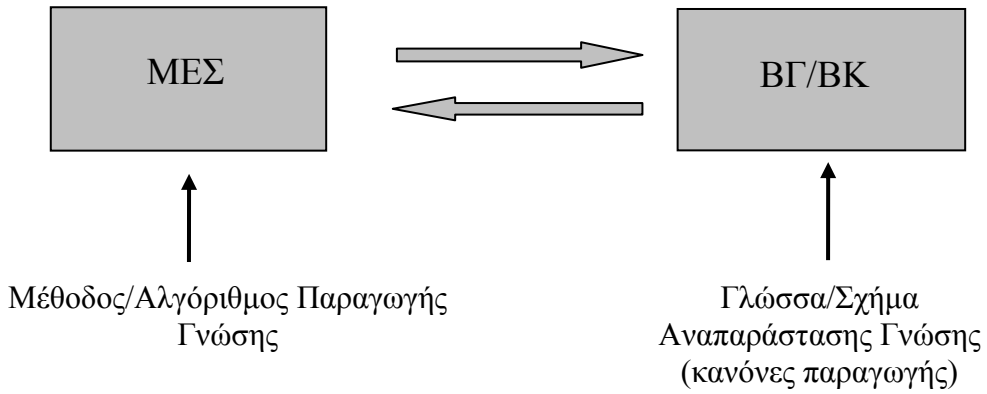
12 ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΒΑΣΙΣΜΕΝΟΣ ΣΕ ΚΑΝΟΝΕΣ

12.1 Εισαγωγή

Οι *κανόνες παραγωγής* (production rules) είναι μια γλώσσα αναπαράστασης γνώσης. Οι γλώσσες αναπαράστασης γνώσης χρησιμοποιούνται σε *ευφυή συστήματα* (intelligent systems). Ένα ευφυές σύστημα αποτελείται από δύο βασικές μονάδες: τη *βάση γνώσης* (knowledge base) και τη *μηχανή εξαγωγής συμπερασμάτων* (inference engine) (βλ. Σχήμα 12.1). Στη βάση γνώσης (ΒΓ) αποθηκεύονται εκφράσεις που αναπαριστούν τη γνώση γύρω από ένα πρόβλημα (ή μάλλον μια κατηγορία προβλημάτων). Αυτές οι εκφράσεις είναι γραμμένες σε κάποια γλώσσα αναπαράστασης γνώσης, όπως η λογική, οι κανόνες παραγωγής, τα πλαίσια κλπ. Η μηχανή εξαγωγής συμπερασμάτων χρησιμοποιεί τη γνώση στη βάση γνώσης με κάποιο τρόπο (μέθοδο) για την εξαγωγή «νέας» γνώσης (συμπερασμάτων), δηλ. γνώσης που ενυπάρχει στη βάση γνώσης, αλλά δεν είναι εμφανής. Κάθε γλώσσα αναπαράστασης συνοδεύεται και από τον δικό της μηχανισμό εξαγωγής συμπερασμάτων, που είναι κατάλληλος γι' αυτήν.

Μια πολύ γνωστή κατηγορία ευφυών συστημάτων είναι τα *έμπειρα συστήματα* (expert systems), τα οποία συνήθως χρησιμοποιούν κανόνες για την αναπαράσταση γνώσης, οπότε και η βάση γνώσης λέγεται βάση κανόνων (rule base). Δηλ. μια βάση κανόνων (ΒΚ) περιέχει ένα αριθμό από κανόνες, που εκφράζουν τη γνώση με την οποία ένας εμπειρογνώμονας σ' ένα πεδίο λύνει προβλήματα αυτού του πεδίου. Εκτός όμως από τους κανόνες, ρόλο στην εξαγωγή συμπερασμάτων παίζουν και τα γεγονότα (facts). Τα γεγονότα αντιπροσωπεύουν είτε αρχικά δεδομένα είτε

ενδιάμεσα συμπεράσματα κατά τη διαδικασία της παραγωγής συμπερασμάτων και αποθηκεύονται σε μια ξεχωριστή βάση, που ονομάζεται *βάση γεγονότων* (fact base) ή *μνήμη εργασίας* (working memory).



Σχήμα 12.1 Βασικές Μονάδες Ευφυούς/Εμπειρου Συστήματος

Από τους κανόνες παραγωγής πηγάζει ένα νέο είδος προγραμματισμού, ο λεγόμενος *προγραμματισμός βασισμένος σε κανόνες* (rule-based programming), όπως από τη λογική πηγάζει ο λογικός προγραμματισμός (logic programming). Υπάρχουν αρκετές γλώσσες ή εργαλεία προγραμματισμού βασισμένου σε κανόνες, όπως OPS5, CLIPS και JESS. Τα εργαλεία αυτά αποτελούν ταυτόχρονα και γενικές γλώσσες προγραμματισμού και μπορούν να υλοποιήσουν και συμβατικά προγράμματα.

Κάθε γλώσσα αναπαράστασης προσδιορίζεται από δύο βασικά στοιχεία, τη σύνταξή (syntax) της και τη σημαντική (semantics) της. Επίσης, συνοδεύεται και από ένα τρίτο στοιχείο, όπως προαναφέραμε, τη μηχανή εξαγωγής συμπερασμάτων.

12.2 Σύνταξη και Σημαντική

Η βασική μορφή ενός κανόνα είναι η ακόλουθη:

if <conditions>
then <actions>/<conclusions>

Υπάρχουν δηλαδή δύο τμήματα. Το πρώτο (τμήμα if) αποτελείται από ένα αριθμό συνθηκών ή υποθέσεων και το δεύτερο (τμήμα then) από μία ή περισσότερες

ενέργειες ή συμπεράσματα. Η σημαντική ενός τέτοιου κανόνα είναι ότι, αν οι συνθήκες ικανοποιούνται (δηλ. αληθεύουν) τότε εκτελούνται οι ενέργειες ή εξάγονται τα συμπεράσματα.

Γενικά υπάρχουν δύο μεγάλες τέτοιες κατηγορίες υλοποιήσεων αυτής της γενικής μορφής. Στη μία, οι σχέσεις μεταξύ κανόνων και γεγονότων είναι προκαθορισμένες και έτσι μια βάση κανόνων μπορεί να παρασταθεί σαν ένα δίκτυο που παριστάνει τις αλληλεξαρτήσεις μεταξύ κανόνων και γεγονότων. Αυτές οι γλώσσες δεν χρησιμοποιούν μεταβλητές στις συνθήκες. Η δεύτερη κατηγορία στηρίζεται στην έννοια του ταιριάσματος προτύπων (pattern matching). Οι σχέσεις μεταξύ κανόνων και γεγονότων σ' αυτές τις υλοποιήσεις δεν είναι προκαθορισμένες, αλλά σχηματοποιούνται κατά τη διαδικασία εξαγωγής συμπερασμάτων. Εδώ έχουμε χρήση μεταβλητών στις συνθήκες. Οι γλώσσες της δεύτερης αυτής κατηγορίας είναι πιο ευέλικτες, μεγαλύτερων δυνατοτήτων από αυτές της πρώτης και καταλληλότερες για προγραμματισμό. Γι' αυτό και όλες οι γνωστές γλώσσες προγραμματισμού βασισμένου σε κανόνες ακολουθούν αυτό το πρότυπο.

Σ' αυτόν τον τύπο γλωσσών, κάθε συνθήκη ανήκει σε κάποιο *πρότυπο* (pattern) ή με άλλα λόγια υλοποιεί κάποιο πρότυπο. Ένα πρότυπο είναι μια n-άδα οντοτήτων (ή στοιχείων) που παριστάνουν αντικείμενα, έννοιες, σχέσεις, ιδιότητες κλπ και αποτελεί σημαντική παράμετρο στην αναπαράσταση της γνώσης του πεδίου ενός προβλήματος. Συνήθως ένα πρότυπο χαρακτηρίζεται από το πρώτο στοιχείο του. Ο συλλογισμός για την εξαγωγή συμπερασμάτων στηρίζεται στη συσχέτιση στιγμιοτύπων των προτύπων αυτών.

Στη συνέχεια, παραθέτουμε τη σύνταξη μιας τέτοιας γλώσσας:

<rule> := **if** <conditions> **then** <conclusions>
<conditions> := <condition> {**and** <condition>}*
<conclusions> := <conclusion> {**and** <conclusion>}*
<condition> := <predicate> <pattern>
<conclusion> := <action> < pattern >
<pattern> := ({<variable>/<constant>}*)
<predicate> := is/isnot/greaterthan/lessthan/…
<action> := assert

<fact> := ({<constant>}*)

Ένα παράδειγμα κανόνα που ακολουθεί την παραπάνω σύνταξη είναι το εξής:

if is(person ?name ?age) **and**
graterthan(?age 12) **and**
lessthan(?age 20)
then assert(teenager ?name ?age)

Παρατηρείστε ότι οι μεταβλητές, για να ξεχωρίζουν, έχουν το «?» σαν πρώτο χαρακτήρα. Φυσικά υπάρχουν διάφορες παραλλαγές της παραπάνω σύνταξης. Π.χ. μπορεί να υπάρχουν και τα λογικά συνδετικά **or** και **not** στις συνθήκες ή εκτός από το 'assert' να υπάρχει και άλλη ενέργεια, π.χ. 'retract'.

12.3 Ταίριασμα Προτύπων

Το ταίριασμα προτύπων (pattern matching) είναι μια βασική διαδικασία στον βασισμένο σε κανόνες προγραμματισμό. Η διαδικασία αυτή συνίσταται στον προσδιορισμό των γεγονότων από τη βάση γεγονότων, που ταιριάζουν με μια συνθήκη, δηλ. στο αν υπάρχουν τιμές για τις μεταβλητές της συνθήκης (που ονομάζονται προσδέσεις) που κάνουν ταυτόσημες τη συνθήκη και κάποιο(α) γεγονός(ότα). Ένας ορισμός της διαδικασίας αυτής είναι ο εξής:

Ορισμός: Ένα *πρότυπο* (*pattern*) *ταιριάζει* (*matches*) με ένα γεγονός (fact) αν υπάρχουν *προσδέσεις* (*bindings*) για τις μεταβλητές στο πρότυπο τέτοιες που, αν αντικαταστήσουμε τις μεταβλητές με τις προσδεδεμένες τιμές (προσδέσεις) τους, το γεγονός και το πρότυπο γίνονται συντακτικά ταυτόσημα.

Ας δούμε ένα παράδειγμα. Έστω η παρακάτω βάση γεγονότων

ΒΓ={(Yannis Kostas Maria),
(Giorgos Kostas Eleni),
(Petros Pavlos Maria)}

και έστω το παρακάτω πρότυπο (συνθήκη):

(?name Kostas Maria)

Τότε, παρατηρούμε ότι το πρότυπο αυτό «ταιριάζει» με πρώτο γεγονός «(Yannis Kostas Maria)», διότι υπάρχει πρόσδεση για τη μεταβλητή «?name»:

{ (?name.Yannis) }

οπότε αντικαθιστώντας στο πρότυπο την τιμή «Yannis» στη θέση της μεταβλητής «?name» παίρνουμε:

(Yannis Kostas Maria)

που αποτελεί ένα *στιγμιότυπο* (instance) ή μια στιγμιοτυποποίηση (instantiation) του προτύπου και είναι ταυτόσημο με το πρώτο γεγονός της ΒΓ.

Παρομοίως, για το παρακάτω πρότυπο:

(?name Kostas ?moth)

έχουμε τις παρακάτω προσδέσεις:

{ ((?name.Yannis), (?moth.Maria))
((?name.Giorgos),(?moth.Eleni)) }

και τα παρακάτω γεγονότα της ΒΓ με τα οποία ταιριάζει το πρότυπο:

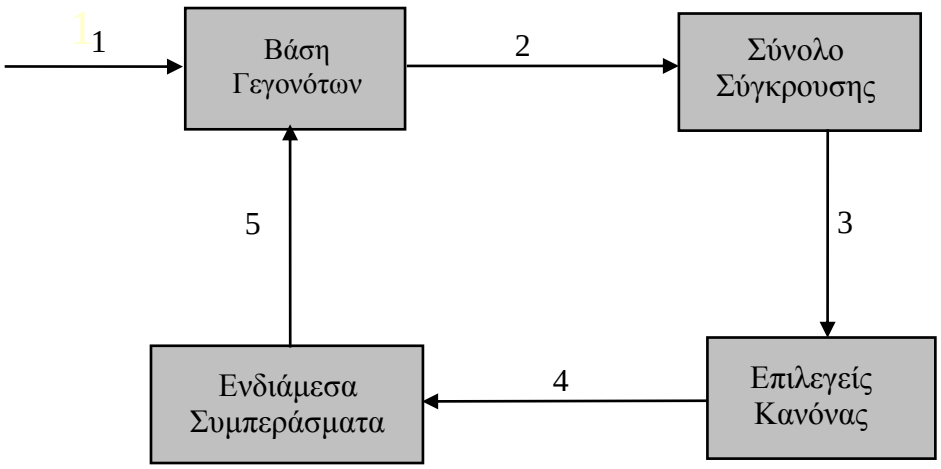
(Yannis Kostas Maria)
(Giorgos Kostas Eleni).

12.4 Διαδικασία Εξαγωγής Συμπερασμάτων

Υπάρχουν δύο μέθοδοι εξαγωγής συμπερασμάτων στον βασισμένο σε κανόνες προγραμματισμό, η *κατευθυνόμενη από το στόχο* (goal driven) ή *ανάστροφη*

αλυσίδωση (backward chaining) και η *κατευθυνόμενη από τα δεδομένα* (data driven) ή *ορθή αλυσίδωση* (forward chaining).

Στην ανάστροφη αλυσίδωση, η μηχανή εξαγωγής συμπερασμάτων ξεκινά από το συμπέρασμα ενός κανόνα που ταιριάζει με το στόχο που θέλουμε πετύχουμε (συμπεράνουμε) και προχωρά στη διερεύνηση της αλήθειας των υποθέσεων. Αν οι υποθέσεις είναι αληθείς, τότε εξάγεται το συμπέρασμα. Αν η αλήθεια ή το ψεύδος κάποιων (ή όλων των) υποθέσεων είναι άγνωστα, τότε αυτές γίνονται υπο-στόχοι, αναζητούνται κανόνες με αυτούς ως συμπεράσματα κ.ο.κ. έως ότου καταλήξουμε σε αληθείς συνθήκες (δηλ. συνθήκες που ταιριάζουν με γεγονότα της ΒΓ ή σε κάποιες ψευδείς, οπότε σταματά η διαδικασία.



Σχήμα 12.2 Διαδικασία εξαγωγής συμπερασμάτων

Στην ορθή αλυσίδωση, η μηχανή εξαγωγής συμπερασμάτων ξεκινά από τις υποθέσεις ενός κανόνα και αν είναι αληθείς προχωρά στην εξαγωγή του συμπεράσματός του. Αυτό συνεχίζεται μέχρις ότου εξαχθεί το ζητούμενο συμπέρασμα. Οι γνωστές γλώσσες προγραμματισμού βασισμένου σε κανόνες ακολουθούν κατά βάση αυτή τη μέθοδο και σ' αυτή θα επικεντρώσουμε εδώ.

Η διαδικασία εξαγωγής συμπερασμάτων έχει ως εξής:

1. Αρχικοποίηση της βάσης γεγονότων
2. Εύρεση Κανόνων που ικανοποιούνται
   (Σύνολο Σύγκρουσης)
3. Επιλογή ενός Κανόνα

4. Εκτίμηση του Κανόνα
5. Ενημέρωση της ΜΕ
6. Αν κατάσταση λύσης, σταμάτα.
   Αλλιώς, πήγαινε στο βήμα 2.

και σχηματικά παριστάνεται στο Σχήμα 12.2. Τα βήματα 2-5 αποτελούν ένα κύκλο, που ονομάζεται κύκλος επιλογής-εκτέλεσης (select-execute cycle). Κατ' αρχήν εισάγονται στη ΒΓ τα δεδομένα του προβλήματος (δηλ. τα γνωστά γεγονότα). Στη συνέχεια (βήμα 2) βρίσκονται οι κανόνες των οποίων όλες οι συνθήκες/υποθέσεις ικανοποιούνται, δηλ. ταιριάζουν με γεγονότα στη ΒΓ. Εδώ συνήθως προκύπτουν περισσότεροι του ενός κανόνες, που δημιουργούν ένα σύνολο σύγκρουσης (conflict set), με την έννοια ότι «μάχονται» για το ποιος θα επιλεγεί. Από το σύνολο σύγκρουσης επιλέγεται (βήμα 3) ένας κανόνας, ο οποίος και «πυροδοτείται», όπως λέγεται, ή «ενεργοποιείται» (βήμα 4) και παράγει ένα (ενδιάμεσο συνήθως) συμπέρασμα, το οποίο και εισάγεται στη ΒΓ (βήμα 5). Αν αυτό αποτελεί τη ζητούμενη λύση (απάντηση), τότε σταματά η διαδικασία, αλλιώς συνεχίζει από το βήμα 2.

12.5 Δημιουργία Συνόλου Σύγκρουσης

Ας δούμε με ένα παράδειγμα, πώς δημιουργείται ένα σύνολο σύγκρουσης. Ας υποθέσουμε ότι έχουμε τη ΒΓ του Σχήματος 12.3.

```
ME={ (person Yannis Kostas Maria),
     (person Petros Pavlos Eleni),
     (person Giorgos Kostas Maria)
     (equal Yannis Yannis)
     (equal Kostas Kostas) … }
```

Σχήμα 12.3. Βάση Γεγονότων Παραδείγματος

```
if is(person ?name1 ?fath1 ?moth1) and
   is(person ?name2 ?fath2 ?moth2) and
   is(equal ?fath1 ?fath2) and
   is(equal ?moth1 ?moth2) and
   isnot(equal ?name1 ?name2)
then assert(brother ?name1 ?name2)
```

Σχήμα 12.4 Κανόνας Παραδείγματος

Ας υποθέσουμε επίσης ότι ένας από τους κανόνες της ΒΚ είναι αυτός που φαίνεται στο Σχήμα 12.4. Για να δούμε αν ο κανόνας αυτός είναι υποψήφιος για πυροδότηση, ή με άλλα λόγια αν κάποια στιγμιότυπά του είναι κανόνες υποψήφιοι για πυροδότηση, δηλ. ανήκουν στο σύνολο σύγκρουσης, κάνουμε το εξής. Εξετάζουμε αν υπάρχουν στιγμιοτυποποιήσεις της πρώτης συνθήκης. Πράγματι, η πρώτη συνθήκη ταιριάζει με τα εξής γεγονότα της ΒΓ:

(person Yannis Kostas Maria),
(person Petros Pavlos Eleni),
(person Giorgos Kostas Maria)

με τις εξής προσδέσεις των μεταβλητών:

{ ((?name1.Yannis), (?fath1.Kostas), (?moth1.Maria))
((?name1.Petros), (?fath1.Pavlos), (?moth1.Eleni))
((?name1.Giorgos), (?fath1.Kostas), (?moth1.Maria)) }

Ομοίως, η δεύτερη συνθήκη ταιριάζει με τα εξής γεγονότα της ΒΓ:

(person Yannis Kostas Maria),
(person Petros Pavlos Eleni),
(person Giorgos Kostas Maria)

με τις εξής προσδέσεις των μεταβλητών:

{ ((?name2.Yannis), (?fath2.Kostas), (?moth2.Maria))
((?name2.Petros), (?fath2.Pavlos), (?moth2.Eleni))
((?name2.Giorgos), (?fath2.Kostas), (?moth2.Maria)) }

Οι παραπάνω δύο συνθήκες δεν έχουν κοινές μεταβλητές και επομένως η διαδικασία ταιριάσματος της μιας δεν σχετίζεται με της άλλης. Οι επόμενες συνθήκες όμως έχουν μεταβλητές που υπάρχουν όλες στις δύο αυτές συνθήκες. Επειδή η κάθε μια από τις δύο αυτές συνθήκες έχουν τρία στιγμιότυπα (δηλ. τρία γεγονότα με τα οποία ταιριάζουν), υπάρχουν εννέα συνδυασμοί τους. Για κάθε ένα από τους συνδυασμούς

αυτούς παράγεται και ένας κανόνας χωρίς μεταβλητές, που ονομάζεται *στιγμιότυπο* (instance) του αρχικού κανόνα. Δηλαδή προκύπτουν εννέα (9) κανόνες-στιγμιότυπα. Π.χ. για τους συνδυασμούς (α) της πρώτης ομάδας προσδέσεων της πρώτης συνθήκης με την τρίτη ομάδα προσδέσεων της δεύτερης συνθήκης και (β) της τρίτης ομάδας προσδέσεων της πρώτης συνθήκης με την πρώτη ομάδα προσδέσεων της δεύτερης συνθήκης προκύπτουν οι παρακάτω κανόνες:

if is(person Yannis Kostas Maria) **and**
is(person Giorgos Kostas Maria) **and**
is(equal Kostas Kostas) **and**
is(equal Maria Maria) **and**
isnot(equal Yannis Giorgos)
then assert(brother Yannis Giorgos)

if is(person Giorgos Kostas Maria) **and**
is(person Yannis Kostas Maria) **and**
is(equal Kostas Kostas) **and**
is(equal Maria Maria) **and**
isnot(equal Giorgos Yannis)
then assert(brother Giorgos Yannis)

Αυτοί οι κανόνες είναι και οι μόνοι από τους εννέα των οποίων ικανοποιούνται οι συνθήκες και επομένως αποτελούν στοιχεία του συνόλου σύγκρουσης.

