρησης και σε αυτή την περίπτωση αυτή απορρίπτεται.

5. Εφαρμογή του λογισμικού *Αντιγόνη*

Στην παρούσα ενότητα δίνονται τα αποτελέσματα της πρακτικής εφαρμογής του λογισμικού *Αντιγόνη*. Το λογισμικό δοκιμάστηκε εντός του περιβάλλοντος της Python 3.3.1, σε φορητό υπολογιστή με επεξεργαστή Intel Core i5-3230M στα 2.6 GHz, μνήμη RAM 4 GB και λειτουργικό σύστημα Windows 8.

5.1 Εκτέλεση του λογισμικού

Για την εκτέλεση του προγράμματος χρησιμοποιήθηκαν τα παρακάτω αρχεία:

- το αρχείο των λεκτικών μονάδων Tokens.xml (Παράρτημα Α)
- το αρχείο αντιστοίχισης των χαρακτηριστικών TaggerXleFeatures.txt (Παράρτημα Β)
- το αρχείο γραμματικής του XLE DemoGrammar.lfg (Παράρτημα Γ)
- το αρχείο ρυθμίσεων Config.ini (Παράρτημα Δ)

Αποτέλεσμα της εκτέλεσης είναι:

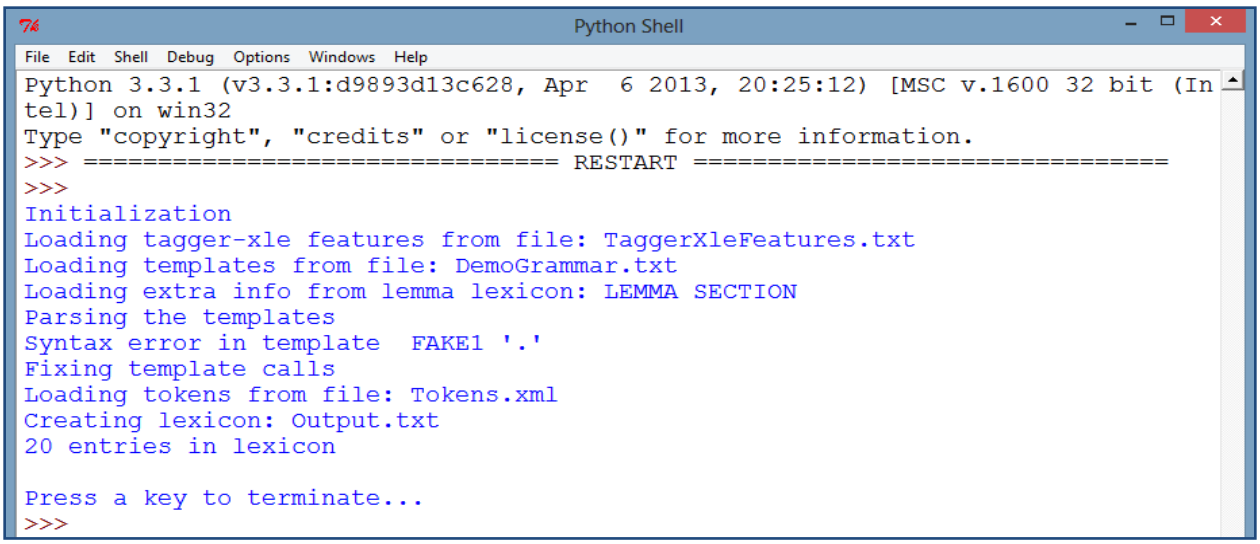
- το αρχείο εξόδου με τις λεξικές καταχωρήσεις Output.txt (Παράρτημα Ε)
- το αρχείο καταγραφής LogFile.txt (Παράρτημα Στ).

Τα παραδείγματα των υποδειγμάτων που χρησιμοποιούνται στη γραμματική εξυπηρετούν την ανάγκη επίδειξης του λογισμικού και ελέγχου του ως προς τις προδιαγραφές που έχουν τεθεί, αναδεικνύοντας τις επιμέρους λειτουργίες του. Δεν αποτελούν τα ίδια πρόταση προς το γλωσσολόγο. Έτσι, πέρα από τα λειτουργικά υποδείγματα, στο αρχείο γραμματικής περιέχονται και τα REC1 και REC2 τα οποία καλούν το ένα το άλλο, το FAKE1 που έχει συντακτικό λάθος και το FAKE2 που κάνει κλήση στο REC1.

Εκτελώντας το πρόγραμμα εμφανίζονται στην οθόνη πληροφορίες για τα αρχεία που χρησιμοποιεί και για την πορεία της εκτέλεσής του, μήνυμα λάθους για το υπόδειγμα FAKE1και το πλήθος των λεξικών καταχωρήσεων που προέκυψαν (Εικ. 11).

Στο αρχείο καταγραφής υπάρχει μήνυμα λάθους για αναδρομική κλήση στα υποδείγματα REC1 και REC2 και η πλήρης ανάλυση των υποδειγμάτων, σε μορφή λογικών εκφράσεων. Το FAKE1 είναι κενό λόγω του συντακτικού του λάθους ενώ στα REC1, REC2 και FAKE2 οι κλήσεις των υποδειγμάτων που έχουν σφάλμα έχουν αντικατασταθεί από την τιμή False.

```
(1)  REC1 = False
     REC2 = False
     FAKE2 = (^ GENDER) = NEU and False and (^ CASE) = ACC
```



Εικ. 11 Η οθόνη εκτέλεσης του λογισμικού

Ελέγχοντας το αρχείο εξόδου μπορούν να γίνουν οι εξής παρατηρήσεις:

- Όλες οι λεκτικές μονάδες του αρχείου εισόδου μεταφέρθηκαν στο λεξικό, συνοδευόμενες από την πληροφορία που δίνει ο επισημειωτής. Εμφανίζονται, δε, ομαδοποιημένες κατά μέρος του λόγου και σε αλφαβητική σειρά.
- Από την παραπάνω διαδικασία εξαιρούνται οι λεκτικές μονάδες οι οποίες ανήκουν σε μέρη του λόγου στα οποία δεν έχει δοθεί τιμή στις στήλες του XLE στο αρχείο αντιστοίχισης χαρακτηριστικών. Επίσης με την ίδια μέθοδο, μπορεί ο χρήστης να αποκρύπτει και συγκεκριμένα χαρακτηριστικά ή τιμές. Στη συγκεκριμένη περίπτωση, εξαιρέθηκαν τα σημεία στίξης, που ανήκουν στα tokenizers καθώς και η πτώση (CASE) και η τιμή ADJ του χαρακτηριστικού POS στα επίθετα (Aj).
- Έχει γίνει σωστή χρήση των υποδειγμάτων ώστε να μην υπάρχει πλεονάζουσα πληροφορία. Πχ στο (2) όλη η μορφολογική πληροφορία του ουσιαστικού 'spiti' εμπεριέχεται στο υπόδειγμα NEU_NOUN.

```
(2)  spiti N * @(NEU_NOUN spiti SG CM).
```

- Στις περιπτώσεις χαρακτηριστικών τα οποία δεν περιέχονται σε κάποιο υπόδειγμα, αυτά εμφανίζονται στη λεξική καταχώρηση αυτούσια.

```
(3)  einai V * @PRES3SG
                (^ TYPE) = MAIN
                (^ ASPECT) = IP
                (^ VOICE) = PASSIVE.
```

- Οι διπλές καταχωρήσεις, όπως το άρθρο 'το', έχουν σβηστεί.
- Οι καταχωρήσεις που προκύπτουν από την ίδια λεκτική μονάδα αλλά αντιστοιχούν σε διαφορετικό λήμμα, όπως το άρθρο και η αντωνυμία 'το', συνοδεύονται από το χαρακτηριστικό ETC.

- Υπάρχει η δυνατότητα κατασκευής υποδειγμάτων με τέτοιο τρόπο ώστε να αντιστοιχίζονται με συγκεκριμένα μόνο μέρη του λόγου. Αυτό επιτυγχάνεται με δυο τρόπους. Με τον έλεγχο της τιμής του χαρακτηριστικού POS ή με τη χρήση χαρακτηριστικού το οποίο απαντάται μόνο στο συγκεκριμένο μέρος του λόγου. Στην πρώτη περίπτωση ανήκει το υπόδειγμα PREP που αντιστοιχεί στις προθέσεις (4α), ενώ στη δεύτερη το ADJECTIVE για τα επίθετα (4β) όπου αντί του POS χρησιμοποιείται το DEGREE.

```
(4)  α. PREP(P) =          @(PRED P)
                           (^ POS) = PREP.

     β. ADJECTIVE(P D) = @(PRED P)
                           (^ DEGREE) = D.
```

- Η συντακτική (ή άλλη) πληροφορία που περιέχεται στο λεξικό λημμάτων προστέθηκε στις καταχωρήσεις που αντιστοιχούν στα λήμματα αυτά, όπως το υπόδειγμα DITRANS στο 'Edwse'.

```
(5)   Edwse V *            @NOT_PRES
                           @3SG
                           @(MOOD INDIC)
                           @(TENSE PAST)
                           (^ TYPE) = MAIN
                           (^ ASPECT) = PE
                           (^ VOICE) = ACTIVE
                           @(DITRANS dinw).
```

Πέρα από τον έλεγχο ορθότητας του λογισμικού πραγματοποιήθηκε και έλεγχος της επίδοσής του. Για το σκοπό αυτό χρησιμοποιήθηκε αρχείο με 5500 περίπου λεκτικές μονάδες. Η εκτέλεση του προγράμματος είχε σαν έξοδο λεξικό 2200 καταχωρήσεων και χρόνο εκτέλεσης 45 δευτερόλεπτα, ο οποίος κρίνεται ικανοποιητικός για το μέγεθος των δεδομένων.

