ΑΤΕΙ ΗΠΕΙΡΟΥ

ΤΜΗΜΑ ΤΗΛΕΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΔΙΟΙΚΗΣΗΣ

ΕΠΕΞΕΡΓΑΣΙΑ ΦΥΣΙΚΗΣ ΓΛΩΣΣΑΣ ΜΕ PROLOG

ΕΥΘΥΜΙΟΣ ΤΣΟΥΤΣΙΑΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:
ΙΩΑΝΝΗΣ ΤΣΟΥΛΟΣ

ΙΟΥΝΙΟΣ 2004

ΕΠΕΞΕΡΓΑΣΙΑ ΦΥΣΙΚΗΣ ΓΛΩΣΣΑΣ ΜΕ PROLOG

ΕΥΘΥΜΙΟΣ ΤΣΟΥΤΣΙΑΣ

20/6/2004

Αφιέρωση: Επιθυμώ να αφιερώσω το παρόν σύγγραμμα στον Ιησού Χριστό, χωρίς την βοήθεια του οποίου ποτέ δεν θα είχε γραφτεί. Επίσης στους γονείς μου, στους φίλους μου για την υποστήριξή τους και στον επιβλέποντα καθηγητή για τις χρήσιμες συμβουλές του σε όλη την διάρκεια του έργου. Ευχαριστώ

Περιεχόμενα

Α. Περιγραφή προβλήματος

1. Εισαγωγή

Η Prolog έχει χρησιμοποιηθεί για πάρα πολλούς σκοπούς, αλλά αυτός που την δημιούργησε, δηλαδή ο Alain Colmerauer, ασχολούνταν με γλωσσολογικούς υπολογισμούς, και οι γλωσσολογικοί υπολογισμοί παραμένουν μία κλασική εφαρμογή για την Prolog. Με τον όρο γλωσσολογικούς υπολογισμούς εννοούμε τρόπους και μεθόδους ανάλυσης των προτάσεων και των λέξεων σε κάποια γλώσσα. Παράδειγμα αποτελεί η συντακτική και λεκτική ανάλυση. Η Prolog παρέχει έναν αριθμό εργαλείων που διευκολύνουν σε σημαντικό βαθμό όσους εργάζονται πάνω σε αυτό το αντικείμενο.

Ένα από τα σημαντικότερα εργαλεία που παρέχει η Prolog είναι τα **Definite Clauses Grammars ή DCGs** για συντομία. Μία προσπάθεια απόδοσης του όρου στα ελληνικά είναι Συγκεκριμένες – Περιορισμένες Γραμματικές Προτάσεις. Δηλαδή κάποιες προτάσεις με περιορισμένη συνήθως έκταση και δομή, οι οποίες όμως είναι ορθές συντακτικά.

Τα DCGs αποτελούν μία ειδική μορφή για να ορίσουμε γραμματικές προτάσεις. Για να καταλάβουμε τι εννοούμε όταν λέμε γραμματική (grammar), πρέπει να καταλάβουμε τι είναι τα **Context Free Grammars** ή για συντομία **CFGs**. Μία ελληνική απόδοση του όρου είναι Γραμματική Ανεξάρτητη των Συμφραζομένων.

Η βασική ιδέα πίσω από τα CFGs είναι πολύ απλή στην κατανόηση. Το μόνο μειονέκτημά τους είναι ότι δεν μπορούν να συμβαδίσουν με την συντακτική δομή όλων των φυσικών γλωσσών (με την έννοια της φυσικής γλώσσας εννοούμε την γλώσσα που χρησιμοποιούν οι άνθρωποι). **Μπορούν όμως να χειριστούν αποδοτικά το συντακτικό από πολλές φυσικές γλώσσες (όπως είναι τα Αγγλικά, τα Γερμανικά και τα Γαλλικά).**

Το ερώτημα που προκύπτει τώρα είναι το εξής: Τι ακριβώς είναι ένα Context Free Grammar ή αλλιώς μία Γραμματική Ανεξάρτητη των

Συμφραζομένων; Στην ουσία του, το CFG αποτελεί ένα περιορισμένο σύνολο κανόνων (rules) οι οποίοι μας επιτρέπουν να διαπιστώσουμε εάν συγκεκριμένες προτάσεις είναι σωστά συντακτικά (πληρούν δηλαδή τους γραμματικούς κανόνες), καθώς επίσης μας επιτρέπουν να μάθουμε την γραμματική δομή των προτάσεων αυτών.

Απομένει να εξηγήσουμε ένα ακόμη συσχετιζόμενο θέμα, συγκεκριμένα τι είναι μία context free language (έχουμε δώσει τον ορισμό για τα context free grammars, αλλά όχι και για τα context free languages). Μία context free language είναι μία γλώσσα που μπορεί να γεννηθεί από ένα context free grammar. Μερικές γλώσσες είναι context free, και μερικές όχι. Για παράδειγμα είναι αληθές ότι τα Αγγλικά είναι μία context free language, επειδή είναι απλούστατα στην δομή τους και έχουν λιγότερους συντακτικούς κανόνες από ότι π.χ. τα αρχαία ελληνικά (αν και έχουν πολλές εξαιρέσεις). Αυτό σημαίνει ότι είναι εφικτά πιθανό να γράψει κάποιος ένα context free grammar που να γεννάει όλες τις προτάσεις που δέχονται οι Εγγλέζοι. Από την άλλη μεριά, κάποιες διάλεκτοι της Γερμανικής γλώσσας δεν είναι context free. Μπορεί να αποδειχτεί μαθηματικά ότι καμία context free grammar δεν μπορεί να δημιουργήσει όλες μα όλες τις προτάσεις που χρησιμοποιούν οι ιθαγενείς. Αν κάποιος δηλαδή επιθυμούσε να γράψει μία γραμματική για τέτοιες διαλέκτους θα έπρεπε να χρησιμοποιήσει επιπρόσθετους γραμματικούς μηχανισμούς, όχι απλά context free κανόνες. (Αναφορά: **Learn Prolog Now!** Patrick Blackburn, Johan Bos, Kristina Striegnitz)

2. Ανθρώπινη γλώσσα: Ένα σύστημα που αποτελείται από κανόνες

- Οι ανθρώπινες γλώσσες (φυσικές γλώσσες) είναι σύνθετα συστήματα, με ευφυείς (έξυπνους στην λογική τους) και λειτουργικούς κανόνες.
- Γνωρίζοντας τους κανόνες μπορούμε να επικοινωνούμε αποτελεσματικά και με ακρίβεια.
- Όσον αφορά τα γλωσσικά μοντέλα στους υπολογιστές, οι κανόνες πρέπει να είναι λεπτομερείς (γλωσσικά μοντέλα εννοούμε κάποια

μοντέλα που ορίζουν τους κανόνες από τους οποίους διέπεται μία γλώσσα, για παράδειγμα τους συντακτικούς κανόνες).

- Οι κανόνες ορίζουν το πώς οι «μονάδες» της γλώσσας, τα κομμάτια της δηλαδή, μπορούν να συνδυαστούν για να δημιουργήσουν ουσιώδεις και πιο σύνθετες μονάδες:
    - Μορφολογικοί κανόνες: ορίζουν το πώς συνθέτονται οι λέξεις π.χ. re+invest+ing = reinvesting
    - Συντακτικοί κανόνες: ορίζουν το πώς οι λέξεις μπορούν να συνδυαστούν σε προτάσεις
    - Εννοιολογικοί κανόνες: ορίζουν το πώς συνδυάζονται οι έννοιες

(Αναφορά: **Natural Language Processing Introduction** http://www.cee.hw.ac.uk/~diana/nl/l01sh Nikos Drakos)

3. Λεκτική Ανάλυση:

3.1 Μορφολογία: Η Δομή των λέξεων

- Η μορφολογία αφορά το πώς οι λέξεις δομούνται από πιο βασικά μέρη:
    - friend + ly = friendly
    - dog + s = s
    - jump + ed = jumped
    - re + introduce = reintroduce
    - de + central + ise + ation = decentralisation
- Οι βασικές μονάδες λέγονται morphemes: friend, ly, dog, s και re είναι όλα αναπαραστάσεις (morphemes).

Οι λέξεις συνθέτονται από μία ρίζα και κάποια προθέματα (όταν μιλάμε για πρόσθεση μπροστά από την λέξη) και κάποια επιθέματα (όταν μιλάμε για πρόσθεση στο τέλος της λέξης).

3.2 Κατηγορίες Λέξεων

Οι λέξεις χωρίζονται σε κατηγορίες ανάλογα με την χρήση τους και με το πώς συνδυάζονται με άλλες λέξεις εντός των προτάσεων:

Οι βασικές κατηγορίες είναι:

- Ουσιαστικά: αναφέρονται σε αντικείμενα ή έννοιες π.χ. cat, beauty, John
- Επίθετα: περιγράφουν ή δίνουν ιδιότητες στα ουσιαστικά π.χ. the *big* man
- Ρήματα: περιγράφουν τι κάνει το ουσιαστικό: John *jumps*
- Επιρρήματα: περιγράφουν το πώς γίνεται: john runs *quickly*

Οι λέξεις που ανήκουν σε αυτές τις κατηγορίες μερικές φορές αναφέρονται ως open class words, γιατί συχνά προστίθενται καινούργιες λέξεις αυτών των κατηγοριών στην γλώσσα.

Οι άλλες κατηγορίες λέξεων είναι «κλειστές»: υπάρχει ένα μικρό προκαθορισμένο όριο λέξεων σε αυτές τις κατηγορίες:

- Άρθρα: the, a, an.
- Αντωνυμίες: it, he etc.
- Προθέσεις: by, on, with.
- Συνδετικά: and, or.

(Αναφορά: 3.1&3.2 **Natural Language Processing Introduction** http://www.cee.hw.ac.uk/~diana/nl/l01sh Nikos Drakos)

4. Συντακτική Ανάλυση:

4.1 Φράσεις:

Μπορούν να χρησιμοποιηθούν μεγαλύτερες φράσεις από μία λέξη:

- Φράσεις με ουσιαστικά (Noun phrases). Αναφέρονται σε αντικείμενα: “Ο άνδρας με το καπέλο”.
- Προτάσεις με ρήματα (Verb phrases). Δηλώνουν τι κάνει η φράση με ουσιαστικά π.χ. “ Ο άνδρας με το καπέλο μιλάει στο σκυλάκι”
- Προτάσεις με επίθετα (Adjective phrases). Περιγράφουν – δίνουν ιδιότητα σε ένα αντικείμενο π.χ. «φτωχός σαν ένας άστεγος»
- Προτάσεις με επιρρήματα (Adverbial phrases). Περιγράφουν τον τρόπο με τον οποίο γίνεται π.χ. “πολύ προσεκτικά”

Η απλούστερη φράση με ουσιαστικά αποτελείται από:

- Πρόθεση
- Κατάλληλο όνομα

Πιο σύνθετες φράσεις με ουσιαστικά χρησιμοποιούν κοινά ουσιαστικά, τα οποία διακρίνονται σε:

- Μετρήσιμα ουσιαστικά π.χ. υπολογιστές, γάτες
- Μη μετρήσιμα ουσιαστικά π.χ. ζάχαρη

Επίσης υπάρχουν φράσεις που αναφέρονται σε ιδιοκτησία, αναφορικές προτάσεις κ.α.

4.2 Φράσεις με ρήματα

Σύνθετες φράσεις μπορούν να περιγράφουν δράση:

- He *talks to Mary*.
- He *eats the pizza*.
- He *quickly walks*.
- He *gives a very nice present to his mother*,

Όλες αυτές οι φράσεις είναι ρηματικές.

4.3 Προτάσεις

Με ποιόν τρόπο οι ρηματικές και οι ουσιαστικές φράσεις συνδυάζονται και προκύπτουν ολόκληρες προτάσεις;
Εξαρτάται από τον τύπο της πρότασης:

- Ισχυρισμός: John looked at Mary
- Ερώτηση Ναί / Όχι: Did he do it?
- Ερώτηση Υποκειμένου: Who did John look at?
- Εντολή - Προστακτική: Look at Mary.

4.4 Προτάσεις: Ισχυρισμοί

Ένας απλός ισχυρισμός αποτελείται από μία ουσιαστική φράση συν μία ρηματική φράση:

- John jumps.
- The tall man eats a large tasty pizza.
- The man eating the pizza jumps.
- He likes Mary.
- He eats the pizza with the extra hot chilli topping.

Αυτός που κάνει την δράση ονομάζεται υποκείμενο, το πράγμα στο οποίο γίνεται είναι το αντικείμενο.

Το συμπέρασμα στο οποίο μπορούμε να φτάσουμε από όσα έχουμε πει μέχρι στιγμής είναι το εξής:

- Η γλώσσα είναι ένα σύστημα με σύνθετους κανόνες
- Η προτάσεις συνθέτονται από λέξεις, με σύνθετους κανόνες που ορίζουν το πώς συνδέονται οι λέξεις

- Χωρίς τους κανόνες δεν θα υπήρχε δομή: δεν θα μπορούσαμε να κατανοήσουμε τι σημαίνει μία πρόταση, όπως και δεν θα μπορούσαμε να γνωρίζουμε το πώς οι λέξεις συνδέονται μεταξύ τους

- Γνωρίζουμε τους κανόνες υποσυνείδητα, δηλαδή τους χρησιμοποιούμε στην καθημερινή μας ομιλία χωρίς πραγματικά να τους σκεφτόμαστε ενεργά, τους έχουμε εμπεδώσει. Ένα υπολογιστικό μοντέλο πρέπει να τους κάνει επακριβείς

(Αναφορά:4.1,4.2,4.3,4.4 **Natural Language Processing Introduction** http://www.cee.hw.ac.uk/~diana/nl/l01sh Nikos Drakos)

5. Επίπεδα της Γλωσσολογικής Ανάλυσης

Η γλωσσολογική ανάλυση μπορεί να χωριστεί στα ακόλουθα επίπεδα:

- Φωνητικά: (ήχοι→λέξεις: /b/ + /o/ + /t/ = boat).
- Μορφολογικά: (μορφήματα→λέξεις: friend + ly=friendly)
- Συντακτικά: ακολουθία λέξεων → δομή πρότασης
- Εννοιολογικά: δομή πρότασης + σημασία λέξεων = νόημα πρότασης
- Κυριολεκτικά: νόημα πρότασης + συμφραζόμενα = καλύτερο νόημα
- Κατανόηση ομιλίας: χρησιμοποιώντας γενική γνώση και γνώση προηγούμενων ομιλιών

(Αναφορά: **Natural Language Processing Introduction** http://www.cee.hw.ac.uk/~diana/nl/l01sh Nikos Drakos)

6. Εφαρμογές της ανάλυσης φυσικής γλώσσας

Η ανάλυση φυσικής γλώσσας έχει ποικίλες σημαντικές εφαρμογές, οι οποίες μπορούν να διακριθούν σε δύο κυρίως κατηγορίες. Οι κατηγορίες αυτές είναι οι εξής:

- Επεξεργασία κειμένου
- Επικοινωνία ανθρώπου – μηχανής

1. Αναφορικά με την επεξεργασία του δοθέντος στην μηχανή κειμένου (written text)
    - Υποστηρίζει και βοηθά τον άνθρωπο που συγγράφει το κείμενο: συγκεκριμένα διεξάγοντας έλεγχο γραμματικής και συγγραφικού ύφους – τύπου.
    - Εξάγει πληροφορίες από αναφορές: συγκεκριμένα παράγει με αυτόματο τρόπο μία περίληψη
    - Κατηγοριοποιεί, ανακτά ή φιλτράρει μηνύματα ή έγγραφα: π.χ. δρομολόγηση εγγράφων ηλεκτρονικού ταχυδρομείου, μηχανές αναζήτησης στον παγκόσμιο ιστό (World Wide Web) κτλ.
    - Μεταφράζει έγγραφα.
    - Μπορεί να καταγράφει μία συζήτηση
    - Δημιουργεί αναφορές κειμένου παίρνοντας πληροφορίες από βάσεις δεδομένων

2. Αναφορικά με την επικοινωνία ανθρώπου – μηχανής

    - Περιβάλλοντα διεπαφής (με βάσεις δεδομένων κτλ.).
    - Αυτόματη εξυπηρέτηση πελατών μέσω τηλεφώνου (π.χ. κρατήσεις).
    - Δυνατότητα φωνητικού ελέγχου μηχανής (Voice Control)
    - Συστήματα διδασκαλίας και εκπαίδευσης.

Τα συστήματα θα πρέπει να είναι σε θέση να ανταποκριθούν και με χρήση της καθημερινής διαλέκτου (η οποία μπορεί να είναι διαφορετική από την κοινά επισήμως αποδεκτή γλώσσα), καθώς επίσης και να μπορούν να συνδιαλεχθούν με τον χρήστη.
Βέβαια, προκειμένου να μπορέσει να υλοποιηθεί ένα σύστημα που να μπορεί να κάνει ανάλυση της φυσικής γλώσσας, απαραίτητο είναι να περάσει κάποιο χρονικό διάστημα, έτσι ώστε να υπάρξει σαφής και σωστός σχεδιασμός. Αν

δεν υπάρξει σαφής και σωστός σχεδιασμός, τότε το στάδιο της υλοποίησης θα περάσει από πολλά στάδια, καθιστώντας πιθανό το γεγονός ότι θα πρέπει να βελτιώνεται και να αλλάζει συνεχώς ο σχεδιασμός, εφόσον δεν είχαν καθοριστεί και προβλεφτεί κατάλληλα από την αρχή οι απαραίτητες ανάγκες-προϋποθέσεις. Ένας σωστός σχεδιασμός λοιπόν, θα μας γλιτώσει από τον επικίνδυνο βρόγχο στον οποίο μπορεί να παγιδευτεί το σύστημα.

Ένα σύστημα επεξεργασίας φυσικής γλώσσας, όπως αναφέρθηκε μπορεί να χρησιμοποιηθεί σε πολλές περιπτώσεις. Για παράδειγμα μπορούμε να έχουμε ένα σύστημα φωνητικής αναγνώρισης, το οποίο να το έχουμε προγραμματίσει να κάνει διάφορα πράγματα στο άκουσμα συγκεκριμένων λέξεων. Μπορούμε να το προγραμματίσουμε π.χ. να καταγράφει την ομιλία μας σε έναν επεξεργαστή κειμένου, να μεταφράζει τα λεγόμενά μας στην γλώσσα που επιλέγουμε, να αλληλεπιδρά φωνητικά με το API του λειτουργικού συστήματος ή του τρέχοντος προγράμματος και να κάνει λειτουργίες όπως άνοιγμα - κλείσιμο παραθύρων, επιλογή στοιχείων οπτικού ελέγχου κ.ο.κ.

Επίσης, συστήματα επεξεργασίας φυσικής γλώσσας έχουν χρησιμοποιηθεί στο παρελθόν – μερικά από αυτά χρησιμοποιούνται ακόμη – σε πανεπιστήμια, ιδιωτικά ερευνητικά κέντρα και φροντιστήρια για την εκμάθηση μιας ή περισσοτέρων ξένης γλώσσας.

Σε ένα τέτοιο σύστημα παραδείγματος χάριν, οι μαθητές πληκτρολογούν στην κονσόλα κείμενο με προτάσεις και το σύστημα τους πληροφορεί εάν έχουν εισάγει την πρόταση σωστά ή τους καθοδηγεί σε περίπτωση που έχουν εισάγει μία πρόταση που δεν στέκει συντακτικά – γραμματικά. Μία προσφιλής εφαρμογή αυτού του τύπου – απλή βέβαια στην μορφή της – είναι ένα παιχνίδι στο οποίο θα εισάγονται λέξεις με την μορφή απλών προτάσεων και το παιχνίδι θα ανταποκρίνεται σε αυτές, ανάλογα με το περιεχόμενό τους. Είναι κάτι που θα κρατάει το ενδιαφέρον των παιδιών αμείωτο, και μπορεί άνετα να χρησιμοποιηθεί σε ένα φροντιστήριο π.χ. για την εκμάθηση μιας ξένης γλώσσας. Ένα τέτοιο παιχνίδι αποτελεί τον καρπό της παρούσης πτυχιακής εργασίας.