Αυτή η διαδικασία γίνεται για κάθε κανόνα της ΒΚ, οπότε γίνεται αντιληπτό πόσο πολύπλοκη καθίσταται η διαδικασία της παραγωγής του συνόλου σύγκρουσης όταν έχουμε ικανό αριθμό κανόνων με αρκετές συνθήκες ο καθένας. Για την αντιμετώπιση αυτής της πολυπλοκότητας έχει βρεθεί ένας αλγόριθμος, ο αλγόριθμος Rete, τον οποίον παρουσιάζουμε στην επόμενη ενότητα.

12.6 Αλγόριθμος Rete

12.6.1 Γενικά

Ο αλγόριθμος Rete επιτυγχάνει να βελτιώσει την απόδοση της διαδικασίας εξαγωγής συμπερασμάτων σε συστήματα κανόνων που στηρίζονται στο ταίριασμα προτύπων που χρησιμοποιούν ορθή αλυσίδωση σαν μέθοδο εξαγωγής συμπερασμάτων.

Βασική αιτία της μειωμένης αποδοτικότητας των παραπάνω συστημάτων είναι η ανάγκη επαναπροσδιορισμού του συνόλου σύγκρουσης μετά από κάθε εκτέλεση κανόνα. Η αναζήτηση για γεγονότα που ταιριάζουν με τις συνθήκες κάθε κανόνα

είναι πολύ χρονοβόρα μια και μια απ' ευθείας προσέγγιση απαιτεί εξαντλητική αναζήτηση στη ΒΓ. Ο αλγόριθμος Rete έρχεται να επιταχύνει αυτή τη διαδικασία. Βασίζεται στο γεγονός ότι συνήθως με την εκτέλεση ενός κανόνα μόνο μικρές αλλαγές γίνονται στη βάση γεγονότων και έτσι το σύνολο των κανόνων που ικανοποιούνται σε κάθε κύκλο αλλάζει ελάχιστα. Έτσι, αντί να συγκρίνει τους κανόνες με τα γεγονότα για να παράγει μια λίστα ικανοποιήσιμων κανόνων (σύνολο σύγκρουσης), ελέγχει τις αλλαγές που υπόκειται η λίστα αυτή σε κάθε κύκλο.

Ο αλγόριθμος Rete επιτυγχάνει επιτάχυνση της διαδικασίας του ταιριάσματος προτύπων δημιουργώντας 2 δίκτυα:

- Το δίκτυο προτύπων (pattern network)
- Το δίκτυο συνδέσεων (join network)

Το δίκτυο προτύπων (ΔΠ) αποτελείται από ένα σύνολο δέντρων (δίκτυο) που σχηματίζονται από όλα τα τμήματα των υποθέσεων των κανόνων του συστήματος. Η ρίζα κάθε τέτοιου δένδρου είναι το πρώτο στοιχείο του πρώτου προτύπου-υπόθεσης ενός κανόνα. Οι επόμενοι κόμβοι του δένδρου αναπαριστούν με τη σειρά τα υπόλοιπα στοιχεία του προτύπου. Ο αλγόριθμος χτίζοντας το δίκτυο εισάγει κάθε νέο πρότυπο επαναχρησιμοποιώντας όσο είναι δυνατό τα ήδη υπάρχοντα δέντρα. Όταν το πρώτο στοιχείο ενός προτύπου υπάρχει ήδη ως ρίζα σε κάποιο δένδρο, ο αλγόριθμος δεν δημιουργεί νέο δένδρο, αλλά κατεβαίνει το ήδη υπάρχον μέχρι να συναντήσει κόμβο που συμβολίζει διαφορετικό αντικείμενο από το αντίστοιχο του προτύπου. Εκεί δημιουργεί μια διακλάδωση και εισάγει τους κόμβους που αντιστοιχούν στα νέα αντικείμενα.

Όταν ολοκληρωθεί το ΔΠ ο αλγόριθμος κατασκευάζει το δίκτυο συνδέσμων (ΔΣ). Το ΔΣ συνδέει τα φύλλα των δένδρων του δικτύου προτύπων που ανήκουν στις υποθέσεις του ίδιου κανόνα. Αν δύο μονοπάτια που συνδέονται με αυτό τον τρόπο περιέχουν κοινές μεταβλητές, τότε ο αλγόριθμος ελέγχει αν η μεταβλητή έχει την ίδια τιμή και στα δύο μονοπάτια.

Ο αλγόριθμος χρησιμοποιεί τα παραπάνω δυο δίκτυα για να βρει ποιοι κανόνες ικανοποιούνται σε κάθε κύκλο ορθής αλυσίδωσης. Στον πρώτο κύκλο περνά όλα τα γεγονότα της ΒΓ από το δίκτυο. Για κάθε γεγονός βρίσκεται η ρίζα, αν υπάρχει, που συμβολίζει το πρώτο στοιχείο της πλειάδας, ή μια μεταβλητή που μπορεί να πάρει αυτό το στοιχείο ως τιμή. Το γεγονός κατεβαίνει το δέντρο μέχρι το κατώτερο σημείο που είναι δυνατό ταιριάζοντας τα αντίστοιχα στοιχεία και δίνοντας τιμές στις αντίστοιχες μεταβλητές. Αφού όλα τα γεγονότα περάσουν από το δίκτυο προτύπων,

στη συνέχεια περνούν από το δίκτυο συνδέσεων, που λειτουργώντας σαν φίλτρο ελέγχει τις όμοιες μεταβλητές δίνοντας το σύνολο των κανόνων που ικανοποιούνται. Όταν εκτελείται ένας κανόνας, ως αποτέλεσμα μπορεί να είναι η προσθήκη, η διαγραφή ή η μεταβολή ενός γεγονότος. Ο αλγόριθμος καθορίζει το σύνολο των ικανοποιήσιμων κανόνων του επόμενου κύκλου ανανεώνοντας στο δίκτυο μόνο αυτά τα γεγονότα.

12.6.2 Παράδειγμα

Θα παρουσιάσουμε τη δημιουργία των δύο δικτύων με βάση ένα παράδειγμα. Έστω ότι έχουμε ένα σύστημα κανόνων τύπου ταιριάσματος προτύπων, όπου χρησιμοποιείται (χάριν απλότητας) μια μορφή προτύπου, η

(τύπος επίπλου, χρώμα, μέγεθος, βάρος)

και το σύστημα αποτελείται (χάριν απλότητας) από δύο μόνο κανόνες, τους εξής:

R1: **if** is(table ?c big ?w1) **and**
is(bed ?c medium light) **and**
is(chair ?c medium ?w2)
then is(Room-Decoration PERFECT)

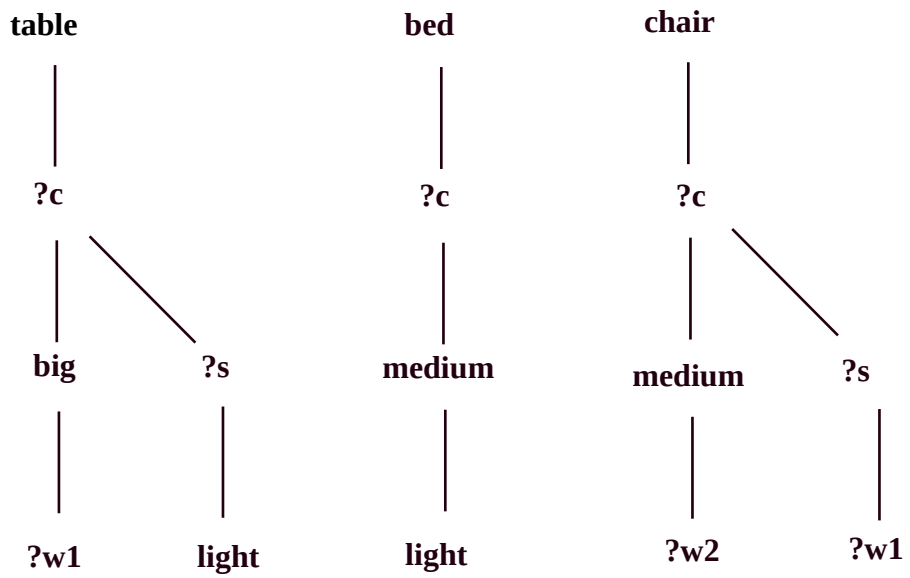
R2: **if** is(table ?c ?s light) **and**
is(chair ?c ?s ?w1)
then is(Room-Decoration GOOD)

οι οποίοι αποφαίνονται για την ποιότητα της διακόσμησης ενός δωματίου. Επίσης, έστω η παρακάτω ΒΓ (που αναπαριστά τα στοιχεία ενός δωματίου):

1. (bed brown big heavy)
2. (bed black medium light)
3. (bed black medium heavy)
4. (table black big heavy)
5. (table brown small light)
6. (table yellow big light)
7. (table black small light)
8. (chair black medium light)
9. (chair brown small light)
10. (chair black small light)

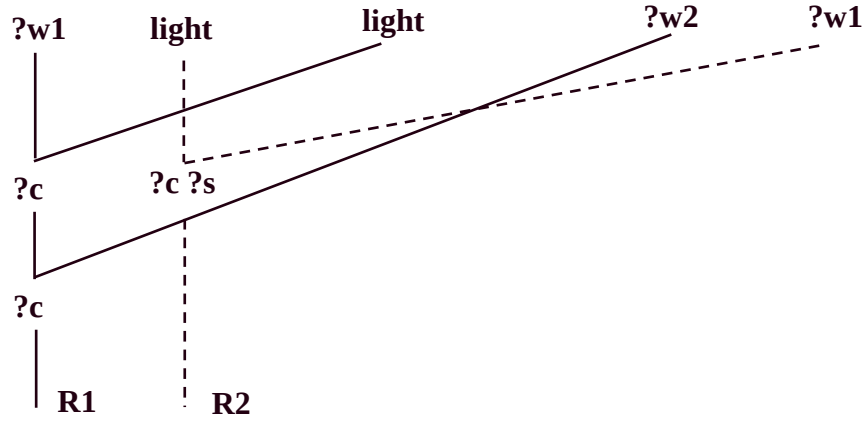
Κατ' αρχήν σχηματίζεται το ΔΠ, όπως αυτό φαίνεται στο Σχήμα 12.5. Στο ΔΠ υπάρχουν τόσα δέντρα όσα και τα διαφορετικά πρώτα στοιχεία των συνθηκών των κανόνων. Εδώ υπάρχουν τρία (με ρίζες τα table, bed και chair). Σε κάθε δέντρο

αναπαριστούμε τις συνθήκες των κανόνων, τοποθετώντας σαν κόμβους τα στοιχεία κάθε συνθήκης με τη σειρά. Οπότε π.χ. η πρώτη συνθήκη του R1 παριστάνεται από τη διαδρομή table-?c-big-?w1. Τα κοινά στοιχεία, όπως είπαμε, χρησιμοποιούνται μια φορά σε κάθε δέντρο. Π.χ. η πρώτη συνθήκη του R2 ακολουθεί την ίδια διαδρομή στο δέντρο table μέχρι και το στοιχείο '?c', μετά παριστάνεται με μια διακλάδωση στο δέντρο table (βλ. σχήμα 12.5).



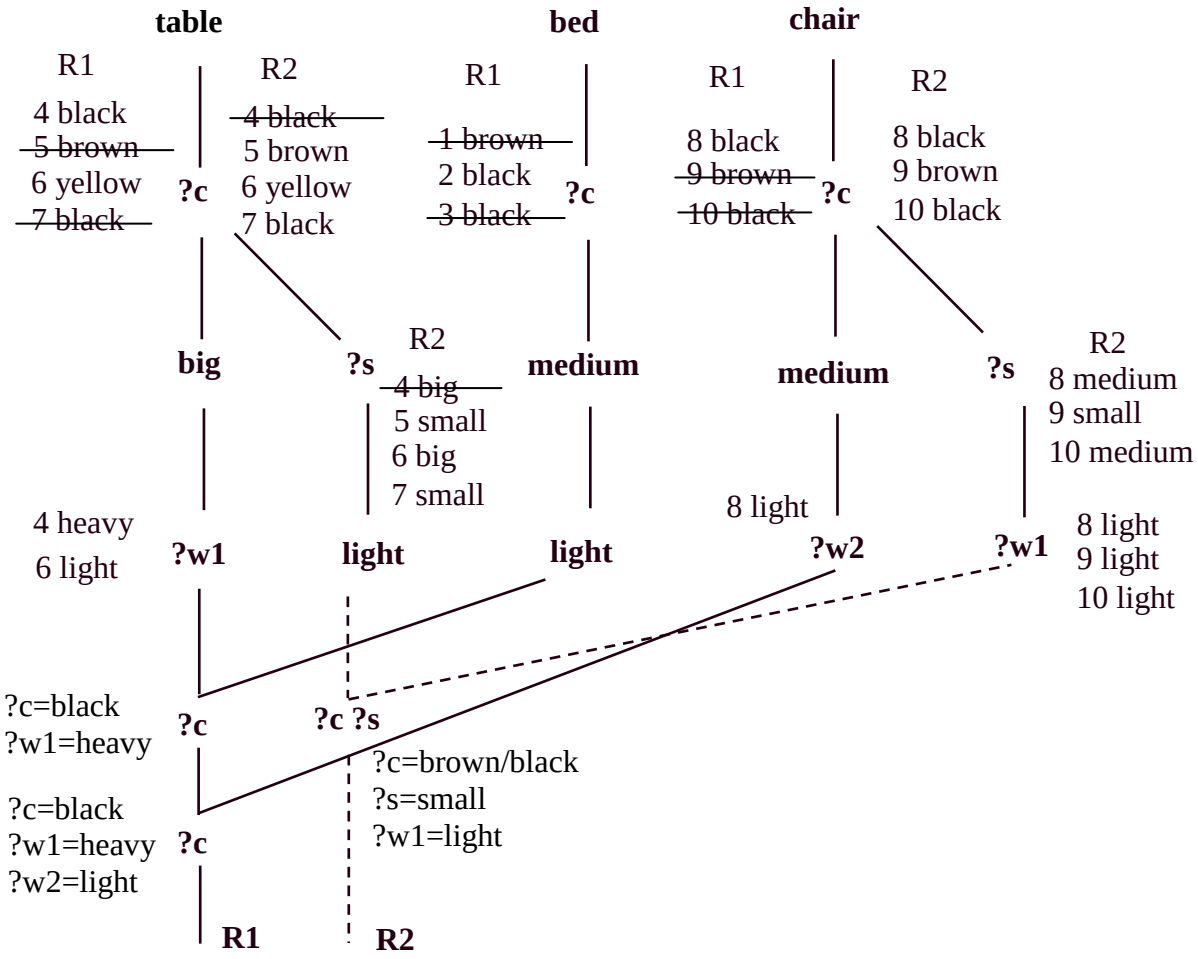
Σχήμα 12.5 Δίκτυο Προτύπων Παραδείγματος

Στη συνέχεια σχηματίζεται το δίκτυο συνδέσεων, όπως φαίνεται στο Σχήμα 12.6.



Σχήμα 12.6 Δίκτυο Συνδέσεων Παραδείγματος

Το δίκτυο αυτό κατασκευάζεται ως εξής. Οι συνθήκες-πρότυπα (table, ?c, big, ?w1) και (bed, ?c, medium, light) ανήκουν στον πρώτο κανόνα και συνδέονται με το ‘and’. Για να ικανοποιηθεί ο κανόνας πρέπει η μεταβλητή «?c» να έχει την ίδια τιμή και στα δύο patterns. Γι’ αυτό, συνδέονται τα δύο μονοπάτια που αντιστοιχούν στα δύο πρότυπα με έναν κόμβο που ελέγχει την τιμή της μεταβλητής «?c». Τώρα σε αυτά τα δύο μονοπάτια θα συνδέσουμε αυτό που αντιστοιχεί στην Τρίτη συνθήκη-πρότυπο του κανόνα, ελέγχοντας πάλι τη μεταβλητή «?c». Όμοια συνδέουμε με τους ανάλογους ελέγχους τις συνθήκες του δεύτερου κανόνα.



Σχήμα 12.7 Διαδικασία Εύρεσης Ικανοποιήσιμων Κανόνων

Το επόμενο βήμα είναι το πέρασμα όλων των γεγονότων από τα δυο δίκτυα. Κάθε γεγονός αρχίζει να κατεβαίνει το δίκτυο προτύπων ξεκινώντας από την ρίζα που ταιριάζει με το πρώτο αντικείμενο της τετράδας και θα κατέβει μέχρι το μέγιστο

βάθος που μπορεί να φτάσει ταιριάζοντας τα αντίστοιχα στοιχεία με τους κόμβους του δέντρου. Στη διαδρομή αυτή το γεγονός αποδίδει τιμές στις μεταβλητές που συναντά. Το πρώτο γεγονός θα μπει στο δένδρο με ρίζα το «bed», θα δώσει στη μεταβλητή «?c» την τιμή «brown» και θα σταματήσει εκεί, μια και ο επόμενος κόμβος του δέντρου είναι «medium», που δεν ταιριάζει με το «big» του γεγονότος. Ομοίως περνάνε όλα τα γεγονότα από το δίκτυο προτύπων.

Όταν όλα τα γεγονότα έχουν περάσει από το δίκτυο προτύπων, αυτά που έχουν φτάσει στα φύλλα θα περάσουν από το δίκτυο συνδέσεων. Κατά το πέρασμα από το δίκτυο συνδέσεων τα γεγονότα φιλτράρονται στους κόμβους συμβολής έτσι ώστε να περνάνε μόνο οι συνδυασμοί γεγονότων που δίνουν ίδια τιμή στις ελεγχόμενες μεταβλητές. Όταν ένα σύνολο γεγονότων φτάσει στον κατώτερο κόμβο του δικτύου τότε ο αντίστοιχος κανόνας είναι ικανοποιήσιμος. Το σύστημα ξέρει ακριβώς ποιος συνδυασμός γεγονότων τον ικανοποιεί και με ποιες αναθέσεις τιμών στις μεταβλητές. Η διαδικασία αυτή αποτυπώνεται στο σχήμα 12.7, όπου οι αριθμοί δίπλα από τις διάφορες τιμές αναφέρονται στο αντίστοιχο γεγονός στη ΒΓ. Οι διαγραμμισμένες τιμές είναι αυτές που έχουν απορριφθεί, δηλ. δεν περνούν το φίλτρο.

13 ΣΥΝΤΕΛΕΣΤΕΣ ΒΕΒΑΙΟΤΗΤΑΣ

13.1 Ορισμός

Οι συντελεστές βεβαιότητας (certainty factors) είναι μια μέθοδος αναπαράστασης της αβεβαιότητας στην εξαγωγή συμπερασμάτων, που χρησιμοποιείται στον βασισμένο σε κανόνες προγραμματισμό. Πρόκειται για εμπειρική μέθοδο, δηλ. μέθοδο που δεν στηρίζεται στη θεωρία πιθανοτήτων, όπως άλλες παρόμοιες μέθοδοι (π.χ. κανόνες Bayes). Οι συντελεστές βεβαιότητας πρωτοεισήχθησαν στο βασισμένο σε κανόνες έμπειρο σύστημα (ΕΣ) MYCIN (Shortliffe and Buchanan, 1975), το οποίο ήταν ένα ΕΣ για διάγνωση μολύνσεων του αίματος και λήψη απόφασης για κατάλληλη φαρμακευτική αγωγή για την αντιμετώπισή τους.

Συνετελεστής βεβαιότητας cf (certainty factor, cf) είναι ένας αριθμός ($-1 \leq cf \leq 1$) που παριστάνει το βαθμό βεβαιότητας του εμπειρογνώμονα ότι μια υπόθεση h ισχύει δεδομένου ενός στοιχείου/γεγονότος e:

if e

then h (cf)Φυσικά ο κανόνας μπορεί να είναι πιο σύνθετος, δηλ. να έχει περισσότερα του ενός γεγονότα/στοιχεία συνδεόμενα μεταξύ τους με λογικό and ή or.

Κάθε συντελεστής βεβαιότητας ορίζεται με βάση δύο μεγέθη/συναρτήσεις, το *μέτρο βεβαιότητας* (measure of belief: MB) και το *μέτρο αβεβαιότητας* (measure of disbelief: MD), ως εξής:

$$cf(h,e) = \frac{MB(h,e) - MD(h,e)}{1 - \min\{MB(h,e), MD(h,e)\}}$$

τα οποία με τη σειρά τους ορίζονται ως εξής:

Μέτρο βεβαιότητας (measure of belief):

$$MB(h,e) = \begin{cases} 1 & \text{αν } p(h) = 1 \\ \max\{0, \dfrac{p(h/e) - p(h)}{1 - p(h)}\} & \text{αλλιώς} \end{cases}$$

Μέτρο αβεβαιότητας (measure of disbelief):

$$MD(h,e) = \begin{cases} 1 & \text{αν } p(h) = 0 \\ \max\{0, \dfrac{p(h) - p(h/e)}{p(h)}\} & \text{αλλιώς} \end{cases}$$

για τα οποία ισχύει $0 \leq MB(h,e), MD(h,e) \leq 1$, οπότε προκύπτει ότι $-1 \leq cf(h, e) \leq 1$.Σε πραγματικές εφαρμογές, συνήθως, οι συντελεστές βεβαιότητας προσδιορίζονται από τον εμπειρογνώμονα, είναι δηλαδή εκτίμηση του εμπειρογνώμονα.