5.2 Αξιολόγηση και προτάσεις

Αξιολογώντας το λογισμικό *Αντιγόνη* μπορεί να εξαχθεί το ασφαλές συμπέρασμα ότι επιτελεί το σκοπό του επιτυχώς. Δημιουργεί λεξικό μεταφέροντας όλη την μορφολογική πληροφορία του επισημειωτή, η οποία είναι απαραίτητη για τη γραμματική του XLE, αξιοποιώντας τόσο τη λειτουργικότητα των υποδειγμάτων όσο και τη συντακτική και σημασιολογική πληροφορία του λεξικού λημμάτων. Η δημιουργία πλήρων λεξικών για το XLE όμως, πιθανά να απαιτεί την αξιοποίηση κι άλλων εξωτερικών πηγών (Crouch & King, 2005).

Οι προτάσεις για περαιτέρω εργασία πάνω στο αντικείμενο θα προκύψουν αναμφισβήτητα από τη χρήση του εργαλείου από γλωσσολόγο. Από της έως τώρα αξιοποίησή του, διαφάνηκε μια πρόσθετη ανάγκη για επεξεργασία της συντακτικής και σημασιολογικής πληροφορίας ώστε να αξιοποιείται σε συνδυασμό με τη μορφολογική που παρέχει ο επισημειωτής. Ένα τέτοιο παράδειγμα είναι η χρήση υποδειγμάτων που περιέχουν και τα δυο είδη χαρακτηριστικών. Για να επιτευχθεί το παραπάνω απαιτείται η επεξεργασία (parsing) του λεξικού λημμάτων.

Μια ακόμη πρόταση για επέκταση του λογισμικού είναι και η διαδικασία επαύξησης του ήδη υπάρχοντος λεξικού, πέρα από τη δημιουργία νέου. Κάτι τέτοιο βέβαια απαιτεί τόσο την επεξεργασία του υπάρχοντος λεξικού όσο και ένα σύστημα ελέγχου των εκδόσεων (version control) των υποδειγμάτων.

Πηγές

Βιβλιογραφία

Butt, Miriam; King, Tracy Holloway; et al. A grammar writer's cookbook. Stanford, CA; CSLI Publications. 1999.

Butt, Miriam; Dipper, Stefanie; Frank, Anette; King, Tracy Holloway. Writing Large-Scale Parallel Grammars for English, French and German. Proceedings of the LFG99 Conference. The University of Manchester. 1999.

Butt, Miriam; Dyvik, Helge; King, Tracy Holloway; Masuichi, Hiroshi; Rohrer, Christian. The Parallel Grammar Project. Proceedings of COLING2002 Workshop on Grammar Engineering and Evaluation. 2002.

Crouch, R. S.; King, T. H. Unifying lexical resources. Proceedings of Interdisciplinary Workshop on the Identification and Representation of Verb Features and Verb Classes. Saarbruecken. 2005.

Dalrymple, Mary; Kaplan, R. M.; King, Tracy Holloway. Linguistic generalizations over descriptions. Proceedings of the LFG04 Conference. University of Canterbury. 2004.

Denis, Pascal; Kuhn, Jonas. Applying an LFG parser in conference resolution: Experiments and analysis. Proceedings of the LFG06 Conference. Konstanz University. 2006.

Dipper, Stefanie. Grammar modularity and its impact on grammar documentation. COLING '04 Proceedings of the 20th international conference on Computational Linguistics. Association for Computational Linguistics. 2004.

Fiotaki, Alexandra. Practice with the XLE platform: development of toy grammars of Modern Greek. Internship Report. Institute for language and speech processing/R.C. "ATHENA". 2012.

Kaplan, R. M., Riezler, S., King, T. H. ; Maxwell, J. T. ; Vasserman, A. Speed and accuracy in shallow and deep stochastic parsing. Human Language Technology Conference/North American Chapter of the Association for Computational Linguistics Meeting. 2004.

Lapointe, Steven Guy. A Theory of Grammatical Agreement. University of Massachusetts. 1980.

Papageorgiou, Harris; Prokopidis, Prokopis; Giouli, Voula; Piperidis, Stelios. A Unified POS Tagging Architecture and its Application to Greek. Proceedings of the 2nd Language Resources and Evaluation Conference. 2000.

Prokopidis Prokopis , Georgantopoulos Byron, Papageorgiou Harris. A suite of NLP tools for Greek. The 10th International Conference of Greek Linguistics. 2011.

Wescoat Michael T. Practical Instructions for Working with the Formalism of Lexical Functional Grammar. Summer Institute, Stanford University. 1987.

Διαδικτυακές διευθύνσεις

PANACEA: Platform for Automatic, Normalized Annotation and Cost-Effective Acquisition of Language Resources for Human Language Technologies

http://www.ilsp.gr/el/infoprojects/meta?view=project&task=show&id=10

(προσπελάστηκε 15.5.2013)

ILSP FBT Tagger tagset with examples

http://nlp.ilsp.gr/nlp/tagset_examples/tagset_en/index.html

(προσπελάστηκε 2.4.2013)

ILSP FBT Tagger overview

http://registry.elda.org/services/180

(προσπελάστηκε 2.4.2013)

XLE Documentation

http://www2.parc.com/isl/groups/nltt/xle/doc/xle_toc.html

(προσπελάστηκε 21.10.2013)

A Byte of Python

http://files.swaroopch.com/python/byte_of_python.pdf

(προσπελάστηκε 11.7.2013)

The Python Standard Library

https://docs.python.org/2/library/

(προσπελάστηκε 12.7.2013)

PLY (Python Lex-Yacc)

http://www.dabeaz.com/ply/ply.html

(προσπελάστηκε 15.5.2013)

Παράρτημα

Α: Το αρχείο των λεκτικών μονάδων

```
<?xml version='1.0' encoding='UTF-8'?>
<cesDoc xmlns="http://www.xces.org/schema/2003" version="0.4"><cesHeader
version="0.4"/>
  <text>
    <body>
      <p id="p1">
        <s id="s1">
          <t id="t1" word="O" tag="AtDfMaSgNm" lemma="o"/>
          <t id="t2" word="Nikos" tag="NoPrMaSgNm" lemma="Nikos"/>
          <t id="t3" word="mou" tag="PnPeMa01SgGeWe" lemma="egw"/>
          <t id="t4" word="to" tag="PnPeNe03SgAcWe" lemma="egw"/>
          <t id="t5" word="eipe" tag="VbMnIdPa03SgXxPeAvXx" lemma="legw"/>
          <t id="t6" word="oti" tag="CjSb" lemma="oti"/>
          <t id="t7" word="to" tag="AtDfNeSgNm" lemma="o"/>
          <t id="t8" word="spiti" tag="NoCmNeSgNm" lemma="spiti"/>
          <t id="t9" word="einai" tag="VbMnIdPr03SgXxIpPvXx" lemma="eimai"/>
          <t id="t10" word="palio" tag="AjBaNeSgNm" lemma="palios"/>
          <t id="t11" word="." tag="PTERM_P" lemma="."/>
        </s>
        <s id="s2">
          <t id="t12" word="Phga" tag="VbMnIdPa01SgXxPeAvXx" lemma="phgainw"/>
          <t id="t13" word="sth" tag="AsPpPaFeSgAc" lemma="stou"/>
          <t id="t14" word="douleia" tag="NoCmFeSgAc" lemma="douleia"/>
          <t id="t15" word="trechontas" tag="VbMnPpXxXxXxXxIpAvXx"
                                                    lemma="trechw"/>
          <t id="t16" word="." tag="PTERM_P" lemma="."/>
        </s>
        <s id="s3">
          <t id="t17" word="Edwse" tag="VbMnIdPa03SgXxPeAvXx" lemma="dinw"/>
          <t id="t18" word="to" tag="AtDfNeSgAc" lemma="o"/>
          <t id="t19" word="vivlio" tag="NoCmNeSgAc" lemma="vivlio"/>
          <t id="t20" word="ston" tag="AsPpPaMaSgAc" lemma="stou"/>
          <t id="t21" word="Kwsta" tag="NoPrMaSgAc" lemma="Kwstas"/>
          <t id="t22" word="me" tag="AsPpSp" lemma="me"/>
          <t id="t23" word="eucharisthsh" tag="NoCmFeSgAc"
                                                  lemma="eucharisthsh"/>
          <t id="t24" word="." tag="PTERM_P" lemma="."/>
        </s>
      </p>
    </body>
  </text>
</cesDoc>
```