Οι τεχνικές επεξεργασίας φυσικής γλώσσας μπορούν να αφορούν τους εξής:

- Γλωσσολόγους – οι οποίοι θέλουν να κατανοήσουν την γλώσσα βαθύτερα

- Ψυχολόγους – οι οποίοι θέλουν να κατανοήσουν τις διαδικασίες της αντίληψης
- Φιλόσοφους – οι οποίοι ερευνούν το πώς οι λέξεις έχουν κάποια εννοιολογική σημασία, και το πώς η νοήμων σκέψη συσχετίζεται με την γλώσσα
- Επαγγελματίες Τεχνητής Νοημοσύνης – οι οποίοι θέλουν να αναπτύξουν μοντέλα ανθρώπινης λογικής, στα οποία συμπεριλαμβάνεται η επεξεργασία φυσικής γλώσσας
- Επιστήμονες της Πληροφορικής – οι οποίοι θέλουν να αναπτύξουν καλύτερες εφαρμογές οι οποίες περιλαμβάνουν την επεξεργασία της φυσικής γλώσσας

Ο λόγος για τον οποίο τα μοντέλα επεξεργασίας φυσικής γλώσσας είναι χρήσιμα, μαζί με όσα αναφέρθηκαν πιο πάνω είναι:

- Να επιτευχθεί η δημιουργία προγραμμάτων Η/Υ τα οποία θα είναι σε θέση να επιτελέσουν διάφορες λειτουργίες που περιλαμβάνουν επεξεργασία γλώσσας π.χ.
    - Ελεγκτές γραμματικής και συλλαβισμού
    - Καλύτερα συστήματα ανάκτησης πληροφορίας
    - Συστήματα που κατανοούν την ομιλία
    - Περιβάλλοντα Φυσικής γλώσσας
- Να μπορέσει να κατανοηθεί καλύτερα η ανθρώπινη επικοινωνία

(Αναφορά: **Natural Language Processing Introduction** http://www.cee.hw.ac.uk/~diana/nl/l01sh Nikos Drakos)

Β. Η γλώσσα Prolog

Η Prolog όπως είπαμε μπορεί να χρησιμοποιηθεί για ανάλυση φυσικής γλώσσας, κάτι το οποίο εκπληρώνει επιτυχώς. Πριν όμως δούμε κάποια παραδείγματα με κώδικα, πρέπει να εντρυφήσουμε στην σύνταξη της Prolog.

1. Η γλώσσα PROLOG:

Η γλώσσα προγραμματισμού PROLOG (PROgramming in LOGic) ακολουθεί μεθόδους της λογικής για την αναπαράσταση γνώσης και την επίλυση προβλημάτων, χρησιμοποιώντας για τον προγραμματισμό την κατηγορηματική λογική.
Ειδικότερα, ο προτασιακός λογισμός και ειδικότερα ένα υποσύνολό του οι προτάσεις Horn, αποτελούν τη βάση ανάπτυξης της PROLOG. Δημιουργός της είναι ο Alain Colmerauer, που με την βοήθεια των Jean Trudel και Philippe Roussel, δημιουργήσανε την Prolog, η οποία στην αρχή ήταν φτιαγμένη σε W-Algo.
Η ανάπτυξη της PROLOG ξεκίνησε στην Ευρώπη. Είναι αρκετά διαδεδομένη στην Ευρώπη και την Ιαπωνία, σε αντίθεση με τις ΗΠΑ, όπου αναπτύχθηκε κυρίως η LISP (LISt Processing).

2. Στοιχεία PROLOG:

Ο προγραμματισμός σε PROLOG διαφέρει από αυτόν στις διαδικασιακές γλώσσες προγραμματισμού. Ο προγραμματιστής δεν περιγράφει κάποιο αλγόριθμο με την κλασσική έννοια, δηλαδή μια σειρά από διαδοχικά βήματα που επιλύουν κάποιο πρόβλημα. Περιγράφει μόνο το ίδιο το πρόβλημα και τις

σχέσεις που ισχύουν μεταξύ των αντικειμένων του προβλήματος. Στη συνέχεια, αφήνεται στον υπολογιστή να διερευνήσει τη λύση και να διαπιστώσει το αληθές των σχέσεων αυτών.
Στη γενική μορφή ο προγραμματισμός σε PROLOG περιλαμβάνει τα εξής στάδια:

- Διατύπωση κάποιων γεγονότων που αφορούν τα αντικείμενα και τις σχέσεις τους.
- Ορισμό κάποιων κανόνων που διέπουν τα αντικείμενα και τις σχέσεις τους.
- Διατύπωση ερωτήσεων σχετικά με τα αντικείμενα και τις σχέσεις τους.

Αφού ορισθούν τα γεγονότα, οι κανόνες και οι σχέσεις που διέπουν τα αντικείμενα του προβλήματος, στη συνέχεια υποβάλλονται ερωτήσεις σε σχέση με τα παραπάνω. Οι απαντήσεις στις ερωτήσεις αυτές (καταφατικές ή αρνητικές) αποτελούν τη λύση του προβλήματος.
Μια συλλογή από γεγονότα και κανόνες καλείται μια **βάση γνώσεων** (ή μια βάση δεδομένων) και ο προγραμματισμός σε Prolog είναι η δημιουργία και το γράψιμο των βάσεων γνώσης. Δηλαδή τα προγράμματα Prolog *είναι* απλά βάσεις γνώσης, συλλογές γεγονότων και κανόνες που περιγράφουν κάποιες σχέσεις που βρίσκουμε ενδιαφέρον. Η απάντηση στο ερώτημα του πώς *χρησιμοποιούμε* ένα πρόγραμμα Prolog είναι η εξής:

Με την υποβολή των ερωτήσεων.

Δηλαδή απευθύνοντας ερωτήσεις για τις πληροφορίες που αποθηκεύονται στη βάση γνώσης. Δεν είναι βεβαίως προφανές ότι συνδέεται άρρηκτα με τον προγραμματισμό, αλλά όπως θα δούμε, ο προγραμματισμός με Prolog είναι πολύ βαθύς, ειδικότερα για ορισμένα είδη εφαρμογών (π.χ. επεξεργασία φυσικής γλώσσας που είναι ένα από πιο τρανταχτά παραδείγματα). Ακολουθούν κάποιες βάσεις γνώσεις, γιατί έτσι μόνο θα αποσαφηνιστεί η έννοια των τριών δομικών στοιχείων της Prolog, δηλαδή των γεγονότων, των κανόνων και των ερωτημάτων.

3. Σύνταξη της Prolog:

Ακριβώς από τι φτιάχνονται τα γεγονότα, οι κανόνες και οι ερωτήσεις;

Η απάντηση είναι όροι, και υπάρχουν τέσσερα είδη όρων σε Prolog:

- άτομα
- αριθμοί
- μεταβλητές
- σύνθετοι όροι (ή δομές).

Τα άτομα και οι αριθμοί αποτελούν τις σταθερές της Prolog, και οι σταθερές μαζί με τις μεταβλητές αποτελούν τους απλούς όρους της Prolog.
Για να γίνουν τα πράγματα ξεκάθαρα πρέπει πρώτα να μιλήσουμε για τους χαρακτήρες (σύμβολα). Οι *κεφαλαίοι χαρακτήρες* είναι οι A, B, ..., Z, *οι πεζοί χαρακτήρες* είναι a, b, ..., z, τα *ψηφία* είναι 1, 2..., 9 και οι *ειδικοί χαρακτήρες* είναι +, -, *, /, <, >, =, : ., &, ~, και _. Το κενό *διάστημα* είναι επίσης χαρακτήρας, αλλά για να χρησιμοποιηθεί απαιτείται να βρίσκεται μέσα σε μονά εισαγωγικά (‘ ‘). Μια σειρά είναι μία συνεχής ακολουθία χαρακτήρων.

3.1 Άτομα:

Ένα άτομο μπορεί να είναι:

1. Μια σειρά χαρακτήρων φτιαγμένων από κεφαλαία γράμματα, πεζά γράμματα, ψηφία, και τον χαρακτήρα κάτω παύλας _, η οποία αρχίζει με ένα πεζό γράμμα. Παραδείγματος χάριν: butch, big_kahuna_burger, και m_monroe2.

2. Μια αυθαίρετη ακολουθία χαρακτήρων που εσωκλείεται στα μονά εισαγωγικά (‘ και ‘). Παραδείγματος χάριν ’Vincent’, ’The Gimp’, ’Five_Dollar_Shake’, ’&^%&#@$ &*’, και ’ ’. Οι χαρακτήρες μεταξύ των μονών εισαγωγικών είναι το όνομα των ατόμων. Σημειώστε ότι έχουμε την άδεια να χρησιμοποιήσουμε τα διαστήματα (πλήκτρο spacebar) σε τέτοια άτομα — στην πραγματικότητα, ένας κοινός λόγος για τα μονά εισαγωγικά είναι ότι με την χρήση τους μπορούμε να κάνουμε ακριβώς αυτό.

3. Μια σειρά ειδικών χαρακτήρων. Παραδείγματος χάριν: @= και ====> και ; και :- είναι όλα άτομα. Όπως έχουμε δει, μερικά από αυτά τα άτομα, όπως το ; και το :- έχουν μία προκαθορισμένη έννοια.

3.2 Αριθμοί:

Η Prolog αντιμετωπίζει τους αριθμούς το ίδιο με τους υπόλοιπους όρους. Οι περισσότερες εφαρμογές Prolog υποστηρίζουν τους ακέραιους αριθμούς και τους αριθμούς κινητής υποδιαστολής (δηλαδή αριθμούς όπως 1657.3087 ή το **π).**

Οι ακέραιοι αριθμοί (π.χ.:... -2, -1, 0 ..1 ..2 ..3…) είναι χρήσιμοι για διάφορους στόχους όπως π.χ. μετρώντας τα στοιχεία ενός καταλόγου.

Η σύνταξη Prolog είναι η προφανής: 23, 1001, 0, -365, και τα λοιπά.

3.3 Μεταβλητές:

Μια μεταβλητή είναι μια σειρά χαρακτήρων φτιαγμένων από κεφαλαία γράμματα, πεζά γράμματα, ψηφία, και τον χαρακτήρα κάτω παύλας _, που αρχίζει *είτε* με ένα κεφαλαίο γράμμα *είτε* με την κάτω παύλα. Παραδείγματος χάριν, X, Y, Variable, _tag, X_526, και List, List24, _head, Tail, _input και Outputείναι όλες μεταβλητές της Prolog. Η μεταβλητή _ (δηλαδή ο χαρακτήρας κάτω παύλας) είναι ειδικός. Καλείται *ανώνυμη μεταβλητή*.

Την χρησιμοποιούμε όταν έχουμε ανάγκη να χρησιμοποιήσουμε μια μεταβλητή, αλλά δεν ενδιαφερόμαστε με τι την αντιστοιχεί η Prolog. Επιπλέον,

κάθε ανώνυμη μεταβλητή είναι *ανεξάρτητη*: κάθε μια είναι συνδεδεμένη σε κάτι διαφορετικό, κάτι που δεν συμβαίνει με τις άλλες μεταβλητές.

3.4 Σύνθετοι Όροι:

Οι σταθερές, οι αριθμοί, και οι μεταβλητές είναι οι δομικές μονάδες: τώρα πρέπει να ξέρουμε πώς ενώνονται για να σχηματίσουν μαζί τους σύνθετους όρους. Οι σύνθετοι όροι καλούνται συχνά δομές.
Οι σύνθετοι όροι αποτελούνται από ένα functor που ακολουθείται από μια ακολουθία ορισμάτων. Τα ορίσματα τίθενται ανάμεσα στις συνηθισμένες παρενθέσεις, χωρίζονται από τα κόμματα, και τοποθετούνται μετά από ένα functor. Το functor *πρέπει* να είναι ένα άτομο. Αφ' ετέρου, τα ορίσματα μπορεί να είναι οποιοδήποτε είδος όρου.
Ένα παράδειγμα σύνθετου όρου είναι το εξής:

```
welcomes(johny,kate).
welcomes(johny,kate,anny).
```

Πρέπει να σημειώσουμε ότι ο αριθμός των ορισμάτων δείχνει την τάξη του σύνθετου όρου. Η τάξη είναι πολύ σημαντική στην Prolog. Δύο σύνθετοι όροι με ίδιο functor αλλά διαφορετική τάξη ή διαφορετικά ορίσματα, αντιμετωπίζονται από την Prolog ως διαφορετικοί.

4. Γεγονότα:

Η ακόλουθη βάση γνώσης αποτελεί μία συλλογή γεγονότων. **Τα γεγονότα χρησιμοποιούνται για να δηλώσουν τα πράγματα που ισχύουν παντοτινά αναφορικά με την εκάστοτε περιοχή ενδιαφέροντος**. Παραδείγματος χάριν, μπορούμε να δηλώσουμε ότι η Mia, η Jody, και η Γιολάντα είναι γυναίκες, και ότι η Jody παίζει κιθάρα, χρησιμοποιώντας τα ακόλουθα τέσσερα γεγονότα:

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
```

Αυτή η συλλογή των γεγονότων είναι μία βάση γνώσης. Είναι το πρώτο παράδειγμά μας ενός προγράμματος Prolog. Σημειώστε ότι το mia, το jody, και το yolanda (τα ονόματα δηλαδή), και οι ιδιότητες woman και playsAirGuitar, έχει γραφτεί έτσι ώστε ο πρώτος χαρακτήρας να είναι πεζός. Αυτό είναι σημαντικό γιατί στην Prolog με κεφαλαίους χαρακτήρες ή με το underscore _, ξεκινούν οι μεταβλητές.
Μπορούμε να χρησιμοποιήσουμε την παραπάνω βάση γνώσης με την τοποθέτηση των ερωτήσεων. Δηλαδή υποβάλλοντας ερωτήσεις πάνω στις πληροφορίες που περιέχονται. Για παράδειγμα μπορούμε να ρωτήσουμε την Prolog εάν το mia είναι γυναίκα με την τοποθέτηση της ερώτησης:

```
?- woman(mia).
```

Η Prolog θα απαντήσει:

```
yes
```

για τον προφανή λόγο ότι αυτό είναι ένα από τα γεγονότα που καταγράφονται ρητά στην βάση γνώσης μας. Το σύμβολο ?- (ή κάτι σαν το ?-, ανάλογα με ποιά έκδοση της Prolog χρησιμοποιούμε) είναι το σύμβολο που εμφανίζει ο διερμηνέας Prolog όταν περιμένει να εισάγουμε μια ερώτηση. Δακτυλογραφούμε ακριβώς την πραγματική ερώτηση
(παραδείγματος χάριν woman(mia)) ακολουθούμενη από . (την τελεία).

Ομοίως, μπορούμε να ρωτήσουμε εάν Jody παίζει κιθάρα, με την τοποθέτηση της ακόλουθης ερώτησης:

```
?- playsAirGuitar(jody).
```

Η Prolog θα απαντήσει πάλι "ναι", επειδή αυτό είναι ένα από τα γεγονότα που καταγράφονται ρητά στην βάση γνώσης μας. Εντούτοις, υποθέστε ότι ρωτάμε εάν η mia παίζει κιθάρα:

```
?- playsAirGuitar(mia).
```

Θα πάρουμε την απάντηση:

```
no
```

Αυτό συμβαίνει γιατί δεν είναι ένα γεγονός που καταγράφεται στην βάση γνώσης μας. Επιπλέον, η βάση γνώσης μας δεν περιέχει καμία άλλη πληροφορία (όπως *κανόνες*) που θα βοηθήσει Prolog να προσπαθήσει να συμπεράνει εάν η mia παίζει κιθάρα. Έτσι η Prolog σωστά καταλήγει στο συμπέρασμα ότι το playsAirGuitar (mia) δεν προκύπτει από την βάση γνώσης μας.

Στην PROLOG τα γεγονότα εκφράζουν (με κωδικοποιημένη μορφή) κάποιες πρωτογενείς γνώσεις σχετικές με το πρόβλημα που δεν μπορούν να παραχθούν από άλλες. Για παράδειγμα οι έννοιες:
Η ΜΑΡΙΑ ΕΙΝΑΙ ΜΗΤΕΡΑ ΤΗΣ ΚΑΙΤΗΣ
Ο ΓΙΩΡΓΟΣ ΤΡΩΕΙ ΜΗΛΑ
Κωδικοποιούνται ως εξής:

```
is_mother(maria,kaiti).
```

Όπου ορίζουμε μια σχέση (μητέρα) που συνδέει δυο αντικείμενα (Μαρία), (Καίτη). Το ΕΙΝΑΙ ΜΗΤΕΡΑ εαποτελεί την φυσική σημασία που δίνουμε εμείς στο κατηγόρημα, και όχι αυτό που δίνει η ίδια η Prolog.Η δεύτερη περίπτωση είναι:

```
eat(george,apple).
```

5. Κανόνες:

Έστω ότι έχουμε την ακόλουθη βάση γνώσης:

```
listensToMusic(mia).
happy(yolanda).
playsAirGuitar(mia) :- listensToMusic(mia).
playsAirGuitar(yolanda) :- listensToMusic(yolanda).
listensToMusic(yolanda):- happy(yolanda).
```

Αυτή η βάση γνώσης περιέχει δύο γεγονότα, το listensToMusic(mia) και happy(yolanda). Τα τελευταία τα τρία στοιχεία είναι κανόνες.

Οι κανόνες δηλώνουν τις πληροφορίες που ισχύουν *υπό όρους* αναφορικά με την εκάστοτε περιοχή ενδιαφέροντος. Παραδείγματος χάριν, ο πρώτος κανόνας λέει ότι η mia παίζει κιθάρα *εάν* ακούει τη μουσική, και ο τελευταίος κανόνας λέει ότι η yolanda ακούει τη μουσική *εάν* αυτή είναι happy. Γενικότερα, το σύμβολο : - πρέπει να διαβάζεται σαν "εάν", ή "υπονοείται από". Το μέρος στην αριστερή πλευρά του συμβόλου : - καλείται το «κεφάλι» του κανόνα, το μέρος στη δεξιά πλευρά καλείται «σώμα». Έτσι στους γενικούς κανόνες συμβαίνει το εξής: *εάν* το σώμα του κανόνα είναι αληθινό, *κατόπιν* το κεφάλι του κανόνα είναι επίσης αληθινό. Και τώρα για το βασικό σημείο: *εάν μια βάση γνώσης περιέχει ένα κανόνα με κεφάλι και σώμα. και η Prolog ξέρει ότι το* σώμα ισχύει - *προκύπτει από τις πληροφορίες στη βάση γνώσεων τότε η Prolog μπορεί να συμπεράνει ότι ισχύει το* κεφάλι.

Για να γίνουν σαφή τα παραπάνω θα δούμε ένα παράδειγμα. Θα ρωτήσουμε την Prolog εάν η mia παίζει κιθάρα :

```
?- playsAirGuitar(mia).
```

Η Prolog θα αποκριθεί "ναι". Γιατί; Αν και playsAirGuitar (mia) δεν είναι ένα γεγονός ρητά καταγραμμένο στην βάση γνώσης μας, η βάση γνώσης μας περιέχει τον κανόνα

```
playsAirGuitar(mia) :- listensToMusic(mia).
```

Επιπλέον, η βάση γνώσης μας επίσης περιέχει το γεγονός listensToMusic(mia). Ως εκ τούτου η Prolog μπορεί να χρησιμοποιήσει την προαναφερθείσα συμπερασματική μέθοδο για να συναγάγει ότι το playsAirGuitar(mia) είναι αληθές.

Οι κανόνες εκφράζουν δευτερογενείς γνώσεις, δηλαδή γνώσεις που μπορούν να παραχθούν από άλλες. Για παράδειγμα η πρόταση:
ΚΑΘΕ ΜΗΤΕΡΑ ΕΙΝΑΙ ΓΥΝΑΙΚΑ
Μπορεί να ερμηνευθεί σαν:
ΑΝ Η Χ ΕΙΝΑΙ ΜΗΤΕΡΑ ΤΟΥ Υ, ΤΟΤΕ Η Χ ΕΙΝΑΙ ΓΥΝΑΙΚΑ.
Που μπορεί να κωδικοποιηθεί σαν:

```
is_woman(X):-is_mother(X,Y).
```

Είναι εξίσου αποδεκτό να γραφεί σαν:

```
is_woman(X):-is_mother(X,_).
```

Όπου η μεταβλητή Υ είναι ανώνυμη μεταβλητή (δεν χρησιμοποιείται πουθενά αλλού στην πρόταση) και άρα δεν χρειάζεται να ονομασθεί, οπότε υποκαθίσταται από την υπογράμμιση (_).
Φυσικά το σώμα των περισσοτέρων προτάσεων αποτελείται από περισσότερες της μιας στοιχειώδεις προτάσεις.
Για παράδειγμα στην περιγραφή ενός απλού γενεαλογικού δένδρου μιας οικογένειας που όλοι γνωρίζουμε:

```
is_mother(eve,abel).
is_mother(eve,kain).
is_father(adam,abel).
is_father(adam,kain).
```

μπορούμε να ορίσουμε την έννοια «γονέας» (parent), με διάζευξη (λογικό OR), δυο προτάσεων:

```
is_parent(X,Y):- is_mother(X,Y) ; is_father(X,Y).
```

Το οποίο μεταφράζεται σαν:
Ο Χ ΕΙΝΑΙ ΓΟΝΕΑΣ ΤΟΥ Υ ΑΝ Ο Χ ΕΙΝΑΙ ΜΗΤΕΡΑ ΤΟΥ Υ Η Ο Χ ΕΙΝΑΙ

ΠΑΤΕΡΑΣ ΤΟΥ Υ.

Το OR συνήθως δεν συναντάται σε προγράμματα Prolog, αντί αυτού ο κανόνας γράφεται δύο ή (ανάλογα με την περίπτωση) παραπάνω φορές.

6. Αναζήτηση λύσεων:

Για να αντλήσουμε γνώση από το σύστημα πρέπει να υποβάλλουμε ερωτήσεις (query).

Σε σχέση με το προηγούμενο παράδειγμα μπορούμε να υποβάλουμε την ερώτηση:

```
?is_mother(eve,abel).
```

Οπότε προκύπτει η απάντηση:

```
Yes
```

Στην ερώτηση:

```
?is_mother(eve,adam).
```

Η απάντηση που προκύπτει είναι:

```
No
```

(σε κάποιες υλοποιήσεις της PROLOG μπορεί η απάντηση να είναι nil, ή κάτι αντίστοιχο).

Στην ερώτηση:

```
?is_father(adam,X).
```

το σύστημα θα απαντήσει:

```
X = abel
X = kain
Yes
```

Δηλαδή όλες τις τιμές της μεταβλητής Χ που επαληθεύουν την πρόταση του ερωτήματος.
Φυσικά μπορούμε να υποβάλλουμε ερωτήματα με συνδυασμό προτάσεων, κλπ.

7. Αναδρομή:

Τα κατηγορήματα (predicates) μπορούν να οριστούν αναδρομικά. **Ένα κατηγόρημα ορίζεται αναδρομικά, όταν κάποιος από τους κανόνες που περιέχει ξανακαλούν το ίδιο κατηγόρημα.**
Παράδειγμα:

Έχουμε την ακόλουθη βάση γνώσης:

```
is_digesting(X,Y) :- just_ate(X,Y).
is_digesting(X,Y) :- just_ate(X,Z),
                     is_digesting(Z,Y).

just_ate(mosquito,blood(john)).
just_ate(frog,mosquito).
just_ate(stork,frog).
```

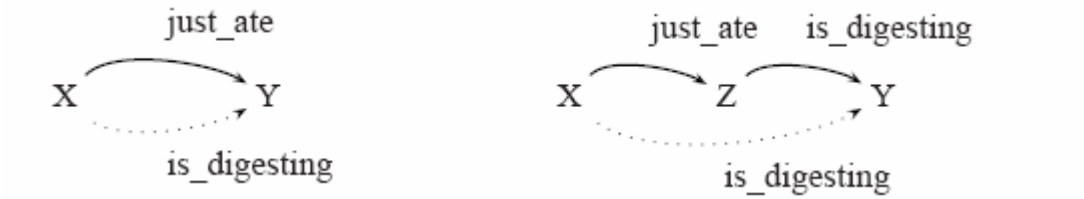
Με μία πρώτη ματιά η βάση αυτή φαίνεται συνηθισμένη, αλλά ο ορισμός του κατηγορήματος is_digesting είναι αναδρομικός. Το is_digesting στον δεύτερο κανόνα εμφανίζεται και στην αριστερή και στην δεξιά μεριά του κανόνα. Παρόλα αυτά όμως, υπάρχει έξοδος από αυτόν τον βρόγχο. Η έξοδος αυτή παρέχεται από το just_ate κατηγόρημα, το οποίο βρίσκεται και στους δύο κανόνες. Από τον πρώτο κανόνα μάλιστα, αποδεικνύεται ότι αν ισχύει το just_ate, τότε ισχύει και το is_digesting. Ο πρώτος κανόνας δεν είναι αναδρομικός.

Αν κοιτάξουμε τον πρώτο κανόνα ερμηνευτικά, θα δούμε ότι λέει το εξής: αν κάτι μόλις έφαγε (just_ate) τότε χωνεύει (is_digesting). Αυτή είναι η δηλωτική

– ερμηνευτική σκοπιά του πρώτου κανόνα. Πρέπει να σημειώσουμε ότι υπάρχουν πολλές περιπτώσεις που η ερμηνευτική πλευρά ενός κανόνα διαφέρει από την πραγματική – πρακτική του μεριά, όταν αυτή τεθεί σε εφαρμογή.

Ο δεύτερος κανόνας, δηλαδή ο αναδρομικός, λέει ότι: εάν το X έφαγε το Z και το Z χωνεύει το Y, τότε το X χωνεύει επίσης το Y.

Θα δούμε τι κάνει ο δεύτερος κανόνας από την πρακτική του πλευρά, την διαδικαστική. Η Prolog θα ψάξει να εντοπίσει κάποιο Z τέτοιο ώστε το X να έχει φάει το Z και το Z να χωνεύει το Y. Ο κανόνας αυτός σπάει δηλαδή σε δύο επιμέρους μέρη. Οι παρακάτω εικόνες δείχνουν καθαρά τον τρόπο με τον οποίο λειτουργούν οι δύο κανόνες.



Για να δούμε την λειτουργία τους, θέτουμε το ακόλουθο ερώτημα:

```
?- is_digesting(stork,mosquito).
```

Η Prolog προσπαθεί να αποδείξει για το εάν το ερώτημα αυτό είναι αληθές. Στην αρχή προσπαθεί να χρησιμοποιήσει τον πρώτο κανόνα που ορίζει το is_digesting. Συγκεκριμένα ο πρώτος κανόνας λέει το X is_digesting το Y εάν το X just_ate το Y. Το X ενοποιείται με το stork και το Y με το mosquito. Οπότε έχουμε:

```
just_ate(stork,mosquito).
```

Αλλά η βάση γνώσης μας δεν έχει τέτοια πληροφορία, οπότε ο κανόνας αυτός δεν επαληθεύεται. Η Prolog προσπαθεί να αποδείξει εάν είναι αληθείς ο

δεύτερος κανόνας. Αντικαθιστώντας το X με το stork και το Y με το mosquito, έχουμε τους ακόλουθους στόχους:

```
just_ate(stork,Z),
is_digesting(Z,mosquito).
```

Για να αποδείξει δηλαδή η Prolog ότι ισχύει is_digesting(stork,mosquito), πρέπει να βρει ένα Z τέτοιο ώστε να ισχύει:

```
just_ate(stork,Z).
```

Και

```
is_digesting(Z,mosquito).
```

Υπάρχει ένα τέτοιο Z, ο frog. Το γεγονός just_ate(stork,frog) είναι αληθές, γιατί βρίσκεται στην βάση γνώσης. Επίσης, από τον πρώτο κανόνα του is_digesting βγαίνει το συμπέρασμα ότι το is_digesting(frog,mosquito) είναι αληθές, από την στιγμή που το γεγονός just_ate(frog,mosquito) βρίσκεται στην βάση γνώσης.

Καταλήγουμε στο συμπέρασμα ότι όποτε συναντάμε αναδρομικά κατηγορήματα, πρέπει να έχουν τουλάχιστον δύο προτάσεις: μία βασική πρόταση, η οποία σταματάει την αναδρομή σε ένα συγκεκριμένο σημείο, και μία πρόταση που περιέχει την αναδρομή. Πρέπει να είμαστε ιδιαίτερα προσεκτικοί για να μην περιπέσουμε σε ατέρμονα βρόγχο λόγω αναδρομής.

Ένα παράδειγμα ατέρμονης αναδρομής είναι ο ακόλουθος απλούστατος «φαινομενικά» κανόνας.

```
p :- p.
```

Αυτός ο κανόνας μπορεί να ισχύει ερμηνευτικά (δηλώνει ότι το p ισχύει όταν ισχύει το p), αλλά όσον αφορά την πρακτική του πλευρά, είναι ένας πολύ επικίνδυνος κανόνας. Και στις δύο πλευρές του κανόνα υπάρχει το ίδιο πράγμα, και δεν υπάρχει κάποιος βασικός κανόνας για να αποδράσουμε από αυτόν τον βρόγχο της αναδρομής.

Αν παραδείγματος χάριν θέσουμε το ερώτημα :

```
?- p.
```

Η Prolog θα προσπαθήσει να αποδείξει εάν ισχύει το p. Από την βάση γνώσης ξέρει ότι το p ισχύει εάν ισχύει το p. Οπότε ψάχνει το πότε ισχύει το p, το οποίο ισχύει εάν ισχύει το p, κ.ο.κ..

Πληκτρολογώντας την εντολή trace, μπορούμε να δούμε τα βήματα που ακολουθεί η Prolog προσπαθώντας να αποδείξει εάν το ερώτημα είναι αληθές (αυτό ισχύει μόνο για την SWI-Prolog).

8. Λίστες:

Οι λίστες είναι μια σημαντική επαναλαμβανόμενη δομή δεδομένων χρησιμοποιούμενη ευρέως σε γλωσσολογικούς υπολογισμούς (π.χ. συντακτική και λεκτική ανάλυση).
Όπως το όνομά της προτείνει, μία λίστα είναι ακριβώς ένας σαφής κατάλογος στοιχείων. Ελαφρώς ακριβέστερα, είναι μια πεπερασμένη ακολουθία στοιχείων. Εδώ είναι μερικά παραδείγματα των καταλόγων σε Prolog:

```
[mia, vincent, jules, yolanda].
[mia, robber(honey_bunny), X, 2, mia].
[].
[mia, [vincent, jules], [butch, girlfriend(butch)]].
[[], dead(zed), [2, [b, chopper]], [], Z, [2, [b, chopper]]].
```

Μπορούμε να μάθουμε μερικά σημαντικά πράγματα από αυτά τα παραδείγματα.

- Μπορούμε να διευκρινίσουμε τις λίστες σε Prolog με το να εσωκλείσουμε τα στοιχεία τους με τα σύμβολα [και]. Τα στοιχεία χωρίζονται από τα κόμματα.
  Παραδείγματος χάριν, το πρώτο παράδειγμά μας [mia, vincent, jules, yolanda] είναι μία λίστα με τέσσερα στοιχεία, δηλαδή mia, vincent, jules, και yolanda. Το μήκος της λίστας είναι ο αριθμός στοιχείων που έχει, έτσι το πρώτο παράδειγμά μας είναι μία λίστα μήκους τέσσερα.

- Από το δεύτερο παράδειγμά μας, [mia, robber(honey_bunny), X, 2, mia], μαθαίνουμε ότι όλα τα είδη των αντικειμένων Prolog μπορούν να είναι στοιχεία μιάς λίστας. Το πρώτο στοιχείο αυτής της λίστας είναι mia (ένα άτομο), το δεύτερο στοιχείο είναι robber(honey_bunny), ένας σύνθετος όρος, το τρίτο στοιχείο είναι X (μια μεταβλητή) και το τέταρτο στοιχείο είναι 2, ένας αριθμός. Επιπλέον, μαθαίνουμε ότι το ίδιο στοιχείο μπορεί να εμφανιστεί περισσότερο από μία φορά στην ίδια λίστα: παραδείγματος χάριν, το πέμπτο στοιχείο αυτού του καταλόγου είναι mia, το οποίο είναι ίδιο με το πρώτο στοιχείο.

- Το τρίτο παράδειγμα δείχνει ότι υπάρχει μία πολύ ειδική λίστα, η κενή λίστα. Η κενή λίστα (όπως το όνομά της προτείνει) είναι η λίστα που δεν περιλαμβάνει κανένα στοιχείο. Ποιό είναι το μήκος της κενής λίστας; Μηδέν, φυσικά (γιατί το μήκος μιας λίστας είναι ο αριθμός των μελών που περιέχει, και η κενή λίστα δεν περιέχει τίποτα).

- Το τέταρτο παράδειγμα μας διδάσκει κάτι εξαιρετικά σημαντικό: οι λίστες μπορούν να περιέχουν άλλες λίστες ως στοιχεία. Παραδείγματος χάριν, το δεύτερο στοιχείο [mia, [vincent, jules], [butch, girlfriend(butch)] είναι η λίστα [vincent, jules], και το τρίτο στοιχείο είναι [butch, girlfriend(butch)]]. Εν ολίγοις, οι λίστες είναι παραδείγματα των αναδρομικών δομών δεδομένων: οι λίστες μπορούν να συνταχτούν από λίστες. Ποιο είναι το μήκος της τέταρτης λίστας; Η απάντηση είναι: τρία. Εάν σκεφτόμασταν ότι ήταν πέντε (ή πράγματι, τίποτ' άλλο) τότε δεν θα σκεφτόμασταν για τις λίστες με τον σωστό

τρόπο. Τα στοιχεία της λίστας είναι τα πράγματα μεταξύ των εξώτερων [και], που χωρίζονται με κόμμα. Έτσι αυτή η λίστα περιλαμβάνει *τρία* στοιχεία: το πρώτο στοιχείο είναι mia, το δεύτερο στοιχείο είναι [vincent, jules], και το τρίτο στοιχείο είναι [butch, girlfriend (butch)].

- Το τελευταίο παράδειγμα αναμιγνύει όλες αυτές τις ιδέες από κοινού. Έχουμε εδώ μία λίστα που περιέχει την κενή λίστα (στην πραγματικότητα, την περιέχει δύο φορές), τον σύνθετο όρο dead(zed), δύο αντίγραφα της λίστας [2, [b, chopper]], και το μεταβλητό Z. Σημειώστε ότι τα τρίτα (και το τελευταίο) στοιχεία είναι λίστες που περιέχουν λίστες συγκεκριμένα [b, chopper]).

Τώρα φτάσαμε σε ένα πολύ σημαντικό σημείο. Οποιαδήποτε μη άδεια λίστα μπορεί να θεωρηθεί αποτελούμενη από δύο μέρη: το κεφάλι και η ουρά. Το κεφάλι είναι απλά το πρώτο στοιχείο στην λίστα, η ουρά είναι όλα τα άλλα. Ή ακριβέστερα, η ουρά είναι η λίστα που παραμένει όταν παίρνουμε το πρώτο στοιχείο μακριά, δηλ. *η ουρά μιας λίστας είναι πάντα μία λίστα* πάλι. Παραδείγματος χάριν, το κεφάλι

```
[mia, vincent, jules, yolanda]
```

είναι mia και η ουρά είναι [vincent, jules, yolanda]. Ομοίως, το κεφάλι

```
[[], dead(zed), [2, [b, chopper]], [], Z, [2, [b, chopper]]]
```

είναι [], και η ουρά είναι [dead(zed), [2, [b, chopper]], [], Z, [2, [b, chopper]]]]. Και ποιο είναι το κεφάλι και η ουρά της λίστας [dead(zed)]; Καλά, το κεφάλι είναι το πρώτο το στοιχείο του καταλόγου, που είναι το dead(zed), και ουρά είναι ο κατάλογος που παραμένει εάν πάρουμε το κεφάλι μακριά, το οποίο, σε αυτήν την περίπτωση, είναι ο κενός κατάλογος [].

Σημειώστε ότι μόνο οι non-empty λίστες έχουν κεφάλια και ουρές. Δηλαδή ο κενός κατάλογος δεν περιέχει κάποια εσωτερική δομή. Για την Prolog, ο κενός κατάλογος [] είναι ένας ειδικός, ιδιαίτερα απλός, κατάλογος.

Η Prolog έχει έναν ειδικό ενσωματωμένο τελεστή | που μπορεί να χρησιμοποιηθεί για να αποσυνθέσει έναν κατάλογο στο κεφάλι και ουρά του. Είναι *πολύ σημαντικό* να γνωρίζει ο προγραμματιστής να ξέρει πώς να χρησιμοποιήσει τον τελεστή |, γιατί είναι ένα βασικό εργαλείο για προγράμματα χειρισμού λιστών σε Prolog.

Η προφανέστερη χρήση του τελεστή | είναι να εξαχθούν οι πληροφορίες από τους καταλόγους - λίστες. Κάνουμε αυτό με να χρησιμοποιήσουμε | από κοινού με το matching. Παραδείγματος χάριν, για να πάρουμε το κεφάλι και την ουρά από την λίστα [mia, vincent, jules, yolanda] μπορούμε να θέσουμε την ακόλουθη ερώτηση:

```
?- [Head| Tail] = [mia, vincent, jules, yolanda].

Head = mia

Tail = [vincent,jules,yolanda]

Yes
```

Δηλαδή το κεφάλι ης λίστας έχει γίνει το Head και η ουρά της λίστας έχει γίνει το Tail. Σημειώστε ότι δεν υπάρχει τίποτα ειδικό για το Head και το Tail, είναι απλά μεταβλητές. Θα μπορούσαμε εξ ίσου καλά να είχαμε θέσει την ερώτηση:

```
?- [X|Y] = [mia, vincent, jules, yolanda].

X = mia

Y = [vincent,jules,yolanda]

yes
```

Όπως αναφέραμε ανωτέρω, μόνο οι non-empty λίστες έχουν κεφάλια και ουρές. Εάν προσπαθούμε να χρησιμοποιήσουμε τον τελεστή | για να χωρίσουμε την κενή λίστα [ ], η Prolog θα αποτύχει:

```
?- [X|Y] = [].

no
```

Δηλαδή η Prolog μεταχειρίζεται την λίστα [] ως ειδική λίστα. Αυτή η παρατήρηση είναι *πολύ* σημαντική.. Εξετάστε μερικά άλλα παραδείγματα. Μπορούμε να εξαγάγουμε το κεφάλι και την ουρά της ακόλουθης λίστας ακριβώς όπως είδαμε ανωτέρω:

```
?- [X|Y] = [[], dead(zed), [2, [b, chopper]], [], Z].

X = []

Y = [dead(zed),[2,[b,chopper]],[],_7800]

Z = _7800

yes
```

Αυτό σημαίνει: το κεφάλι της λίστας είναι συνδεδεμένο - Matched στο X, η ουρά είναι συνδεδεμένη στο Y. (Παίρνουμε επίσης πληροφορίες ότι Prolog έχει δεσμεύσει το ζ στην εσωτερική μεταβλητή _ 7800.) Πρέπει να σημειώσουμε ότι το _7800 είναι τυχαίος αριθμός και μπορεί να διαφέρει ανάλογα με την έκδοση της Prolog.

Αλλά μπορούμε να κάνουμε πολύ περισσότερα με τον τελεστή | , ο οποίος είναι πραγματικά ένα πολύ εύκαμπτο εργαλείο. Παραδείγματος χάριν, υποθέστε ότι θέλαμε να ξέρουμε τι ήταν τα πρώτα *δύο* στοιχεία της λίστας, και επίσης ότι απέμενε από την λίστα μετά από το δεύτερο στοιχείο. Θα θέταμε την ακόλουθη ερώτηση:

```
?- [X,Y | W] = [[], dead(zed), [2, [b, chopper]], [], Z].

X = []

Y = dead(zed)

W = [[2,[b,chopper]],[],_8327]

Z = _8327

Yes
```

ένο στο X, το δεύτερο στοιχείο είναι συνδεδεμένο στο Y, και το υπόλοιπο του καταλόγου μετά από το δεύτερο στοιχείο είναι συνδεδεμένο στο W. Το W είναι ο κατάλογος που παραμένει όταν παίρνουμε μαζί τα πρώτα δύο στοιχεία. Έτσι, ο τελεστής | μπορεί όχι μόνο να χρησιμοποιηθεί για να χωρίσει έναν κατάλογο στο κεφάλι του και στην ουρά του, αλλά μπορούμε στην πραγματικότητα να τον χρησιμοποιήσουμε για να χωρίσουμε έναν κατάλογο σε οποιοδήποτε σημείο. Αριστερά του τελεστή |, εμείς πρέπει απλά να απαριθμήσουμε πόσα στοιχεία θέλουμε να πάρουμε μαζί από την αρχή του καταλόγου, και δεξιά του τελεστή | θα πάρουμε θα πάρουμε τα υπόλοιπα στοιχεία του καταλόγου - λίστας. Σε αυτό το παράδειγμα, παίρνουμε επίσης τις πληροφορίες ότι Prolog έχει δεσμεύσει το Z στην εσωτερική μεταβλητή _ 8327.

Ήρθε η στιγμή να μάθουμε για το τι εστί η ανώνυμη μεταβλητή. Υποθέστε ότι ενδιαφερόμαστε να πάρουμε το δεύτερο και το τέταρτο στοιχείο της λίστας:

```
[[], dead(zed), [2, [b, chopper]], [], Z].
```

Μπορούμε να το εντοπίσουμε ως εξής:

```
?- [X1,X2,X3,X4 | Tail] = [[], dead(zed), [2, [b, chopper]], [], Z].