13.2 Αβέβαιη Παρατήρηση

Πολλές φορές όμως δεν είναι αβέβαιο μόνο το αποτέλεσμα (υπόθεση) του κανόνα, αλλά είναι αβέβαιη και η παρατήρηση ενός γεγονότος/στοιχείου. Τοτε, διακρίνουμε τις παρακάτω περιπτώσεις:

Κανόνας με απλό στοιχείο

if e (cf_e)
then h (cf_h)

Τότε, ο συνδυασμένος συντελεστής βεβαιότητας του κανόνα είναι:

$cf = cf_e * cf_h$

Κανόνας με πολλαπλά στοιχεία:

α) Με σύξευξη στοιχείων

if e1 (cf_1) and e2 (cf_2)
then h (cf_h)

οπότε ο συνδυασμένος συντελεστής βεβαιότητας είναι:

$$cf_e = \min\{cf_1, cf_2\}$$

$$cf = cf_e * cf_h$$

β) Με διάζευξη στοιχείων

if e1 (cf_1) or e2 (cf_2)
then h (cf_h)

οπότε ο συνδυασμένος συντελεστής βεβαιότητας είναι:

$$cf_e = \max\{cf_1, cf_2\}$$

$$cf = cf_e * cf_h$$

Παράδειγμα:

if sky is clear
and forecast is sunny (0.8)
then action is ‘leave-umbrella’ (0.8)
cf_1=1.0, cf_2=0.8, cf_h=0.8$cf_e = \min\{1.0, 0.8\} = 0.8$
$cf = cf_e*cf_h = 0.8*0.8 = 0.64$Κανόνες με Πολλαπλά Στοιχεία

Κανόνες με ίδιο συμπέρασμα

Είναι σύνηθες σε συστήματα κανόνων με χρήση συντελεστών βεβαιότητας να έχουμε κανόνες με το ίδιο συμπέρασμα. Στην περίπτωση αυτή θα πρέπει να συνδυαστούν οι συντελεστές βεβαιότητας των δύο κανόνων στο τελικό συμπέρασμα. Έστω οι παρακάτω κανόνες:

if e1	if e2
then h (cf_{h1})	then h (cf_{h2})

Τότε, ο συνδυασμένος συντελεστής βεβαιότητας cf του συμπεράσματος υπολογίζεται με βάση τα παρακάτω:

$$cf = \begin{cases} cfh1+cfh2*(1-cfh1) & \text{αν } cfh1, cfh2 > 0 \\ cfh1+cfh2*(1+cfh1) & \text{αν } cfh1, cfh2 < 0 \\ (cfh1+cfh2)/(1-\min\{|cfh1|,|cfh2|\}) & \text{αν } cfh1*cfh2 < 0 \end{cases}$$

<u>Διαδοχικοί κανόνες</u>

Μια άλλη περίπτωση συνδυασμού κανόνων είναι να υπάρχουν διαδοχικοί κανόνες, δηλ. κανόνες όπου το συμπέρασμα του ενός αποτελεί στοιχείο του άλλου. Π.χ., έστω οι δύο κανόνες:

if e1	if e2
then e2 (cf_1)	then h (cf_2)

Στην περίπτωση αυτή, ο συνδυασμένος συντελεστής βεβαιότητας cf υπολογίζεται ως εξής:

$$cf_e = \max\{0, cf_1\}$$
$$cf = cf_e * cf_2$$

<u>Παράδειγμα</u>:

if today is rain
then tomorrow is rain (0.5)

if today is rain
and temperature is high
then tomorrow is rain (0.7)

Επειδή cfh1, cfh2 >0 είναι cf = cfh1+cfh2*(1-cfh1)= 0.5 + 0.7*(1-0.5) = 0.85Εξαγωγή Συμπερασμάτων

Ας δούμε τώρα πως γίνεται η εξαγωγή συμπερασμάτων σ' ένα ΕΣ με συντελεστές βεβαιότητας. Ας θεωρήσουμε ένα υποτυπώδες ΕΣ, που κάνει πρόβλεψη του καιρού της επόμενης ημέρας με βάση τα στοιχεία της τρέχουσας ημέρας και διαθέτει τους παρακάτω κανόνες:

R1
if today is rain
then tomorrow is rain (0.5)
R2
if today is dry
then tomorrow is dry (0.5)
R3
if today is rain
and rainfall is low
then tomorrow is dry (0.6)
R4
if today is rain
and rainfall is low
and temperature is low
then tomorrow is dry (0.7)
R5
if today is dry
and temperature is high
then tomorrow is rain (0.65)

Για την εξαγωγή συμπερασμάτων, το ΕΣ κάνει ερωτήσεις στο χρήστη, για να πάρει τιμές για τις μεταβλητές/παραμέτρους. Ας παρακολουθήσουμε μια τέτοια διαδικασία (όπου ΧΡ παριστάνει τον χρήστη του συστήματος, ΜΕ τη μνήμη εργασίας και

ακολουθείται αλυσίδωση προς τα εμπρός). Πρέπει να σημειωθεί ότι στην περίπτωση των συντελεστών βεβαιότητας λαμβάνονται υπ' όψιν όλοι οι κανόνες που έχουν την ίδια μεταβλητή συμπεράσματος και μπορούν να πυροδοτηθούν.

ΕΣ: What is the weather today?
ΧΡ: rain

Πυροδοτείται ο R1: cf_{R1}= cfe*cfh= 1.0*0.5= 0.5
ΜΕ= {tomorrow is rain (0.5)}
ΕΣ: What is the rainfall today?
ΧΡ: low
ΕΣ: To what degree you believe the rainfall is low?
ΧΡ: 0.85

Πυροδοτείται ο R3: cfR3= min(cfe1, cfe2)*cfh= min(1.0, 0.85)*0.6 = 0.51
ΜΕ= {tomorrow is rain (0.5)
tomorrow is dry (0.51)}
ΕΣ: What is the temperature today?
ΧΡ: low
ΕΣ: To what degree you believe the temperature is low?
ΧΡ: 0.95

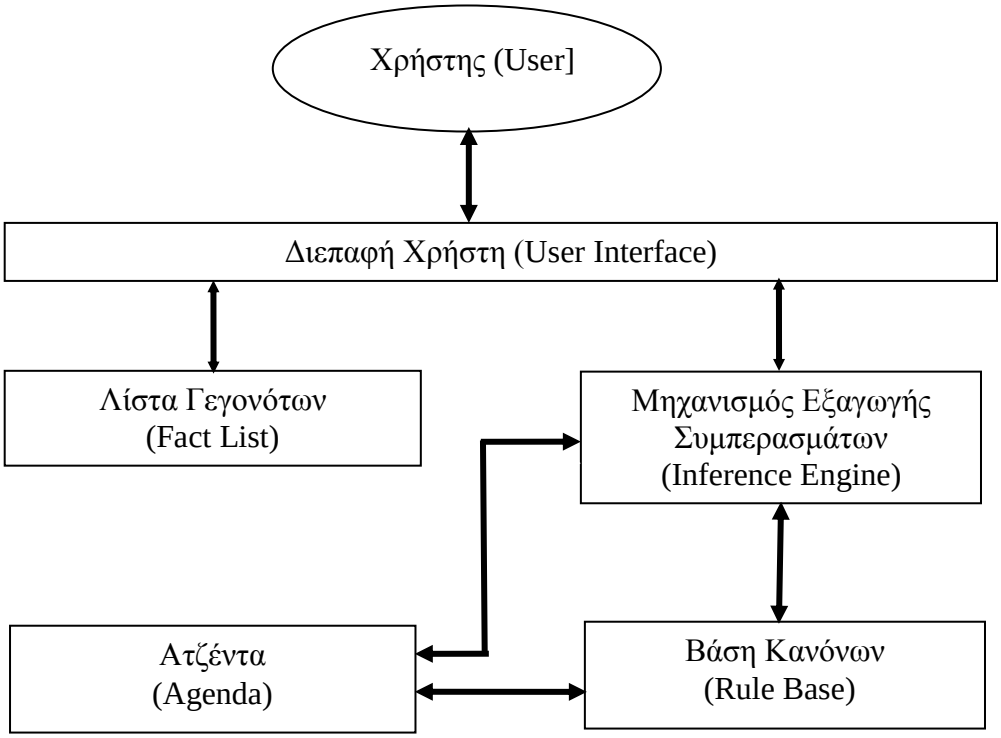
Πυροδοτείται ο R4: cfR4= min(cfe1, cfe2, cf3) * cfh= min(1.0,0.85,0.95)*0.7=0.595
Επειδή όμως ήδη υπάρχει ίδιο συμπέρασμα στη ΜΕ από τον R3 έχουμε:
cfR34= cfR3+cfR4*(1-cfR3) = 0.51+0.595*(1-0.51)= 0.8
ΜΕ= {tomorrow is rain (0.5)
tomorrow is dry (0.8)}
Επόμένως, η απάντηση είναι ότι είναι βεβαιότερο ότι αύριο δεν θα βρέξει (cf = 0.8) από το να βρέξει (cf = 0.5)

14 CLIPS ΚΑΙ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΜΕ ΚΑΝΟΝΕΣ

Το CLIPS (C Language Integrated Production System) είναι ένα περιβάλλον που αναπτύχθηκε από τη NASA και αποτελεί μια χαμηλού κόστους πλατφόρμα ανάπτυξης έμπειρων συστημάτων, αντικαθιστώντας τα ήδη υπάρχοντα συστήματα τα οποία βασίζονταν στη γλώσσα LlSP και προσφέρει δυνατότητες για προγραμματισμό με κανόνες, συναρτήσεις και αντικείμενα. Με το CLIPS επομένως είναι δυνατόν να υλοποιηθούν εφαρμογές που βασίζονται σε τεχνικές ευρετικού, διαδικαστικού αλλά και αντικειμενοστραφούς προγραμματισμού. Το CLIPS διαθέτει μια αντικειμενοστρεφή γλώσσα προγραμματισμού, που ονομάζεται COOL (CLIPS Object-Oriented Language). Η τελευταία έκδοση του CLIPS είναι η 6.21. Είναι σημαντικό να σημειωθεί επιπρόσθετα ότι οι διάφορες εκδόσεις του είναι διαθέσιμες στο διαδίκτυο με τη μορφή ανοικτού κώδικα γραμμένες σε γλώσσα C. Στο κεφάλαιο αυτό θα μας απασχολήσει το CLIPS κυρίως ως εργαλείο ανάπτυξης έμπειρων συστημάτων.

14.1 Δομή του CLIPS

Το CLIPS μπορεί να θεωρηθεί σαν ένα γενικό εργαλείο ανάπτυξης συστημάτων λογισμικού. Ανάμεσα στα άλλα, το CLIPS εμπεριέχει ένα κέλυφος έμπειρου συστήματος, και επομένως διαθέτει τα βασικά μέρη ενός έμπειρου συστήματος (Σχήμα 14.1):



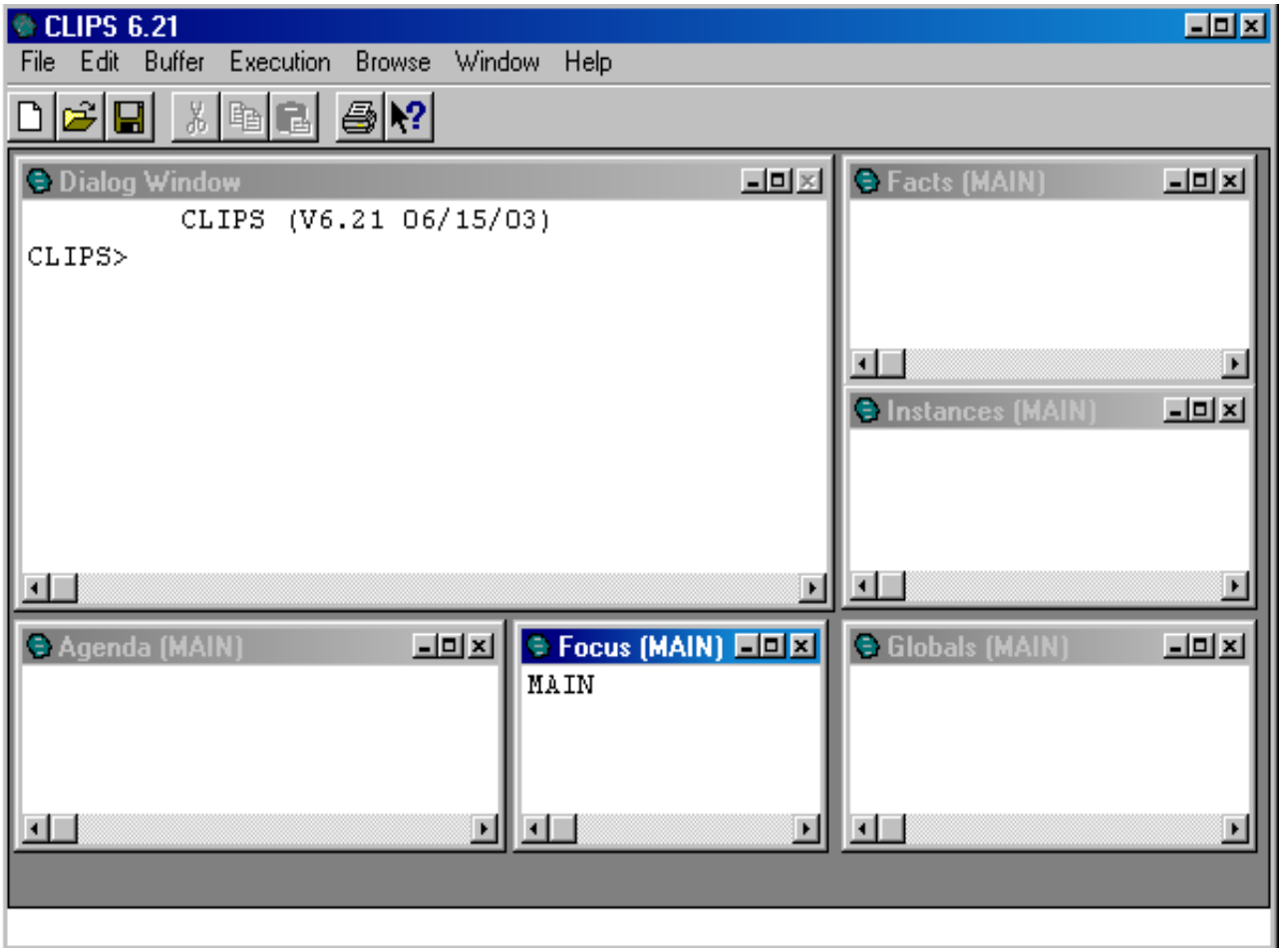
Σχήμα 14.1. Δομή Κελύφους ΕΣ του CLIPS

- *Λίστα γεγονότων (fact-list).* Είναι ο χώρος στον οποίο αποθηκεύονται τα γεγονότα (facts), τόσο εκείνα που ορίζονται κατά την εκκίνηση του συστήματος (δεδομένα ή αρχικές συνθήκες), όσο και εκείνα που δημιουργούνται κατά την εκτέλεση του. Κάθε γεγονός παίρνει και ένα χαρακτηριστικό ακέραιο αριθμό με την καταχώρησή του στη λίστα, που το χαρακτηρίζει μοναδικά.
- *Βάση κανόνων (rule-base).* Αποτελείται από ένα σύνολο κανόνων, που αναπαριστούν γνώση για την εξαγωγή συμπερασμάτων. Οι κανόνες είναι συνήθως αποθηκευμένοι σε κάποιο αρχείο απλού κειμένου, το οποίο φορτώνεται στο σύστημα.
- *Μηχανισμός εξαγωγής συμπερασμάτων (inference engine).* Είναι ο μηχανισμός με τη βοήθεια του οποίου χρησιμοποιούνται οι κανόνες για εξαγωγή συμπερασμάτων. Ο μηχανισμός αυτός υλοποιεί την ορθή αλυσίδωση και προσφέρει αρκετές στρατηγικές επίλυσης συγκρούσεων (conflict resolution strategies).

- *Ατζέντα (agenda).* Είναι μια στοίβα στην οποία αποθηκεύονται κάθε φορά οι υποψήφιοι για πυροδότηση κανόνες, δηλ. το σύνολο σύγκρουσης (conflict set). Η τοποθέτηση των κανόνων γίνεται με βάση την ακολουθούμενη στρατηγική επίλυσης συγκρούσεων, οπότε επιλέγεται κάθε φορά ο ευρισκόμενος στην κορυφή της στοίβας.

14.2 Το περιβάλλον του CLIPS

Το περιβάλλον του CLIPS εκκινεί σε περιβάλλον MS-Windows με την εκτέλεση του αρχείου CLIPSWIN.EXE. Αμέσως μετά εμφανίζεται στην οθόνη το γραφικό περιβάλλον της γλώσσας, όπως φαίνεται στο Σχήμα 14.2, όπου είναι ανοικτά όλα τα παράθυρα του περιβάλλοντος.



Σχήμα 14.2. Το περιβάλλον του CLIPS 6.21

Από τα παράθυρα αυτά, μπορεί να είναι ανοικτά ταυτόχρονα όσα θέλουμε. Στο Dialog Window γίνεται το τρέξιμο των προγραμμάτων και εμφανίζονται τα αποτελέσματα. Στο Facts εμφανίζονται τα περιεχόμενα της λίστας γεγονότων, ενώ στο Agenda τα περιεχόμενα της ατζέντας. Αυτά είναι και τα σπουδαιότερα παράθυρα.

Για να «φορτωθεί» ένα πρόγραμμα CLIPS επιλέγεται από το μενού *File* η επιλογή *Load* και μέσα από το σχετικό παράθυρο επιλέγεται το επιθυμητό αρχείο που συνήθως έχει την κατάληξη *.clp.

Για να εκτελεστεί κάποιο πρόγραμμα γραμμένο σε CLIPS κατ' αρχήν επιλέγουμε και εκτελούμε την εντολή Reset από το μενού της επιλογής Execution του CLIPS. Η εντολή αυτή καταχωρεί στη μνήμη όλα τα γεγονότα τα οποία περιγράφονται στο αρχείο που φορτώθηκε. Στη συνέχεια, επιλέγεται και εκτελείται η εντολή Run από το ίδιο μενού.

Έξοδος από το CLIPS επιτυγχάνεται με χρήση της εντολής *exit* :

CLIPS> (exit)

Οι παρενθέσεις προσδιορίζουν την *exit* ως *εντολή* και την διαχωρίζουν από το *σύμβολο exit*, που δεν είναι σε παρενθέσεις και έχει εντελώς διαφορετική σημασία.

14.3 Ο Κύκλος Εξαγωγής Συμπερασμάτων

Ένα πρόγραμμα στο CLIPS είναι ένα σύνολο από κανόνες και γεγονότα και η εκτέλεση του συνίσταται σε μια ακολουθία από πυροδοτήσεις κανόνων, των οποίων οι συνθήκες ικανοποιούνται. Η ικανοποίηση των συνθηκών γίνεται μέσω ταυτοποίησης τους με τα γεγονότα που υπάρχουν στη λίστα γεγονότων. Η εκτέλεση τερματίζεται όταν δεν υπάρχουν άλλοι κανόνες προς πυροδότηση ή όταν κληθεί συγκεκριμένη εντολή τερματισμού. Ο κύκλος λειτουργίας μιας εφαρμογής στο εργαλείο αυτό είναι ο τυπικός κύκλος λειτουργίας ενός συστήματος παραγωγής :

1. Εύρεση όλων των κανόνων των οποίων οι συνθήκες ικανοποιούνται και προσθήκη τους στην ατζέντα (agenda - conflict set).
2. Αν η ατζέντα είναι κενή ή ο μέγιστος αριθμός πυροδοτήσεων έχει συμπληρωθεί τότε η εκτέλεση τερματίζεται.
3. Διάταξη των κανόνων στην ατζέντα με βάση τη στρατηγική επίλυσης συγκρούσεων (conflict resolution strategy).
4. Πυροδότηση του πρώτου κανόνα στην ατζέντα.

5. Επιστροφή στο βήμα 1, εκτός αν υπάρχει εντολή τερματισμού (halt).

Το CLIPS ακολουθεί την ορθή αλυσίδωση σαν μέθοδο εξαγωγής συμπερασμάτων.

14.4 Τα Βασικά Στοιχεία του CLIPS

Τα CLIPS διαθέτει μια σειρά από στοιχεία κατάλληλα για την υλοποίηση έμπειρων συστημάτων που αφορούν τα δεδομένα, τις μεταβλητές, τους κανόνες, τις συναρτήσεις κλπ. Η σύνταξη που χρησιμοποιεί το CLIPS είναι επηρεασμένη από αυτή της γλώσσας LISP, χρησιμοποιεί δηλ. σαν βάση διαχωρισμού των δομικών μονάδων τις παρενθέσεις. Ο ορισμός των διαφόρων στοιχείων του CLIPS θυμίζει ορισμό συναρτήσεων της LISP.