Β: Το αρχείο αντιστοίχισης των χαρακτηριστικών

```
#
#         ILSP Tagger                         XLE
#---------------------------------- ------------------------------
#POS      Feature        Value     POS       Feature        Value
#---      -------        -----     ---       -------        ----
No,       Pos,           No,       N,        POS,           N
No,       Type,          Cm,       N,        TYPE,          CM
No,       Type,          Pr,       N,        TYPE,          PR
No,       Gender,        Ma,       N,        GENDER,        MASC
No,       Gender,        Fe,       N,        GENDER,        FEM
No,       Gender,        Ne,       N,        GENDER,        NEU
No,       Number,        Sg,       N,        NUM,           SG
No,       Number,        Pl,       N,        NUM,           PL
No,       Case,          Nm,       N,        CASE,          NOM
No,       Case,          Ge,       N,        CASE,          GEN
No,       Case,          Ac,       N,        CASE,          ACC
No,       Case,          Da,       N,        CASE,          DAT
No,       Case,          Vo,       N,        CASE,          VOC

Aj,       Pos,           Aj,       ADJ,      POS,
Aj,       Degree,        Ba,       ADJ,      DEGREE,        BA
Aj,       Degree,        Cp,       ADJ,      DEGREE,        CP
Aj,       Degree,        Su,       ADJ,      DEGREE,        SU
Aj,       Gender,        Ma,       ADJ,      GENDER,        MASC
Aj,       Gender,        Fe,       ADJ,      GENDER,        FEM
Aj,       Gender,        Ne,       ADJ,      GENDER,        NEU
Aj,       Number,        Sg,       ADJ,      NUM,           SG
Aj,       Number,        Pl,       ADJ,      NUM,           PL
Aj,       Case,          Nm,       ADJ,
Aj,       Case,          Ge,       ADJ,
Aj,       Case,          Ac,       ADJ,
Aj,       Case,          Da,       ADJ,
Aj,       Case,          Vo,       ADJ,

Nm,       Pos,           Nm,       NUMERAL,  POS,           NUM
Nm,       Type,          Cd,       NUMERAL,  TYPE,          CD
Nm,       Type,          Od,       NUMERAL,  TYPE,          OD
Nm,       Type,          Ml,       NUMERAL,  TYPE,          ML
Nm,       Type,          An,       NUMERAL,  TYPE,          AN
Nm,       Type,          Ct,       NUMERAL,  TYPE,          CT
Nm,       Gender,        Ma,       NUMERAL,  GENDER,        MASC
Nm,       Gender,        Fe,       NUMERAL,  GENDER,        FEM
Nm,       Gender,        Ne,       NUMERAL,  GENDER,        NEU
Nm,       Number,        Sg,       NUMERAL,  NUM,           SG
Nm,       Number,        Pl,       NUMERAL,  NUM,           PL
```

```
Nm,     Case,       Nm,     NUMERAL,  CASE,       NOM
Nm,     Case,       Ge,     NUMERAL,  CASE,       GEN
Nm,     Case,       Ac,     NUMERAL,  CASE,       ACC
Nm,     Case,       Da,     NUMERAL,  CASE,       DAT
Nm,     Case,       Vo,     NUMERAL,  CASE,       VOC
Nm,     Function,   Aj,     NUMERAL,  FUNCTION,   AJ
Nm,     Function,   No,     NUMERAL,  FUNCTION,   NO
Nm,     Function,   Ad,     NUMERAL,  FUNCTION,   AD

At,     Pos,        At,     D,        POS,        D
At,     Type,       Df,     D,        DEF,        +
At,     Type,       In,     D,        DEF,        -
At,     Gender,     Ma,     D,        GENDER,     MASC
At,     Gender,     Fe,     D,        GENDER,     FEM
At,     Gender,     Ne,     D,        GENDER,     NEU
At,     Number,     Sg,     D,        NUM,        SG
At,     Number,     Pl,     D,        NUM,        PL
At,     Case,       Nm,     D,        CASE,       NOM
At,     Case,       Ge,     D,        CASE,       GEN
At,     Case,       Ac,     D,        CASE,       ACC
At,     Case,       Da,     D,        CASE,       DAT
At,     Case,       Vo,     D,        CASE,       VOC

Vb,     Pos,        Vb,     V,        POS,
Vb,     Type,       Mn,     V,        TYPE,       MAIN
Vb,     Type,       Im,     V,        TYPE,       IMPERS
Vb,     Mood,       Id,     V,        MOOD,       INDIC
Vb,     Mood,       Mp,     V,        MOOD,       IMPER
Vb,     Mood,       Nf,     V,        MOOD,       INFIN
Vb,     Mood,       Pp,     V,        MOOD,       PARTICIPLE
Vb,     Tense,      Pr,     V,        TENSE,      PRES
Vb,     Tense,      Pa,     V,        TENSE,      PAST
Vb,     Tense,      Xx,     V,        TENSE,
Vb,     Person,     01,     V,        SUBJ PERS,  1
Vb,     Person,     02,     V,        SUBJ PERS,  2
Vb,     Person,     03,     V,        SUBJ PERS,  3
Vb,     Person,     Xx,     V,        SUBJ PERS,
Vb,     Number,     Sg,     V,        SUBJ NUM,   SG
Vb,     Number,     Pl,     V,        SUBJ NUM,   PL
Vb,     Number,     Xx,     V,        SUBJ NUM,
Vb,     Gender,     Ma,     V,        GENDER,     MASC
Vb,     Gender,     Fe,     V,        GENDER,     FEM
Vb,     Gender,     Ne,     V,        GENDER,     NEU
Vb,     Gender,     Xx,     V,        GENDER,
Vb,     Aspect,     Ip,     V,        ASPECT,     IP
Vb,     Aspect,     Pe,     V,        ASPECT,     PE
Vb,     Voice,      Av,     V,        VOICE,      ACTIVE
```

```
Vb,     Voice,          Pv,     V,      VOICE,          PASSIVE
Vb,     Case,           Nm,     V,      CASE,           NOM
Vb,     Case,           Ge,     V,      CASE,           GEN
Vb,     Case,           Ac,     V,      CASE,           ACC
Vb,     Case,           Da,     V,      CASE,           DAT
Vb,     Case,           Vo,     V,      CASE,           VOC
Vb,     Case,           Xx,     V,      CASE,

Pn,     Pos,            Pn,     PRON,   POS,            PRON
Pn,     Type,           Pe,     PRON,
Pn,     Type,           Dm,     PRON,
Pn,     Type,           Po,     PRON,
Pn,     Type,           Id,     PRON,
Pn,     Type,           Ir,     PRON,
Pn,     Type,           Re,     PRON,
Pn,     Type,           Ri,     PRON,
Pn,     Person,         01,     PRON,   PERS,           1
Pn,     Person,         02,     PRON,   PERS,           2
Pn,     Person,         03,     PRON,   PERS,           3
Pn,     Gender,         Ma,     PRON,   GENDER,         MASC
Pn,     Gender,         Fe,     PRON,   GENDER,         FEM
Pn,     Gender,         Ne,     PRON,   GENDER,         NEU
Pn,     Number,         Sg,     PRON,   NUM,            SG
Pn,     Number,         Pl,     PRON,   NUM,            PL
Pn,     Case,           Nm,     PRON,   CASE,           NOM
Pn,     Case,           Ge,     PRON,   CASE,           GEN
Pn,     Case,           Ac,     PRON,   CASE,           ACC
Pn,     Case,           Da,     PRON,   CASE,           DAT
Pn,     Case,           Vo,     PRON,   CASE,           VOC
Pn,     Inflection,     St,     PRON,
Pn,     Inflection,     We,     PRON,
Pn,     Inflection,     Xx,     PRON,

Ad,     Pos,            Ad,     ADV,    POS,            ADV
Ad,     Type,           Xx,     ADV,
Ad,     Degree,         Ba,     ADV,
Ad,     Degree,         Cp,     ADV,
Ad,     Degree,         Su,     ADV,

As,     Pos,            As,     PREP,   POS,            PREP
As,     Type,           Pp,     PREP,
As,     Form,           Sp,     PREP,
As,     Form,           Pa,     PREP,
As,     Gender,         Ma,     PREP,   GENDER,         MASC
As,     Gender,         Fe,     PREP,   GENDER,         FEM
As,     Gender,         Ne,     PREP,   GENDER,         NEU
As,     Number,         Sg,     PREP,   NUM,            SG
```

```
As,     Number,     Pl,     PREP,     NUM,     PL
As,     Case,       Nm,     PREP,     CASE,    NOM
As,     Case,       Ge,     PREP,     CASE,    GEN
As,     Case,       Ac,     PREP,     CASE,    ACC
As,     Case,       Da,     PREP,     CASE,    DAT
As,     Case,       Vo,     PREP,     CASE,    VOC

Cj,     Pos,        Cj,     CONJ,     POS,     CONJ
Cj,     Type,       Co,     CONJ,     TYPE,    COORD
Cj,     Type,       Sb,     CONJ,     TYPE     SUBCOORD

Ij,     Pos,        Ij,     INTERJECT,POS,     INTERJECT

Pt,     Pos,        Pt,     PARTICLE, POS,     PARTICLE
Pt,     Type,       Fu,     PARTICLE, TYPE,    PART_FU
Pt,     Type,       Ne,     PARTICLE, TYPE,    PART_NE
Pt,     Type,       Sj,     PARTICLE, TYPE,
Pt,     Type,       Ot,     PARTICLE, TYPE,

Rg,     Pos,        Rg,
Rg,     Type,       Fw,
Rg,     Type,       Ab,
Rg,     Type,       An,
Rg,     Type,       Sy,
Rg,     Translit,   Tr,
Rg,     Translit,   Or,
Rg,     Translit,   Xx,
```