X1 = []

X2 = dead(zed)
```

Έχουμε τις πληροφορίες που θελήσαμε: οι τιμές που ενδιαφερόμαστε είναι συνδεδεμένες στις μεταβλητές X2 και X4. Αλλά έχουμε πολλές άλλες πληροφορίες επίσης (συγκεκριμένα τιμές που είναι δεσμευμένες από τα X1, X3 και Tail). Και ίσως δεν ενδιαφερόμαστε για όλες αυτές τις άλλες πληροφορίες. Σε αυτή την περίπτωση, είναι λίγο ανόητο να πρέπει ρητά να εισαχθούν οι μεταβλητές X1, X3 και Tail. Και στην πραγματικότητα, υπάρχει ένας απλούστερος τρόπος να λάβουμε *μόνο* τις πληροφορίες που θέλουμε:
Μπορούμε να θέσουμε την ακόλουθη ερώτηση αντ' αυτού:

```
?- [_,X,_,Y|_] = [[], dead(zed), [2, [b, chopper]], [], Z].

X = dead(zed)

Y = []

Z = _9593

yes
```

Το _ underscore σύμβολο (δηλαδή *κάτω παύλα*) είναι η ανώνυμη μεταβλητή. Το χρησιμοποιούμε όταν έχουμε ανάγκη να χρησιμοποιήσουμε μια μεταβλητή, αλλά δεν ενδιαφερόμαστε για το τι αντικαθιστεί η Prolog σε αυτή. Όπως μπορείτε να δείτε στο ανωτέρω παράδειγμα, Prolog δεν μας ενημέρωσε με τι ήταν συνδεδεμένο το _ . Επιπλέον, σημειώστε ότι κάθε σύμβολο _ είναι *ανεξάρτητο*: κάθε φορά είναι συνδεδεμένο σε κάτι διαφορετικό. Αυτό δεν θα μπορούσε να συμβεί με μια συνηθισμένη μεταβλητή φυσικά, αλλά η ανώνυμη μεταβλητή δεν προορίζεται να είναι συνηθισμένη. Είναι απλά ένας τρόπος να λέμε στην Prolog να δεσμεύσει κάτι σε μια δεδομένη θέση, εντελώς ανεξάρτητα από οποιεσδήποτε άλλες συνδέσεις.

Εξετάστε ένα τελευταίο παράδειγμα. Το τρίτο στοιχείο του παραδείγματός μας είναι μία λίστα (συγκεκριμένα [2, [b, chopper]]). Υποθέστε ότι θελήσαμε να εξαγάγουμε την ουρά αυτής της εσωτερικής λίστας, και ότι δεν ενδιαφερόμαστε για οποιεσδήποτε άλλες πληροφορίες. Πώς θα μπορούσαμε να κάνουμε αυτό; Όπως
ακολουθεί:

```
?- [_,_,[_|X]|_] =
        [[], dead(zed), [2, [b, chopper]], [], Z, [2, [b, chopper]]].

X = [[b,chopper]]

Z = _10087

Yes
```

9. Συναρτησιακοί Όροι:

Στους συναρτησιακούς όρους ανήκουν και οι λίστες. Με συναρτησιακούς όρους μπορούμε να προσομοιώσουμε μια λίστα. Για παράδειγμα έστω ότι έχουμε τα εξής στοιχεία για πέντε υπαλλήλους:

```
Ο John εργάζεται στην Nixon.
Ο George εργάζεται στην Sony.
Ο John αμείβεται 5000 Ευρώ.
Ο George αμείβεται 1000 Ευρώ.
```

Τα στοιχεία αυτά τα προσομοιώνουμε στην Prolog ως εξής:

```
works(john,nixon).
works(george,sony).
gets(john,5000).
gets(george,1000).
```

Μπορούμε να ενσωματώσουμε τα στοιχεία αυτά σε μια λίστα :

```
[works(john,nixon),works(george,sony),gets(john,5000),gets(george,1000)].
```

Έχουμε την ακόλουθη βάση γνώσης:

```
woman(mia).

woman(jody).

woman(yolanda).

loves(vincent,mia).

loves(marcellus,mia).

loves(pumpkin,honey_bunny).

loves(honey_bunny,pumpkin).
```

Σε αυτήν την βάση γνώσης δεν υπάρχει κανένας κανόνας, μόνο μια συλλογή από γεγονότα. Εντάξει, βλέπουμε μια σχέση που έχει δύο ονόματα ως ορίσματα για πρώτη φορά (συγκεκριμένα η σχέση loves).
Όχι, η καινοτομία βρίσκεται αυτή τη φορά όχι στη βάση γνώσεων, αλλά στις ερωτήσεις που θα θέσουμε. Ειδικότερα, για πρώτη φορά πρόκειται να χρησιμοποιήσουμε τις μεταβλητές. Έχουμε το ακόλουθο παράδειγμα:

```
?- woman(X).
```

Το X είναι μια μεταβλητή (στην πραγματικότητα, οποιαδήποτε λέξη αρχίζοντας με ένα κεφαλαίο χαρακτήρα είναι μια μεταβλητή στην Prolog). Τώρα μια μεταβλητή δεν είναι ένα όνομα, αλλά είναι ένα "placeholder" για τις πληροφορίες (δηλαδή ένας χώρος εις τον οποίον μπορούν να τοποθετηθούν πληροφορίες).
Δηλαδή αυτή η ερώτηση ρωτά ουσιαστικά την Prolog: εμφάνισέ μου όποιο άτομο γνωρίζεις το οποίο είναι γυναίκα.
Η Prolog απαντά σε αυτήν την ερώτηση ψάχνοντας μέσα στην βάση γνώσης, από πάνω έως κάτω, προσπαθώντας να ταιριάξει με (ή να ενοποιήσει) την έκφραση woman(X)με τις πληροφορίες που περιέχει η βάση γνώσης. Τώρα το πρώτο στοιχείο στη βάση γνώσεων είναι το woman(mia). Έτσι, αντιστοιχίζει η Prolog το X στο mia, κάνοντας κατά συνέπεια την ερώτηση να συμφωνήσει τέλεια με αυτό το πρώτο στοιχείο. (Μπορούμε επίσης να χρησιμοποιήσουμε διαφορετική ορολογία για αυτήν την διαδικασία: μπορούμε π.χ. να πούμε ότι η Prolog instantiates το X στο mia, ή ότι δεσμεύει το X στο mia.). Η Prolog μας αναφέρει μετά το εξής:

```
X = mia
```

Δηλαδή όχι μόνο λέει ότι υπάρχουν πληροφορίες για τουλάχιστον μια γυναίκα στην βάση γνώσης μας, αλλά μας λέει πραγματικά ποια είναι. Όχι μόνο είπε "ναι", αλλά μας έδωσε πραγματικά τη δέσμευση της μεταβλητής, που οδήγησε στην επιτυχία. Το σημαντικό σημείο των μεταβλητών — και όχι μόνο στην Prolog — είναι ότι μπορούν "να αντιπροσωπεύσουν" ή "να ταιριάξουν με"

διαφορετικά πράγματα. Και όντως, υπάρχουν πληροφορίες για άλλες γυναίκες στη βάση γνώσεων. Μπορούμε να έχουμε πρόσβαση σε αυτές τις πληροφορίες με τη δακτυλογράφηση της ακόλουθης απλής ερώτησης:

```
?- ;
```

Το σύμβολο ; σημαίνει «ή», δηλαδή πατώντας το ; ζητάμε από την Prolog να μας δώσει τις άλλες λύσεις, εάν υπάρχουν, δηλαδή είναι σαν να θέτουμε το παρακάτω ερώτημα: υπάρχουν άλλες γυναίκες; Έτσι η Prolog αρχίζει μέσω της βάσης γνώσης πάλι (θυμάται από ποιο σημείο είχε πάρει την απάντηση την τελευταία φορά και ξεκινάει την αναζήτηση από εκεί) και βλέπει ότι εάν ταιριάξει με X με το jody, τότε η ερώτηση συμφωνεί τέλεια με τη δεύτερη γραμμή στη βάση γνώσης. Έτσι αποκρίνεται:

```
X = jody
```

Μας λέει ότι υπάρχουν πληροφορίες για μια δεύτερη γυναίκα στην βάση γνώσης μας, και μας δίνει πραγματικά την λύση που οδήγησε στην επιτυχία. Και φυσικά, εάν πιέσουμε το ; μία δεύτερη φορά, η Prolog επιστρέφει την απάντηση :

```
X = yolanda
```

Αλλά τι θα συμβεί εάν πιέσουμε το ; μια τρίτη φορά; Η Prolog αποκρίνεται "no". Καμία άλλη αντιστοιχία δεν είναι δυνατή. Δεν υπάρχει κανένα άλλο γεγονός που αρχίζει από τη συμβολοσειρά woman. Τα τελευταία τέσσερα στοιχεία στη βάση γνώσεων αφορούν τη σχέση loves, και δεν υπάρχει κανένας τρόπος να ταιριάξουν με μια ερώτηση της μορφής της woman(X).

Δοκιμάζουμε μια πιό περίπλοκη ερώτηση, δηλαδή:

```
loves(marcellus,X),woman(X).
```

Τώρα, ήρθε η στιγμή να πούμε ότι το σύμβολο , (το κόμμα) σημαίνει «*και*», έτσι αυτή η ερώτηση λέει: *είναι εκεί οποιοδήποτε μεμονωμένο* X *τέτοιο ώστε ο marcellus να αγαπά* το X *και* το X να *είναι γυναίκα*; Εάν εξετάσουμε τη βάση γνώσης θα δούμε ότι υπάρχει: Το mia είναι γυναίκα (όπως ισχύει από το πρώτο γεγονός) και ο marcellus αγαπά την mia (γεγονός 5). Στην πραγματικότητα, η Prolog είναι σε θέση να το εντοπίσει αυτό. Δηλαδή μπορεί να ψάξει μέσω της βάσης γνώσης και εάν ταιριάξει το X με το mia, τότε και τα δύο μέλη της ερώτησης ικανοποιούνται. Έτσι η Prolog δίνει την απάντηση:

```
X = mia
```

Αυτό το ταίριασμα των μεταβλητών με τις πληροφορίες στη βάση γνώσεων είναι η καρδιά της Prolog. Οπωσδήποτε, η Prolog έχει πολλές άλλες ενδιαφέρουσες πτυχές — αλλά το ταίριασμα των μεταβλητών με τις πληροφορίες στη βάση γνώσεων, είναι η πιο κρίσιμη δυνατότητα της Prolog.
(Αναφορά:Κεφάλαιο Β **Learn Prolog Now!** Patrick Blackburn, Johan Bos, Kristina Striegnitz)

Γ. Prolog και ανάλυση φυσικής γλώσσας

1.Parsing σε Prolog:

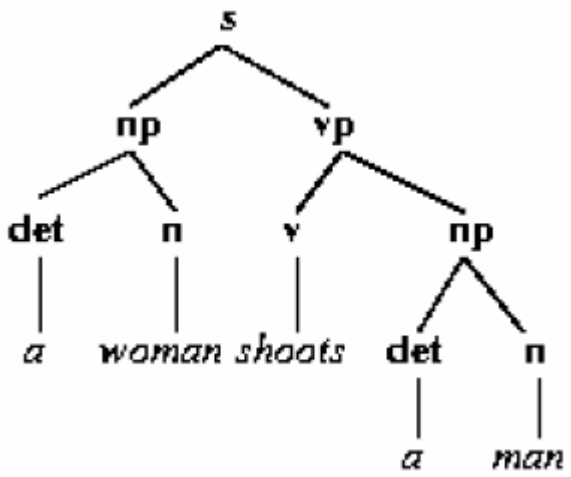
Ένα παράδειγμα ενός Context Free Grammar για ένα κομμάτι της αγγλικής γλώσσας είναι το εξής:

s ⇒ np vp
np ⇒ det n
vp ⇒ v np
vp ⇒ v
det ⇒ **a**
det ⇒ **the**
n ⇒ **woman**
n ⇒ **man**
v ⇒ **shoots**

Για να καταλάβουμε τα περιεχόμενα αυτής της μικρής γραμματικής, πρέπει πρώτα να καταλάβουμε τα σύμβολα. Υπάρχει το ->, το οποίο χρησιμοποιείται για να ορίζει τους κανόνες. Επίσης υπάρχουν και τα ακόλουθα σύμβολα: s, np, vp, det, n, v. Αυτά αποτελούν μη τερματικά σύμβολα και το καθένα από αυτά έχει την ερμηνεία του. Πιο συγκεκριμένα, το s είναι συντομογραφία για την αγγλική λέξη sentence (πρόταση), το np είναι συντομογραφία για την φράση noun phrase (φράση με ουσιαστικό), το vp είναι συντομογραφία για την φράση verb phrase (φράση με ρήμα) και το det είναι συντομογραφία για την λέξη determiner (άρθρο). Δηλαδή, καθ' ένα από αυτά τα σύμβολα αναφέρεται σε μία γραμματική κατηγορία. Επίσης έχουμε και τα εξής σύμβολα με το μαύρο χρώμα: a, the, woman, man, shoots. Αυτά τα σύμβολα αποτελούν τις λέξεις.

Το παραπάνω παράδειγμα γραμματικής περιέχει εννέα κανόνες. Ένας κανόνας ανεξάρτητος από τα συμφραζόμενα (context free rule) αποτελείται από ένα μη τερματικό σύμβολο, ακολουθούμενο από το ->, και έπειτα από μία πεπερασμένη ακολουθία χαρακτήρων, οι οποίοι μπορεί να είναι τερματικοί ή μη τερματικοί χαρακτήρες. Στο παράδειγμά μας έχουμε χτίσει εννέα γραμματικούς κανόνες. Το σύμβολο -> ερμηνεύεται ως *αποτελείται από* ή *παράγεται από*. Δηλαδή στο παράδειγμά μας ο πρώτος κανόνας ορίζει ότι μία πρόταση μπορεί να αποτελείται από μία noun phrase (φράση με ουσιαστικό), ακολουθούμενη από μία verb phrase (φράση με ρήμα). Ο τρίτος κανόνας ορίζει ότι μία φράση με ρήμα (verb phrase) μπορεί να αποτελείται από ένα ρήμα ακολουθούμενο από μία φράση με ουσιαστικό. Ο τέταρτος κανόνας ορίζει ότι μία φράση με ρήμα μπορεί επίσης να αποτελείται και από ένα ρήμα μόνο. Οι τελευταίοι πέντε κανόνες ορίζουν ότι το *a* και το *the* είναι άρθρα, ότι το *man* και το *woman* είναι ουσιαστικά και ότι το *shoots* είναι ρήμα.

Ας υποθέσουμε ότι έχουμε την εξής σειρά λέξεων: *a woman shoots a man*. Το παράδειγμά μας υποστηρίζει αυτή την σειρά λέξεων. Για να δούμε την δομή της πρότασης αυτής σχεδιάζουμε το ακόλουθο δέντρο.



Στην κορυφή έχουμε έναν κόμβο που ονομάζεται s. Αυτός ο κόμβος έχει δύο απολήξεις, η μία ονομάζεται np, και η άλλη vp. Μέχρι εδώ αυτά ορίζονται από τον πρώτο κανόνα. Κάθε κομμάτι στο εικονιζόμενο δέντρο περιγράφεται από τους κανόνες που είχαμε στο παράδειγμά μας. Για παράδειγμα οι δύο κόμβοι που σημειώνονται με np, εκφράζονται με τον

κανόνα που λέει ότι ένα np (noun phrase) μπορεί να αποτελείται από ένα det (determiner) ακολουθούμενο από ένα n(noun). Στην βάση του δέντρου, όλες οι λέξεις στην πρόταση *a woman shoots a man* εκφράζονται από έναν κανόνα. Πρέπει σε αυτό το σημείο να παρατηρήσουμε ότι τα τερματικά σύμβολα βρίσκονται στην βάση του δέντρου, ενώ τα μη-τερματικά σύμβολα βρίσκονται πιο ψηλά στο δέντρο.

Ένα τέτοιο δέντρο ονομάζεται parse tree (δέντρο που αναλύει γραμματικά λέξεις κειμένου) και μας δίνει δύο ειδών πληροφορίες: από την μια μεριά πληροφορίες για τα strings (τις συμβολοσειρές) και από την άλλη πληροφορίες για την δομή.

Για παράδειγμα έστω ότι έχουμε ένα string από λέξεις, και μία γραμματική, και βλέπουμε ότι μπορούμε να φτιάξουμε ένα parse tree όπως το παραπάνω (δηλαδή ένα δέντρο που θα έχει το s ως κορυφαίο κόμβο, και κάθε κόμβος στο δέντρο εκφράζεται από την γραμματική, και το string από λέξεις που έχουμε είναι διατεταγμένο στην σωστή σειρά μεταξύ των τερματικών κόμβων). Σε αυτήν την περίπτωση μπορούμε να πούμε ότι το string είναι γραμματικά (σε συμφωνία με την γραμματική μας) ορθό.

Από την άλλη μεριά, εάν δεν μπορούμε να φτιάξουμε ένα τέτοιο δέντρο, τότε το string δεν είναι γραμματικά ορθό (με βάση την δοθείσα γραμματική). Για παράδειγμα, το string *woman a woman man a shoots* δεν είναι γραμματικά ορθό με βάση την δοθείσα γραμματική (και με βάση οποιαδήποτε Αγγλική γραμματική). Η γλώσσα που γεννάται από μία γραμματική αποτελείται από όλα τα strings τα οποία η γραμματική αποδέχεται ως σωστά.

Για παράδειγμα, *a woman shoots a man* είναι μία φράση που ανήκει στην γραμματική μας, όπως είναι και η φράση *a man shoots the woman*. Ένας context free recognizer (ανιχνευτής φράσεων αδέσμευτες από τα συμφραζόμενα) είναι ένα πρόγραμμα που κατηγοριοποιεί τα strings ως έγκυρα ή μη έγκυρα γραμματικά.

Πολλές φορές, οι γλωσσολόγοι και οι επιστήμονες της πληροφορικής, δε ενδιαφέρονται μόνο για το εάν ένα string είναι γραμματικό ή μη, αλλά και γιατί είναι γραμματικό. Για να εκφραστούμε με περισσότερη ακρίβεια, επιθυμούμε να γνωρίζουμε τι δομή ακριβώς έχει, μία πληροφορία που μας την παρέχει λεπτομερώς το parse tree. Για παράδειγμα, το παραπάνω parse tree μας

δείχνει το πώς οι λέξεις στο string a woman shoots a man δένουν μεταξύ τους, κομμάτι-κομμάτι, για να σχηματίσουν την φράση. Αυτή η πληροφορία θα ήταν πολύ σημαντική εάν χρησιμοποιούσαμε αυτήν την πρόταση σε κάποια εφαρμογή και έπρεπε να δώσουμε την ερμηνεία της (δηλαδή σημασιολογικά). Ένα context free parser είναι ένα πρόγραμμα που αποφασίζει σωστά πότε ένα string ανήκει σε μια γλώσσα που προκύπτει από ένα context free grammar, καθώς επίσης μας πληροφορεί για το ποια είναι η δομή του. Δηλαδή, εκεί όπου ένας recognizer θα μας έλεγε εάν το string είναι γραμματικά ορθό ή όχι, ένας parser θα έχτιζε το συσχετιζόμενο parse tree.

2. Δυναμική προσθήκη κατηγορημάτων σε Prolog:

Ήρθε η στιγμή να αναφερθούμε στις διαδικασίες διαχείρισης του προγράμματος. Με τις διαδικασίες αυτές μπορούμε να διαχειριστούμε γεγονότα ή κανόνες του προγράμματος σαν όρους, να ενσωματώσουμε δυναμικά στο πρόγραμμα νέες προτάσεις, ή να διαγράψουμε από το πρόγραμμα γεγονότα ή κανόνες.

Στην κατηγορία αυτή ανήκουν οι ακόλουθες ενσωματωμένες διαδικασίες:

- asserta(X): Το γεγονός ή ο κανόνας X ενσωματώνεται στο πρόγραμμα πριν από τις προτάσεις του προγράμματος με το ίδιο κατηγόρημα με κεφαλή
- assertz(X): Το γεγονός ή ο κανόνας X ενσωματώνεται στο πρόγραμμα κάτω από τις προτάσεις του προγράμματος με το ίδιο κατηγόρημα με κεφαλή
- retract(X): Το πρώτο γεγονός ή κανόνας του προγράμματος που η κεφαλή του ταυτοποιείται με το X, διαγράφεται από το πρόγραμμα

Οι προτάσεις που εισάγονται στο πρόγραμμα με την βοήθεια των διαδικασιών αυτών συμπεριφέρονται με τον ίδιο ακριβώς τρόπο όπως και όλες οι άλλες προτάσεις του προγράμματος. Η Swi-Prolog –ή για συντομία Swipl- πάνω

στην οποία συντάχθηκε η παρούσα εργασία, απαιτεί πριν την χρησιμοποίηση του assert ή του retract να έχει δηλωθεί το predicate ως dynamic.

3. Η Αποκοπή:

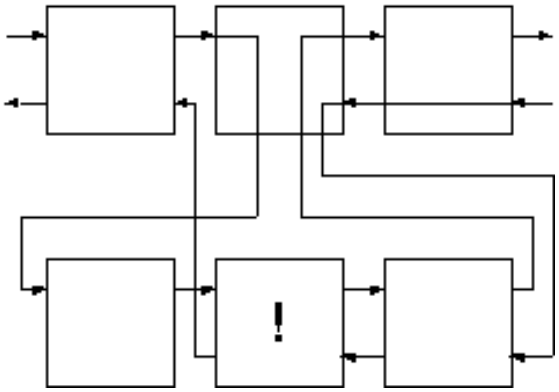
Η αυτόματη οπισθοδρόμηση είναι ένα από τα χαρακτηριστικότερα χαρακτηριστικά γνωρίσματα της Prolog. Αλλά οπισθοδρομώντας η Prolog μπορεί να οδηγηθεί σε μη αποτελεσματικότητα. Μερικές φορές η Prolog μπορεί να σπαταλήσει χρόνο διερευνώντας δυνατότητες που μπορεί να μην οδηγούν πουθενά. Θα ήταν ευχάριστο να υπάρχει κάποιος τρόπος ελέγχου αυτής της πτυχής της συμπεριφοράς της πέρα από τους ακόλουθους δύο:

- αλλαγή της σειράς των κανόνων και
- αλλαγή της σειράς των προτάσεων στο σώμα των κανόνων.

Ένας άλλος τρόπος πράγματι υφίσταται, υπάρχει μία ενσωματωμένη λειτουργία στην Prolog (με σύμβολο το θαυμαστικό) ! , που ονομάζεται αποκοπή. Η αποκοπή προσφέρει έναν πιο άμεσο τρόπο για να ελέγχουμε το πώς ψάχνει η Prolog τις λύσεις.
Τι ακριβώς είναι η αποκοπή και τι κάνει; Είναι απλά ένα ειδικό άτομο (atom) που μπορούμε να χρησιμοποιήσουμε όταν γράφουμε προτάσεις. Παραδείγματος χάριν,