14.4.1 Τύποι δεδομένων

Οι βασικοί τύποι δεδομένων που υποστηρίζονται από το CLIPS είναι:

Σύμβολα (symbols): Οποιαδήποτε ακολουθία εκτυπώσιμων χαρακτήρων που δεν ξεκινά από <, I , &, (,), $, ?, +, - και δεν περιέχει κανένα από τους <, | , &, (,), ; .

Αλφαριθμητικά (strings): Οποιαδήποτε ακολουθία εκτυπώσιμων χαρακτήρων ανάμεσα σε διπλά εισαγωγικά. Π.χ. "This is a program" , “2A”

Αριθμοί (numbers): 23, -23, +23 (*integers*)
23.34, +23.0, 23e4, -23.2e-5 (*floats*)

Σχόλια: από ";" μέχρι το τέλος της γραμμής

14.4.2 Μεταβλητές

Οι μεταβλητές που μπορούμε να χρησιμοποιήσουμε στο CLIPS μπορεί να είναι:

Μονότιμες (singlevalue) : ξεκινούν με ? (Π.χ. ?x, ?day). Τιμές: 32, flight34, mary, “a12”

Πολλαπλών τιμών (multivalue) : ξεκινούν με $?. Π.χ. $?days. Τιμές : (28 29 30 31), (mon tues wedn)

Καθολικές μεταβλητές (global variables)

Είναι προσπελάσιμες από παντού. Ορίζονται με τη δήλωση defglobal:

(defglobal ?*<σύμβολο>* = <έκφραση>)

Π.χ. (defglobal ?*x* = 0)

Τοπικές μεταβλητές (local variables)

Είναι προσπελάσιμες μόνο μέσα από τη δομή (π.χ. κανόνας, συνάρτηση) που χρησιμοποιούνται.

Οι μεταβλητές χρησιμοποιούνται και στις συνθήκες και στις ενέργειες των κανόνων και φυσικά στις συναρτήσεις χρήστη και είναι συνήθως τοπικές μεταβλητές. Τιμές παίρνουν οι μεταβλητές που βρίσκονται στις συνθήκες ενός κανόνα μέσω της διαδικασίας του ταιριάσματος/ενοποίησης (matching) με γεγονότα από τη λίστα γεγονότων. Οι τιμές αυτές αποδίδονται στις ενέργειες των κανόνων είτε μέσω εκτυπώσεων στην οθόνη είτε μέσω εισαγωγής νέων γεγονότων.

14.4.3 Γεγονότα (facts)

Τα γεγονότα είναι αποθηκευμένα στη λίστα γεγονότων, που αντιστοιχεί στη μνήμη εργασίας των συστημάτων παραγωγής. Η λίστα αυτή συνήθως αλλάζει περιεχόμενο, διότι κατά τη διάρκεια εκτέλεσης ενός προγράμματος εισάγονται νέα και διαγράφονται παλιά γεγονότα.

Κάθε γεγονός στη λίστα έχει ένα μοναδικό αριθμό που το χαρακτηρίζει, που ονομάζεται δείκτης του γεγονότος (fact index). Ο δείκτης αυτός αποδίδεται από το σύστημα κατά τη δημιουργία του γεγονότος, και αποτελεί και μια ετικέτα που δείχνει την παλαιότητά του. Η διαγραφή κάποιου γεγονότος γίνεται με βάση τον δείκτη του.

Τα γεγονότα αποτελούν τα δεδομένα του προβλήματος που επιλύει ένα πρόγραμμα. Είναι η αρχική γνώση στην οποία βασίζεται ώστε να εξάγει συμπεράσματα. Τα γεγονότα δεν περιέχουν μεταβλητές και μπορεί να είναι δύο τύπων:

Διατεταγμένα: Λίστες από τιμές, που παριστάνουν σχετικά απλά γεγονότα, όπου η σειρά των τιμών παίζει ρόλο.

Π.χ. (flight 734 DELTA), (person George), (class “A1” 1995).

Σε κάθε τέτοιο γεγονός υπονοείται μια σημασιολογία για κάθε θέση στη λίστα. Π.χ. αν θέλω να εισάγω μια σειρά από γεγονότα που αφορούν αεροπορικές πτήσεις και θέλω να υπάρχουν πέντε στοιχεία σ’ αυτά, π.χ. η λέξη flight, ο αριθμός πτήσης, ο τόπος αναχώρησης, ο τόπος προορισμού και η εταιρία, τότε υπονοώ την παρακάτω δομή:

(flight <αριθμός-πτήσης> <τόπος-αναχώρησης> <προορισμός> <εταιρία>)

και τα αντίστοιχα γεγονότα πρέπει να την ακολουθούν, αλλιώς θα έχω προβλήματα στην επίλυση του προβλήματος.

Μη διατεταγμένα ή προτύπου: Ονοματοποιημένες λίστες, που αποτελούνται από λίστες δύο στοιχείων, και παριστάνουν πιο σύνθετα γεγονότα. Συνήθως οι λίστες αυτές των δύο στοιχείων παριστάνουν μια δυάδα «ιδιότητα-τιμή».

Π.χ. (student (age 19) (year a) (sex male))

Τα γεγονότα αυτά σχετίζονται με αντίστοιχο πρότυπο γεγονότων (fact template) (βλ. παρακάτω) που ορίζει το πλήθος και τον τύπο των λιστών που θα έχουν τα γεγονότα. Τα γεγονότα προτύπου μπορούν να έχουν λιγότερες λίστες από ότι το πρότυπο, δηλ. δεν πειράζει να λείπουν ζεύγη «ιδιότητα-τιμή», γι' αυτό και ονομάζονται μη διατεταγμένα. Δηλ. δεν παίζει ρόλο η ακριβής θέση μιας λίστας, αλλά το όνομα της ιδιότητας που έχει η λίστα.

14.4.4 Κανόνες (Rules)

Είναι δομές της μορφής:

if *(συνθήκες)* then *(ενέργειες)*

που αναπαριστούν ευρετική γνώση (heuristic knowledge) στη βάση γνώσης.

Οι κανόνες βρίσκονται στη βάση κανόνων στη μνήμη απ' όπου μετακινούνται στη στοίβα κανόνων (agenda) όταν όλες οι συνθήκες τους ικανοποιούνται, δηλ. οι συνθήκες τους ενοποιούνται (match) με γεγονότα στη λίστα γεγονότων. Στη στοίβα κανόνων μπαίνουν στην κατάλληλη θέση (σειρά) με βάση την προτεραιότητά τους και την στρατηγική επίλυσης συγκρούσεων που ακολουθείται.

Επειδή οι κανόνες περιέχουν μεταβλητές, η ενοποίηση/ταίριασμα των συνθηκών τους με γεγονότα από τη λίστα γεγονότων μπορεί να γίνεται με διαφορετικές τιμές κάθε φορά, δηλ. από ένα κανόνα CLIPS να έχουμε ένα ή περισσότερα στιγμιότυπα του κανόνα, αυτά που προκύπτουν αν αντικαταστήσουμε τις μεταβλητές με τις τιμές που προκύπτουν από την ενοποίηση/ταίριασμα. Αυτά τα στιγμιότυπα των κανόνων είναι που εισέρχονται στη στοίβα κανόνων.

Η διαδικασία της ενοποίησης/ταιριάσματος (matching) έχει σαν στόχο να βρει αν υπάρχει γεγονός με το οποίο μπορεί να ταυτοποιηθεί μια συνθήκη, δηλ. να βρεθούν τιμές για τις μεταβλητές, αν υπάρχουν, που τα κάνουν ταυτόσημα (εκτός αν η συνθήκη δεν έχει μεταβλητές και είναι ταυτόσημη). Παρακάτω δίνονται κάποια

παραδείγματα (Πίνακες 14.1 και 14.2) που αποσαφηνίζουν τη διαδικασία ταυτοποίησης του CLIPS.

Πίνακας 14.1: Παραδείγματα ενοποίησης/ταιριάσματος συνθηκών-γεγονότων.

Συνθήκη	Γεγονός	Μεταβλητές-Τιμές
(month ?m ?d)	(month jan 10)	?m←jan, ?d←10
(color $?col)	(color gray blue)	$?col←(gray blue)
(flight ?f dest ?d comp $?c)	(flight 123 dest boston comp klm delta)	?f←123 ?d ←boston $?c←(klm delta)

Ιδιαίτερη προσοχή πρέπει να δίνεται όταν στις συνθήκες ενός κανόνα εμφανίζονται μεταβλητές πολλαπλών τιμών.

Πίνακας 14.2: Παραδείγματα μη ενοποίησης/ταιριάσματος συνθηκών-γεγονότων.

Συνθήκη	Γεγονός	Αιτία
(month ?m ?d)	(months jan 10)	Διαφορετικό πρώτο σύμβολο
(color green $?col)	(color red gray blue)	Το 2ο σύμβολο είναι διαφορετικό.
(flight ?f dest ?d comp ?c)	(flight 123 dest boston comp klm delta)	Η μονότιμη μεταβλητή ?c δε μπορεί να πάρει 2 τιμές (klm και delta).

14.4.5 Συναρτήσεις (Functions)

Είναι τμήματα προγράμματος για την αναπαράσταση διαδικαστικής γνώσης (procedural knowledge). Συνήθως επιστρέφουν κάποια τιμή, ως πλευρικό αποτέλεσμα.

14.5 Ορισμός και Διαχείριση Στοιχείων του CLIPS

14.5.1 Διατεταγμένα Γεγονότα

Οι βασικές εντολές διαχείρισης των γεγονότων είναι:

assert: Εισαγωγή ενός γεγονότος.

Σύνταξη: (assert <fact>)

Παράδειγμα: (assert (water-tank empty))

deffacts: Εισαγωγή πολλών γεγονότων, ως ομάδα με κάποιο όνομα.

Σύνταξη:

(deffacts <όνομα>
 ["<σχόλιο>"]
  <fact1>
  <fact2>
    …
  <factn>)

Παράδειγμα:

(deffacts pref-cars
  (car AlfaRomeo156 1600 16 nai idrayliki 22910)
  (car AudiA4 1600 8 nai idrayliki 25700)
  (car FordMondeo 1800 16 nai idrayliki 19289)
  (car NissanPrimera 1600 16 oxi idrayliki 17990)
  (car VWPassat 1600 8 oxi idrayliki 18910))

retract: Διαγραφή ενός ή όλων των γεγονότων.

Σύνταξη: retract <fact-index>, όπου <fact-index> είναι ο δείκτης (ακέραιος αριθμός) του γεγονότος στη λίστα γεγονότων.

Παράδειγμα1:

(retract 2)

Συνήθως όμως, σ' ένα κανόνα, δεν ξέρουμε τον δείκτη του γεγονότος. Τότε χρησιμοποιούμε τον ειδικό τελεστή '<-' για να τον αποθηκεύσουμε σε μια μεταβλητή.

Παράδειγμα2:

?x <- (car AudiA4 1600 8 nai idravliki 25700)

(retract ?x)

Το (retract *) αφαιρεί όλα τα γεγονότα από τη λίστα γεγονότων.

14.5.2 Πρότυπα γεγονότων-Μη διατεταγμένα Γεγονότα

Ορισμός:

```
(deftemplate <όνομα προτύπου>
  (slot/multislot <slot-name1> <constraint-form>*)
  (slot/multislot <slot-name2> <constraint-form>*)
                  …
  (slot/multislot <slot-namen> <constraint-form>*))
```

Το slot δηλώνει μονότιμες μεταβλητές/παραμέτρους, ενώ το multislot μεταβλητές/παραμέτρους πολλαπλών τιμών. Οι περιορισμοί (δηλ. τα <constraint form>) στην παραπάνω σύνταξη μπορεί να είναι:

- Τύπου

  (type <type>)

  όπου <type> ∈ {SYMBOL, STRING, LEXEME, INTEGER, FLOAT, NUMBER,?VARIABLE}

- Απαρίθμησης τιμών

  (allowed-<type> <value1> … <valuen>)

  όπου <type> ∈ {symbols, strings, lexems, integers, floats, numbers, values}

και

  τα <valuei> είναι αντίστοιχου τύπου.

- Περιοχής τιμών

  (range <init-val> <final-val>)

  όπου <init-val>, <final-val> μπορεί να είναι ?VARIABLE (για ανοικτό κάτω,

πάνω όριο)

- Εξ' ορισμού (ή Προκαθορισμένης) τιμής

  (default <value>)

  όπου <value> συγκεκριμένη τιμή ή ?DERIVE ή ?NONE (απαιτεί την εισαγωγή τιμής)

- Πλήθους τιμών

  (cardinality <min> <max>)

  όπου <min> και <max> ο ελάχιστος και μέγιστος αριθμός τιμών μιας multislot.

Παράδειγμα

```
(deftemplate student
  (slot name (type STRING) (default ?NONE))
  (slot sex (type SYMBOL) (allowed-symbols male female))
  (slot age (type INTEGER) (range 18 40))
  (multislot courses (type SYMBOL) (cardinality 1 4)))
```

Η εισαγωγή μη διατεταγμένων γεγονότων γίνεται όπως και των διατεταγμένων, με την επαναληπτική χρήση της εντολής **assert**.

Παραδείγματα

(assert (student (name "petros mixos") (sex male) (age 19)))

(assert (student (name "giannis panou") (sex male) (courses (ai db))))

(assert (student (sex female) (age 20))) → ERROR!!! (λόγω ?NONE στους περιορισμούς του name, βλ. ορισμό πιο πάνω)

modify: Τροποποιεί ένα ή περισσότερα στοιχεία ενός μη διατεταγμένου γεγονότος στη λίστα γεγονότων (αφαιρεί το παλιό στοιχείο και εισάγει το νέο)

Σύνταξη: (modify <fact-index> (<slot> <new-value>)*)

Παράδειγμα:

Έστω ότι το γεγονός

(student (name "giannis panou") (sex male) (age 19) (courses (ai db)))

υπάρχει στη λίστα γεγονότων. Τότε, αν εκτελέσουμε

?x <- (student (name “giannis panou”))

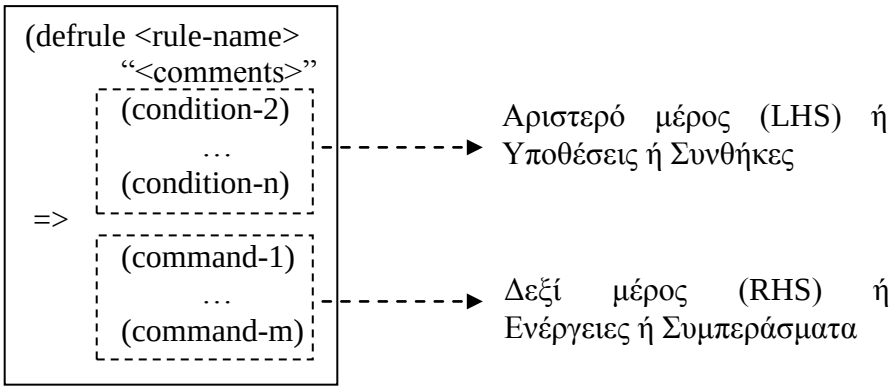
(modify ?x (courses (age 20) (math ai db)))

το CLIPS αντικαθιστά το παραπάνω γεγονός με το

(student (name “giannis panou”) (sex male) (age 20) (courses (math ai db)))

14.5.3 Κανόνες

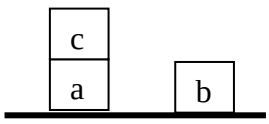
Το βασικό δομικό στοιχείο ενός έμπειρου συστήματος είναι η βάση γνώσης που αποτελείται από μια σειρά από κανόνες. Για το λόγο αυτό η σύνταξη ενός κανόνα είναι η ακόλουθη:



Το σύμβολο «= =>» διαχωρίζει το αριστερό (συνθήκες) από το δεξί (ενέργειες) μέρος του κανόνα. Το όνομα του κανόνα πρέπει να είναι μοναδικό. Συνήθως όλοι οι κανόνες είναι αποθηκευμένοι σε κάποιο αρχείο κειμένου απ’ όπου τους «φορτώνει» το σύστημα (εντολή Load, είτε από τη γραμμή εντολών CLIPS, είτε από το μενού του File).

Παράδειγμα

```
(defrule pick-up-cube
        “pick up a free cube”
        (free hand)
        (on table ?x)
        ?n1 <- (free hand)
        ?n2 <- (on table ?x)
 =>
        (assert (hand contains ?x))
        (retract ?n1)
        (retract ?n2)
```



Ο κανόνας αυτός αφορά μια ενέργεια στο γνωστό πρόβλημα των κύβων, πιο συγκεκριμένα το πιάσιμο στο χέρι ενός κύβου που δεν έχει τίποτα επάνω του.

Τύποι συνθηκών κανόνων

Οι κύριοι τύποι των συνθηκών που χρησιμοποιούνται στο CLIPS είναι:

(α) *Συνθήκες Προτύπου (pattern)*

- Το πρώτο στοιχείο είναι πάντα ένα σύμβολο που καθορίζει αν η συνθήκη προτύπου εφαρμόζεται σε διατεταγμένα γεγονότα ή γεγονότα προτύπου.
- Στα υπόλοιπα μπορούν να χρησιμοποιηθούν
  - σταθερές τιμές
  - τα ?, $? σαν wildcards
  - μεταβλητές (μονότιμες ή πολλαπλών τιμών)
  - περιορισμοί λογικής σύνδεσης (&, |, ~)
  - περιορισμοί κατηγορήματος (:<κλήση-συνάρτησης>)
  - περιορισμοί επιστρεφόμενης τιμής (= <κλήση-συνάρτησης>)

(β) *Συνθήκες διεύθυνσης προτύπου*

Ανατίθεται η τιμή της διεύθυνσης ενός γεγονότος σε μια μεταβλητή:

<μεταβλητή> <- <συνθήκη προτύπου>

(γ) *Εκτιμήσιμες Συνθήκες*

Για περιπτώσεις που έχουμε ως υποθέσεις συγκρίσεις αριθμητικών τιμών:

(test <κλήση-συνάρτησης>)

(δ) *Λογικά Συνδεόμενες Συνθήκες*

Χρήση των λογικών συνδετικών (and, or, not) για δημιουργία σύνθετων συνθηκών.

Παράδειγμα

Έστω ότι εισάγουμε τα παρακάτω διατεταγμένα γεγονότα:

```
(deffacts data facts
  (data 1.0 blue “red”)
  (data 1 green)
  (data 1 blue red)
  (data 1 gray RED)
  (data 1 blue red 6.9 “05”))
```

Επίσης δηλώνουμε το παρακάτω πρότυπο γεγονότων:

```
(deftemplate person
```

```
  (slot name)
  (slot age)
  (multislot friends))
```

και εισάγουμε τα παρακάτω μη διατεταγμένα γεγονότα:

```
(deffacts people
   (person (name paul) (age 20))
   (person (name john) (age 20))
   (person (name paul) (age 30))
   (person (name niki) (age 35) (friends maria helen)))
```

Οι παρακάτω θα μπορούσαν να είναι συνθήκες σε κανόνες CLIPS:

```
(data 1 green)
(person (name paul))
(person (name paul) (age 20))
(data ? blue red $?)
(person (name ?) (age 20))
(person)
(data ?x blue red $?y)
(data 1 ~blue&~green $?)
(data 1 ?x~blue&~green $?y)
(data 1 gray|green $?y)
(person (name ?y&~paul) (age 20))
(data ?x&:(floatp ?x) blue $?y)
(data 1 blue $?x&:(> (length$ ?x) 2))
(person (name paul) (age ?z&:(> ?z 20)))
```

και παρακάτω δείτε δύο κανόνες, που χρησιμοποιούν τέτοιες συνθήκες.

```
(defrule example-1
  (person (name paul) (age ?y))
  (test (> ?y 20))
  =>
  (printout t “This paul is “ ?y crlf))
```

Ο κανόνας αυτός βρίσκει και τυπώνει (με κάποιο τρόπο) όσους έχουν όνομα paul και ηλικία μεγαλύτερη των 20. Στην περίπτωσή μας θα τυπώσει:

```
This paul is 30.
```

```
(defrule example-2
  (person (name paul) (age ?x))
  (person (name ?y) (age ?z&:(> ?z ?x)))
  =>
  (printout t ?y “is older than pauls“ crlf))
```

Ο κανόνας αυτός βρίσκει και τυπώνει (με κάποιο τρόπο) όσους έχουν ηλικία μεγαλύτερη από όλους τους Paul. Στην περίπτωσή μας θα τυπώσει:

niki is older than pauls.

Ο παρακάτω κανόνας χρησιμοποιεί τους λογικούς τελεστές or και and στις συνθήκες του.

```
(defrule example-3
  (or (and (temp high) (valve closed))
      (and (temp low) (valve open)))
  =>
  (printout t "there is a problem" crlf))
```

14.5.4 Συναρτήσεις

Η κλήση μιας συνάρτησης γίνεται με την δήλωση:

(<όνομα-συνάρτ.> όρισμα1 όρισμα2 .. όρισμαΝ).