Γ: Το αρχείο γραμματικής XLE

```
DEMO   GREEK   CONFIG (1.0)
ROOTCAT   S.
FILES  output.txt.
LEXENTRIES   (LEMMA LEXICON) (TAGGER LEXICON).
RULES   (DEMO ENGLISH).
TEMPLATES   (MY TEMPLATES) (OTHER TEMPLATES).
GOVERNABLERELATIONS SUBJ OBJ OBJ2 COMP XCOMP OBL OBL-?+.
SEMANTICFUNCTIONS ADJUNCT  TOPIC FOCUS POSS STANDARD.
NONDISTRIBUTIVES NUM PERS CONJ-FORM.
EPSILON e.
OPTIMALITYORDER NOGOOD.
CHARACTERENCODING utf-8.
----

DEMO   ENGLISH   RULES (1.0)

   S --> e: (^ TENSE);
      NP: (^ SUBJ)=!
          (! CASE)=NOM;
        VP.

   VP --> V
       NP: (^ OBJ)=!
            (! CASE)=ACC.

   NP --> (D)
       N
       PP*:! $ (^ ADJUNCT).
----

MY TEMPLATES (1.0)

DET(P) =       (^ DEF) = P.

PRED(P) =      (^ PRED) = P.

CASE(C) =      (^ CASE) = C.

NUMB(N) =      (^ NUM) = N.
```

```
GEND(G) =       (^ GENDER) = G.

PERS(P) =       (^ PERS) = P.

TENSE(T) =      (^ TENSE) = T.

MOOD(M) =       (^ MOOD) = M.

3PERSUBJ =      (^ SUBJ PERS) = 3.

SINGSUBJ =      (^ SUBJ NUM) = SG.

ADJECTIVE(P D) =
                @(PRED P)
                (^ DEGREE) = D.

PREP(P) =       @(PRED P)
                (^ POS) = PREP.

3SG =           @3PERSUBJ
                @SINGSUBJ.

PRESENT =       @(TENSE PRES)
                @(MOOD INDIC).

PRES3SG =       @PRESENT
                @3SG.

NEUTRAL =       {@(CASE NOM)
                |@(CASE ACC)}
                @(GEND NEU).

NOUN(P NR TY) =@(PRED P)
                @(NUMB NR)
                (^ POS) = N
                (^ TYPE) = TY.

NEU_NOUN(P NR TY) =
                @(NOUN P NR TY)
                @NEUTRAL.
```

```
NOT_PRES =      ~@PRESENT
                (^ TENSE)
                (^ MOOD).

REC1 =          @REC2.

REC2 =          @REC1.

FAKE1 =         @(NOUN.

FAKE2 =         @(GEND NEU)
                @REC1
                @(CASE ACC).
----

OTHER TEMPLATES

INTRANS(P) =    (^ PRED) = 'P<(^SUBJ)>'.

TRANS (P) =     @(PASS (^ PRED)='P<(^ SUBJ) (^ OBJ)>').

OPT-TRANS(P) = { @(TRANS P)
                | @(INTRANS P) }.

DITRANS(P) =    {@(PASS @(DAT-SHIFT (^ PRED)='P<(^ SUBJ) (^
           OBJ)( ^ OBJ2) >'))
           |@(PASS @(DAT-SHIFT (^ PRED)='P<(^ SUBJ) (^ OBJ)>'))
           |@(PASS @(DAT-SHIFT (^ PRED)='P<(^ SUBJ) (^ OBJ) (^
           OBL-TO)>'))}.

PASS(SCHEMATA) =
           {SCHEMATA
           |SCHEMATA
           (^ PASSIVE) = +
           (^ PARTICIPLE)=c past
           (^ OBJ)-->(^ SUBJ)
           (^ OBJ2) --> NULL
           {(^ SUBJ)-->(^ OBL-AG)
           |(^ SUBJ)-->NULL}}.
```

```
DAT-SHIFT(SCHEMATA) =
               {SCHEMATA
               (^ OBL PCASE) =c TO
               |SCHEMATA
               (^ OBJ) --> (^ OBJ)
               (^ OBL) --> (^ OBJ2)}.
----

LEMMA LEXICON

trww      V * @(OPT-TRANS trww).

dinw      V * @(DITRANS dinw).

trechw    V * @(INTRANS trechw).
----
```

Δ: Το αρχείο ρυθμίσεων του λογισμικού *Αντιγόνη*

```
#
# This is the configuration file for the  ILSP PoS Tagger to XLE
# Parser connection program, built in Python
# The user has to give valid values for all the parameters
# listed before executing the program
#

TagSetFile=TaggerXleFeatures.txt
XleGrammarFile=DemoGrammar.txt
ILSPTaggerFile=Tokens.xml
OutputFile=Output.txt
LogFile=Logfile.txt
TemplateSection=MY TEMPLATES
LexiconSection=LEXICON SECTION
LemmaSection=LEMMA LEXICON
```

Ε: Το αρχείο εξόδου του λογισμικού *Αντιγόνη*

```
TOKEN GREEK LEXICON (1.0)

palio ADJ *
      @(GEND NEU)
      @(NUMB SG)
      @(ADJECTIVE palios BA).

me PREP *
      @(PREP me).

sth PREP *
      @(GEND FEM)
      @(CASE ACC)
      @(NUMB SG)
      @(PREP stou).

ston PREP *
      @(GEND MASC)
      @(CASE ACC)
      @(NUMB SG)
      @(PREP stou).

O D *
      @(GEND MASC)
      @(DET +)
      @(CASE NOM)
      @(NUMB SG)
      (^ POS) = D.

to D *
      @(DET +)
      @(NUMB SG)
      @NEUTRAL
      (^ POS) = D
      ETC.

oti CONJ *
      (^ POS) = CONJ.
```

```
Kwsta N *
      @(GEND MASC)
      @(CASE ACC)
      @(NOUN Kwstas SG PR).

Nikos N *
      @(GEND MASC)
      @(CASE NOM)
      @(NOUN Nikos SG PR).

douleia N *
      @(GEND FEM)
      @(CASE ACC)
      @(NOUN douleia SG CM).

eucharisthsh N *
      @(GEND FEM)
      @(CASE ACC)
      @(NOUN eucharisthsh SG CM).

spiti N *
      @(NEU_NOUN spiti SG CM).

vivlio N *
      @(NEU_NOUN vivlio SG CM).

mou PRON *
      @(PERS 1)
      @(PRED egw)
      @(GEND MASC)
      @(CASE GEN)
      @(NUMB SG)
      (^ POS) = PRON.

to PRON *
      @(PERS 3)
      @(PRED egw)
      @(NUMB SG)
      @NEUTRAL
      (^ POS) = PRON
      ETC.
```

```
Edwse V *
      @(TENSE PAST)
      @3SG
      @NOT_PRES
      @(MOOD INDIC)
      (^ TYPE) = MAIN
      (^ ASPECT) = PE
      (^ VOICE) = ACTIVE
      @(DITRANS dinw).

Phga V *
      @(TENSE PAST)
      @NOT_PRES
      @SINGSUBJ
      @(MOOD INDIC)
      (^ TYPE) = MAIN
      (^ SUBJ PERS) = 1
      (^ ASPECT) = PE
      (^ VOICE) = ACTIVE.

einai V *
      @PRES3SG
      (^ TYPE) = MAIN
      (^ ASPECT) = IP
      (^ VOICE) = PASSIVE.

eipe V *
      @(TENSE PAST)
      @3SG
      @NOT_PRES
      @(MOOD INDIC)
      (^ TYPE) = MAIN
      (^ ASPECT) = PE
      (^ VOICE) = ACTIVE.

trechontas V *
      @(MOOD PARTICIPLE)
      (^ TYPE) = MAIN
      (^ ASPECT) = IP
      (^ VOICE) = ACTIVE
      @(INTRANS trechw).
```