```
p(X) :- b(X),c(X),!,d(X),e(X).
```

έχουμε τον παραπάνω κανόνα σε Prolog. Όσον αφορά την λειτουργία της αποκοπής, καταρχήν, είναι ένας στόχος που πάντα πετυχαίνει. Δεύτερον, και το πιο σημαντικό, έχει μια παρενέργεια. Υποθέστε ότι κάποιος στόχος χρησιμοποιεί αυτήν την πρόταση (καλούμε αυτόν τον στόχο parent goal). Κατόπιν η αποκοπή δεσμεύει την Prolog σε οποιεσδήποτε επιλογές είχε κάνει από την στιγμή που το parent goal ενοποιήθηκε με την αριστερή πλευρά του κανόνα.



Η λειτουργία της αποκοπής.

Εξετάζουμε ένα παράδειγμα για να δούμε τι σημαίνει αυτό.

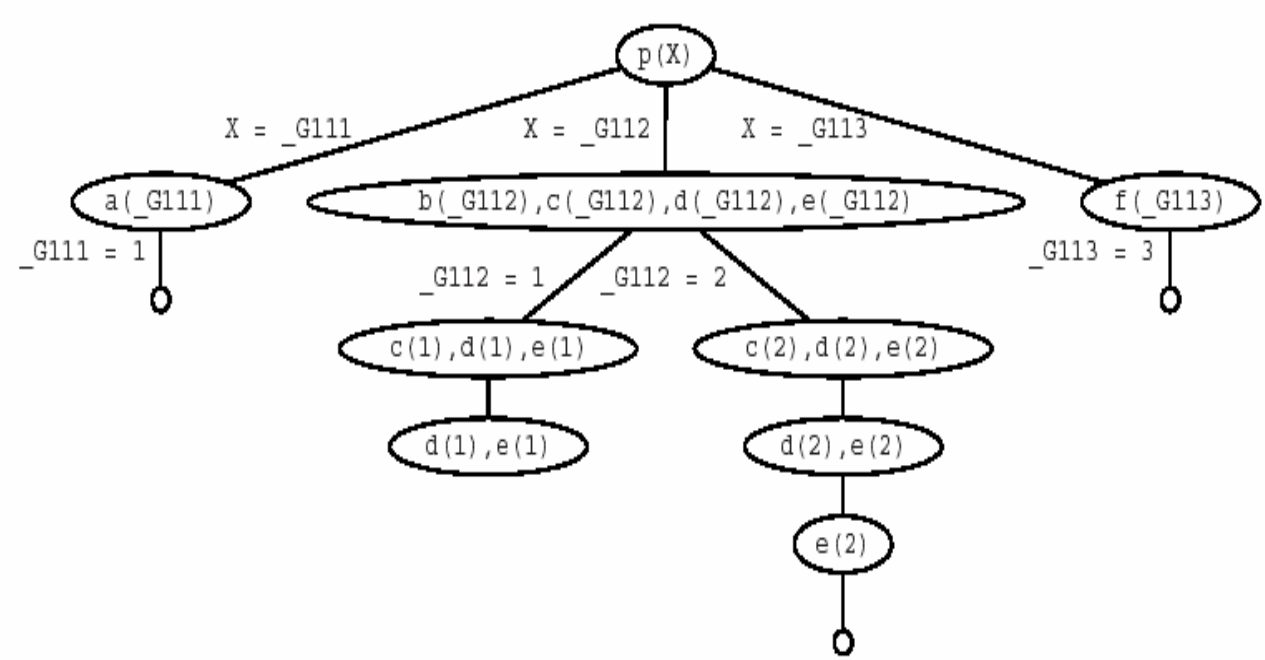
Αρχικά εξετάζουμε το ακόλουθο κομμάτι κώδικα το οποίο δεν έχει αποκοπή:

```
p(X) :- a(X).
p(X) :- b(X),c(X),d(X),e(X).
p(X) :- f(X).
a(1).
b(1).
c(1).
b(2).
c(2).
d(2).
e(2).
f(3).
```

Εάν θέσουμε την ερώτηση p(X) θα πάρουμε τις ακόλουθες απαντήσεις:

```
X = 1 ;
X = 2 ;
X = 3 ;
no
```

Εδώ είναι το δέντρο αναζήτησης που εξηγεί πώς η Prolog βρίσκει αυτές τις τρεις λύσεις. Πρέπει να οπισθοδρομήσει μιά φορά, κάτι που γίνεται όταν ο στόχος ενώνεται με την δεύτερη πρόταση για p/1 και αποφασίζει για να ταιριάξει τον πρώτο στόχο με το b(1) αντί με το b(2).



Αλλά τώρα παρεμβάλλουμε μια αποκοπή στη δεύτερη πρόταση:

p(X) :- b(X),c(X),!,d(X),e(X).

Εάν θέσουμε τώρα την ερώτηση p(X) θα πάρουμε τις ακόλουθες απαντήσεις:

X = 1 ;

no

Βγάζουμε τα ακόλουθα συμπεράσματα:

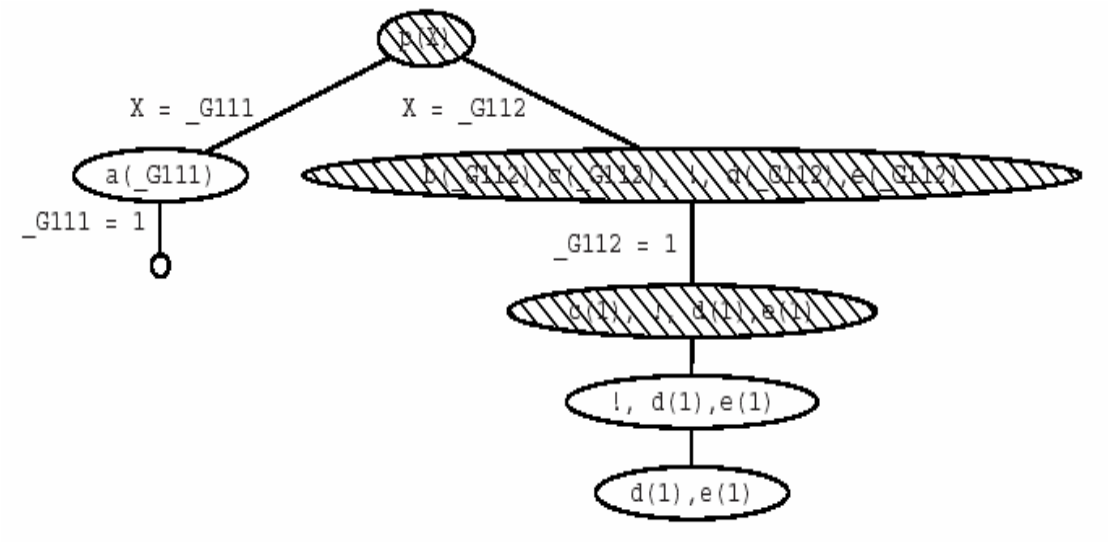
1. Το p(X) αρχικά αντιστοιχείται με τον πρώτο κανόνα, έτσι παίρνουμε έναν νέο στόχο, τον a(X). Αντιστοιχίζοντας το X με το 1, η Prolog αντιστοιχεί το a(X) με το γεγονός a(1) και έχουμε βρεί μια λύση. Μέχρι τώρα, αυτό ακριβώς συνέβη στην πρώτη έκδοση του προγράμματος.

2. Συνεχίζουμε έπειτα και ψάχνουμε μια δεύτερη λύση. Το p(X) αντιστοιχείται με τον δεύτερον κανόνα, έτσι παίρνουμε τους νέους στόχους b(X),c(X),!,d(X),e(X). Αντιστοιχίζοντας το X με το 1, η Prolog αντιστοιχεί το b(X) με το γεγονός b(1), έτσι έχουμε τώρα τους στόχους c(1),!,d(1),e(1). Αλλά το c(1) είναι στη βάση δεδομένων έτσι συνεχίζουμε στο !,d(1),e(1).

3. Τώρα για τη μεγάλη αλλαγή. Με την αποκοπή ! ο στόχος μας πετυχαίνει (όπως πάντα) και δεσμεύει όλες τις επιλογές έχουμε κάνει μέχρι τώρα. Ειδικότερα, είμαστε δεσμευμένοι στο ότι ισχύει X = 1, και είμαστε επίσης δεσμευμένοι στη χρησιμοποίηση του δεύτερου κανόνα.

4. Αλλά το d(1) αποτυγχάνει. Και δεν υπάρχει κανένας τρόπος με τον οποίο να μπορούμε να ξαναικανοποιήσουμε τον στόχο p(X). Βεβαίως, εάν μας επιτρεπόταν να δοκιμάσουμε την ισότητα X=2 θα μπορούσαμε να χρησιμοποιήσουμε το δεύτερο κανόνα για να παραγάγουμε μια λύση (ότι συνέβη στην αρχική έκδοση του προγράμματος). Αλλά εμείς δεν μπορούμε να το κάνουμε αυτό: η αποκοπή μας έχει δεσμεύσει στην επιλογή X=1. Και, βεβαίως εάν μπορούσαμε να δοκιμάσουμε τον τρίτο κανόνα, θα μπορούσαμε να παραγάγουμε τη λύση X=3. Αλλά δεν μπορούμε να κάνουμε αυτό: η περικοπή μας έχει δεσμεύσει στη χρησιμοποίηση του δεύτερου κανόνα.

Εξετάζοντας το δέντρο αναζήτησης, αυτό σημαίνει ότι η αναζήτηση σταματά όταν ο στόχος d(1) δεν μπορεί να εμφανιστεί καθώς ανεβαίνοντας το δέντρο δεν οδηγούμαστε σε οποιοδήποτε κόμβο όπου μια εναλλακτική επιλογή είναι διαθέσιμη. Οι μαρκαρισμένοι κόμβοι στο δέντρο όλοι εμποδίζονται για την οπισθοδρόμηση λόγω της αποκοπής.



Πρέπει να δώσουμε προσοχή σε ένα σημείο: η αποκοπή μας δεσμεύει μόνο στις επιλογές που γίνονται από την στιγμή που ο parental(=πρωταρχικός) στόχος ενοποιήθηκε με την αριστερή πλευρά του κανόνα που περιέχει την αποκοπή. Για παράδειγμα, σε έναν κανόνα της μορφής

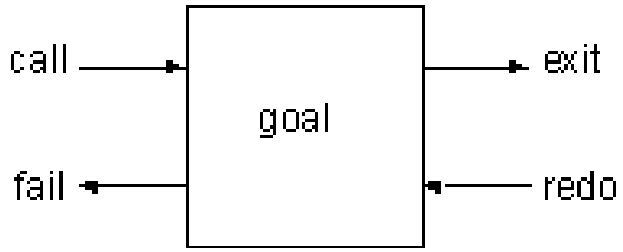
q :- p1,...,pn,!,r1,...,rm

μόλις φθάσουμε στην αποκοπή, μας δεσμεύει στη χρησιμοποίηση αυτής της ιδιαίτερης πρότασης για το q και μας δεσμεύει στις επιλογές που έγιναν κατά τον υπολογισμό των p1,...,pn. Εντούτοις, είμαστε ελεύθεροι να οπισθοδρομήσουμε μεταξύ των r1,...,rm και είμαστε επίσης ελεύθεροι να οπισθοδρομήσουμε μεταξύ των εναλλακτικών λύσεων για τις επιλογές που έγιναν πρίν φτάσουμε τον στόχο q.
(Αναφορά:5.1,5.2,5.3 **Learn Prolog Now!** Patrick Blackburn, Johan Bos, Kristina Striegnitz, **Adventure In Prolog**)

4. Εσωτερική Δομή των στόχων της Prolog:

Ένας στόχος στην Prolog έχει τέσσερις **θύρες** που αντιπροσωπεύουν τη ροή του ελέγχου μέσω του στόχου: **call, exit, redo** και **fail**. Πρώτα ο στόχος καλείται (θύρα call). Εάν επιτυχής γίνεται έξοδος (exit). Εάν όχι αποτυγχάνει (fail). Εάν ο στόχος ξαναδοκιμάζεται τότε διέρχεται από την θύρα redo, με την

είσοδο ενός ερωτηματικού (;). Τα σχήματα στην παρακάτω εικόνα παρουσιάζουν το στόχο και τις θύρες του.



Οι θύρες ενός στόχου Prolog

Οι συμπεριφορές σε κάθε θύρα είναι :

call

Αναζητεί τις προτάσεις που ταιριάζουν με το στόχο

exit

Δείχνει ότι ο στόχος έχει ικανοποιηθεί, θέτει έναν δείκτη θέσης στην πρόταση και δεσμεύει τις μεταβλητές κατάλληλα

redo

Ξαναδοκιμάζει το στόχο, αποδεσμεύει τις μεταβλητές και επαναλαμβάνει την αναζήτηση ξεκινώντας από των δείκτη θέσης

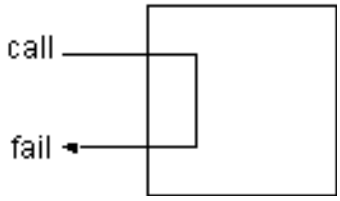
fail

Δείχνει ότι δεν υπάρχουν άλλες προτάσεις που να ικανοποιούν τον στόχο

5. Η Άρνηση:

Υπάρχουν ενσωματωμένα predicates που έχουν επιπτώσεις στην οπισθοδρόμηση. Ένα από αυτά είναι το **fail**/0, και, όπως το όνομά του υπονοεί, αποτυγχάνει πάντα.

Εάν το fail/0 (το /0 σημαίνει ότι δεν δέχεται κανένα όρισμα) καλεστεί από τα αριστερά, αμέσως δίνει τον έλεγχο στην θύρα redo του στόχου στα αριστερά. Δεν καλείται ποτέ από τα δεξιά, από την στιγμή που ποτέ δεν αφήνει την ροή να περάσει δεξιά.



Εσωτερική ροή του ελέγχου του fail/0 predicate
(Αναφορά:5.4,5.5 **Adventure in Prolog**
http://www.amzi.com/AdventureInProlog/advfrtop.htm)

6.Παραδείγματα εφαρμογών επεξεργασίας φυσικής γλώσσας με Prolog:

Ένα παράδειγμα που εξομοιώνει την μορφολογία της Αγγλικής γλώσσας βρίσκεται στην διεύθυνση

http://inertia.curvedspaces.com/Projects/NLP/Morphology/morphology.zip

Με το πρόγραμμα αυτό προσθέτουμε καταλήξεις στις λέξεις, ή τις μετατρέπουμε από τον ενικό στον πληθυντικό.

Π.χ.

```
?- gen_morph(dog,s,Word).
Word = dogs

?- gen_morph(peel,ing,Word).
Word = peeling
```

Επίσης, ένα άλλο παράδειγμα που προέρχεται από την ίδια σελίδα, (προέρχεται από τον ακόλουθο σύνδεσμο)

http://inertia.curvedspaces.com/Projects/NLP/DCG/dcg.zip

το παράδειγμα αυτό δημιουργεί προτάσεις (με ήδη δοθέντα στο πρόγραμμα άρθρα και ουσιαστικά) που αφορούν πτήσεις αεροπλάνων:

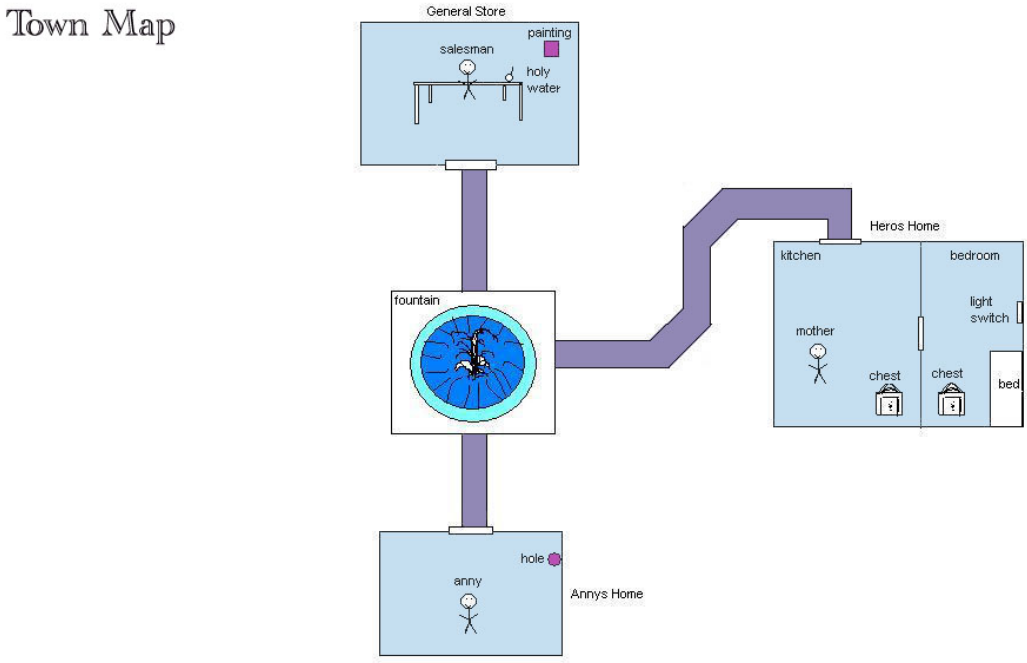
```
1: s
[np, [det, the], [n, passenger]]
[vp, [v, boarded], [np, [det, the], [n, plane], [pp, [p, without], [np, [det, a], [n, ticket]]]]]

12: s
[np, [det, the], [n, flight], [pp, [p, to], [np, [pn, Adelaide]]]]
[vp, [v, had], [adjp, [adjs, [adj, no], [adjs, [adj, empty]]], [n, seats]]]