Για παράδειγμα αν μέσα σε ένα πρόγραμμα CLlPS υπάρχει η εντολή (assert (The product is (* 5 4))) τότε το γεγονός το οποίο θα αποθηκευτεί στη μνήμη θα είναι το (The product is 20). Το «(* 5 4)» είναι κλήση της συνάρτησης '*' με δύο ορίσματα, που είναι ακέραιοι αριθμοί.

Οι κυριότερες συναρτήσεις είναι οι:

Αριθμητικές

+, -, *, /

Σύγκρισης

<, >, =, >=, <=, <>

Λογικές

and, or, not, eq, neq

Ελέγχου τύπου

numberp, symbolp, floatp, integerp, stringp

Χειρισμού Πολλαπλών Τιμών

create$

Σύνταξη: (create$ <values>). Δημιουργεί πολλαπλή τιμή, που μπορεί να ανατεθεί σε μεταβλητή πολλαπλής τιμής.

Π.χ. (create$ a b c) → (a b c)

(create$ a (b c) d) → (a b c d)

explode$

Σύνταξη: (explode$ <string>). Δημιουργεί πολλαπλή τιμή από αλφαριθμητικό.

Π.χ. (explode$ "give the ball") → (give the ball)

implode$

Σύνταξη: (implode$ <multivalue>). Επιστρέφει το αντίστοιχο αλφαριθμητικό από μια πολλαπλή τιμή.

Π.χ. (implode (give the ball)) → "give the ball"

nth$

Σύνταξη: (nth$ N <multivalue>). Επιστρέφει το N-οστό πεδίο μιας πολλαπλής τιμής.

Π.χ. (nth$ 2 (a b c)) → b

member$

Σύνταξη: (member$ <symbol> <multivalue>). Επιστρέφει τη θέση του στοιχείου <symbol> μέσα στην πολλαπλή τιμή <multivalue> εφόσον αυτό υπάρχει. Εάν δεν υπάρχει επιστρέφει FALSE.

Π.χ. (member$ c (a b c)) → 3

(member$ d (a b c)) → FALSE

first$

Σύνταξη: (first$ <multivalue>). Επιστρέφει το πρώτο στοιχείο μιας πολλαπλής τιμής μέσα σε λίστα.

Π.χ. (first$ (a b c)) → (a)

rest$

Σύνταξη: (rest$ <multivalue>). Επιστρέφει τα στοιχεία της πολλαπλής τιμής, πλην του πρώτου στοιχείου, σαν λίστα.

Π.χ. (rest$ (a b c)) → (b c)

Εξόδου

printout

Σύνταξη: (printout <συσκευή> <έκφραση>). Η <έκφραση> αποστέλλεται στη <συσκευή>, που μπορεί να είναι μια συσκευή I/O, όπως π.χ. η οθόνη, ένα αρχείο ή ένας εκτυπωτής.

Π.χ. (printout t "The vehicle is" ?type crlf)→

The vehicle is train

Υποτίθεται ότι η μεταβλητή ?type έχει την τιμή 'train'. (Το 't' σημαίνει εκτύπωση στην οθόνη: terminal).

Εισόδου

read

Σύνταξη: (read). Με την εκτέλεση της εντολής το σύστημα περιμένει είσοδο από το πληκτρολόγιο (που είναι η κύρια είσοδος δεδομένων). Χρησιμοποιείται συνήθως μαζί με την εντολή bind (βλ. αμέσως παρακάτω).

Ανάθεσης

bind

Σύνταξη: (bind <μεταβλητή> <τιμή>). Καταχωρεί την <τιμή> στην <μεταβλητή>. Συνήθως χρησιμοποιείται στις ενέργειες των κανόνων καθώς και με την εντολή read.

Π.χ. (bind ?name (read))

giannis

(printout t ?name crlf)

giannis

Ελέγχου Ροής

if

Σύνταξη: (if (συνθήκη)
then (εντολή 1) … (εντολή ν)
else (εντολή 1) … (εντολή μ))

Χρησιμοποιείται όπως η εντολή if στις συμβατικές γλώσσες προγραμματισμού. Αν αληθεύει η (συνθήκη) εκτελούνται οι εντολές 1 έως ν, αλλιώς οι 1 έως μ. Χρησιμοποιείται είτε σε συναρτήσεις ορισμένες από τον χρήστη (βλ. παρακάτω) είτε σε ενέργειες (κυρίως) κανόνων.

Π.χ. (defrule age-type
(age ?x)
=>
(if (< ?x 30)
then (printout t “young” crlf)
else (if (> ?x 60)
then (printout t “old” crlf)
else (printout t “middle” crlf))))

while

Σύνταξη: (while (συνθήκη) do
(εντολή 1)
…
(εντολή ν))

Χρησιμοποιείται όπως η εντολή while στις συμβατικές γλώσσες προγραμματισμού. Ενόσω η (συνθήκη) αληθεύει εκτελούνται οι εντολές 1 έως ν. Χρησιμοποιείται είτε σε συναρτήσεις ορισμένες από τον χρήστη είτε σε ενέργειες (κυρίως) κανόνων.

Π.χ. (defrule print-color
(color $?col)
=>
(bind ?n (length $?col))
(bind ?i 0)
(while (< ?i ?n) do
(printout t (implode$ (first$ $?col)) crlf)
(bind ?i (+ ?i 1))
(bind $?col (rest$ $?col))))

Αν υποθέσουμε ότι στη λίστα γεγονότων υπάρχει το γεγονός (color red blue gray), το αποτέλεσμα του παραπάνω κανόνα θα είναι:

red

blue

gray

δηλ. η εκτύπωση των χρωμάτων του γεγονότος (όσα κι αν είναι αυτά).

Ορισμός Συναρτήσεων Χρήστη

```
(deffunction <όνομα-συν> (<μεταβλητές>)
        (εντολή 1)
           …
        (εντολή ν))
```

Παραδείγματα

```
(deffunction hypotenusa (?a ?b)
     (sqrt (+ (* ?a ?a) (* ?b ?b))))
```

Κλήση της συνάρτησης:

(hypotenuse 3 4) → 5

```
(deffunction initialize ()
     (clear)
     (assert (today is sunday)))
```

Κλήση της συνάρτησης:

(initialize) → Καθαρισμός περιβάλλοντος CLIPS (διαγραφή γεγονότων και κανόνων) και εισαγωγή του γεγονότος (today is sunday).

```
(deffunction findmax3 (?x ?y ?z)
    (if (?x > ?y)
            then if (?x > ?z)
              then (return ?x)
              else (return ?z)
       else if (?y > ?z)
              then (return ?y)
              else (return ?z)))
```

Κλήση της συνάρτησης:

(findmax 5 6 2) → 6

14.5.5 Συναρτήσεις ελέγχου περιβάλλοντος

facts

Σύνταξη: (facts)

Επιστρέφει όλα τα γεγονότα που βρίσκονται στη λίστα γεγονότων.

watch

Η εντολή αυτή είναι ιδιαίτερα χρήσιμη για debugging.

Σύνταξη: (watch <watch-item>)

όπου <watch-item> είναι ένα από τα σύμβολα *facts, rules, activations, statistics, compilations* ή *all*. Ανάλογα με το τι θέτουμε στο <watch-item> παρακολουθούνται τα αντίστοιχα στοιχεία κατά την εκτέλεση του προγράμματος.

Παράδειγμα:

```
CLIPS> (facts)
f-1     (person (name “Giannis Papas”) (age 25)
                (eyes brown) (hair black))
For a total of 1 fact.
```

(Δηλ. υποθέτουμε ότι υπάρχει μόνο ένα γεγονός στη μνήμα, με δείκτη 3).

```
CLIPS> (watch facts)
CLIPS> (modify 1 (age 30))
<= = f-1        (person (name “Giannis Papas”) (age 25)
                        (eyes blue) (hair black))
= => f-2        (person (name “Giannis Papas”) (age 30)
                        (eye blue) (hair black))
<Fact-2>
```

Το = => σημαίνει τη διαγραφή ενός γεγονότος, ενώ το = => την εισαγωγή ενός γεγονότος.

unwatch

Σύνταξη: (unwatch <watch-item>)

Χρησιμοποιείται για την απενεργοποίηση της παρακολούθησης. Πρέπει να σημειωθεί ότι με τη χρήση της το CLIPS θα τυπώσει αυτόματα ένα μήνυμα που δείχνει ότι μια αναπροσαρμογή έχει γίνει στον κατάλογο γεγονότων όποτε τα γεγονότα βεβαιώνονται ή αποσύρονται.

reset

Σύνταξη: (reset)

Αφαιρεί όλα τα γεγονότα από τη λίστα γεγονότων, εισάγει το γεγονός (intial-fact) και εισάγει τα γεγονότα από τις υπάρχουσες δηλώσεις deffacts του προγράμματος. Επομένως, αν είχε προηγηθεί εκτέλεση του προγράμματος, τότε με το reset επανερχόμεθα στην αρχική κατάσταση.

Παράδειγμα:

Ας θεωρήσουμε ότι υπάρχει η παρακάτω δήλωση deffacts στο πρόγραμμά μας.

```
(deffacts people
    (person (name “Giannis Papas”) (age 25) (eyes brown) (hair black))
    (person (name “Giorgos Hatzis”) (age 20) (eyes blue) (hair brown))

CLIPS> (reset)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (person (name “Giannis Papas”) (age 25)
                (eyes brown) (hair black))
f-2     (name“Giorgos Hatzis”) (age 20)
                (eyes blue) (hair brown))
For a total of 3 facts.
```

Δηλαδή έχουν εισαχθεί και τα δύο γεγονότα της εντολής deffacts.

Με τη εκκίνηση το CLIPS αυτόματα ορίζει δύο δομές:

```
(deftemplate initial-fact)
(deffacts initial-fact (initial-fact))
```

Ακόμη κι αν δεν χρησιμοποιήσουμε δηλώσεις deffacts, η reset θα εισάγει το γεγονός (initial-fact). Ο δείκτης του θα είναι πάντα f-0.

clear

Σύνταξη: (clear)

Καθαρίζει το περιβάλλον, δηλ. διαγράφει από τη μνήμη γεγονότα και κανόνες. Το πρόγραμμα πρέπει να ξαναφορτωθεί για να τρέξει.

run

Σύνταξη: (run)

Ξεκινά την εκτέλεση των κανόνων που έχουν φορτωθεί στη μνήμη. Υπάρχει δυνατότητα να χρησιμοποιηθεί ως

(run N)

όπου N είναι ακέραιος αριθμός. Στην περίπτωση αυτή, γίνονται μόνο N κύκλοι εκτέλεσης κανόνων και κατόπιν σταματά. Η εκτέλεση συνεχίζεται με μια νέα εντολή run. Αυτό χρησιμοποιείται κυρίως για αποσφαλμάτωση (debugging) ενός προγράμματος.

14.6 Επίλυση Συγκρούσεων

Η επίλυση συγκρούσεων αναφέρεται στην επιλογή ενός κανόνα από το σύνολο των κανόνων (σύνολο σύγκρουσης: conflict set) που βρίσκονται σε κάθε βήμα στη στοίβα κανόνων (agenda) και είναι υποψήφιοι για πυροδότηση.

Η επιλογή στηρίζεται (α) στην προτεραιότητα κάθε κανόνα και (β) στη στρατηγική επίλυσης σύγκρουσης που έχει επιλεγεί. Το CLIPS διαθέτει επτά τις στρατηγικές επίλυσης συγκρούσεων, από τις οποίες μόνο μια είναι ενεργή και χρησιμοποιείται για την επιλογή του κανόνα από την ατζέντα.

14.6.1 Προτεραιότητα Κανόνων

Η προτεραιότητα ενός κανόνα ορίζεται μέσω ειδικής δήλωσης, που εισάγεται στη συνάρτηση ορισμού του κανόνα..

Σύνταξη: (declare (salience <number>))

Όσο μεγαλύτερη είναι η τιμή <number> τόσο μεγαλύτερη είναι και η προτεραιότητα του συγκεκριμένου κανόνα. Οι επιτρεπτές τιμές της προτεραιότητας είναι από -10000 έως 10000. Εάν δεν υπάρχει δήλωση, ο κανόνας θεωρείται ότι έχει ως τιμή προτεραιότητας μηδέν.

Παράδειγμα

```
(defrule number
  (declare (salience 60))
  (number ?a)
  (number ?b)
```

```
=>
  (printout t "Numbers: " ?a " " ?b crlf)
)
```

14.6.2 Στρατηγικές Επίλυσης Συγκρούσεων

Όπως είπαμε, το CLIPS διαθέτει επτά στρατηγικές επίλυσης συγκρούσεων, που είναι η εξής:

1. **depth:** Σύμφωνα με αυτή τη στρατηγική οι «νέοι» κανόνες μπαίνουν πάνω από τους «παλαιούς». Π.χ. αν το fact-1 ενεργοποιεί τους r1, r2 και το fact-2 ενεργοποιεί τους r3 και r4 τότε οι r3, r4 τοποθετούνται πάνω από τους r1, r2, διότι το fact-1 έχει εισαχθεί πριν από το fact-2 (δηλ. το fact-2 είναι πιο πρόσφατο, πιο επίκαιρο και επομένως οι r3, r4 λέμε ότι έχουν μεγαλύτερη επικαιρότητα-recency). Μεταξύ τους οι r1, r2 και r3, r4 τοποθετούνται αυθαίρετα.
2. **breadth:** Αντίθετα με την προηγούμενη στρατηγική, οι «νέοι» κανόνες μπαίνουν κάτω από τους «παλαιούς».
3. **simplicity:** Ένας νεοεισερχόμενος κανόνας τοποθετείται υψηλότερα από όλους με ίση ή μεγαλύτερη εξειδίκευση (specificity). Εξειδίκευση = αριθμός συγκρίσεων ή/και κλήσεων συναρτήσεων στο LHS. Π.χ. ο κανόνας

   ```
   (defrule r-example
           (item ?x ?y ?x)
           (test (and (numberp ?x) (> ?x (+ 10 ?y)) (< ?x 100)))
   =>)
   ```

   έχει εξειδίκευση = 5.

4. **complexity:** Ένας νεοεισερχόμενος κανόνας τοποθετείται υψηλότερα από όλους με ίση ή μικρότερη εξειδίκευση. (το αντίστροφο της simplicity)
5. **LEX:** Ένας νεοεισερχόμενος κανόνας με μεγαλύτερη επικαιρότητα (recency) τοποθετείται υψηλότερα. Σε περίπτωση ίδιας επικαιρότητας χρησιμοποιείται ως κριτήριο η εξειδίκευση. Ουσιαστικά αποτελεί συνδυασμό των στρατηγικών depth και complexity.
6. **MEA:** Ένας νεοεισερχόμενος κανόνας με μεγαλύτερη επικαιρότητα (recency) για την πρώτη συνθήκη μόνο τοποθετείται υψηλότερα.. Σε περίπτωση σύμπτωσης χρησιμοποιείται η LEX.
7. **random:** Ένας τυχαίος αριθμός χρησιμοποιείται για να ξεχωρίσουν κανόνες

της ίδιας προτεραιότητας.

Ορισμός στρατηγικής: (set-strategy <strategy>)

Η <strategy> είναι μια από τις τιμές: depth, breadth, simplicity, complexity, lex, mea, random.

Ανίχνευση στρατηγικής: (get-strategy).

Βιβλιογραφία

1. Ιστότοπος CLIPS: http://www.ghg.net/clips/CLIPS.html.
2. Ι. Βλαχάβας, Π. Κεφαλας, Ν. Βασιλειάδης, Ι. Ρεφανίδης, Φ. Κόκκορας και Η. Σακελλαρίου. Τεχνητή Νοημοσύνη, Εκδ. ΓΑΡΤΑΓΑΝΗ, Θεσ/νίκη, 2002.
3. Joseph Giarratano. CLIPS User's Guide. Lyndon B. Johnson Space Center Information Systems Directorate Software Technology Branch, NASA, May 28, 1993.
4. A. Gonzalez, D. Dankel The Engineering of Knowledge-Based Systems: Theory and Practice, Prentice-Hall, 1993.
5. P. Lucas and L. van der Gaag. Principles of Expert Systems, Addison Wesley, 1991.

ΠΑΡΑΡΤΗΜΑ

Λίστα Εντολών CLIPS

(Από την ιστοσελίδα του CLIPS)

Ακολουθεί λίστα των κυριότερων εντολών του CLIPS και σχετικό παράδειγμα γιά την καλύτερη κατανόησή της.

Γεγονότα

1. assert

Παράδειγμα:

```
 CLIPS> (assert (tweetie))
< Fact-1 >
```

2. assert-string

Παράδειγμα:

```
CLIPS> (assert-string "(hello there guy)")
< Fact-2 >
CLIPS> (facts)
f-0 (initial-fact)
f-1 (duck)
f-2 (hello there guy)
For a total of 3 facts.
```

3. facts

Παράδειγμα:

```
CLIPS> (facts)
f-0 (clyde)
f-1 (tweetie)
For a total of 2 facts.
```

4. reset

Παράδειγμα:

```
CLIPS> (reset)
CLIPS> (facts)
f-0  (initial-fact)
```

5. clear

Παράδειγμα:

```
CLIPS> (clear)
CLIPS> (facts)
CLIPS>
```

6. retract

Παράδειγμα:
```
CLIPS> (retract 0)
CLIPS> (facts)
CLIPS>
```

7. save-facts

Παράδειγμα:
```
CLIPS> (save-facts < filename >)
TRUE
```

8. load-facts

Παράδειγμα:
```
CLIPS> (load-facts < filename >)
TRUE
```

Κανόνες

9. refresh

Παράδειγμα:
```
CLIPS> (refresh clyde)
CLIPS>
```

10. matches

Παράδειγμα:
```
CLIPS> (matches clyde)
Matches for Pattern 1
f-1
Activations
f-1
```

11. set-break

Παράδειγμα:
```
CLIPS> (set-break a)
CLIPS>
```

12. show-breaks

Παράδειγμα:
```
CLIPS> (show-breaks)
clyde
```

13. remove-break

Παράδειγμα:
```
CLIPS> (remove-break clyde)
```

```
CLIPS>
```

14. bind

Παράδειγμα:
```
CLIPS> (bind ?killer (rabbit))
Rabbit
CLIPS> (printout t ?killer crlf)
[EVALUATN1] Variable killer is unbound
```

Διόρθωση

15. agenda

Παράδειγμα:
```
CLIPS> (agenda)
a: f-0
For a total of 1 activation.
```

16. run

Παράδειγμα:
```
CLIPS> (run [integer])
CLIPS> (facts)
f-0    (initial-fact)
```

17. exit

Παράδειγμα:
```
CLIPS> (exit)
```

18. watch

Παράδειγμα:
```
CLIPS> (watch facts)
CLIPS> (reset)
==> f-0 (initial-fact)
```

19. unwatch

Παράδειγμα:
```
CLIPS> (unwatch facts)
CLIPS>
```

20. help

Παράδειγμα:
```
CLIPS> (help)
```

Διαχείριση αρχείων

21. save

Παράδειγμα:
```
CLIPS> (save < filename >)
TRUE
```

22. load

Παράδειγμα:
```
CLIPS> (load < filename >)
Defining defrule: myrule +j
TRUE
```

23. bsave

Παράδειγμα:
```
CLIPS> (bsave < filename >)
TRUE
```

24. bload

Παράδειγμα:
```
CLIPS> (bload < filename >)
TRUE
```

25. batch

Παράδειγμα:
```
CLIPS> (batch < filename >)
TRUE
CLIPS> (reset)
CLIPS> (facts)
f-0    (initial-fact)
```

Διαχείριση εισόδου-εξόδου

26. system

Παράδειγμα:
```
CLIPS> (system {system command})
CLIPS>
```

27. 2. str-cat

Παράδειγμα:
```
CLIPS> (str-cat "clydes" " red" " toe-nails" " fell" " off"
"clydes red toe-nails fell off"
```

28. create$

Παράδειγμα:
```
CLIPS> (create$ Dopey Dorky Dinky)
```

```
(Dopey Dorky Dinky)
```

29. implode$

Παράδειγμα:

```
CLIPS> (implode$ (create$ duck quacked tweetie))
"duck quacked tweetie"
```

30. read

Παράδειγμα:

```
CLIPS> (read)
killer bunnies love clyde
killer
```

31. readline

Παράδειγμα:

```
CLIPS>(readline)
clyde is too big for the kitchen
"clyde is too big for the kitchen"
```

32. printout

Παράδειγμα:

```
CLIPS> (printout t "one small step for Clyde, one giant leap
for the rest of us." crlf) one small step for Clyde, one giant
leap for the rest of us.
```

15 ΕΜΠΕΙΡΑ ΣΥΣΤΗΜΑΤΑ

15.1 Ορισμός Δομή και Λειτουργία ΕΣ

Τα *έμπειρα συστήματα* (expert systems) είναι υπολογιστικά συστήματα που εμφανίστηκαν σαν πρακτική εφαρμογή της έρευνας και των πορισμάτων της ΤΝ στα μέσα της δεκαετίας του 70. Λειτουργούν σαν προσομοιωτές της ανθρώπινης σκέψης για τη λύση προβλημάτων, οι οποίοι προσφέρουν ορισμένα πλεονεκτήματα έναντι του ανθρώπου-εμπειρογνώμονα. Τα ΕΣ (έμπειρα συστήματα) χρησιμοποιούνται κυρίως σαν σύμβουλοι των ειδικών/εμπειρογνωμόνων για τη λήψη αποφάσεων.