Στ: Το αρχείο καταγραφής του λογισμικού *Αντιγόνη*

```
Fixing Template Calls

Error: Recursive template REC1 in REC2 found!

Printing Templates

MOOD (M)= (^ MOOD) = M

DET (P)= (^ DEF) = P

PRED (P)= (^ PRED) = P

PRES3SG = (^ TENSE) = PRES  and (^ MOOD) = INDIC  and (^ SUBJ
PERS) = 3  and (^ SUBJ NUM) = SG

PREP (P)= (^ PRED) = P  and (^ POS) = PREP

FAKE2 = (^ GENDER) = NEU  and False and (^ CASE) = ACC

SINGSUBJ = (^ SUBJ NUM) = SG

ADJECTIVE (P D)= (^ PRED) = P  and (^ DEGREE) = D

NEUTRAL = ((^ CASE) = NOM  or (^ CASE) = ACC ) and (^ GENDER) =
NEU

NOUN (P NR TY)= (^ PRED) = P  and (^ NUM) = NR  and (^ POS) = N
and (^ TYPE) = TY

3PERSUBJ = (^ SUBJ PERS) = 3

FAKE1 =

PRESENT = (^ TENSE) = PRES  and (^ MOOD) = INDIC

CASE (C)= (^ CASE) = C

GEND (G)= (^ GENDER) = G

REC2 = False

REC1 = False

3SG = (^ SUBJ PERS) = 3  and (^ SUBJ NUM) = SG

NUMB (N)= (^ NUM) = N
```

```
PERS (P)= (^ PERS) = P

TENSE (T)= (^ TENSE) = T

NEU_NOUN (P NR TY)= (^ PRED) = P  and (^ NUM) = NR  and (^ POS)
= N  and (^ TYPE) = TY  and ((^ CASE) = NOM  or (^ CASE) = ACC )
and (^ GENDER) = NEU

NOT_PRES =  ~ ((^ TENSE) = PRES  and (^ MOOD) = INDIC ) and (^
TENSE) and (^ MOOD)
```