Sentence 12: Number of parses = 1
```

Επίσης, στην ακόλουθη ιστοσελίδα (http://www.cs.cofc.edu/%7Emanaris/ai-education-repository/nlp-tools.html) υπάρχει ένα πρόγραμμα, το SAX (τα αρχικά προκύπτουν από τις λέξεις Sequential Analyzer for syntaX and semantics). Το πρόγραμμα κατεβαίνει με ftp από το ακόλουθο link: ftp://clr.nmsu.edu/CLR/tools/ling-analysis/syntax/SAX/ . Λειτουργία του προγράμματος αποτελεί η συντακτική ανάλυση των DCG's.

Δ. Περιγραφή του παιχνιδιού

1. Ιστορία του παιχνιδιού

Το παιχνίδι διαδραματίζεται το 800 μ.Χ. στο Innsmouth της Νέας Αγγλίας. Είναι ένα μικρό επαρχιακό χωριό, όπου όλα κυλούν ευχάριστα και όμορφα. Το παιχνίδι έχει τρείς περιοχές, την πόλη, το σκοτεινό δάσος και την βαθιά θάλασσα. Ο χρήστης υποδύεται έναν ήρωα, ο οποίος ονομάζεται Tim. Η υπόθεση ξεκινά από το σπίτι του ήρωα, συγκεκριμένα από το υπνοδωμάτιό του. Παρακάτω δίνεται ο χάρτης της πόλης.



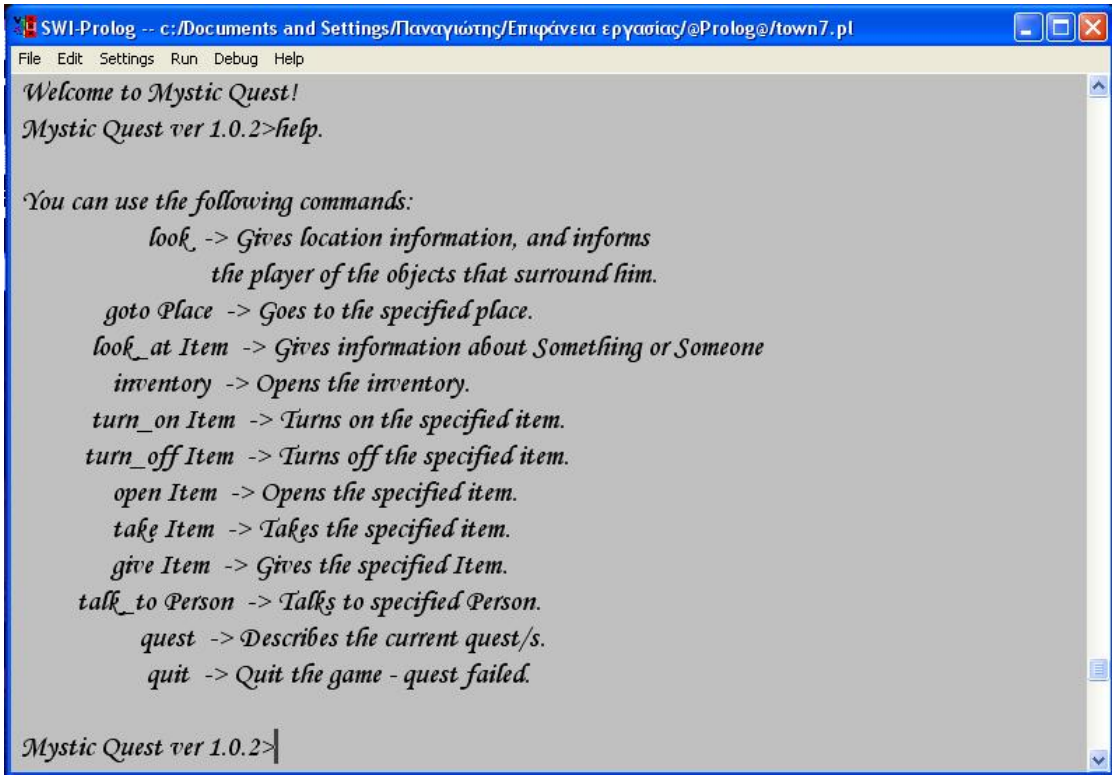
Δεξιά βρίσκεται το σπίτι του ήρωα, στον βορρά βρίσκεται το κεντρικό μαγαζί της πόλης, και στο νότο το σπίτι της Anny's. Στο κέντρο της πόλης βρίσκεται το συντριβάνι. Στον χάρτη εικονίζονται τα συμμετέχοντα πρόσωπα εις το παιχνίδι, καθώς επίσης και τα διάφορα αντικείμενα που υπάρχουν σε κάθε χώρο (π.χ. σεντούκια κ.α.). Ο χάρτης αυτός φτιάχτηκε για να διευκολύνει τον χρήστη στην νοερή απεικόνιση του περιβάλλοντος, καθότι το παιχνίδι διαδραματίζεται σε κονσόλα.

Κατά την διάρκεια του παιχνιδιού η υπόθεση εξελίσσεται και ο ήρωας αναλαμβάνει να εκπληρώσει διάφορες αποστολές. Το παιχνίδι ονομάζεται Mystic Quest – Μυστική Αναζήτηση, και η βασική αποστολή του παιχνιδιού είναι να εντοπίσει ο ήρωας την αδερφή του, Daphne. Για να εκπληρώσει τον σκοπό αυτό, θα πρέπει να καταφέρει να λύσει τους διάφορους γρίφους που θα συναντάει κατά την διάρκεια του παιχνιδιού. Οι γρίφοι αυτοί περιλαμβάνουν την χρησιμοποίηση αντικειμένων που μπορεί να κουβαλά ο χρήστης, την επίσκεψη περιοχών και τις συνομιλίες με τους NPC's (Non Player Characters), δηλαδή τους παίκτες που δεν τους ελέγχει ο χρήστης και κατευθύνονται από τον Η.Υ.

2. Γιατί το παιχνίδι συνδέεται με την επεξεργασία φυσικής γλώσσας

Το παιχνίδι είναι άρρηκτα συνδεδεμένο με την επεξεργασία φυσικής γλώσσας, γιατί ο χρήστης καλείται να γράφει προτάσεις (απλές μεν, ουσιώδεις δε) για να εξελίσσεται το παιχνίδι. Οι προτάσεις αυτές αποτελούν και τις εντολές του παιχνιδιού. Ένα παράδειγμα τέτοιας πρότασης είναι η προστακτική εντολή «πάρε αυτό το αντικείμενο» ή «κοίταξε γύρω σου» (take an object ή look, αντίστοιχα) κτλ.

Αν ο χρήστης γράψει μία εντολή που αφορά ένα αντικείμενο που δεν υπάρχει, τότε το πρόγραμμα απαντά ανάλογα (έλεγχος λαθών εισόδου), δηλαδή ο χρήστης καλείται να δίνει έμφαση στο περιβάλλον του και να θέτει σωστές προτάσεις. Οι προτάσεις που είναι διαθέσιμες στον χρήστη εμφανίζονται με την εντολή help. Ένα δείγμα εκτέλεσης της εντολής αυτής φαίνεται στην παρακάτω εικόνα:



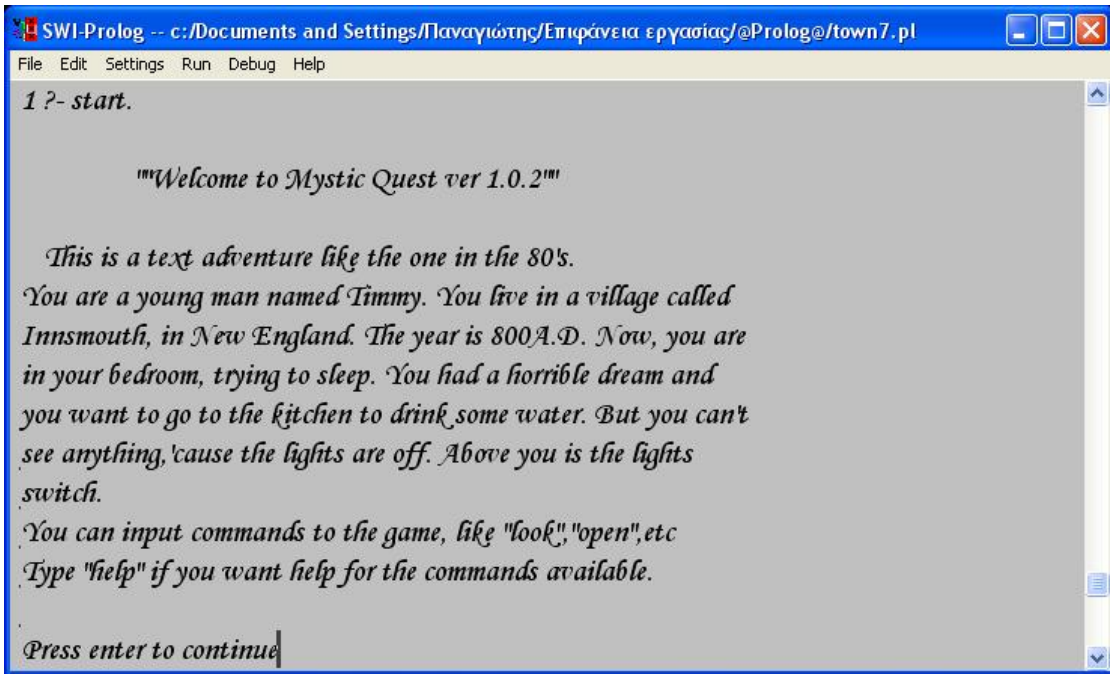
Όπως βλέπουμε οι διαθέσιμες εντολές και η λειτουργεία τους είναι η εξής:

- look – περιγράφει τον χώρο στον οποίο κάθε στιγμή βρισκόμαστε, μας πληροφορεί για τους ανθρώπους που μπορεί να βρίσκονται γύρω κτλ
- goto Place – πήγαινε σε ένα μέρος. Μετακινεί τον ήρωα και αλλάζει την δυναμική βάση του προγράμματος
- look_at Item – περιγράφει κάποιο αντικείμενο
- inventory – περιγράφει τα αντικείμενα που κάθε φορά κουβαλάει μαζί του ο ήρωας
- turn on/off Item – ανοίγει/κλείνει αντικείμενα όπως διακόπτες
- open Item – ανοίγει κάποιο αντικείμενο
- take Item – παίρνει το αντικείμενο, με επίδραση φυσικά στην δυναμική βάση
- give Item – δίνει το αντικείμενο σε όποιον βρίσκεται μπροστά του κάθε στιγμή
- talk_to Person – μιλάει με κάποιον άνθρωπο
- quest – περιγράφει την τρέχουσα αποστολή του ήρωα
- quit – τερματίζει το παιχνίδι

Ένα τέτοιο παιχνίδι μπορεί να χρησιμοποιηθεί για την εκμάθηση μιάς ξένης γλώσσας σε ένα φροντιστήριο. Αυτό συμβαίνει γιατί για την εξέλιξη του παιχνιδιού απαιτείται η κατανόηση των κειμένων. Επίσης, με την ύπαρξη των γρίφων χρειάζεται η διανοητική διεργασία του χρήστη. Λόγω του ότι το πρόγραμμα αυτό είναι παιχνίδι, οι μαθητές θα δείχνουν περισσότερο ενδιαφέρον από ότι θα έδειχναν σε ένα απλό πρόγραμμα. Η υπόθεση φροντίζει δε, να κρατά το ενδιαφέρον αμείωτο.

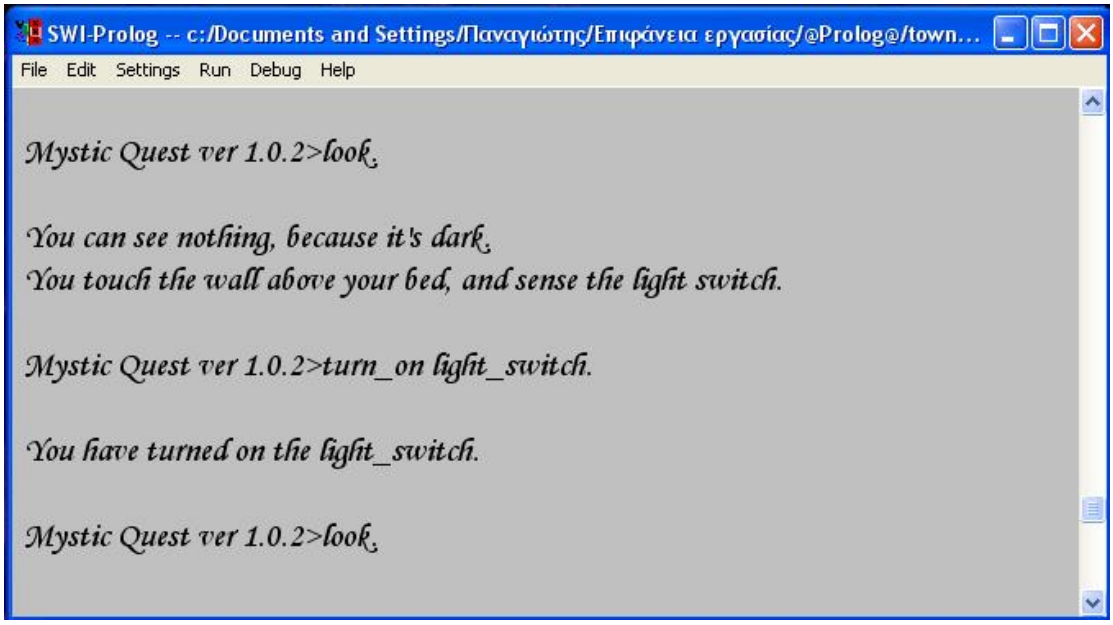
3. Παραδείγματα από την εξέλιξη του παιχνιδιού

Για να αποσαφηνιστεί η χρησιμοποίηση του παιχνιδιού (δηλαδή το user input), χρειάζεται να δώσουμε μερικά παραδείγματα. Το παιχνίδι ξεκινά με την εντολή start. Όταν ο χρήστης (αφού έχει κάνει ήδη consult το αρχείο) πληκτρολογήσει την εντολή start, θα δει την ακόλουθη εικόνα:

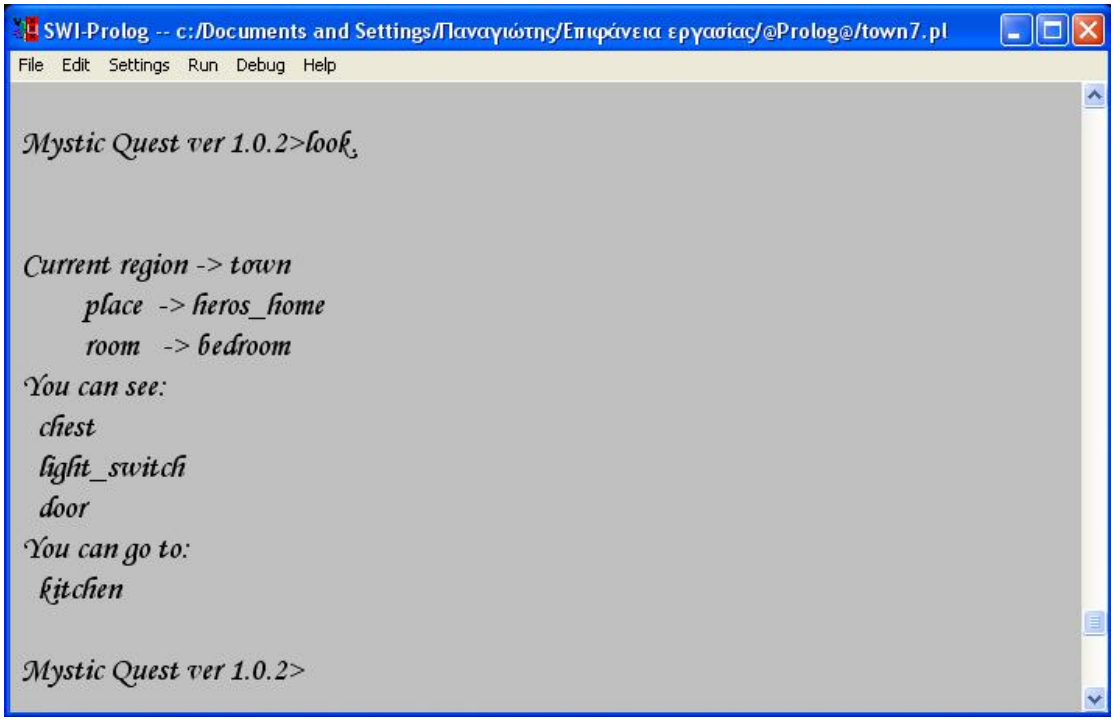


Η εικόνα αυτή αποτελεί την εισαγωγή του παιχνιδιού, μας πληροφορεί για το πότε και για το πού βρισκόμαστε. Επίσης, μας πληροφορεί ότι με την εντολή help βλέπουμε τις διαθέσιμες εντολές.

Αν ο χρήστης πληκτρολογήσει look, θα διαπιστώσει ότι δεν μπορεί να κοιτάξει γύρω του, γιατί τα φώτα είναι σβηστά. Μπορεί, όμως να ανοίξει τον διακόπτη με τα φώτα. Αυτός είναι και ο πρώτος γρίφος του παιχνιδιού.



Αφού ανοίξει τα φώτα, μπορεί να κοιτάξει γύρω του. Βλέπει λοιπόν τα εξής:



Η εντολή look λοιπόν μας πληροφορεί για το που βρισκόμαστε, ποια αντικείμενα υπάρχουν γύρω μας και που μπορούμε να πάμε. Κατά την πορεία

του παιχνιδιού ο ήρωας καλείται να χρησιμοποιήσει διάφορες εντολές. Η συνέχεια επί της οθόνης..!!

Πρέπει σε αυτό το σημείο να σημειώσουμε ότι το παιχνίδι χρησιμοποιεί μια δυναμική βάση, η οποία κρατά στοιχεία όπως η τρέχουσα θέση του ήρωα, τα αντικείμενα που κρατά, και άλλα χρήσιμα στο παιχνίδι στοιχεία. Επίσης, λέξεις που ξεκινάν με αριθμούς, πρέπει να συμπεριλαμβάνονται μέσα σε μονά εισαγωγικά, π.χ. ‘2000’.

4. Προτάσεις για εξέλιξη

Το παιχνίδι έχει φυσικά πολλά περιθώρια εξέλιξης. Μπορούν να προστεθούν περισσότερες εντολές, καθώς επίσης και μεγαλύτερες προτάσεις, οι οποίες μπορούν να έχουν και ορθογραφικό-συντακτικό έλεγχο. Μπορεί επίσης το πρόγραμμα να προτείνει και διορθώσεις συντακτικού περιεχομένου. Αυτό θα καθιστούσε πιο εύκολη την εκμάθηση μιας ξένης γλώσσας και των γραμματικών κανόνων που την διέπουν.

ΠΑΡΑΡΤΗΜΑ

<u>Ο κώδικας του παιχνιδιού</u>

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% We start our game be specifying the three regions it will have
%% These are "Town","Darkwood","Deepsea". The town is Innsmouth,
%% located in New England. It is year 800 A.D. There are four houses
%% in the town. North of town lies the Darkwood forest, a terrifying
%% place which noone likes to visit. South of town lies the Deepsea,
%% a sea with a very strange past.
%% We define the region\1 predicate, with each region as argument

region(town).
region(darkwood).
region(deepsea).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% As written above, the town has three houses-buildings:
%% -The Hero's Home, -Anny's Home, -A General Store.
%% The house\1 predicate is defined with each house as
%% argument

house(heros_home).
house(annys_home).
house(general_store).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Start is the predicate that must be run at start (logical). The
%% start\0 predicate describes the current version of the game, and
```

```
%% tries to inform the player about the hypothesis of the game. It
%% also tries to give a first hint (about the light switch) to guide
%% the player. One useful action (maybe the most useful) of start is
%% that it initialises the dynamic database, a database which is
% needed the most during the game, affecting its flow.

start:-nl,
      tab(15),
      write("""Welcome to Mystic Quest ver 1.0.2"""),
      nl,
      nl,
      tab(3),
      write('This is a text adventure like the one in the 80''s.'),
      nl,
      write('You are a young man named Timmy. You live in a village called '),
      nl,
      write('Innsmouth, in New England. The year is 800A.D. Now, you are '),
      nl,
      write('in your bedroom, trying to sleep. You had a horrible dream and '),
      nl,
      write('you want to go to the kitchen to drink some water. But you can''t'),
      nl,
      write('see anything,''cause the lights are off. Above you is the lights'),
      nl,
      write('switch.'),
      nl,
      write('You can input commands to the game, like "look","open",etc'),
      nl,
      write('Type "help" if you want help for the commands available.'),
      nl,
      nl,
      write('Press enter to continue'),
      get0(_),
      nl,
```

```
    initialise_dynamic_data,
    command_loop.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% We define the dynamic predicates, by calling the in-built dynamic
%% function, followed by the predicates, having as argument the
%% anonymous variable, because we simply do not need to name one here.

:-dynamic(current_region(_)).
:-dynamic(current_place(_)).
:-dynamic(current_room(_)).
:-dynamic(quest_state(_)).
:-dynamic(object(_,_,_,_)).
:-dynamic(stable_object(_,_,_,_)).
:-dynamic(person(_,_,_,_)).
:-dynamic(item_in_inventory(_)).
:-dynamic(door(_,_,_,_,_,_,_,_)).
:-dynamic(closed(_,_,_,_)).
:-dynamic(opened(_,_,_,_)).
:-dynamic(in_object(_,_,_,_,_)).
:-dynamic(turned_off(_)).
:-dynamic(turned_on(_)).
:-dynamic(painting_state(_)).
:-dynamic(holy_water_state(_)).
:-dynamic(anny_speech_state(_)).
:-dynamic(map_state(_)).
:-dynamic(monk_speech_state(_)).
:-dynamic(can_go_darkwood(_)).
:-dynamic(elder_speech_state(_)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The initialise_dynamic_data predicate asserts the following predicates
```

```
%% into the dynamic database. Notice that all of the following predicates
%% should be defined as dynamic above.

initialise_dynamic_data:-
    asserta(current_region(town)),
    asserta(current_place(heros_home)),
    asserta(current_room(bedroom)),
    asserta(quest_state(0)),
    asserta(stable_object(light_switch,town,heros_home,bedroom)),
    asserta(stable_object(chest,town,heros_home,bedroom)),
    asserta(closed(chest,town,heros_home,bedroom)),
    asserta(turned_off(light_switch)),
    asserta(in_object(chest,town,heros_home,bedroom,dagger)),
    asserta(in_object(chest,town,heros_home,bedroom,'100GP')),

asserta(door(door,town,heros_home,bedroom,town,heros_home,kitchen,clos
ed)),
    % The next refer to kitchen.
    asserta(stable_object(chest,town,heros_home,kitchen)),
    asserta(closed(chest,town,heros_home,kitchen)),
    asserta(in_object(chest,town,heros_home,kitchen,apple)),
    asserta(person(mother,town,heros_home,kitchen)),

asserta(door(door_to_town,town,heros_home,kitchen,town,town,town,closed)
),
    % The next refer to town.
    asserta(stable_object(fountain,town,town,town)),
    % The next refer to general store
    asserta(object(painting,town,general_store,general_store)),
    asserta(painting_state(0)),
    asserta(holy_water_state(0)),
    asserta(person(salesman,town,general_store,general_store)),
    asserta(object(holy_water,town,general_store,general_store)),
```

```
asserta(door(door_to_town,town,general_store,general_store,town,town,town
,opened)),
    % The next refer to annys_home
    asserta(map_state(0)),
    asserta(person(anny,town,annys_home,annys_home)),
    asserta(anny_speech_state(0)),
    asserta(stable_object(hole,town,annys_home,annys_home)),

asserta(door(door_to_town,town,annys_home,annys_home,town,town,town,o
pened)),
    % The next refer to deepsea
    asserta(monk_speech_state(0)),
    asserta(person(monk,deepsea,deepsea,deepsea)),
    % The next refer to darkwood
    asserta(can_go_darkwood(0)),
    asserta(elder_speech_state(0)),
    asserta(person(elder,darkwood,darkwood,darkwood)),
    asserta(stable_object(parrot,darkwood,darkwood,darkwood)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%% We define the operators, so that the user can use a more friendly
environment

:-op(1200,fx,open).
:-op(1200,fx,look_at).
:-op(1200,fx,turn_on).
:-op(1200,fx,turn_off).
:-op(1200,fx,take).
:-op(1200,fx,goto).
:-op(1200,fx,talk_to).
:-op(1200,fx,give).
```

```
:-op(1200,xfx,to). % We will see if we use this

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%% The main control loop

command_loop:-write('Welcome to Mystic Quest!'),
        nl,
        repeat,
        write('Mystic Quest ver 1.0.2>'),
        read(Command),
        do(Command),
        nl,
        end_condition(Command).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%% The list of the commands which are acceptable by the game engine,
entered
%% by the main command control loop

do(goto(Someroom)):-goto(Someroom),!.
do(look):-look,!.
do(look_at(Something)):-look_at(Something),!.
do(inventory):-inventory,!.
do(turn_on(Item)):-turn_on(Item),!.
do(turn_off(Item)):-turn_off(Item),!.
do(open(Item)):-open(Item),!.
do(take(Item)):-take(Item),!.
do(give(Item)):-give(Item),!.
do(talk_to(Person)):-talk_to(Person),!.
do(quest):-quest,!.
```

```
do(help):-help,!.
do(quit):-delay(1),write('Quitting..'),!.
do(_):-write('Invalid command. View help.'),nl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%% Defines the end condition/s.

end_condition(quit).
end_condition(end).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%% We define the look at item predicate, which gives us information
%% about the item which will be the argument. It is defined many
%% times, to react differently on different items. The last definition
%% with the anonymous variable, ensures us that the goal always succeeds

look_at(Item):-current_region(Region),
        current_place(Place),
        current_room(Room),
        stable_object(Item,Region,Place,Room),
        item_properties(Item,Region,Place,Room).

look_at(Item):-current_region(Region),
        current_place(Place),
        current_room(Room),
        object(Item,Region,Place,Room),
        item_properties(Item,Region,Place,Room).

look_at(Person):-current_region(Region),
          current_place(Place),
```

```
                current_room(Room),
                person(Person,Region,Place,Room),
                item_properties(Person,Region,Place,Room).

look_at(Door):-current_region(Region),          %State represents whether
            current_place(Place),           %a door is opened or closed.
            current_room(Room),
            door(Door,Region,Place,Room,Region2,Place2,Room2,State),

item_properties(Door,Region,Place,Room,Region2,Place2,Room2,State).

look_at(Door):-current_region(Region),          %look at door re-written,
            current_place(Place),           %to establish connection
            current_room(Room),             %between the rooms that
            door(Door,Region,Place,Room2,Region2,Place2,Room,State),
%the door connects

item_properties(Door,Region,Place,Room,Region2,Place2,Room2,State).

look_at(Item):-nl,
            write('There is no '),
            write(Item),
            write(' here.'),
          nl.

look_at(_).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The item properties pred. gives us info on each item

item_properties(Door,Region,Place,Room,Region2,Place2,Room2,State):-
State=closed, %Must be re-written
```

```
                                        nl,             %incase the place
changes
                                write('You can see a door connecting '),
                                        write(Room),
                                        write(' with '),
                                        write(Room2),
                                        write(', but it is closed.'),
                                nl.

item_properties(Door,Region,Place,Room,Region2,Place2,Room2,State):-
State=opened, %Must be re-written
                                        nl,             %incase the place
changes
                                write('You can see a door connecting '),
                                        write(Room),
                                        write(' with '),
                                        write(Room2),
                                        write(', and it is opened.'),
                                nl.

item_properties(Item,Region,Place,Room):-Item=light_switch,
                                turned_off(light_switch),
                                nl,
                                write('The switch is within your reachable area,'),
                                write(' and it is turned off.'),
                                nl.

item_properties(Item,Region,Place,Room):-Item=light_switch,
                                turned_on(light_switch),
                                nl,
                                write('The switch is turned on.'),
                                nl.

item_properties(Person,Region,Place,Room):-Person=mother,
```

```
                    nl,
                    write('You can see your mother.'),
                    nl,
                    write('She seems she wants to talk to you.'),
                    nl.

item_properties(Item,Region,Place,Room):-Item=fountain,
                        nl,
                    write('You see a wondrous fountain '),
                    write('in the middle of the town square.'),
                    nl,
                    write('The water relaxes you with its delight.'),
                        nl.

item_properties(Item,Region,Place,Room):-Item=painting,
                        painting_state(0),
                        nl,
                        write('An image of a fairy riding a horse.'),
                    nl,
                        write('Strange,but you feel that you have
seen '),
                        nl,
                    write('this painting before..'),
                        nl,
                        retract(painting_state(0)), % The hero has
seen the
                        assert(painting_state(1)).  % painting

item_properties(Item,Region,Place,Room):-Item=painting,
                        painting_state(1),
                        nl,
                        write('An image of a fairy riding a horse.'),
                    nl,
```

```
                write('Strange,but you feel that you have
seen '),
                nl,
        write('this painting before..'),
                nl.

item_properties(Item,Region,Place,Room):-Item=painting,
                painting_state(2),
                nl,
                write('An image of a fairy riding a horse.'),
        nl,
                write('Strange,but you feel that you have
seen '),
                nl,
        write('this painting before..'),
                nl.

item_properties(Item,Region,Place,Room):-Item=holy_water,
                nl,
        write('You see a bottle full of Holy water.'),
        nl,
        write('It''s price is 100 Gold Pieces.'),
        nl.

item_properties(Person,Region,Place,Room):-Person=salesman,
          nl,
          write('You can see the old man warming his hands
'),
          nl,
                write('above the fireplace. He seems
excited of '),
                nl,
          write('your visit.'),
          nl.
```

```
item_properties(Person,Region,Place,Room):-Person=anny,
                                nl,
                                write('Anny seems very happy of your visit, '),
                                nl,
                                        write('but she is nervous about
something.'),
                                        nl.

item_properties(Item,Region,Place,Room):-
        Item=hole,
          nl,
          write(' You look thoroughly at the wall behind Anny.'),
          nl,
          write(' You find out that there is a concavity in the'),
          nl,
        write(' shape of a rectangle. That concavity has a lighter '),
        nl,
        write(' paint color,than the rest of the wall.'),
        nl,
        write(' Seems that something used to hang there.'),
        nl.


item_properties(Person,Region,Place,Room):-
          Person=monk,
          nl,
            write(' An old monk is standing in front of you.'),
            nl,
          write(' He seems to be talking to the sea..'),
          nl,
            write(' You hear him whisper: "Oh great sea of happiness and pain '),
            nl,
          write('                        deliver us from darkness."    '),
```

```
        nl.

item_properties(Person,Region,Place,Room):-
        Person=elder,