'Ενα έμπειρο σύστημα είναι ένα πρόγραμμα Η/Υ που μιμείται, σε κάποιο βαθμό, τις διαδικασίες λήψης αποφάσεων ενός ανθρώπου-εμπειρογνώμονα.

Αυτό το επιτυγχάνει στηριζόμενο σε εκτεταμένη γνώση γύρω από ένα στενό πεδίο προβλημάτων, που το σύστημα καλείται να αντιμετωπίσει.

Τα βασικά *φυσικά χαρακτηριστικά* που ξεχωρίζουν ένα ΕΣ από ένα συμβατικό πρόγραμμα είναι:

α) μπορεί και χειρίζεται όχι μόνο αριθμούς, αλλά και σύμβολα

β) υπάρχει διαχωρισμός μεταξύ γνώσης και χρήσης της γνώσης

γ) Χρήση ευριστικής γνώσης (heuristic knowledge) σχετικής με το πεδίο της εφαρμογής.

Η ακόλουθη εξίσωση συχνά χρησιμοποιείται για την περιγραφή ενός ΕΣ:

Έμπειρο σύστημα = Γνώση + Συλλογισμός

Έτσι, ο *πυρήνας* ενός ΕΣ αποτελείται από δύο βασικές συνιστώσες, τη *βάση γνώσης* (knowledge base) και τον *μηχανισμό ή μηχανή εξαγωγής συμπερασμάτων* (inference mechanism/engine). Πρακτικά όμως, ένα ΕΣ συνήθως περιλαμβάνει επιπλέον μια *συνιστώσα επικοινωνίας χρήστη* (user

interface), μια *βάση εργασίας* (working database), και, ίσως, ένα *μηχανισμό επεξηγήσεων* (explanation mechanism). Η δομή ενός πλήρους ΕΣ απεικονίζεται στο σχήμα 15.1, όπου ο βασικός πυρήνας σημειώνεται μέσα σε ορθογώνιο με διακεκομένη γραμμογράφηση.

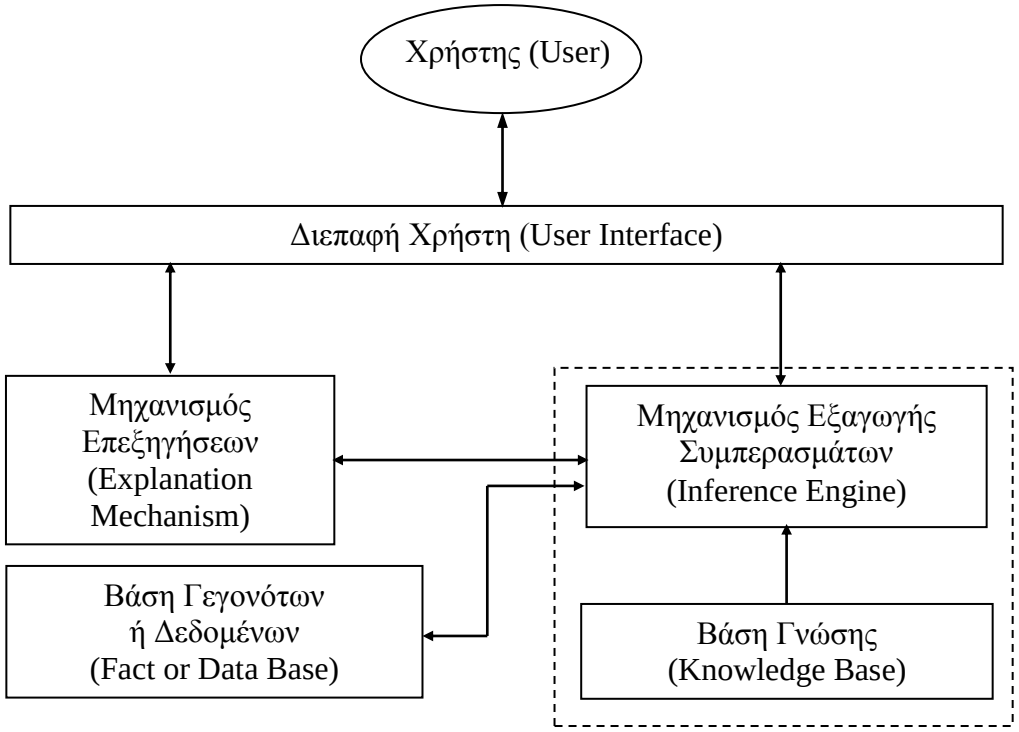
Βάση Γνώσης (ΒΓ)

Περιέχει τη γνώση γύρω από το πεδίο εφαρμογής για το οποίο έχει σχεδιασθεί το ΕΣ, και η οποία συνήθως αποκτάται από τη σχετική βιβλιογραφία και από εμπειρογνώμονες στο αντικείμενο της εφαρμογής. Η γνώση αυτή κωδικοποιείται σαν γεγονότα (facts) και κανόνες (rules), μέσω κάποιας γλώσσας αναπαράστασης της γνώσης (ΓΑΓ), και αποτελεί τη μόνιμη γνώση του συστήματος.

Η πιό διαδεδομένη ΓΑΓ στα ΕΣ είναι οι κανόνες παραγωγής (production rules). Γι' αυτό και συνήθως ταυτίζονται οι όροι «προσέγγιση/τεχνολογία έμπειρων συστημάτων» (expert system approach/technology) και «προσέγγιση/τεχνολογία βασισμένη σε κανόνες» (rule-based approach/technology). Έτσι, η ΒΓ αποτελείται συνήθως από κανόνες και καλείται *βάση κανόνων* (rule base). Όμως, σε κάποιες περιπτώσεις, χρησιμοποιούνται και άλλες ΓΑΓ, όπως τα πλαίσια (frames), τα σημαντικά δίκτυα (semantic nets) και η λογική πρώτης τάξεως (first-order logic). Αυτές οι ΓΑΓ χρησιμοποιούνται συνήθως μαζί με τους κανόνες παραγωγής, σαν δεύτερο συστατικό υβριδικής ΓΑΓ. Είναι γεγονός ότι τα τελευταία χρόνια άρχισαν να χρησιμοποιούνται υβριδικές ΓΑΓ, δηλαδή γλώσσες που συνδιάζουν στοιχεία από δύο ή περισσότερες από τις αναφερθείσες βασικές ΓΑΓ. Μια συνήθης τέτοια περίπτωση είναι ο συνδυασμός πλαισίων και κανόνων παραγωγής.

Βάση Εργασίας

Η ΒΕ περιέχει κάθε φορά γνώση σχετική με γεγονότα που αφορούν το υπό εξέταση συγκεκριμένο πρόβλημα/ερώτημα. Αυτή η γνώση αλλάζει όταν αλλάζει το υπό εξέταση πρόβλημα, σε αντίθεση με τη γνώση στη ΒΓ, που παραμένει αμετάβλητη.Στη ΒΕ δηλαδή καταχωρούνται πληροφορίες για την αρχική κατάσταση (αρχικές υποθέσεις/συνθήκες) του προβλήματος, απαντήσεις του χρήστη σε ερωτήσεις του ΕΣ, ενδιάμεσα συμπεράσματα και τελικά συμπεράσματα. Πολλές φορές η ΒΕ δεν αναφέρεται σαν ξεχωριστή μονάδα αλλά θεωρείται ενσωματωμένη στη ΒΓ.



Σχήμα 15.1. Δομή 'Εμπειρου Συστήματος

Μηχανισμός Εξαγωγής Συμπερασμάτων (ΜΕΣ)

Ο ΜΕΣ εξάγει συμπεράσματα χρησιμοποιώντας τα δεδομένα της ΒΔ και τη γνώση της ΒΓ. Καθορίζει το "πως" θα χρησιμοποιειθεί η γνώση που βρίσκεται στη ΒΓ (και στη ΒΕ) για τη λύση του εκάστοτε προβλήματος. Για κάθε συγκεκριμένο πρόβλημα καθορίζει ποιά δεδομένα θα χρησιμοποιηθούν

σε ποιά δεδομένη στιγμή της διαδικασίας, ποιοί κανόνες θα ενεργοποιηθούν και με ποιά σειρά, για ποιά δεδομένα θα ερωτηθεί ο χρήστης, κλπ.

Ο ΜΕΣ εργάζεται ανεξάρτητα από τη ΒΓ. Δηλαδή, υπάρχει διαχωρισμός της γνώσης (ΒΓ) από τον τρόπο χρήσης της (ΜΕΣ). Αυτό είναι ένα από τα κύρια χαρακτηριστικά των ΕΣ, όπως προαναφέραμε. Λόγω αυτού, ο ίδιος ΜΕΣ μπορεί να χρησιμοπιηθεί για περισσότερες από μια ΒΓ. Αυτό αποτελεί και τη βάση στην οποία στηρίζονται τα κελύφη (shells) ΕΣ.

Συνιστώσα Επικοινωνίας Χρήστη (ΣΕΧ)

Είναι το μέσον δια του οποίου ο χρήστης επικοινωνεί με τις διάφορες μονάδες του ΕΣ. Μια βασική μορφή επικοινωνίας είναι η δυνατότητα διερεύνησης ή/και μεταβολής του περιεχομένου της ΒΓ. Μια άλλη συνήθης μορφή επικοινωνίας είναι αυτή που γίνεται μέσω ερωτήσεων-απαντήσεων κατά την διάρκεια μιας διαδικασίας εξαγωγής κάποιου συμπεράσματος, για την συλλογή δεδομένων προς λύση του υπό εξέταση προβλήματος. Τέλος, μια άλλη μορφή είναι η δυνατότητα παροχής επεξηγήσεων για την εξαγωγή κάποιου συμπεράσματος.

Συνιστώσα Επεξηγήσεων (ΣΕΠ)

Η συνιστώσα αυτή είναι υπεύθυνη για την παροχή επεξηγήσεων όσον αφορά το "πως" (how) εξήχθη κάποιο συμπέρασμα ή το "γιατί" (why) γίνεται κάποια ερώτηση ή και το ποιό θα ήταν το συμπέρασμα σε περίπτωση που διαφορετικά δεδομένα είχαν δοθεί, γνωστή σαν "τι εάν" (what if) διαδικασία.

Τα βασικά λειτουργικά χαρακτηριστικά ενός ΕΣ είναι τα εξής:

- *Εξαγωγή συμπερασμάτων (inference), χωρίς να είναι απαραίτητη όλη η διαθέσιμη πληροφορία-γνώση.*

  Δηλαδή, για την εξαγωγή κάποιου συμπεράσματος δεν απαιτείται να έχουμε γνώση/πληροφορία για τις τιμές όλων των μεταβλητών του συστήματος, αλλά μόνο για αυτές που αφορούν τη συγκεκριμένη εξαγωγή συμπεράσματος.

- *Διαδραστική καθοδήγηση της εισόδου δεδομένων στο σύστημα.*
  Αυτό συνδέεται με το προηγούμενο, καθώς το σύστημα ζητά από τον χρήστη κατά τη διάρκεια της λειτουργίας του, δηλ. της εξαγωγής συμπερασμάτων, την απαιτούμενη γνώση/πληροφορία.
- *Επεξήγηση των συμπερασμάτων.*
  Ένα ΕΣ πρέπει να είναι ικανό να δίνει εξηγήσεις για τα συμπεράσματα που εξάγει με απλό και κατανοητό τρόπο.
- *Τμηματικότητα.*
  Η τμηματικότητα αναφέρεται σε δύο επίπεδα. Πρώτον, η βάση γνώσης έχει τμηματικότητα αναπαράστασης, δηλ. η γνώση αποτελείται από ανεξάρτητες μονάδες γνώσης (π.χ. κανόνες) που συνδέονται μεν μεταξύ τους εννοιολογικά, αλλά όχι συντακτικά. Αυτό δημιουργεί μια ευελιξία στην αναπαράσταση γνώσης και στη δημιουργία της ΒΓ. Δεύτερον, υπάρχει τμηματικότητα στη δομή ενός ΕΣ. Αυτό συνδέεται με το διχωρισμό της γνώσης από τη χρήση της, πο αναφέραμε πιο πάνω.

15.2 Κριτήρια Καταλληλότητας

Τα ΕΣ δεν είναι κατάλληλα για όλες τις εφαρμογές. Εφαρμογές στις οποίες οι διαδικασίες λύσης του προβλήματος είναι από τη φύση τους αλγοριθμικές, δηλαδή μπορούν να εκφραστούν σαν μια ακολουθία αριθμητικών υπολογισμών, ή οι λύσεις μπορούν να δοθούν μέσω επίλυσης μαθηματικών εξισώσεων, δεν είναι κατάλληλες για χρησιμοποίηση ΕΣ. Π.χ. η εύρεση του μέσου όρου ενός συνόλου τιμών ενός μεγέθους είναι ένα αλγοριθμικό πρόβλημα. Επίσης, προβλήματα που απαιτούν κατά μεγάλο μέρος χρησιμοποίηση γενικών γνώσεων (general knowledge) ή κοινότυπου συλλογισμού (commonsense reasoning) για τη λυση τους δεν είναι κατάλληλα για χρήση ΕΣ.

Εφαρμογές κατάλληλες για χρήση ΕΣ είναι εκείνες στις οποίες για τη λύση των προβλημάτων απαιτείται ανθρώπινη εμπειρία, δηλαδή δεν υπάρχει σαφής θεωρία επίλυσης, και οι διαδικασίες επίλυσης μπορούν καλύτερα να παρασταθούν μέσω του συμβολικού συλλογισμού (symbolic reasoning). Π.χ.

η ερμηνεία του μέσου όρου ενός συνόλου τιμών μπορεί να είναι πρόβλημα συμβολικού συλλογισμού. Επίσης, όταν οι λύσεις των προβλημάτων μιας εφαρμογής στηρίζονται σε σχετικά στενό, καλά καθωρισμένο πεδίο γνώσεων. Ακόμη, προβλήματα των οποίων η λύση στηρίζεται σε αβέβαια ή ασαφή δεδομένα, είναι κατάλληλα για προσέγγιση μέσω ΕΣ.

Τα παραπάνω συνιστούν τεχνολογικούς παράγοντες καταλληλότητος της χρήσης ΕΣ για μια εφαρμογή. Όμως, παράλληλα υπάρχουν και ανθρώπινοι παράγοντες. Έτσι, παρ'όλο που ένα πρόβλημα μπορεί να είναι τεχνολογικά κατάλληλο για λύση μέσω ΕΣ, αν οι εμπειρογνώμονες του είδους είναι φθηνοί ή δεν είναι διαθέσιμοι (δεν προτίθενται) να συνεργαστούν, τότε δεν συνίσταται η κατασκευή ΕΣ. Το ίδιο μπορεί να ισχύσει και στη περίπτωση που σημαντικό έγγραφο υλικό (π.χ. εγχειρίδια, μελέτες παραδειγματικών περιπτώσεων κλπ) δεν υπάρχει.

Στον πίνακα 10 συνοψίζονται τα βασικά κριτήρια καταλληλότητας μιας εφαρμογής για χρήση ΕΣ.

Πίνακας 15.1. Κριτήρια Καταλληλότητας ΕΣ

Ειδος Χαρακτηριστικών	**Χαρακτηριστικά Προβλημάτων Κατάλληλων για ΕΣ**	**Χαρακτηριστικά Προβλημάτων Ακατάλληλων για ΕΣ**
Τεχνολογικά Χαρακτηριστικά	• Συμβολική φύση διαδικασίας επίλυσης • Καλά καθορισμένο και σχετικά στενό πεδίο γνώσης • Απαιτείται ανθρώπινη εμπειρία • Απαιτούνται λύσεις από ελλειπή, ασαφή ή αβέβαια δεδομένα	• Αλγοριθμική φύση διαδικασίας επίλυσης • Απαιτούνται γενικές γνώσεις ή κοινότυπος συλλογισμός • Υπάρχει μαθηματική περιγραφή • Απαιτούνται λύσεις από απόλυτα γνωστά δεδομένα
Ανθρώπινοι Παράγοντες	• Εμπειρογνώμονες σπάνιοι και ακριβοί • Εμπειρογνώμονες διαθέσιμοι	• Εμπειρογνώμονες πολλοί και φθηνοί • Εμπειρογνώμονες μη διαθέσιμοι

Μερικά παραδείγματα εφαρμογών κατάλληλων για ΕΣ είναι:

* διάγνωση ασθενειών

* διάγνωση τεχνολογικών συστημάτων

* σχεδίαση ηλεκτρονικών κυκλωμάτων

* εκτίμηση γεωλογικών ευρημάτων
* απόδειξη θεωρημάτων
* ανάλυση χημικών ενώσεων

15.3 Τύποι Έμπειρων Συστημάτων

Τα ΕΣ εξυπηρετούν διαφορετικούς ρόλους σε διαφορετικές εφαρμογές. Έτσι, έχουμε διάφορους τύπους ΕΣ, ανάλογα με το ρόλο που εξυπηρετεί ο καθένας από αυτούς. Οι σπουδαιότεροι τύποι παρουσιάζονται στη συνέχεια εν συντομία.

ΕΣ Βοηθοί

Καλούνται να εκτελέσουν μια συγκεκριμένη εργασία, που είναι μέρος ενός μεγαλυτέρου έργου που εκτελείται από ένα ή περισσότερους ανθρώπους ή άλλα συστήματα.

Κριτικά ΕΣ

Επανεξετάζουν ένα έργο που ήδη έχει εκτελεστεί και κάνουν σχόλια για την ακρίβεια, την συνέπεια, την πληρότητα κλπ του έργου. ΕΣ αυτού του τύπου είναι από τα πιό σπάνια.

ΕΣ Ειδικοί Σύμβουλοι

Προσφέρουν συμβουλές ή γνώμες για κάποιο θέμα, βασισμένα σε πληροφορίες που παρέχει ο χρήστης. Είναι ο πιό διαδεδομένος τύπος ΕΣ.

ΕΣ Δάσκαλοι

Εκπαιδεύουν τον χρήστη στην εκτέλεση ενός ειδικού έργου, όπως π.χ. ο έλεγχος ενός συστήματος βαλβίδων πίεσης σ'ένα εργοστάσιο.

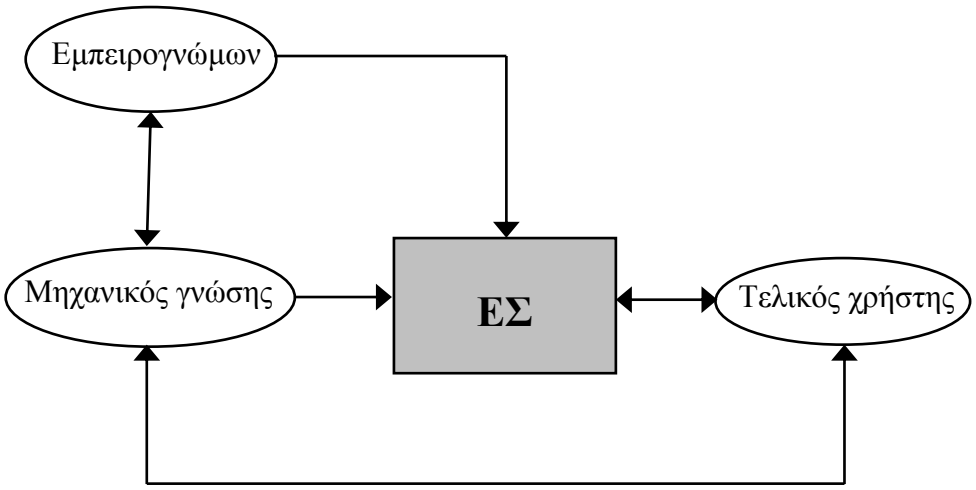
Αυτοματοποιημένα ΕΣ

Εκτελούν ένα έργο αυτόματοποιημένα, ανεξάρτητα από τον χρήστη, και αναφέρουν αποτελέσματα. Είναι ο δεύτερος πιό διαδεδομένος τύπος ΕΣ μετά από τα ΕΣ συμβούλους.

15.4 Μηχανολογία της Γνώσης

Στην ανάπτυξη/κατασκευή (development/building) ενός ΕΣ εμπλέκονται τρείς κατηγορίες ανθρώπων, ο μηχανικός γνώσης, ο εμπειρογνώμων και ο τελικός χρήστης.

Ο μηχανικός γνώσης (knowledge engineer) είναι αυτός που έχει τη διαχείριση της κατασκευής και κατευθύνει τις διάφορες φάσεις της. Είναι αυτός που επιλέγει τα εργαλεία ανάπτυξης (είτε αφορούν το υλικό είτε το λογισμικό), εκμαιεύει την απαραίτητη γνώση από τον εμπειρογνώμονα και την κωδικοποιεί (αναπαριστά) με αποδοτικό τρόπο στη ΒΓ του συστήματος. Ο μηχανικός γνώσης μπορεί να μην έχει καμμία απολύτως γνώση του γνωστικού πεδίου της εφαρμογής εκ των προτέρων.