Z: Ο κώδικας του λογισμικού *Αντιγόνη*

```
#-------------------------------------------------------------------------------------
#
#                              Antigone Lexicon Creator Tool
#
# Source 1:     ilsp_npl POS tagger tool (http://nlp.ilsp.gr/soaplab2-axis/)
# Source 2:     XLE grammar templates
# Source 3:     XLE grammar lemmas
# Destination:  XLE grammar lexicon
# Description:  Reads the output file of ilsp_npl and for each token (word)
#               creates a lexicon entry in the XLE grammar file. Each entry
#               includes a list of the templates that match to the
#               features of the corersponding token
#
#
#-------------------------------------------------------------------------------------

#--------------------------------------
#
# SECTION 1: Initialization
#
#--------------------------------------

print("Initialization")

#
# S1.1 General Declarations
#

import codecs, sys, copy, re
import xml.etree.ElementTree as ET
from GrtoEng import ConvertGrToEng, lTokenizerTags
from operator import attrgetter

sys.path.insert(0,"../..")

if sys.version_info[0] >= 3:
    raw_input = input

#
# S1.2 Reading config.ini
#

sTagSetFile = ''
sXleGrammarFile = ''
sTaggerFile = ''
sOutputFile = ''
sLogFile = ''
sTemplateSection = ''
sLexiconSection = ''
sLemmaSection = ''
#
configFile = open('config.ini')
while True:
    sLine = configFile.readline()
    if len(sLine) != 0:
        if sLine.find('=')>=0:
            sValue = sLine.split('=')[1].rstrip('\n')
            if sLine.startswith('TagSetFile'): sTagSetFile = sValue
            elif sLine.startswith('XleGrammarFile'): sXleGrammarFile = sValue
            elif sLine.startswith('TemplateSection='): sTemplateSection = sValue
```

```
            elif sLine.startswith('ILSPTaggerFile'): sTaggerFile = sValue
            elif sLine.startswith('LogFile'): sLogFile = sValue
            elif sLine.startswith('OutputFile'): sOutputFile = sValue
            elif sLine.startswith('LexiconSection'): sLexiconSection = sValue
            elif sLine.startswith('LemmaSection'): sLemmaSection = sValue
    else:
        break

logFile = open(sLogFile, 'wt', 1, 'utf-8')



#--------------------------------------
#
# SECTION 2: PLY parser
#
#--------------------------------------


#
# S2.1 Classes of the Abstract Syntax Tree nodes
#

class Template:
    '''The object which holds all the data of each template.
       It is the root node of the template's AST.
    '''
    def __init__(self, String, ParamList, Fixed, AndList, IncludeList):
        self.String = String
        self.ParamList = list(ParamList)
        self.Fixed = Fixed
        self.AndList = AndList
        self.IncludeList = list(IncludeList)


class TemplateHead:
    '''
        A template head consists of the template name and a list of the parameters.
        Terminal node.
    '''
    def __init__(self, Name, ParamList):
        self.Name = Name
        self.ParamList = list(ParamList)


class CharAssign:
    ''' A CharAsign consists of the characteristic/feature, the operator (= or ~=) and
the value.
        Terminal node.
    '''
    def __init__(self, Char, Operator, Value):
        self.Char = Char
        self.Operator = Operator
        self.Value = Value


class Negation(list):
    ''' Negation is defined as a list althougth there is always one element. This is
because of the need
        of treating it the same way as AndList and OrList
    '''
    def __init__(self):
```

```
        list.__init__(self)


class AndList(list):
    '''
        A list of items (other lists, charassigns, template calls) conected with the
logical AND.
        Non terminal node.
    '''
    def __init__(self):
        list.__init__(self)


class Conjunction(AndList):
    '''
        A list of items IN BRACKETS (other lists, charassigns, template calls)
conected with the logical AND.
        Non terminal node.
    '''
    def __init__(self):
        AndList.__init__(self)


class OrList(list):
    '''
        A list of items (other lists, charassigns, template calls) conected with the
logical OR.
        Non terminal node.
    '''
    def __init__(self):
        list.__init__(self)


class TemplateCall:
    '''
        A template call consists of the template name and a list of the variables
passed.
        Terminal node.
    '''
    def __init__(self, Template, VarList):
        self.Template = Template
        self.VarList = list(VarList)


#
# S2.2 Tokenizer code
#

tokens = ('NAME', 'EXTRA')
literals = ['=', '~', '@', '(', ')', '.', '^', '|', "{", "}", "'", "<", ">", "+", "-",
"[", "]"]

t_NAME    = r'[_a-zA-Z0-9][\-a-zA-Z0-9_]*'
t_EXTRA  = r'<.*>'
t_ignore = " \t"

# dictionary of names
names = { }

def t_newline(t):
    r'\n+'
    t.lexer.lineno += t.value.count("\n")
```

```
def t_error(t):
    print("Illegal character '%s'" % t.value[0])
    t.lexer.skip(1)


#
# S2.3 Parsing rules
#

def p_template(p):
    'template : temp_head "=" and_list "."'
    p[0]= Template(p[1].Name, p[1].ParamList, False, p[3], [])


def p_temp_head(p):
    '''temp_head : temp_name
                 | temp_name "(" param_list ")"
    '''
    if len(p) == 5:
        lParams = p[3]
    else:
        lParams = []
    p[0] = TemplateHead(p[1], lParams)


def p_temp_name(p):
    'temp_name : NAME'
    p[0]= p[1]


def p_param_list(p):
    '''param_list : param param_list
                  | param
    '''
    p[0] = list()
    p[0].append(p[1])
    if len(p) > 2:
        p[0].extend(p[2])


def p_param(p):
    'param : NAME'
    p[0]= p[1]


def p_and_list(p):
    '''and_list : item and_list
                | item
    '''
    p[0] = AndList()
    p[0].append(p[1])
    if len(p) > 2:
        p[0].extend(p[2])


def p_conjunction(p):
    '''conjunction : "[" and_list "]"
    '''
    p[0] = p[2]
    p[0].__class__ = Conjunction


def p_negation(p):
```

```
    '''negation : "~" item'''
    p[0] = Negation()
    p[0].append(p[2])


def p_item(p):
    '''item : char_assign
            | temp_call
            | negation
            | disjunction
            | conjunction
    '''
    p[0]=p[1]


def p_disjunction(p):
    '''
       disjunction : "{" or_list "}"
    '''
    p[0] = p[2]


def p_or_list(p):
    '''or_list : and_list "|" or_list
               | and_list
    '''
    p[0] = OrList()
    p[0].append(p[1])
    if len(p) > 2:
        p[0].extend(p[3])


def p_char_assign(p):
    '''char_assign : "(" "^" char ")" operator NAME
                   | "(" "^" char ")" operator "+"
                   | "(" "^" char ")" operator "'" NAME EXTRA "'"
                   | "(" "^" char ")" operator "'" NAME "'"
                   | "(" "^" char ")"
    '''
    if len(p) > 5:
        if p[6].startswith("'"):
            val = p[7]
            extra = p[8].lstrip("<").rstrip(">")
        else:
            val = p[6]
            extra = ""
        p[0]=CharAssign(p[3], p[5], val)
    else:
        p[0]=CharAssign(p[3], "", "")
    #
    global lIncItems
    lIncItems.append(p[0])


def p_char(p):
    '''char : NAME
            | NAME NAME
            | NAME NAME NAME
    '''
    if len(p)==2:
        p[0]=p[1]
    elif len(p)==3:
        p[0]=p[1] + " " + p[2]
```

```
    elif len(p)==4:
            p[0]=p[1] + " " + p[2] + " " + p[3]


def p_operator(p):
    '''operator : "="
                | "~" "="
    '''
    if len(p) == 2:
        p[0] = p[1]
    else:
        p[0] = p[1] + p[2]


def p_temp_call(p):
    '''temp_call : "@" NAME
                 | "@" "(" NAME var_list ")"'''
    #
    if p[2].strip() !="(":
        sTemp = p[2]
        lVars = []
    else:
        sTemp = p[3]
        lVars = p[4]
    #
    p[0] = TemplateCall(sTemp, lVars)
    global lIncItems
    lIncItems.append(p[0])


def p_var_list(p):
    '''var_list : var var_list
                | var '''
    p[0] = list()
    p[0].append(p[1])
    if len(p) > 2:
        p[0].extend(p[2])


def p_var(p):
    '''var : NAME
           '''
    p[0] = p[1]


def p_error(p):
    if p:
        print("Syntax error in template ", sTempName ,"'%s'" % p.value)
    else:
        print("Syntax error at EOF")
    sys.exit()

#--------------------------------------
#
# SECTION 3: Functions
#
#--------------------------------------

def FixTemplateCallsOf(inTempName, inList):
    ''' Searches througth the items of the inTemp's inList for template calls and
calls the
        'ReplaceParameters' function when it finds one. Returns the inList with each
of the
```

```
        template calls replaced by its body.
    '''
    #
    #logFile.write("Calling FixTemplateCallsOf " + inTempName + "\n")
    lRecursion.append(inTempName)
    i = -1
    for myItem in inList:
        i += 1
        if isinstance(myItem, TemplateCall):
            #
            myTempCallName = myItem.Template
            #logFile.write("      Found template call" + myTempCallName + " in " +
inTempName + "\n")
            if myTempCallName in dTemplates.keys():
                if myTempCallName not in lRecursion:
                    if not dTemplates[myTempCallName].Fixed:
                        dTemplates[myTempCallName].AndList =
FixTemplateCallsOf(myTempCallName, dTemplates[myTempCallName].AndList)
                        dTemplates[myTempCallName].Fixed = True
                        #logFile.write("Template " + inRootTemp + " fixed! \n")
                    inList[i] = ReplaceParameters(myItem,
dTemplates[myTempCallName].AndList )
                else:
                    logFile.write("Error: Recursive template " + myTempCallName +" in
" + inTempName + " found!\n\n")
                    inList[i] = False
            else:
                logFile.write("Template " + myTempCallName + " not found!\n\n")
                inList[i] = [False]
        elif isinstance(myItem, list):
            inList[i] = FixTemplateCallsOf(inTempName, myItem)
    #
    lRecursion.pop()
    return inList


def ReplaceParameters(inTempCall, inList):
    ''' Returns a list of the items of the AndList of the template inTempName
        All the parameters are replaced by the values in inVarList
        Template of inTempCall must have been already fixed
    '''
    #
    inTempName = inTempCall.Template
    inVarList = inTempCall.VarList
    #
    i = -1
    try:
        myList = copy.deepcopy(inList)
    except:
        myList = [False]
    else:
        for myItem in myList:
            i += 1
            if isinstance(myItem, list):
                myList[i] = ReplaceParameters(inTempCall, myItem)
            elif isinstance(myItem, CharAssign):
                # if Value is a parameter then replace it by the corresponding Var
                if myItem.Value in dTemplates[inTempName].ParamList:
                    iParam = dTemplates[inTempName].ParamList.index(myItem.Value)
                    myList[i].Value = inVarList[iParam]
            elif myItem == False:
                pass