        nl,
          write(' You are standing in front of the ELDER.'),
          nl,
        write(' His presence makes you feel strange..'),
        nl.

item_properties(Item,Region,Place,Room):-
        Item=parrot,
        nl,
          write(' The ELDERS parrot seems hungry.'),
        nl.

item_properties(Item,Region,Place,Room):-Item=chest,
                        closed(Item,Region,Place,Room),
                                nl,
                        write('You can see a fabulous chest and '),
                                write('you wonder what it could contain.'),
                        nl,
                        write('You have to open it, "cause it"s closed.'),
                                nl.

item_properties(Item,Region,Place,Room):-Item=chest,
                        opened(Item,Region,Place,Room),
                        nl,
                        write('You have opened the chest.'),
                        nl,
                        write('Inside it you find:'),
                        items_in_object(Item,Region,Place,Room),
                        nl.
```

```
items_in_object(Item,Region,Place,Room):-
in_object(Item,Region,Place,Room,Object),
                                        nl,
                              tab(3),
                                        write(Object),
                              fail.

items_in_object(Item,Region,Place,Room).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The open predicate opens an item, for instance a door or a chest

open(Item):-current_region(Region),
        current_place(Place),
          current_room(Room),
        stable_object(Item,Region,Place,Room),
          can_open(Item,Region,Place,Room).

open(Item):-turned_on(light_switch),
        current_region(Region), %the doors might be a problem..
        current_place(Place),
          current_room(Room),
        door(Item,Region,Place,Room,Region2,Place2,Room2,closed),
          can_open(Item,Region,Place,Room,Region2,Place2,Room2,closed).

open(Item):-turned_on(light_switch),
        current_region(Region), %the doors might be a problem..
        current_place(Place),
          current_room(Room),
        door(Item,Region,Place,Room,Region2,Place2,Room2,opened),

can_open(Item,Region,Place,Room,Region2,Place2,Room2,opened).
```

```
open(Item):-turned_off(light_switch),
            nl,
            write('You cannot see nothing in the dark.'),
           nl.

open(Item):-nl,
            write('You cannot open the '),
           write(Item),
           write('.'),
           nl.

open(_).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The can open predicate checks whether an item can be opened or not

can_open(Item,Region,Place,Froom,Region2,Place2,Sroom,closed):-
            door(Item,Region,Place,Froom,Region2,Place2,Sroom,closed),

retract(door(Item,Region,Place,Froom,Region2,Place2,Sroom,closed)),

assert(door(Item,Region,Place,Froom,Region2,Place2,Sroom,opened)),
            nl,
           write('You have opened the '),
            write(Item),
            write('.'),
            nl.

can_open(Item,Region,Place,Froom,Region2,Place2,Sroom,opened):- nl,
                                        write('The door is already opened.'),
                                              write('.'),
                                            nl.
```

```
can_open(Item,Region,Place,Room):-closed(Item,Region,Place,Room),
                                retract(closed(Item,Region,Place,Room)),
                                assert(opened(Item,Region,Place,Room)),
                        nl,
                        write('You have successfully opened the '),
                        write(Item),
                        write('.'),
                                nl.

can_open(Item,Region,Place,Room):-opened(Item,Region,Place,Room),
                        nl,
                        write('The '),
                        write(Item),
                        write(' is already opened.'),
                                nl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% The turn on predicate turns an item on, like a lights switch

turn_on(Item):-current_region(Region),
           current_place(Place),
           current_room(Room),
           stable_object(Item,Region,Place,Room),
             turned_off(Item),
           retract(turned_off(Item)),
           assert(turned_on(Item)),
           nl,
           write('You have turned on the '),
             write(Item),
             write('.'),
           nl.
```

```
turn_on(Item):-current_region(Region),
            current_place(Place),
            current_room(Room),
            stable_object(Item,Region,Place,Room),
            turned_on(Item),
            nl,
            write('The '),
            write(Item),
            write(' is already turned on.'),
            nl.

turn_on(Item):-nl,
              write('You cannot turn on the '),
              write(Item),
            write('.'),
            nl.

turn_on(_).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The turn off predicate turns an item off, like a lights switch

turn_off(Item):-current_region(Region),
             current_place(Place),
             current_room(Room),
             stable_object(Item,Region,Place,Room),
             turned_on(Item),
             retract(turned_on(Item)),
             assert(turned_off(Item)),
             nl,
             write('You have turned off the '),
             write(Item),
             write('.'),
```

```
        nl.

turn_off(Item):-current_region(Region),
        current_place(Place),
        current_room(Room),
        stable_object(Item,Region,Place,Room),
        turned_off(Item),
        nl,
        write('The '),
        write(Item),
        write(' is already turned off.'),
        nl.

turn_off(Item):-nl,
        write('You cannot turn off the '),
        write(Item),
        write('.'),
        nl.

turn_off(_).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The take predicate tests if an item can be taken by the player,
%% and if so, it puts it into the player's inventory,
%% and retracts that item from the dynamic database.

take(Item):-current_region(Region),
      current_place(Place),
      current_room(Room),
      object(Item,Region,Place,Room),
      can_take(Item,Region,Place,Room).

take(Item):-current_region(Region),
```

```
        current_place(Place),
        current_room(Room),
        in_object(_,Region,Place,Room,Item),
        can_take(Item,Region,Place,Room).

take(Item):-current_region(Region),
        current_place(Place),
        current_room(Room),
        stable_object(Item,Region,Place,Room),
        nl,
        write('You cannot move the '),
          write(Item),
        write('.'),
        nl.

take(Item):-current_region(Region),
        current_place(Place),
        current_room(Room),
          person(Item,Region,Place,Room),
        nl,
        write('You cannot take the '),
        write(Item),
          write('.'),
        nl.

take(Item):-current_region(Region),
        current_place(Place),
        current_room(Room),

door(Item,Region,Place,Room,Toregion,Toplace,To_room,Doorstate),
        nl,
          write('You cannot move the door.'),
          nl.
```

```
take(Item):-nl,
        write('There is no '),
        write(Item),
        write(' here.'),
        nl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The can take predicate tests if an object can be taken into the
%% Hero's inventory.

can_take(Item,Region,Place,Room):-Item\=painting,
                Item\=holy_water,
                retract(object(Item,Region,Place,Room)),
                        assert(item_in_inventory(Item)),
                nl,
                        write('You have taken the '),
                write(Item),
                write('.'),
                inventory.

can_take(Item,Region,Place,Room):-Item=holy_water,
                holy_water_state(0), % The Hero has not taken the holy
water
                        nl,
                write('Salesman:"Want the holy water,heh?" '),
                        nl,
                        write(' "It costs 100 Gold Pieces.. " '),
                nl.

can_take(Item,Region,Place,Room):-Item=holy_water,
                holy_water_state(1), % The Hero can take the holy
water
                nl,
```

```
                    retract(object(Item,Region,Place,Room)),
                    assert(item_in_inventory(Item)),
                    nl,
                    write('You have taken the '),
                    write(Item),
                    write('.'),
                    inventory.

can_take(Item,Region,Place,Room):-Item=painting,
                    painting_state(State),
                    State\=2,
                    nl,
                    write(' Salesman:"I could give you the painting,'),
                    write(' if you find me a dagger.'),
                    nl.

can_take(Item,Region,Place,Room):-
        Item=painting,
          painting_state(2),
          assert(item_in_inventory(painting)), % Exchange dagger for painting
          retract(object(painting,town,general_store,general_store)),          %
Remove painting from
        retract(painting_state(2)),                              % the wall
        assert(painting_state(3)), % The hero has taken the painting
          nl,
          write('You have taken the '),
          write(Item),
          write('.'),
          inventory.

can_take(Item,Region,Place,Room):-
retract(in_object(_,Region,Place,Room,Item)),
                              assert(item_in_inventory(Item)),
                    nl,
```

```
                write('You have taken the '),
        write(Item),
        write('.'),
                inventory.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The give predicate gives an Item to the person which is currently
%% in the same room. If the person accepts that Item, the necessary
%% changes in the dynamic database of the inventory will be made.

give(Item):-current_region(Region),
        current_place(Place),
        current_room(Room),
        Region=town,
          Place=general_store,
          Room=general_store,
          Item=dagger,
        item_in_inventory(Item), % We check whether the hero has dagger in
his inventory
        painting_state(1),      % The hero has seen the painting
        retract(painting_state(1)),
          assert(painting_state(2)), % The painting is ready to be taken
          retract(item_in_inventory(Item)),
          nl,
        write(' Salesman:"Thanks, you can take the painting now." '),
        nl.

give(Item):-current_region(Region),
        current_place(Place),
        current_room(Room),
          Region=town,
        Place=general_store,
          Room=general_store,
```

```
        Item='100GP',
          item_in_inventory(Item),
        holy_water_state(0),
        retract(holy_water_state(0)),
          assert(holy_water_state(1)),
        retract(item_in_inventory(Item)),
        nl,
          write(' Salesman:"Thanks, you can take the holy water now." '),
        nl.

give(Item):-current_region(Region),
        current_place(Place),
        current_room(Room),
          Region=town,
        Place=annys_home,
          Room=annys_home,
        Item=painting,
        item_in_inventory(Item),
        retract(item_in_inventory(Item)),
        retract(stable_object(hole,town,annys_home,annys_home)),
        nl,
          write(' Anny: "I cannot believe it! This is my painting! I"ve lost'),
        nl,
        write('        it somewhere in town.I"ll put it back where it was, '),
          nl,
          write('            covering the hole." '),
          nl.

give(Item):-current_region(Region),
        current_place(Place),
        current_room(Room),
          Region=town,
        Place=annys_home,
          Room=annys_home,
```

```
        Item=apple,
        item_in_inventory(Item),
        retract(item_in_inventory(Item)),
        assert(item_in_inventory(map)),
          retract(map_state(0)),
          assert(map_state(1)),
        nl,
          write(' Anny: "You are so kind! I hope this would stop the stomach
ache.., '),
          nl,
          write('          Oh,  I forgot!! Here is the map that leads to deepsea.'),
        nl,
        write('    Take it with you,it will be more valuable in your hands.'),
          nl,
          inventory.

give(Item):-current_region(Region),
        current_place(Place),
        current_room(Room),
          Region=deepsea,
        Place=deepsea,
          Room=deepsea,
        Item=holy_water,
        item_in_inventory(Item),
          retract(item_in_inventory(Item)),
        retract(monk_speech_state(0)),
        assert(monk_speech_state(1)),
          speech(monk,deepsea,deepsea,deepsea).

give(Item):-current_region(Region),
        current_place(Place),
        current_room(Room),
          Region=darkwood,
        Place=darkwood,
```

```
		Room=darkwood,
	Item=sandwich,
	item_in_inventory(Item),
		retract(item_in_inventory(Item)),
	retract(elder_speech_state(0)),
	assert(elder_speech_state(1)),
		speech(elder,darkwood,darkwood,darkwood).

give(Item):-current_region(Region),
	current_place(Place),
	current_room(Room),
	item_in_inventory(Item), % If the Item exists in the Hero's inventory
		nl,
	write(' "There is no reason to give that now." '),
		nl.

give(Item):-current_region(Region),
	current_place(Place),
	current_room(Room),
		nl,
	write(' "You don''t have such thing." '),
		nl.

give(Item).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% When "inventory" are called, it returns the list of
%% objects in inventory.

inventory:-nl,
	 write('Your inventory has:'),
	 nl,
	 item_in_inventory(Item),
```

```
        tab(2),
        write(Item),
        nl,
        fail.

inventory.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The goto predicate,if it succeeds, it takes the character to another
%% location. The dynamic database is used to refresh the predicates
%% that represent the location, f.i. current_room.

goto(Town):-Town=town,     %this goto is for going to the town only
       can_go(town,town,town),
       look.

goto(House):-current_region(Region), %this goto is for the homes that are in town
        current_place(Place),
        current_room(Room),
        Region=town,
        Place=town,
        Room=town,
       House\=deepsea,
        house(House),
        can_go(House),
        look.

goto(Someroom):-current_region(Region), %must be renewed
           current_place(Place),
           current_room(Room),
        Someroom\=town,
           Someroom\=deepsea,
```

```
                can_go(Region,Place,Someroom), %the can_go predicate calls
the
                look.                   %move predicate

goto(Deepsea):-current_region(Region),
              current_place(Place),
              current_room(Room),
              Region=town,
            Place=town,
              Room=town,
            Deepsea=deepsea,
            can_go(Deepsea).

goto(Darkwood):-current_region(Region),
               current_place(Place),
               current_room(Room),
               Region=town,
             Place=town,
               Room=town,
             Darkwood=darkwood,
             can_go(Darkwood).

goto(Someroom):-nl,
             write('You cannot go to the '),
             write(Someroom),
                 write('.'),
             nl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The can go predicate tests if a character can go to the specified
%% place. If it succeeds, it calls the move predicate
```

```
can_go(town,town,town):-current_region(Region),  %the next 2 can_go predicates refer to town
                current_place(Place), %checking if we are not
                    Place\=town,        %already in town
                current_room(Room),
                      Room\=town,
                door(Doorname,Region,Place,Room,town,town,town,opened),
                move(town).

can_go(town,town,town):-current_region(Region),
                current_place(Place), %checking if we are not
                    Place\=town,        %already in town
                current_room(Room),
                      Room\=town,
                door(Doorname,Region,Place,Room,town,town,town,closed),
                      nl,
                      write('The door to the town is closed'),
                fail.

can_go(town,town,town):-current_region(Region),
                Region=deepsea,
                move(town).

can_go(town,town,town):-current_region(Region),
                Region=darkwood,
                move(town).

can_go(House):-current_region(Region),
              current_place(Place),
              current_room(Room),
            House\=deepsea,
            House\=darkwood,
            Region=town,
            Place=town,
```

```
            Room=town,
              move(House).

can_go(Toregion,Toplace,Toroom):-current_region(Region),
                          current_place(Place),
                            current_room(Room),
                          Toregion=Region, %same region,same place
                          Toplace=Place,   %different room
                          Toroom\=Room,
                                  Room\=town,

door(Doorname,Region,Place,Room,Region2,Place2,Toroom,opened),
                          %check door opened
                          move(Toroom). %only the argument that differs

can_go(Toregion,Toplace,Toroom):-current_region(Region),
                          current_place(Place),
                            current_room(Room),
                          Toregion=Region, %same region,same place
                          Toplace=Place,   %different room
                          Toroom\=Room,
                                  Room\=town,

door(Doorname,Region,Place,Toroom,Region2,Place2,Room,opened),
                          %check door opened
                          move(Toroom). %only the argument that differs

can_go(Toregion,Toplace,Toroom):-current_region(Region),
                          current_place(Place),
                            current_room(Room),
                          Toregion=Region, %same region,same place
                          Toplace=Place,   %different room
                          Toroom\=Room,
```

```
door(Doorname,Region,Place,Room,Region2,Place2,Toroom,closed),
                        %check door opened
                        nl,
                        write('The door to the '),
                        write(Toroom),
                        write(' is closed'),
                                fail.

can_go(Deepsea):-Deepsea=deepsea,
            map_state(0),
            nl,
            write(' "You don"t know how to get there." '),
            nl.

can_go(Deepsea):-Deepsea=deepsea,
                map_state(1),
            nl,
            write(' "You"ve found the map, so let"s go!.." '),
            delay(2),
                nl,
            move(deepsea),
            look.

can_go(Darkwood):-Darkwood=darkwood,
             can_go_darkwood(0),
             nl,
             write(' "You don"t know how to get there." '),
             nl.

can_go(Darkwood):-Darkwood=darkwood,
                  can_go_darkwood(1),
                  nl,
             write(' "You"ve the ELDER"s approval, so let"s go!.." '),
```

```
              delay(2),
                   nl,
              move(darkwood),
              look.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% We define the move predicate,which retracts the previous location
%% and asserts the new.

move(town):-retract(current_region(Region)),
          assert(current_region(town)),
          retract(current_place(Place)),
            assert(current_place(town)),
          retract(current_room(Room)),
          assert(current_room(town)).

move(House):-current_place(Place),
           Place=town,
           House\=heros_home,
           House\=deepsea,
             House\=darkwood,
             retract(current_place(Place)), % For instance,it would be something
           assert(current_place(House)),    % like town,home,home. Must be
renewed
           retract(current_room(Room)),   % in case a house has more than one