Σχήμα 15.2 Κατηγορίες εμπλεκομένων ανθρώπων στην κατασκευή ΕΣ

Ο εμπειρογνώμων (expert) ή ειδικός (specialist) είναι αυτός που παρέχει την γνώση για το γνωστικό πεδίο της εφαρμογής. Είναι άτομο που έχει εργασθεί στο πεδίο της εφαρμογής για ικανό χρόνο, ώστε ξέρει καλά τα (πιθανά) προβλήματα και τον τρόπο αντιμετώπισής τους, καθώς και διάφορα

τεχνάσματα που αποκτήθηκαν μέσω της εμπειρίας του. Δηλαδή έχει όλα εκείνα τα στοιχεία που τον χαρακτηρίζουν σαν ειδικό.

Τέλος, ο τελικός χρήστης (end user) είναι εκείνος που θέτει περιορισμούς από πλευράς χρήστη κατά την ανάπτυξη του συστήματος, ιδιαίτερα όσον αφορά την συνιστώσα επικοινωνίας χρήστη.

Στο σχήμα 15.2 απεικονίζεται η αλληλεπίδραση των τριών κατηγοριών ανθρώπων στην κατασκευή ενός ΕΣ.

Σχήμα 15.3 Στάδια μηχανολογίας της γνώσης

Η διαδικασία απόκτησης γνώσης από τον εμπειρογνώμονα και η μεταφορά της σε μορφή εκτελέσιμη από τον Η/Υ είναι γνωστή σαν μηχανολογία της γνώσης (knowledge engineering). Η διαδικασία της ΜΓ (μηχανολογίας της γνώσης) περιλαμβάνει τρία στάδια: απόκτηση γνώσης (knowledge acquisition), αναπαράσταση γνώσης (knowledge representation) και μηχανιστική υλοποίηση (machine implementation). Τα περιεχόμενα των τριών αυτών σταδίων απεικονίζονται στο σχήμα 15.3.

Όσον αφορά στη αναπαράσταση της γνώσης ασχοληθήκαμε στα προηγούμενα κεφάλαια. Στη συνέχεια θα ασχοληθούμε με την απόκτηση γνώσης. Η υλοποίηση στον Η/Υ δεν είναι από τα θέματα αυτών των σημειώσεων.

15.5 Απόκτηση Γνώσης

Η απόκτηση γνώσης αποτελεί το δυσκολώτερο στάδιο ΜΓ. Η απόκτηση γνώσης γίνεται από διάφορες πηγές γνώσης (knowledge sources), όπως εμπειρογνώμονες, βιβλιογραφία και βάσεις δεδομένων. Η δυσκολία έγκειται κυρίως στην απόκτηση γνώσης από εμπειρογνώμονες, γνωστή σαν εκμαίευση

της γνώσης (knowledge elicitation), διότι υπάρχει μια εγγενής αδυναμία των εμπειρογνωμόνων στο να εκφράσουν τη γνώση (εμπειρία) που κατέχουν. Γι' αυτό και η απόκτηση γνώσης έχει χαρακτηριστεί σαν το σημείο συμφόρησης (bottleneck) στην εν γένει ανάπτυξη ενός ΕΣ.

Υπάρχουν αρκετές τεχνικές για την απόκτηση γνώσης από ένα ειδικό, μερικές δανεισμένες από την ανάλυση των συμβατικών συστημάτων επεξεργασίας δεδομένων. Σκοπός των μεθόδων αυτών είναι να γίνουν φανερά τα δεδομένα, οι κανόνες και οι διαδικασίες συλλογισμού (λύσης) που χρησιμοποιεί ο ειδικός για τη λύση προβλημάτων σχετικών με την υπ' όψιν εφαρμογή. Οι τεχνικές αυτές περιγράφονται εν συντομία στη συνέχεια.

Δομημένες συνεντεύξεις (Structured Interviews)

Στην πιό απλή μορφή τους, σ' ενα προκαταρτικό στάδιο, οι συνεντεύξεις είναι μια σειρά από αδόμητες συνομιλίες με τον ειδικό, ώστε να σκιαγραφηθεί το πρόβλημα. Στα επόμενα στάδια απόκτησης γνώσης μπορεί να είναι ημι-δομημένες συζητήσεις ή συζητήσεις εντοπισμένες σε ορισμένα θέματα, χωρίς όμως συγκεκριμένες ερωτήσεις. Όμως, στη συνέχεια απαιτείται η διενέργεια δομημένων συνεντεύξεων, με καθορισμένες ερωτήσεις που καθοδηγούν τον εμπειρογνώμονα. Καλό είναι οι συνεντεύξεις να μαγνητοφωνούνται, με την άδεια του ειδικού.

Ανάλυση πρωτοκόλλων (Protocol Analysis)

Στην τεχνική αυτή, η διαδικασία δεν καθοδηγείται από τον μηχανικό γνώσης, όπως στην προηγουμένη, αλλά από τον ειδικό. Ο ειδικός υποχρεούται να σκέπτεται "φωναχτά", ενώ επεξεργάζεται κάποιες περιπτώσεις προβλημάτων, κατά προτίμηση πραγματικές. Η μαγνητοφώνηση συνίσταται και εδώ. Η ανάλυση πρωτοκόλλου είναι πολύ χρήσιμη για την εξαγωγή της γενικής δομής της γνώσης που χρησιμοποιεί ο ειδικός και του τρόπου με τον οποίο την εφαρμόζει. Πιθανόν να χρειάζεται να συμπληρωθεί και από άλλη τεχνική (π.χ. δομημένες συνεντεύξεις) για την εξαγωγή λεπτομερειών της γνώσης.

Παρατήρηση (Observation)

Στην τεχνική αυτή ο μηχανικός γνώσης είναι απλός παρατηρητής του τρόπου με τον οποίο ο ειδικός επιτελεί κάποια νοητική εργασία που πρέπει να

αποτυπωθεί στη ΒΓ. Συνήθως ο μηχανικός γνώσης δεν διακόπτει καθόλου, αλλά αυτό εξαρτάται από τη φύση της εφαρμογής. Εδώ συνίσταται η χρήση μαγνητοφώνησης ή και βιντεοσκόπησης.

Αυτοεξέταση (Introspection)

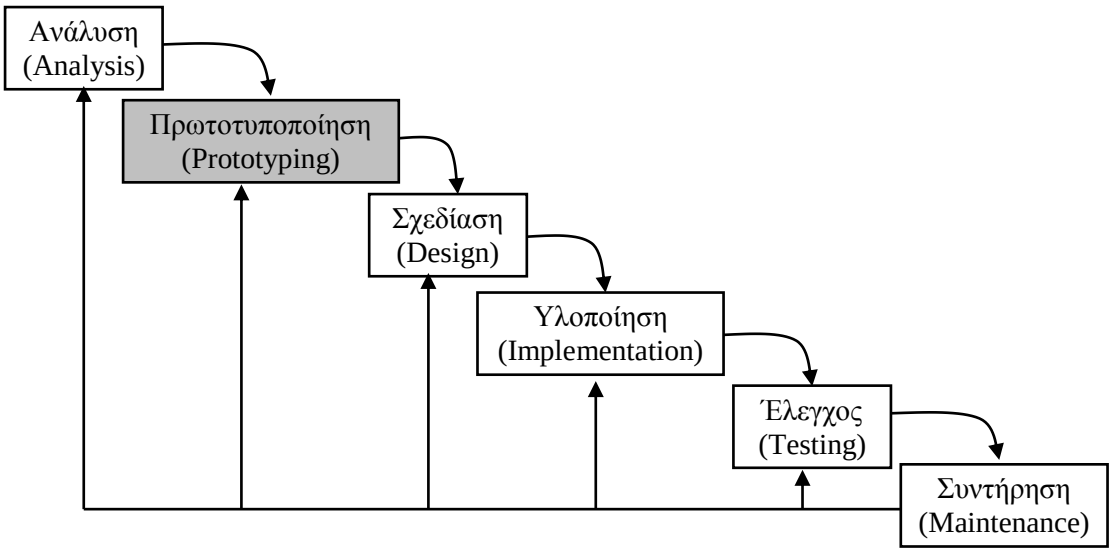
Συχνά αποκαλείται τεχνική τελευταίας καταφυγής. Στην τεχνική αυτή, κατά τη διάρκεια επεξεργασίας μιας περίπτωσης, ο μηχανικός διακόπτει τον ειδικό για να κάνει ερωτήσεις του τύπου: "πες μου τι σκέπτεσαι τώρα" ή "πως θα μπορούσα να κάνω αυτό". Αυτή η τακτική έρχεται σε αντίθεση με τη μέθοδο πρωτοκόλλου, όπου ο μηχανικός διακόπτει μόνο για να θυμίσει στον εμπειρογνώμονα να συνεχίσει ομιλών.

15.6 Μεθοδολογία Ανάπτυξης ΕΣ

Ενώ για την ανάπτυξη ενός συμβατικού συστήματος επεξεργασίας δεδομένων μπορεί να καθοριστεί μια συγκεκριμένη μεθοδολογία ανάπτυξης, σαν ένα σύνολο διαδοχικών διακεκριμένων μεταξύ τους φάσεων, γνωστό σαν κύκλος ζωής (life cycle) του συστήματος, δεν είναι εύκολο να γίνει το ίδιο με ένα ΕΣ. Οι βασικοί λόγοι γι’ αυτό είναι οι εξής:

- ο μη αλγοριθμικός χαρακτήρας της λύσης των προβλημάτων
- η μη ντετερμινιστική συμπεριφορά της διαδικασίας συλλογισμού
- η μη ύπαρξη ακριβούς σχέσεως εισόδου-εξόδου.

Για τους λόγους αυτούς, η ανάπτυξη ενός ΕΣ βασίζεται κυρίως στην ανάπτυξη ενός προτύπου συστήματος (prototype system) και την επαναληπτική του βελτίωση. Μπορεί όμως, παρ' όλα αυτά να δοθεί μια κατά προσέγγιση μεθοδολογία ανάπτυξης ενός ΕΣ κατ' αντιστοιχία με αυτή ενός συμβατικού συστήματος, όπως φαίνεται στο σχήμα 15.4. Όπως είναι φανερό, σε σχέση με τη συμβατική μεθοδολογία υπάρχει μεγάλύτερη αλληλοκάλυψη των φάσεων και έντονότερα επαναληπτική προσέγγιση.



Σχήμα 15.4 Κύκλος ζωής έμπειρων συστημάτων

Στη συνέχεια παρουσιάζουμε εν συντομία τις διαδικασίες που συμπεριλαμβάνει κάθε φάση.

Ανάλυση

Η φάση της ανάλυσης περιλαμβάνει δύο υποφάσεις, την *ανάλυση προβλήματος* (problem analysis) και τον *προσδιορισμό απαιτήσεων* (requirements specification), που περιλαμβάνουν τις παρακάτω διαδικασίες η κάθε μια.

- *Ανάλυση προβλήματος*
  - ✓ Εκτίμηση εφαρμοσιμότητας ΕΣ
  - ✓ Εκτίμηση διαθεσιμότητας πόρων
  - ✓ Εκτίμηση κόστους-ωφέλειας
- *Προσδιορισμός απαιτήσεων*
  - ✓ Προσδιορισμός στόχων και μέσων επίτευξης
  - ✓ Προσδιορισμός απαιτήσεων χρήστη (είσοδοι-έξοδοι)
  - ✓ Προσδιορισμός απαιτήσεων συστήματος (περιορισμοί)

Προτυποποίηση

Επίσης, η φάση της προτυποποίησης περιλαμβάνει κι' αυτή δύο υποφάσεις, την *προκαταρτική σχεδίαση* και την *δημιουργία και αξιολόγηση πρωτοτύπου*. Στη συνέχεια αναλύονται επιγραμματικά οι δύο αυτές υποφάσεις.

- Προκαταρτική Σχεδίαση
  - ✓ Σχεδίαση αρχιτεκτονικής πρωτοτύπου
  - ✓ Μικρής κλίμακας απόκτηση γνώσης
  - ✓ Επιλογή Γλώσσας Αναπαράστασης
  - ✓ Επιλογή Εργαλείου Ανάπτυξης
- Δημιουργία και Αξιολόγηση Πρωτοτύπου
  - ✓ Αναπαράσταση Γνώσης
  - ✓ Υλοποίηση στον Η/Υ
  - ✓ Εγκυροποίηση πρωτοτύπου

Το πρωτότυπο είναι ένα μικρό ΕΣ, που όμως καλύπτει όλες τις βασικές λειτουργίες που θα έχει το τελικό προϊόν. Η τύχη του πρωτότυπου μπορεί να είναι (α) ολική απόρριψη και επανασχεδίαση του τελικού ΕΣ ή (β) συνέχιση της ανάπτυξής του και ως προς την ολοκλήρωση της αναπαράστασης γνώσης, με τη λεγόμενη *αυξητική ανάπτυξη* (incremental development) της ΒΓ, και ως προς τις άλλες λειτπυργικότητες του συστήματος. Η δεύτερη αυτή επιλογή γίνεται όταν οι επιλογές που έγιναν ως προς τη ΓΑΓ και τα εργαλεία ανάπτυξης αποδειχθούν σωστές. Η αυξητική ανάπτυξη, δηλ. η πρόσθεση επι πλέον γνώσης χωρίς να πειράχθεί η υπάρχουσα, μπορεί να υλποιηθεί λόγω της ιδιότητας της τμηματικότητας των κανόνων που αναφέραμε πιο πάνω. Όλα αυτά φυσικά γίνονται στην επόμενες δύο φάσεις.

<u>Σχεδίαση</u>

Η σχεδίαση δεν έχει σχέση μόνο με την τελική σχεδίαση του συστήματος, αλλά και με τις τελικές επιλογές ΓΑΓ και εργαλείου ανάπτυξης. Περιλαμβάνει τα εξής:

- Τελική επιλογή γλώσσας αναπαράστασης
- Τελική επιλογή εργαλείου ανάπτυξης
- Τελική/Λεπτομερής σχεδίαση αρχιτεκτονικής
- Προσδιορισμός εισόδων-εξόδων υποσυστημάτων

<u>Υλοποίηση</u>

Η φάση της υλοποίησης περιλαμβάνει:

- Πλήρης Απόκτηση Γνώσης (Knowledge Acquisition)

- Αναπαράσταση Γνώσης (Knowledge Representation)
- Αυξητική Ανάπτυξη (Incremental Development)

Έλεγχος

Η φάση του ελέγχου περιλαμβάνει δύο υποφάσεις, την *επαλήθευση* (verification) και την *εγκυροποίηση* (validation). Και οι δύο αναφέρονται κυρίως στη βάση γνώσης (κανόνων). Η πρώτη αφορά κυρίως τη συνέπεια και την πληρότητα μιας ΒΓ, ενώ η δεύτερη την ορθότητα της ΒΓ. Μια επαληθευμένη βάση γνώσης σημαίνει τουλάχιστον ότι αναπαριστά σωστά τη γνώση που αποκτήθηκε από εμπειρογνώμονες ή άλλες πηγές γνώσης. Δεν σημαίνει όμως ότι θα παρέχει και σωστές λύσεις/απαντήσεις. Αυτό το δεύτερο το εγγυάται μια εγκυροποιημένη ΒΓ.

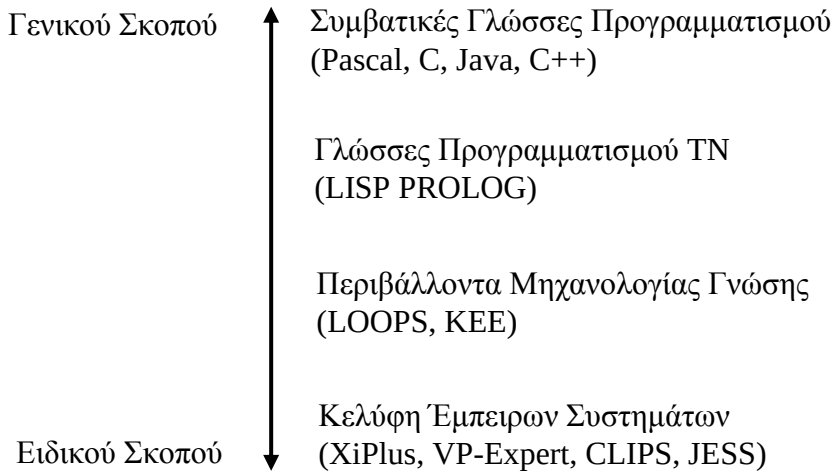
Οι επί μέρους διαδικασίες που περιλαμβάνουν έχουν ως εξής:

- Επαλήθευση
  - ✓ Συμφωνία με προδιαγραφές
  - ✓ Συνέπεια και πληρότητα βάσης γνώσης
- Εγκυροποίηση
  - ✓ Ορθότητα αναπαράστασης
  - ✓ Ορθότητα συμπερασμάτων

15.7 Εργαλεία Ανάπτυξης ΕΣ

Μπορούμε να διακρίνουμε τέσσερεις κατηγορίες εργαλείων για την κατασκευή ΕΣ (expert system building tools). Οι κατηγορίες αυτές παρουσιάζονται στο σχήμα 15.5.

Ένα ΕΣ μπορεί να υλοποιθεί σε μια οποιαδήποτε συμβατική γλώσσα προγραμματισμού, όπως π.χ. C, Pascal κλπ. Μια τέτοια επιλογή προσφέρει ευελιξία στη σχεδίαση και στην υλοποίηση του συστήματος, καθώς και μεγαλύτερες ταχύτητες εκτέλεσης. Αυτό οφείλεται στο ότι η κατασκευή του συστήματος μπορεί να προσαρμοστεί στις ιδιαίτερες ανάγκες της εφαρμογής, όχι μόνο όσον αφορά θέματα υψηλού επιπέδου, αλλά και τις δομές και τους μηχανισμούς χαμηλού επιπέδου.



Σχήμα 15.5. Εργαλεία Ανάπτυξης ΕΣ

Το ίδιο ισχύει και για τις γλώσσες ΤΝ, που προσφέρουν πλεονεκτήματα ανάλογα (αλλά σε μικρότερο βαθμό) με αυτά των συμβατικών γλωσσών. Το μειονέκτημα της χρήσεως τέτοιων εργαλείων (συμβατικές γλώσσες ή γλώσσες ΤΝ) είναι ο μεγάλος χρόνος ανάπτυξης του συστήματος (μικρότερος για γλώσσες ΤΝ) που απαιτείται. Αυτό συμβαίνει, πρώτον διότι ο μηχανικός γνώσης πρέπει να ασχοληθεί με την υλοποίηση όλων των μηχανισμών υψηλού και χαμηλού επιπέδου, και δεύτερον διότι δεν προσφέρουν καμμία βοήθεια όσον αφορά την απόκτηση, μοντελοποίηση ή την αναπαράσταση της γνώσης. Επίσης, το παραχθέν σύστημα είναι δύσκολο, αν όχι αδύνατον, να χρησιμοποιηθεί και σε άλλες εφαρμογές.

Ένα περιβάλλον ΜΓ (KE environment) αποτελείται από ένα σύνολο διαφόρων τύπων γλωσσών αναπαράστασης και προγραμματισμού της γνώσης. Οι γλώσσες που παρέχονται είναι οι ακόλουθοι:

- βασισμένες σε λογική (logic-based)
- βασισμένες σε πλαίσια (frame-based)
- βασισμένες σε κανόνες (rule-based)
- διαδικαστικές (procedural)
- προσανατολισμένες σε αντικείμενα (object-oriented)
- προσανατολισμένες σε προσπελάσεις (access-oriented)

Κάθε περιβάλλον ΜΓ δεν παρέχει όλες τις παραπάνω γλώσσες, αλλά μερικές από αυτές ολοκληρωμένες σε ένα περιβάλλον ανάπτυξης, υποστηριζόμενες και από άλλες διευκολύνσεις, όπως διορθωτή (editor) , ανιχνευτή λαθών (debugger) κλπ. Παραδείγματα ευρέως χρησιμοποιούμενων περιβαλλόντων ΜΓ είναι τα LOOPS και ΚΕΕ. Το περιβάλλον LOOPS π.χ. παρέχει διαδικασίες, αντικείμενα, κανόνες και προσπελάσεις.

Τα περιβάλλοντα ΜΓ προσφέρουν πραγματικά μεγάλη ευελιξία στην αναπαράσταση και τον προγραμματισμό της γνώσης. Ο μηχανικός γνώσης έχει να επιλέξει από μια ποικιλία τυποποιήσεων και τρόπων αναπαράστασης. Επίσης προσφέρουν ευελιξία και όσον αφορά την υλοποίηση. Όμως παρέχουν λίγη, αν όχι καθόλου, καθοδήγηση για το ποιά γλώσσα είναι κατάλληλη για ποιό είδος γνώσεως, υποστηρίζοντας έτσι ένα ευκαιριακό (ad hoc) στυλ αναπαράστασης και προγραμματισμού. Κάτω δε από αυτές τις συνθήκες, η συντήρηση του προγράμματος αποβαίνει ένα δύσκολο έργο.