            else:
```

```
                print("ReplaceTemplateCall -> Template", inTempName, "not fixed!",
myItem)
    return myList


def PrintTemplate(sTemp):
    '''
    '''
    global dTemplates
    #try:
    if len(dTemplates[sTemp].ParamList) > 0:
        sParamString='('
        i = -1
        for sParam in dTemplates[sTemp].ParamList:
            i += 1
            if i > 0:
                sParamString += ' '
            sParamString += sParam
        sParamString += ')'
    else:
        sParamString=''
    #
    sAndList = ListToString(dTemplates[sTemp].AndList)
    logFile.write("\n" + sTemp + " " + sParamString + '= ' + sAndList + "\n")
    #except:
    #    print("Cannot print template", sTemp)


def ListToString(inList):
    '''Returns a string representation of the inList (AndList, Conjunction, OrList or
Negation) '''
    sReturn = ''
    #
    if isinstance(inList, list):
        if isinstance(inList, AndList):
            sBool = " and "
        elif isinstance(inList, OrList):
            sBool = " or "
        #
        i = -1
        sReturn = ""
        for myItem in inList:
            i += 1
            if i > 0:
                sReturn += sBool
            #
            if isinstance(myItem, list):
                sReturn += ListToString(myItem)
            elif isinstance(myItem, CharAssign):
                if myItem.Operator:
                    sReturn += '(^ ' + myItem.Char + ') ' + myItem.Operator + ' ' +
myItem.Value + ' '
                else:
                    sReturn += '(^ ' + myItem.Char + ')'
            else:
                sReturn += str(myItem)
        #
        if isinstance(inList, Conjunction) or isinstance(inList, OrList) or
isinstance(inList, Negation) and len(inList[0]) > 1 :
            sReturn = '(' + sReturn + ')'
        #
        if isinstance(inList, Negation):
                sReturn = " ~ " + sReturn
```

```
    else:
        print("Not a list:", str(inList))
        sReturn = str(inList)
    return sReturn

def GetXleCharFromIndex(sTagPOS, iIndex):
    #
    sXleFeat = ''
    i = -1
    iPos = -1
    for myFeature in lFeatures:
        i += 1
        if sTagPOS == myFeature.TagPos:
            iPos += 1
            if iIndex == iPos:
                sXleFeat = myFeature.XleFeat
                break
    #
    return sXleFeat

def GetXLEChar(sTagPOS, sTagCharValue):
    ''' Returns the XLE feature by giving the tagger pos and value '''
    #
    sXleFeat = ''
    for myFeature in lFeatures:
        if sTagPOS == myFeature.TagPos and sTagCharValue == myFeature.TagVal:
            sXleFeat = myFeature.XleFeat
            break
    #
    return sXleFeat


def GetXLEValue(sXlePOS, sXleFeat, sTagVal):
    ''' Returns the corresponding XLE characteristic value of sXleChar, sXleFeat,
sTagVal '''
    #
    sXLEValue = ''
    for myFeature in lFeatures:
        if sXlePOS == myFeature.XlePos and sXleFeat == myFeature.XleFeat and sTagVal
== myFeature.TagVal:
            sXLEValue = myFeature.XleVal
            bFound = True
            break
    #
    return sXLEValue


def IncludedInTokenTemplates(inToken, inItem):
    ''' Returns True if the inItem is included in any of the inToken templates.
        An inItem may be either a MatchedTemplate or a CharAssign.
    '''
    bIncluded = False
    #
    for myMatchTemp in inToken.MatchedTemplates:
        sTemplate = myMatchTemp.Template
        #
        for myItem in dIncludes[sTemplate]:
            #
            if isinstance(myItem, TemplateCall) and isinstance(inItem,
MatchedTemplate):
                #
                if myItem.Template == inItem.Template:
                    #
```

```
                    #
                    # * * * * * ti ginetai otan exei parametrous??? * * * * * *
                    #
                    bIncluded = True
                    break
            elif isinstance(myItem, CharAssign) and isinstance(inItem, CharAssign):
                #
                # * * * * * ti ginetai otan exei anisotita??? * * * * * *
                #
                myTemplate = dTemplates[sTemplate]
                if myItem.Char == inItem.Char:
                    if myItem.Operator == '=' and ((myItem.Value == inItem.Value) or
(myItem.Value in myTemplate.ParamList)):
                        bIncluded = True
                        break
            if bIncluded:
                break
    #
    return bIncluded


def CreateLexiconEntry(inToken):
    ''' Returns the sLexEntry of the inToken. The sLexEntry consists of
        - the word
        - the POS
        - a list of all templates the token has and are not included in other
templates
        - a list of all features which are not included in the previous templates
        - the extra info taken from the lemmas lexicon (if there is an entry)


    '''
    sLexEntry = ("\n" + inToken.Word + " * " + sXlePos + " " )
    for myMatchedTemplate in inToken.MatchedTemplates:
        # if the temlpate is NOT included in another of the inToken, then print it
(avoid redundancy)
        if not IncludedInTokenTemplates(inToken, myMatchedTemplate):
            sLexEntry += "\n \t \t @"
            iVars = len(myMatchedTemplate.Vars)
            if iVars > 0:
                sLexEntry += "(" + myMatchedTemplate.Template
                i = 0
                while i < iVars:
                    sLexEntry += " " + myMatchedTemplate.Vars[i]
                    i += 1
                sLexEntry += ")"
            else:
                sLexEntry += myMatchedTemplate.Template
    #
    # Adding token characteristics/features
    #
    sXlePOS = dXLETagPOSMap[inToken.POS]
    for sTagVal in inToken.Chars:
        if sTagVal != "Xx":
            sXleChar = GetXLEChar(inToken.POS, sTagVal)
            if sXleChar != "":
                sXleValue = GetXLEValue(sXlePOS, sXleChar, sTagVal)
                myCharAssign = CharAssign(sXleChar,'=', sXleValue)
                if sXleValue != "" and not IncludedInTokenTemplates(inToken,
myCharAssign ):
                    sLexEntry += "\n \t \t (^ " + sXleChar + ") = " + sXleValue
    #
    # Appending lex entries info
```

```
    #

    # Searching the lemma lexicon for the token's lemma. If match found copy the
information
    sLexKey = inToken.Lemma + "*" + dXLETagPOSMap.get(inToken.POS, '###')
    if sLexKey in dLexEntries.keys():
        sLexEntry += "\n \t \t " + dLexEntries[sLexKey].rstrip('.').rstrip('ETC')
    else:
        sLexEntry += "\n \t \t "
    return sLexEntry


def CheckTokenTemplates(inToken):
    ''' Checks all the templates of each token. If a template matches to the token's
data,
        then it is added to the token's MatchedTemplatesList
    '''
    #
    for sTemp in dTemplates.keys():
        myMatchedTemplate  = ListToEvaluationString(sTemp, inToken,
dTemplates[sTemp].AndList, {})
        try:
            #print("\n\n" + sTemp + ": " + myMatchedTemplate.String)
            if eval(myMatchedTemplate.String) == 1:
                inToken.MatchedTemplates.append(myMatchedTemplate)
                #logFile.write("\n Matched: " + myMatchedTemplate.Template)
        except:
            #logFile.write("Error at template evaluation: " + sTemp + " for token " +
inToken.Id)
            pass


def ListToEvaluationString(inTemp, inToken, inList, inVars):
    ''' Returns an evaluation string of inList, which is a boolean expression,
something like (1 and 0 and (1 or 0)))
        where each 0 or 1 derives by a characteristic assignment (1 if there is a
match with the value of inToken,
        0 if not)
    '''
    #print("\n\n* * * ListToEvaluationString", inTemp)
    #print("\n", inToken.Word, inList[0], str(inVars))
    sReturn = ''
    myMatchedTemplate = MatchedTemplate(inTemp, sReturn, inVars)
    #
    if isinstance(inList, list):
        if isinstance(inList, AndList):
            sBool = " and "
        elif isinstance(inList, OrList):
            sBool = " or "
        #
        i = 0
        sReturn = ""
        for myItem in inList:
            i += 1
            if i > 1:
                sReturn += sBool
            elif isinstance(inList, Negation):
                sReturn = " not "
            #
            if isinstance(myItem, list):
                myMatchedTemplate = ListToEvaluationString(inTemp, inToken, myItem,
inVars)
                sReturn += " ( " + myMatchedTemplate.String + " ) "
```

```
                inVars = myMatchedTemplate.Vars
            elif isinstance(myItem, CharAssign):
                if myItem.Char != "PRED":
                    bCharMatched = False
                    iIndex = -1
                    for myTagCharValue in inToken.Chars:
                        iIndex += 1
                        #
                        if myTagCharValue != "Xx":
                            sXleChar = GetXLEChar(inToken.POS, myTagCharValue)
                            if myItem.Char == sXleChar:
                                #
                                #logFile.write(inTemp + " " + inToken.Word
myTagCharValue, myItem.Char, myItem.Value, dTemplates[inTemp].ParamList)
                                #
                                bCharMatched = True
                                sXlePOS = dXLETagPOSMap.get(inToken.POS, '###')
                                # if the item is not an existential constraint
                                if myItem.Operator:
                                    sXleCharValue = GetXLEValue(sXlePOS, sXleChar,
myTagCharValue)
                                    if ((myItem.Operator == "=" and myItem.Value ==
sXleCharValue) or
                                        (myItem.Operator == "~=" and myItem.Value !=
sXleCharValue)):
                                        sReturn += ' 1 '
                                        #
                                    elif myItem.Value in dTemplates[inTemp].ParamList:
                                        sReturn += ' 1 '

inVars[dTemplates[inTemp].ParamList.index(myItem.Value)] = sXleCharValue
                                    else:
                                        sReturn += ' 0 '
                                    break
                                else:
                                    sReturn += ' 1 '
                                    break
                else:
                    sReturn += ' 1 '
                    inVars[dTemplates[inTemp].ParamList.index(myItem.Value)] =
inToken.Lemma
                    bCharMatched = True
                #
                if not bCharMatched:
                    sReturn += ' 0 '
            else:
                sReturn += str(myItem)
        #
        if isinstance(inList, Conjunction) or isinstance(inList, OrList):
            sReturn = '(' + sReturn + ')'
    else:
        print("Not a list:", str(inList))
        sReturn = str(inList)
    #
    myMatchedTemplate.String = sReturn