room
           assert(current_room(House)).

move(House):-current_place(Place),
           Place=town,
           House=heros_home,
             retract(current_place(Place)),
             assert(current_place(House)),
```

```
            retract(current_room(Room)),
            assert(current_room(kitchen)).

move(Toroom):-Toroom\=deepsea,
             Toroom\=darkwood,
             retract(current_room(Room)),
               assert(current_room(Toroom)).

move(deepsea):-retract(current_region(Region)),
              assert(current_region(deepsea)),
              retract(current_place(Place)),
                assert(current_place(deepsea)),
              retract(current_room(Room)),
              assert(current_room(deepsea)).

move(darkwood):-retract(current_region(Region)),
               assert(current_region(darkwood)),
               retract(current_place(Place)),
                 assert(current_place(darkwood)),
               retract(current_room(Room)),
               assert(current_room(darkwood)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% As it name tells, it gives us information about our current location

location_info(town):-current_region(Region),
                    current_place(Place),
                    current_room(Room),
                    nl,
                    write('Current region -> '),
                    write(Region),
                      nl.
```

```
location_info(deepsea):-current_region(Region),
                        current_place(Place),
                        current_room(Room),
                        nl,
                        write('Current region -> '),
                        write(Region),
                          nl.

location_info(darkwood):-current_region(Region),
                         current_place(Place),
                         current_room(Room),
                         nl,
                         write('Current region -> '),
                         write(Region),
                           nl.

location_info:-current_region(Region),
               current_place(Place),
               current_room(Room),
               Place\=Room,
               nl,
               write('Current region -> '),
               write(Region),
                 nl,
               tab(8),
               write('place  -> '),
                 write(Place),
                 nl,
                 tab(8),
               write('room   -> '),
                 write(Room),
               nl.

location_info:-current_region(Region), % We overload location_info
```

```
        current_place(Place),   % in case the place and room
        current_room(Room),     % equals,because we don't want
        Place=Room,             % the room to be described
        nl,
        write('Current region -> '),
        write(Region),
          nl,
        tab(8),
        write('place  -> '),
          write(Place),
          nl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Maybe the most useful predicate. Look tells us where we currently are,
%% what objects surround us, and where we can go from there.

look:-nl,
    current_region(Region),
    current_place(Place),
    current_room(Room),
    Room=bedroom,
    turned_off(light_switch),
    write('You can see nothing, because it''s dark.'),
    nl,
    write('You touch the wall above your bed, and sense the light switch.'),
    nl.

look:-current_region(Region), %this look is for the town only
    current_place(Place),
    current_room(Room),
    Region=town,
    Place=town,
    Room=town,
```

```
    delay(1),
    location_info(town),
%we overload location_info for the town
    write('You can see:'),
    nl,
    nearby_items(town),
    write('You can go to:'),
    nl,
    nearby_places(town).

look:-current_region(Region), %this look is for the deepsea only
    current_place(Place),
    current_room(Room),
    Region=deepsea,
    Place=deepsea,
    Room=deepsea,
    delay(1),
    location_info(deepsea),
%we overload location_info for the deepsea
    write('You can see:'),
    nl,
    nearby_items,
    write('You can go to:'),
    nl,
    nearby_places(deepsea).

look:-current_region(Region), %this look is for darkwood only
    current_place(Place),
    current_room(Room),
    Region=darkwood,
    Place=darkwood,
    Room=darkwood,
    delay(1),
```

```
    location_info(darkwood),
%we overload location_info for darkwood
    write('You can see:'),
    nl,
    nearby_items,
    write('You can go to:'),
    nl,
    nearby_places(darkwood).

look:-delay(1),
    location_info,
    write('You can see:'),
    nl,
    nearby_items,
    write('You can go to:'),
    nl,
    nearby_places.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The nearby items return the items that surround us
%% We don't deal with doors while we are in town, to
%% avoid complexity

nearby_items(town):-current_region(Region),
        current_place(Place),
        current_room(Room),
        object(Object,Region,Place,Room),
        tab(2),
        write(Object),
        nl,
        fail.

nearby_items(town):-current_region(Region),
```

```
                current_place(Place),
                current_room(Room),
                stable_object(Object,Region,Place,Room),
                tab(2),
                write(Object),
                nl,
                fail.

nearby_items(town):-current_region(Region),
                current_place(Place),
                current_room(Room),
                person(Person,Region,Place,Room),
                tab(2),
                write(Person),
                nl,
                fail.

nearby_items(town).

%% We can avoid the above code,if in the nearby_items\0
%% we specify that doors are described in anywhere but
%% the town

nearby_items:-current_region(Region),
            current_place(Place),
            current_room(Room),
            object(Object,Region,Place,Room),
            tab(2),
            write(Object),
            nl,
            fail.

nearby_items:-current_region(Region),
            current_place(Place),
```

```
        current_room(Room),
        stable_object(Object,Region,Place,Room),
        tab(2),
        write(Object),
        nl,
        fail.

nearby_items:-current_region(Region),
        current_place(Place),
        current_room(Room),
        person(Person,Region,Place,Room),
        tab(2),
        write(Person),
        nl,
        fail.

nearby_items:-current_region(Region),
        current_place(Place),
        current_room(Room),
        door(Doorname,Region,Place,Room,Region2,Place2,Toroom,State),
        tab(2),
        write(Doorname),
        nl,
        fail.

nearby_items:-current_region(Region),
        current_place(Place),
        current_room(Room),
        door(Doorname,Region,Place,Toroom,Region2,Place2,Room,State),
        tab(2),
        write(Doorname),
        nl,
        fail.
```

```
nearby_items.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The connect predicate checks whether two rooms are connected via a
%% door

connect(Region,Place,Room,Region2,Place2,Room2):-
current_region(Region),
                                    current_place(Place),
                                  current_room(Room),
                                  Room\=town,

door(Doorname,Region,Place,Room,Region2,Place2,Room2,_).

connect(Region,Place,Room,Region2,Place2,Room2):-
current_region(Region),
                                      current_place(Place),
                                  current_room(Room),
                                          Room\=town,

door(Doorname,Region,Place,Room2,Region2,Place2,Room,_).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The list_regions predicate returns all regions in the game, except
%% the one that we are currently in

list_regions:-current_region(Region),
        region(Region2),
        Region2\=town,
        tab(2),
        write(Region2),
        nl,
```

```
            fail.

list_regions.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The list_houses predicate returns all houses in town

list_houses:-house(House),
              tab(2),
              write(House),
           nl,
           fail.

list_houses.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% nearby places return the places that are connected to our current
%% place

nearby_places(town):-current_region(Region),
                 current_place(Place),
                      current_room(Room),
                 Region=town,
                 Place=town,
                      Room=town,
                      list_houses,
                 write('Regions :'),
                 nl,
                 list_regions,
                 fail.

nearby_places(town).
```

```
nearby_places(deepsea):-current_region(Region),
                    current_place(Place),
                        current_room(Room),
                    Region=deepsea,
                    Place=deepsea,
                        Room=deepsea,
                        write('Regions :'),
                    nl,
                          tab(2),
                          write('town'),
                    nl,
                        fail.

nearby_places(deepsea).

nearby_places(darkwood):-current_region(Region),
                     current_place(Place),
                          current_room(Room),
                     Region=darkwood,
                     Place=darkwood,
                          Room=darkwood,
                          write('Regions :'),
                     nl,
                           tab(2),
                           write('town'),
                     nl,
                          fail.

nearby_places(darkwood).

nearby_places:-current_region(Region),
               current_place(Place),
               current_room(Room),
```

```
            Room\=town,
            connect(Region,Place,Room,Region2,Place2,Room2), %!!!
            tab(2),
            write(Room2),
            nl,
            fail.

nearby_places.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The delay predicate takes as argument the seconds that we want
%% prolog to sleep.

delay(Seconds):-sleep(Seconds).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The help command describes the commands that a player can use

help:-nl,
     delay(1),
     write('You can use the following commands:'),
     nl,
     write('                  look  -> Gives location information, and informs'),
     nl,
     write('                          the player of the objects that surround him.'),
     nl,
     write('          goto Place  -> Goes to the specified place.'),
     nl,
     write('       look_at  Item     ->  Gives  information  about  Something  or
Someone'),
     nl,
     write('          inventory  -> Opens the inventory.'),
```

```
    nl,
    write('        turn_on Item  -> Turns on the specified item.'),
    nl,
    write('       turn_off Item  -> Turns off the specified item.'),
    nl,
    write('           open Item  -> Opens the specified item.'),
    nl,
    write('           take Item  -> Takes the specified item.'),
    nl,
    write('           give Item  -> Gives the specified Item.'),
    nl,
    write('      talk_to Person  -> Talks to specified Person.'),
    nl,
    write('               quest    -> Describes the current quest/s.'),
    nl,
    write('                   quit   -> Quit the game - quest failed. '),
    nl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
%% Talk to a person, if a person is near you

talk_to(Person):-current_region(Region),
            current_place(Place),
                current_room(Room),
            person(Person,Region,Place,Room),
            speech(Person,Region,Place,Room).

talk_to(Person):-nl,
                write('There''s noone near you.'),
            nl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% The speech of a person. If a quest is upgraded during a speech,
%% we must control the upgrade quest rule, so that, if we talk to
%% that person again, the upgrade will not be executed twice.We
%% accomplish this, by putting a flag into the program, which
%% represents the current quest state and putting it as a goal
%% in the speech which upgrades that flag.

speech(Person,Region,Place,Room):-
        Person=mother,
          quest_state(0),
        nl,
          write(' "Goodmorning son. I"ve seen you"ve woken up early.'),
        nl,
        write('I can see you are tired.Here, take this sandwich.It"ll do you
good.'),
        nl,
        assert(item_in_inventory(sandwich)),
             write('Go and tell your sister to come here for breakfast.You will
find her'),
             nl,
        write('outdoors." '),
        nl,
             nl,
        retract(quest_state(0)),
        assert(quest_state(1)),
        write(' "Quest Upgraded!!"  Enter quest for info.'),
        nl.

speech(Person,Region,Place,Room):-
        Person=mother,
        quest_state(State),
        State\=0,
          nl,
          write(' "Goodmorning son. I"ve seen you"ve woken up early.'),
```

```
        nl,
            write('Go and tell your sister to come here for breakfast.You will
find her'),
            nl,
        write('outdoors." '),
        nl.

speech(Person,Region,Place,Room):-
            Person=salesman,
            painting_state(0),
            nl,
            write(' "Welcome child! Please inform me if anything in my store
'),
        nl,
        write('could be helpful to you. By the way, have you looked at the
painting?" '),
        nl.

speech(Person,Region,Place,Room):-
            Person=salesman,
            painting_state(1),
            nl,
            write(' "Welcome child! Please inform me if anything in my store
'),
        nl,
        write('could be helpful to you. I"ve found this painting near the
fountain.'),
        nl,
        write('Maybe someone had dropped it." '),
        nl.

speech(Person,Region,Place,Room):-
            Person=salesman,
            painting_state(3), %The hero has taken the painting
```

```
                nl,
                write(' "Welcome child! Please inform me if anything in my store
'),
            nl,
            write('could be helpful to you.'),
            nl.

speech(Person,Region,Place,Room):-
            Person=anny,
                anny_speech_state(0),
                retract(anny_speech_state(0)),
            assert(anny_speech_state(1)),
                nl,
            write(' "Oh, you cannot imagine how happy I am to see you.'),
                nl,
                write(' But I"m afraid I have to tell you the bad news." '),
                nl.

speech(Person,Region,Place,Room):-
            Person=anny,
            anny_speech_state(1),
            quest_state(1), %in case quest is in the first state
            nl,
            write(' "Oh, ok. I guess you want to hear the news.'),
                nl,
            write('   I was out playing with your sister at the dock '),
            nl,
            write('   which lies at the deepsea. But then my stomach '),
            nl,
            write('   started to ache, and I came home. But I"ve heard'),
            nl,
                write('   that your sister has not come back since." '),
            nl,
            nl,
```

```
        retract(quest_state(1)),
        assert(quest_state(2)), % the quest now is to visit deepsea
            write('              Quest Updated!!                 '),
        nl.

speech(Person,Region,Place,Room):-
        Person=anny,
        anny_speech_state(1),
        quest_state(0),
        nl,
            write(' Anny: "Have you talked to your mother??'),
        nl.

speech(Person,Region,Place,Room):-
        Person=anny,
        anny_speech_state(1),
        quest_state(2),
        nl,
            write(' Anny: "Did you find your sister? '),
        nl.

speech(Person,Region,Place,Room):-
          Person=monk,
        monk_speech_state(0),
        nl,
        write(' "Hello young one, what wind brought you here? '),
        nl,
        write('        Excuse me,but I want to pray,so please leave '),
        nl,
        write('   me alone child.'),
        nl.

speech(Person,Region,Place,Room):-
          Person=monk,
```

```
        monk_speech_state(1),
        quest_state(2),
        nl,
        write(' "Oh thank you my dear child. I will throw this at sea '),
        nl,
        write('        so that the sea will be pleased and help you in your '),
        nl,
        write('   quest. Wait,she speaks to me saying that you are free'),
            nl,
            write('   to go to the darkwood,and meet the ELDER.He will tell'),
        nl,
        write('   you whatever you need.'),
        nl,
            nl,
            write('                  Quest Upgraded!!                  '),
        nl,
        retract(quest_state(2)),
            assert(quest_state(3)),
        retract(can_go_darkwood(0)),
        assert(can_go_darkwood(1)).

speech(Person,Region,Place,Room):-
          Person=monk,
        monk_speech_state(1),
        quest_state(3),
        nl,
        write(' "Oh thank you my dear child. I will throw this at sea '),
        nl,
        write('        so that the sea will be pleased and help you in your '),
        nl,
        write('   quest. Wait,she speaks to me saying that you are free'),
            nl,
            write('   to go to the darkwood,and meet the ELDER.He will tell'),
        nl,
```

```
        write('   you whatever you need.'),
        nl.

speech(Person,Region,Place,Room):-
        Person=elder,
            elder_speech_state(0),
            nl,
        write(' ''Hello to thee,I''m searching for food to give to my '),
        nl,
        write('   parrot.'),
        nl.

speech(Person,Region,Place,Room):-
        Person=elder,
            elder_speech_state(1),
            nl,
        write(' ''Thanks,the parrot will be happy now. '),
        nl,
        write('   You are searching for your sister? Well,ok,'),
        nl,
            write('    I will tell you where she is.She was playing '),
        nl,
        write('   Hide and Seek with her friend,Maria.See is still'),
            nl,
        write('   Hiding here,Daphne come out.Your brother is looking'),
        nl,
        write('   for you..'),
        nl,
        nl,
            write(' Daphne: Hallo,sorry for making you feel nervous.'),
        nl,
        write('   Hero: It''s ok,but don''t do it again.Let''s go home now..'),
        nl,
        nl,
```

```
        nl,
            delay(5),
        write('Congratulations!! '),
        nl,
            nl,
            nl,
        write('            Mystic Quest Ended..    Hope you enjoyed it.'),
            nl,
        write('                Thanks for your time. '),
            nl,
        write('                                             Created by Tim.(E.T.)'),
        nl,
        delay(2),
            end_condition(end).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The quest state represents, as logical, the current primary/secondary quests.

quest:-quest_state(0),
    nl,
    write('You don''t have a quest at the moment.'),
    nl.

quest:-quest_state(1),
    nl,
    write('Current quest: Find your sister and inform her for breakfast.'),
    nl.

quest:-quest_state(2),
    nl,
    write('Current quest: Search for your sister in the deepsea.'),
```

```
    nl.

quest:-quest_state(3),
    nl,
    write('Current quest: Go in darkwood,the darkest of forests.'),
    nl.
```

Βιβλιογραφία - Πηγές

- **Learn Prolog Now!** Patrick Blackburn, Johan Bos, Kristina Striegnitz
- **Adventure in Prolog** http://www.amzi.com/AdventureInProlog/advfrtop.htm
- **Prolog Lectures** by Gareth Lullaby
- **"Sleepy" -- A Sample Adventure Game in Prolog** David Matuszek, Villanova University
- **Some Coding Guidelines for Prolog** Michael A. Covington, Artificial Intelligence Center, The University of Georgia
- **SWI-Prolog 5.2 Reference Manual** Jan Wielemaker
- **Natural Language Processing Introduction** http://www.cee.hw.ac.uk/~diana/nl/l01sh Nikos Drakos, Computer Based Learning Unit, University of Leeds.
- **Logic, Language and Learning: Programming in Prolog** Stefan Kramer (original slides by Peter Flach)