    myMatchedTemplate.Vars = inVars
    return myMatchedTemplate
```

```
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
# *
# *                                M A I N   P R O G R A M
# *
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


#
# SECTION 4: Loading tagger-xle features
#

class FeatureMap:
    def __init__(self, TagPos, TagFeat, TagVal, XlePos, XleFeat, XleVal):
        '''Initializes the data.'''
        self.TagPos = TagPos
        self.TagFeat = TagFeat
        self.TagVal = TagVal
        self.XlePos = XlePos
        self.XleFeat = XleFeat
        self.XleVal = XleVal

print("Loading tagger-xle features from file:", sTagSetFile)
lFeatures = list()
dXLETagPOSMap = dict()

featFile = open(sTagSetFile)
while True:
    sLine = featFile.readline()
    if len(sLine) != 0:
        if not sLine.startswith('#'):
            sLine = sLine.rstrip('\n')
            sLine = re.sub('\t', ' ', sLine)
            lLine = sLine.split(',')
            if len(lLine) > 2:
                sTagPos = lLine[0].strip()
                sTagFeature = lLine[1].strip()
                sTagValue = lLine[2].strip()
                if len(lLine) > 3:
                    sXlePos = lLine[3].strip()
                else:
                    sXlePos = ''
                if len(lLine) > 4:
                    sXleFeature = lLine[4].strip()
                else:
                    sXleFeature = ''
                if len(lLine) > 5:
                    sXleValue = lLine[5].strip()
                else:
                    sXleValue = ''
                #
                dXLETagPOSMap[sTagPos] = sXlePos
                lFeatures.append(FeatureMap(sTagPos, sTagFeature, sTagValue, sXlePos,
sXleFeature, sXleValue))

    else:
        break

#
# SECTION 5: Loading templates from the XLE grammar file to the dTemplates dictionary
#

print("Loading templates from file:", sXleGrammarFile)
dTemplates = dict()
```

```
sTemplate = ""
bWholeTemplate = False
bTemplateSectionFound = False
bNotEndOfTemplates = True
grammarFile = open(sXleGrammarFile,  'rt', 1, 'utf-8')
# Find the templates section
while (not bTemplateSectionFound):
    grammarLine = grammarFile.readline()
    if grammarLine.find(sTemplateSection) >= 0:
        bTemplateSectionFound = True
    elif len(grammarLine) == 0:
        break
#
if not bTemplateSectionFound:
    print("No template section in xle grammar file!")
else:
    while bNotEndOfTemplates:
        grammarLine = grammarFile.readline()
        if len(grammarLine) != 0:
            if grammarLine.find("----") < 0:
                sTemplate += grammarLine
                #
                if grammarLine.find(".") >= 0:
                    bWholeTemplate = True
                #
                if bWholeTemplate:
                    # Cleaning paragraphs(\t), new liles (\n), leading and trailing
spaces and comments ("...")
                    sTemplate = re.sub("\t", " ", sTemplate)
                    sTemplate = re.sub("\n", " ", sTemplate)
                    sTemplate = sTemplate.strip()
                    sTemplate = re.sub('".*?"', ' ',sTemplate)
                    # Split the template to head and body
                    listTemplate = sTemplate.partition("=")
                    sTemplateHead = listTemplate[0].strip()
                    # Get the template name
                    iParenthesis = sTemplateHead.find("(")
                    if  iParenthesis >= 0:
                        sTemplateName = sTemplateHead[0:iParenthesis]
                    else:
                        sTemplateName = sTemplateHead
                    #
                    if sTemplateName in dTemplates.keys():
                        print("WARNING! Double template definition:", sTemplateName)
                    #
                    dTemplates[sTemplateName] = Template(sTemplate, [], False, [], [])
                    #print("'" + sTemplateName + "'", sTemplate)
                    #
                    bWholeTemplate = False
                    sTemplate = ""
            else:
                bNotEndOfTemplates = False
        else:
            break
#
# SECTION 6: Loading lemma lexicon to dLexEntries dictionary (key is Lemma + "*" +
POS)
#            These lemmas carry syntax or other info which the tagger does not give
#            but is essential to the user so it is copied to the lexicon.
#

bLemmaSectionFound = False
if sLemmaSection != "":
```

```
    print("Loading extra info from lemma lexicon:", sLemmaSection)
    while (not bLemmaSectionFound):
        grammarLine = grammarFile.readline()
        if grammarLine.find(sLemmaSection) >= 0:
            bLemmaSectionFound = True
        elif len(grammarLine) == 0:
            break
else:
    print("No lemma lexicon defined!")
#
dLexEntries = dict()
bNotEndOfLemmaLexicon = True
bWholeEntry = False
sLexEntry = ""
if bLemmaSectionFound:
    while bNotEndOfLemmaLexicon:
        sLine = grammarFile.readline()
        if len(sLine) != 0:
            if sLine.find("----") < 0:
                sLexEntry += sLine
                # The lexicon entry ends with a dot (.)
                if sLine.find(".") >= 0:
                    bWholeEntry = True
                #
                if bWholeEntry:
                    # Split the lexicon entry to head and body
                    listEntry = sLexEntry.partition("*")
                    sEntryHead = listEntry[0].strip()
                    sEntryBody = listEntry[2].strip()
                    # Split the head to lemma and POS
                    listHead = sEntryHead.split()
                    sLemma = listHead[0].strip()
                    sPOS = listHead[1].strip()
                    #
                    dLexEntries[sLemma + "*" + sPOS] = sEntryBody
                    #print(sLemma + "*" + sPOS, sEntryBody)
                    #
                    bWholeEntry = False
                    sLexEntry = ""
            else:
                bNotEndOfLemmaLexicon = False
        else:
            break
#
# SECTION 7: Template prossecing
#
print("Parsing the templates")

#
# S7.1 Building the lexer and the parser
#
import ply.lex as lex
lex.lex()

import ply.yacc as yacc
yacc.yacc()

#
# S7.2 Parsing the templates and building the ASTs
#
bError = False
dIncludes = dict()
for sTempName in dTemplates.keys():
```

```
    #print("* Parsing: " + sTempName + "\n")
    sTempString = dTemplates[sTempName].String
    lIncItems = list()
    #myTemplate = yacc.parse(sTempString)
    try:
        myTemplate = yacc.parse(sTempString)
    except:
        #print("Parsing error in template", sTempName)
        bError = True
    else:
        dTemplates[sTempName] = myTemplate
        dIncludes[sTempName] = copy.deepcopy(lIncItems)
#
#if bError: sys.exit(0)
#
# S7.3 Fixing Template Calls
#
lRecursion = list()
print("Fixing template calls")
logFile.write("Fixing Template Calls \n\n")
for sTemp in dTemplates.keys():
    if not dTemplates[sTemp].Fixed:
        dTemplates[sTemp].AndList = FixTemplateCallsOf(sTemp,
dTemplates[sTemp].AndList)
        dTemplates[sTemp].Fixed = True

logFile.write("Printing Templates \n\n")
for sTemp in dTemplates.keys():
    PrintTemplate(sTemp)

#
# SECTION 8: Loading tokens from tagger output file to the lTagTokens list
#

# Token class

class Token:
    def __init__(self, Id, Word, POS, Chars, Lemma, MatchedTemplates, LexEntry):
        '''Initializes the data.'''
        self.Id = Id
        self.Word = Word
        self.POS = POS
        self.Chars = list(Chars)
        self.Lemma = Lemma
        self.MatchedTemplates = list(MatchedTemplates)
        self.LexEntry = LexEntry

# Token matched template class

class MatchedTemplate:
    def __init__(self, Template, String, Vars):
        '''Initializes the data.'''
        self.Template = Template
        self.String = String
        self.Vars = dict(Vars)


print("Loading tokens from file:", sTaggerFile)
ConvertGrToEng(sTaggerFile)
lTagTokens = list()
tree = ET.parse(sTaggerFile)
root = tree.getroot()
```

```
for token in root.iter('{http://www.xces.org/schema/2003}t'):
    #
    lChars = list()
    sId = token.get("id")
    sWord = token.get("word")
    sTag = token.get("tag")
    sLemma = token.get("lemma")
    #
    # Check if the token already exists (same Word and Tag).
    #
    bTokenExists = False

    for myToken in lTagTokens:
        myTag = myToken.POS
        for sCh in myToken.Chars: myTag += sCh
        if myToken.Word == sWord and sCh == sTag:
            bTokenExists = True
            break
    #
    # If it does not exist, add it
    #
    if not bTokenExists:
        if sTag[0:2] in dXLETagPOSMap.keys():
            sPos = sTag[0:2]
            #sTag = sTag[2:]
            for i in range(0, int(len(sTag)/2)):
                lChars.append(sTag[2*i:2*i+2])
        elif sTag in lTokenizerTags:
            sPos = sTag
            lChars = []
        else:
            #print(lTokenizerTags, sTag)
            sPos = '**'
            lChars.append(sTag)
        #
        lTagTokens.append(Token(sId, sWord, sPos, lChars, sLemma, [], ''))
#
# SECTION 9: Writing the lexicon entries to the output file
#
print("Creating lexicon:", sOutputFile)
fOutputFile = open(sOutputFile,  'wt', 1, 'utf-8')
fOutputFile.write(sLexiconSection + "\n\n")

lTokenTemplates = list()

lTemp = sorted(lTagTokens, key=attrgetter('Word'))
lTagTokens = sorted(lTemp, key=attrgetter('POS'))


iEntries = 0
for myToken in lTagTokens:
    sXlePos = dXLETagPOSMap.get(myToken.POS)
    # if the user has not defined an XLE pos then all the tokens which belong to this
pos do not appear in the output file
    if sXlePos:
        #
        CheckTokenTemplates(myToken)
        #
        sEntry = CreateLexiconEntry(myToken)
        #
        bDoubleEntry = False
        bDoubleWord = False
        # check all tokens
```

```
        for checkToken in lTagTokens:
            # if same word exists
            if checkToken.Word == myToken.Word and checkToken.Id != myToken.Id:
                # check if exactly the same lexicon entry already exists
                if checkToken.POS == myToken.POS and
checkToken.LexEntry.rstrip('.').rstrip('ETC') == sEntry:
                    bDoubleEntry = True
                    break
                else:
                    bDoubleWord = True
        if not bDoubleEntry:
            if not bDoubleWord:
                sEntry += '.'
            else:
                sEntry += 'ETC.'
            myToken.LexEntry = sEntry
            fOutputFile.write(sEntry)
            iEntries += 1
print(str(iEntries), "entries in lexicon\n")
s = raw_input("Press a key to terminate...")

fOutputFile.write("\n\n" + "----")
fOutputFile.close()
logFile.close()
```

Η: Πίνακας μετεγγραφής χαρακτήρων

Ελληνικός χαρακτήρας	Λατινικός χαρακτήρας
Α	A
Β	V
Γ	G
Δ	D
Ε	E
Ζ	Z
Η	H
Θ	TH
Ι	I
Κ	K
Λ	L
Μ	M
Ν	N
Ξ	X
Ο	O
Π	P
Ρ	R
Σ	S
Τα	T
Υ	U
Φ	F
Χ	CH
Ψ	PS
Ω	W