Ένα κέλυφος ΕΣ (ES shell) βασικά αποτελείται από ένα ΜΕΣ και μια κενή ΒΓ. Ο ΜΕΣ έχει σχεδιαστεί σύμφωνα με τις ανάγκες ενός συνόλου ομοειδών προβλημάτων. Τα κελύφη ΕΣ προσφέρουν μια, αλλά συχνά υβριδική, γλώσσα αναπαράστασης (π.χ. μια που συνδιάζει κανόνες, αντικείμενα ή πλαίσια και διαδικασίες). Με εκφράσεις της γλώσσας αυτής γεμίζει ο μηχανικός γνώσης τη ΒΓ, κωδικοποιώντας (αναπαριστώντας) τη γνώση τη σχετική με μια συγκεκριμένη εφαρμογή. Επίσης, τα κελύφη ΕΣ είναι εφοδιασμένα με ορισμένα βοηθητικά εργαλεία που υποβοηθούν αφ' ενός μεν την απόκτηση γνώσης από τον εμπειρογνώμονα, αφ' ετέρου δε τον έλεγχο του συστήματος. Τέτοια βοηθητικά εργαλεία είναι:

- μονάδα απόκτησης γνώσης
- μονάδα επικοινωνίας χρήστη
- μονάδα παροχής εξηγήσεων
- μονάδα ελέγχου εγκυρότητος

Έτσι, ένα κέλυφος ΕΣ μπορεί να θεωρηθεί σαν ένα λιγώτερο ή περισσότερο πλήρες εργαλείο ανάπτυξης ΕΣ, που υποβηθεί και καθοδηγεί τον μηχανικό στο έργο του.

Όμως, ένα κέλυφος ΕΣ παρέχει μικρότερη ευελιξία από ένα περιβάλλον ΜΓ στην αναπαράσταση της γνώσης. Από την άλλη όμως, προσφέρει σημαντική καθοδήγηση όχι μόνο για την απόκτηση και αναπαράσταση της γνώσης, αλλά και για τη χρήση και συντήρηση του συστήματος. Ένα άλλο μειονέκτημα ενός κελύφους ΕΣ είναι ότι δεν είναι κατάλληλο για οποιαδήποτε εφαρμογή.

16 ΑΝΑΠΤΥΞΗ ΕΣ ΔΙΑΓΝΩΣΗΣ ΚΑΡΚΙΝΟΥ

16.1 Το Πρόβλημα και η Γνώση

Στο παρόν παράδειγμα παρουσιάζεται η δημιουργία ενός έμπειρου συστήματος για την διάγνωση της μορφής καρκίνου ενός εξεταζόμενου με βάση συγκεκριμένες φυσικές διαταραχές που διαπιστώνονται στον ασθενή και τα συμπτώματα που καταγράφονται από την κλινική εξέτασή του. Το παρόν σύστημα, που μπορεί και να στεγασθεί σε έναν ιστοτόπο για χρήση από ενδιαφερόμενους μη ειδικούς ιατρούς, πρέπει να τονισθεί ότι δεν περιέχει γνώση εμπειρογνωμόνων, αλλά μόνο γνώση από διάφορες ηλεκτρονικές πηγές, και θα πρέπει ο κάθε ενδιαφερόμενος να παραπέμπεται σε ειδικούς για την πιστοποίηση των διαγνώσεων του συστήματος.

Για την δημιουργία του παρόντος παραδείγματος χρησιμοποιήθηκε γενική γνώση για την συμπτωματολογία διαφόρων μορφών καρκίνου που συγκεντρώθηκε από τους παρακάτω ιστοτόπους:

www.cancerindex.org

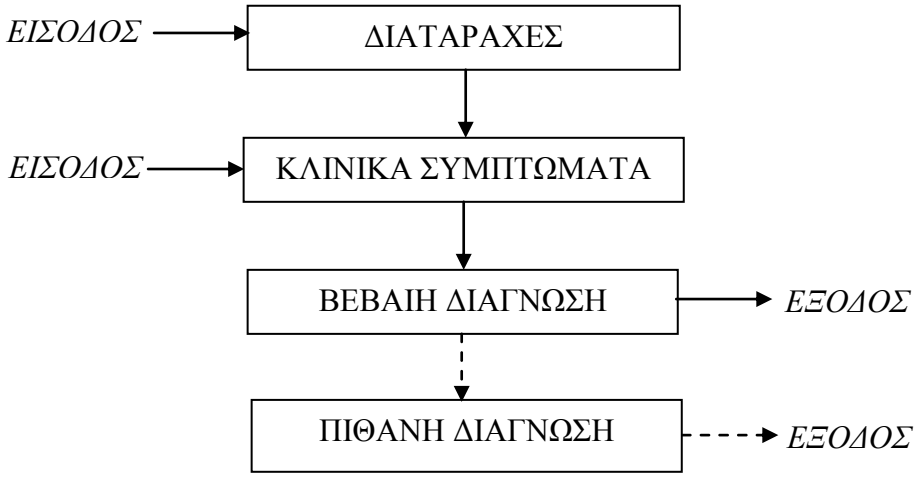
www.acor.org/types.html

www.cancerguide.org/basic.html

www.cancure.org/home.htm

Η διαδικασία διάγνωσης του (ενδεχόμενου) είδους καρκίνου για τις ανάγκες του συστήματός μας (σύμφωνα με τις πηγές γνώσης) απεικονίζεται στο Σχήμα 16.1. Σύμφωνα μ’ αυτήν, κατ’ αρχήν ζητείται από τον εξεταζόμενο η διαταραχή (ή οι διαταραχές) που έχει διαπιστώσει. Στη συνέχεια ζητούνται/διερευνώνται τα (κλινικά) συμπτώματα που πιθανώς συνδέονται με (ή προκάλεσαν) τη διαταραχή αυτή (ή τις διαταραχές). Κατόπιν, με βάση τα συμπτώματα, αν είναι επαρκή, προσδιορίζουμε (με βεβαιότητα) το είδος του καρκίνου. Αν όμως τα κλινικά δεδομένα (συμπτώματα) δεν

επαρκούν ή είναι αλληλοσυγκρουόμενα, τότε γίνεται προσδιορισμός του είδους του καρκίνου με κάποια πιθανότητα (ή ποσοστό βεβαιότητας).



Σχήμα 16.1 Διαδικασία διάγνωσης είδους καρκίνου

Πίνακας 16.1. Διαταραχές και αντίστοιχα συμπτώματα

ΔΙΑΤΑΡΑΧΕΣ	**URINE**	**WEAKNESS**	**CHANGE OF SHAPE**	**FOCUSED PAIN**
ΣΥΜΠΤΩΜΑΤΑ	Bloody	Fatigue	Bone fracture	Bone pain
	Pain	Weakness	Enlarged lymphs	Numbness
	Cloudy	Paleness	Mouth infections	Headache
	Frequent	Loss of appetite	Abdominal swellings	Breast pain
	Bladder with urine			Cramp
				Backside pain
				Chest pain
				Oral pain
				Pelvic pain

Δεδομένης της παραπάνω διαδικασίας, έπρεπε να προσδιοριστούν αφ'ενός μεν οι διαταραχές, αφ' ετέρου δε τα συμπτώματα με τα οποία συνδέεται (ή από τα οποία προκαλείται) κάθε διαταραχή. Οι διαταραχές και τα συμπτώματα αποτελούν τις παραμέτρους εισόδου του συστήματος, ενώ τα είδη καρκίνου τις παραμέτρους εξόδου του συστήματος. Με βάση την υπάρχουσα γνώση δημιουργήθηκαν οι Πίνακες 16.1 και 16.2. Ο μεν πίνακας 16.1 καταγράφει ποια συμπτώματα συνδέονται με κάθε διαταραχή, ο δε 16.2 συσχετίζει τα είδη καρκίνου με τα συμπτώματα (κυρίως) και με

τις διαταραχές, δηλ. προσδιορίζει ποια συμπτώματα και διαταραχές οδηγούν σε διάγνωση ποιού είδους καρκίνου. Οι πίνακες είναι ενδεικτικοί και δεν καταγράφονται π.χ. όλες οι διαταραχές ή όλα τα είδη καρκίνου.

Πίνακας 16.2 Συσχέτιση συμπτωμάτων/διαταραχών και είδους καρκίνου

ΕΙΔΟΣ ΚΑΡΚΙΝΟΥ	**BLADDER CANCER**	**BONE CANCER**	**BRAIN CANCER**	**UTERINE CANCER**	**… CANCER**
ΔΙΑΤΑΡΑΧΕΣ & ΣΥΜΠΤΩΜΑΤΑ	Bloody urine	Bone fracture	Dizziness	Pelvic pain	…
	Cloudy urine	Bone pain	Vomiting	Vaginal bleeding	…
	Urination Pain	Weakness	Nausea	Urination pain	…
	Frequent urination	Fatigue	Weakness		…
		Weight loss	Abnormal eye movements		…
		Nausea	Loss of control		…
		Vomiting	Personality change		…
		Constipation	Memory-speech change		…
		Numbness			…

16.2 Σχεδίαση των κανόνων

Με βάση τη διαδικασία του σχήματος 16.1 και τα δεδομένα των πινάκων 16.1 και 16.2 αποφασίζουμε να δημιουργήσουμε τέσσερις ομάδες κανόνων, που είναι οι εξής:

- Κανόνες εισαγωγής διαταραχών
- Κανόνες εισαγωγής συμπτωμάτων
- Κανόνες βέβαιης διάγνωσης
- Κανόνες πιθανής διάγνωσης

Επίσης, δημιουργήσαμε και μια πέμπτη ομάδα κανόνων για την εκτύπωση του αποτελέσματος:

- Κανόνες εκτύπωσης αποτελεσμάτων

Οι κανόνες εισαγωγής διαταρχών ουσιαστικά αλληλεπιδρούν με τον χρήστη μέσω ερωτήσεων για την καταγραφή των διαταραχών που έχει εντοπίσει. Στην

πραγματικότητα, δημιουργήσαμε ένα τέτοιο κανόνα, που είναι και ο κανόνας εκκίνησης του συστήματος. Ο κανόνας αυτός κάνει τις κατάλληλες ερωτήσεις και εισάγει τα αντίστοιχα γεγονότα που αφορούν διαταραχές μέσω μιας συνάρτησης, της get-distortions. Ο κανόνας αυτός και η συνάρτηση είναι:

```
(defrule start-get-distortions
   "initialize and get distortions"
=>
 (set-strategy depth)
 (printout t "Simple instructions you must follow!!!" crlf)
 (printout t "In simple yes or no questions just answer with a y or n"
      crlf)
 (printout t "In questions that need more complicated answers type
      whatever you want" crlf)
 (printout t "What's your name patient?" crlf)
 (bind ?name (read))
 (assert (user-name ?name))
 (printout t "Well " ?name " what's your sex?" crlf)
 (bind ?sex (read))
 (assert (user-sex ?sex))
 (assert (symptoms nil))
 (get-distortions "Have you got problems with urination?" urination)
)


(deffunction get-distortions (?question ?distortion)
 (bind ?i 1)
 (bind ?questions
      (create$ "Have you ever felt focused pain?" focused_pain
               "Have you ever felt loss of control?" loss_of_control
               "Have you ever had some kind of emotional change?"
                                                  emotional_change
               "Have you ever felt noisy feelings?" noisy_feelings
               "Do you often notice severe bleeding in some part of your
                                                       body?" bleeding
               "Are you often feeling weak?" weakness
               "Have you notived ill-like-symptoms?" ill_like_symptoms
               "Have you ever noticed some abnormality on your shape?"
                                                       change_of_shape
               "Finally have you ever noticed some change of colour or
                                  shape on your skin?" change_of_skin))
 (printout t ?question " " crlf)
 (bind ?answer (read))
 (if (eq ?answer y)
  then (assert (distortion ?distortion)))
 (printout t "Do you want to continue with the distortions?" crlf)
 (bind ?answer (read))
;Oso exoume diataraxes synexizoume tis erwtiseis
 (while (and (eq ?answer y) (<= ?i 18))
  do
  (printout t (nth$ ?i ?questions) crlf)
  (bind ?answer (read))
  (if (eq ?answer y)
   then (assert (distortion (nth$ (+ 1 ?i) ?questions))))
```

```
  (printout t "Do you want to mention more distortions?" crlf)
  (bind ?answer (read))
  (bind ?i (+ 2 ?i))
))
```

Οι κανόνες εισαγωγής συμπτωμάτων ουσιαστικά αλληλεπιδρούν με τον χρήστη μέσω ερωτήσεων για την καταγραφή των συμπτωμάτων των διαταραχών που ανέφερε ο εξεταζόμενος. Χρειαζόμαστε ένα κανόνα για κάθε διαταραχή. Κάθε τέτοιος κανόνας κάνει τις κατάλληλες ερωτήσεις και εισάγει τα αντίστοιχα γεγονότα που αφορούν τα συμπτώματα των διαταραχών (με βάση τον Πίνακα 16.1). Και δω χρησιμοποιούμε μια βοηθητική συνάρτηση, την get-symptoms. Ο κανόνας για την διαταραχή ‘urination’, καθώς και η συνάρτηση είναι:

```
(defrule has-urination
"urination problems"
(declare (salience 100))
(distortion urination)
=>
(printout t "Let's take a look at your urination problems" crlf crlf)
(find-symptoms "Have you noticed blood in the urine?" blood_urine)
(find-symptoms "Are you feeling pain or burning upon urination?"
   pain_urination)
(find-symptoms "Have you cloudy urine?" cloudy_urine)
(find-symptoms "Have you frequent urination?" frequent_urination)
(find-symptoms "Does bladder retains urine?" bladder_with_urine)
(printout t crlf crlf)
)


(deffunction get-symptoms (?question ?symptom)
 (printout t ?question " " crlf)
 (bind ?answer (read))
 (if (eq ?answer y)
  then (assert (symptom ?symptom)))
 (if (eq ?symptom weakness)
  then (printout t "Does the weakness appear in one of the below parts
of your body? If yes type carefully the specific part." crlf)
       (printout t "legs arms chest" crlf)
       (bind ?part (read))
       (assert (symptom ?part weakness))
  else (if (eq ?symptom enlarged_lymph)
       then (printout t "Where do the lymphs appear? If yes type
carefully one of the provided parts." crlf)
            (printout t "arms chest legs neck?" crlf)
            (bind ?part (read))
            (assert (symptom ?part lymph))))
)
```

Οι κανόνες βέβαιης διάγνωσης συμπεραίνουν μετά βεβαιότητος το είδος του καρκίνου, εφ' όσον οι διαταραχές και τα συμπτώματα που έχουν παρατηρηθεί είναι τέτοια που δεν αφήνουν καμμιά αμφιβολία. Χρειαζόμαστε ένα κανόνα για κάθε είδος καρκίνου. Οι κανόνες σχεδίαζονται με βάση τον πίνακα 16.2. Δύο τέτοιοι κανόνες είναι οι παρακάτω:

```
(defrule bladder-cancer
"Karkinos tis ourodoxou kistis"
(declare (salience 60))
(symptom blood_urine)
(symptom pain_urination)
(symptom frequent_urination)
(symptom cloudy_urine)
=>
(assert (cancer bladder_cancer))
)

(defrule bone-cancer
"karkinos ostwn"
(declare (salience 60))
(symptom bone_pain)
(symptom bone_fracture)
(symptom fatigue)
(symptom weight_loss)
(symptom nausea)
(symptom constipation)
(or (symptom legs weakness) (symptom arms weakness))
=>
(assert (cancer bone_cancer))
```

Οι κανόνες πιθανής διάγνωσης συμπεραίνουν το είδος του καρκίνου με κάποια πιθανότητα, αλλά όχι βεβαιότητα, όταν οι διαταραχές και τα συμπτώματα που έχουν παρατηρηθεί είναι τέτοια που δεν οδηγούν με βεβαιότητα σε ένα είδος καρκίνου. Χρειαζόμαστε ένα κανόνα για κάθε είδος καρκίνου. Δύο τέτοιοι κανόνες (οι αντίστοιχοι με τους παραπάνω) είναι οι παρακάτω:

```
(defrule bladder-cancer-possibility
"Choose the possibility for that cancer to occur with the provided
 symptoms"
(declare (salience 50))
(end_of_test true)
(not (cancer bladder_cancer))
(distortion urination)
=>
(printout t "There is a small indication of bladder cancer to occur"
   crlf)
(printout t "I will list a few symptoms that are stricly linked with
```

```
   that type of cancer" crlf crlf)
(printout t "blood_urine" crlf "pain_urination" crlf
   "frequent_urination" crlf "cloudy_urine" crlf crlf)
(printout t "Give me the number of the symptoms above that you have
   experienced:")
(bind ?num (read))
(bind ?poss (* (/ (- ?num 1) 4) 100))
(assert (cancer_possibility bladder_cancer ?poss))
)


(defrule bone-cancer-possibility
"Choose the possibility for that cancer to occur with the provided
   symptoms"
(declare (salience 50))
(end_of_test true)
(not (cancer bone_cancer))
(or (symptom bone_pain) (symptom bone_fracture))
(distortion weakness)
=>
(printout t "There is a small indication of bone cancer to occur"
   crlf)
(printout t "I will list a few symptoms that are stricly linked with
   that type of cancer" crlf crlf)
(printout t "bone_pain" crlf "bone_fracture" crlf "nausea weakness"
   crlf crlf)
(printout t "Give me the number of the symptoms above that you have
   experienced:")
(bind ?num (read))
(bind ?poss (* (/ (- ?num 1) 4) 100))
(assert (cancer_possibility bone_cancer ?poss))
)
```

Τέλος, υπάρχουν τρεις κανόνες εκτύπωσης αποτελεσμάτων, ένας για την περίπτωση βέβαιας διάγνωσης, ένας για την περίπτωση πιθανής διάγνωσης και ένας για την περίπτωση που δεν μπορεί να εξαχθεί συμπέρασμα για κάποιο είδος καρκίνου, ούτε με κάποια πιθανότητα:

```
(defrule cancer-results
"prints the results of the run"
(declare (salience 55))
(cancer ?x)
=>
(printout t "With the symptoms provided so far I have come to a
conclusion" crlf)
(printout t "Grade of danger: severe. Possible cancer type: " ?x crlf)
(printout t "Type a key to continue" crlf)
(read)
(assert (end_of_test true))
)
```

```
(defrule possibility-of-cancer
"finds all the possibilities for the cancers"
(declare (salience 45))
(cancer_possibility ?cancer ?poss)
=>
(printout t "The possibility for " ?cancer " is " ?poss "%." crlf crlf
   crlf)
)

(defrule no-results
"not some particular cancer found"
(declare (salience 45))
(and (not (cancer ?x)) (user-name ?name))
=>
(printout t ?name " there is no adequate  indication for a specific
cancer type." crlf "Nevertheless we will provide below a list of
possible cancer types that correspond to the symptoms provided." crlf)
(assert (end_of_test true))
)
```

16.3 Προτεραιότητες - Στρατηγική επίλυσης συγκρούσεων

Οι ομάδες κανόνων που προσδιορίσαμε προηγουμένως θα πρέπει να εκτελεστούν με τέτοια σειρά ώστε να αντανακλάται η διαδικασία του σχήματος 16.1. Δηλ. πρώτα ο κανόνας αρχικοποίησης-εισαγωγής διαταραχών, μετά οι κανόνες εισαγωγής συμπτωμάτων, στη συνέχεια οι κανόνες βέβαιης διάγνωσης, κατόπιν ο κανόνας εκτύπωσης βέβαιης διάγνωσης, μετά οι κανόνες πιθανής διάγνωσης και τέλος οι υπόλοιποι κανόνες εκτύπωσης. Αυτό επετεύχθη με τη χρήση κατάλληλων προτεραιοτήτων:

Αρχικός κανόνας: 0

Κανόνες εισαγωγής διαταραχών: 100

Κανόνες βέβαιης διάγνωσης: 60

Κανόνες εκτύπωσης βέβαιης διάγνωσης :55

Κανόνες πιθανής διάγνωσης: 50

Υπόλοιποι κανόνες εκτύπωσης :45

Επίσης ως στρατηγική επίλυσης συγκρούσεων επιλέχθηκε εδώ η depth (τα νέα γεγονότα είναι πιο σημαντικά για την ενεργοποίηση των κανόνων)

Βιβλιογραφία

1. A. Gonzalez, D. Dankel The Engineering of Knowledge-Based Systems: Theory and Practice, Prentice-Hall, 1993.
2. F. Hayes-Roth, D. A. Waterman and D. B. Lenat (Eds). Building Expert Systems, Addison Wesley, 1983.

ΠΑΡΑΡΤΗΜΑ

Κώδικας CLIPS

```
;; Empeiro Systhma Diagnosis Eidous Karkinou
;; Eyfyhs Programmatismos 2005-2006

;;Synartisi gia thn eisagwgi twn diataraxwn tou eksetazomenou mesw erwthsewn

(deffunction get-distortions (?question ?distortion)
 (bind ?i 1)
 (bind ?questions
      (create$ "Have you ever felt focused pain?" focused_pain
               "Have you ever felt loss of control?" loss_of_control
               "Have you ever had some kind of emotional change?" emotional_change
               "Have you ever felt noisy feelings?" noisy_feelings
               "Do you often notice severe bleeding in some part of your body?" bleeding
               "Are you often feeling weak?" weakness
               "Have you notived ill-like-symptoms?" ill_like_symptoms
               "Have you ever noticed some abnormality on your shape?" change_of_shape
               "Finally have you ever noticed some change of colour or shape on your skin?" change_of_skin))
 (printout t ?question " " crlf)
 (bind ?answer (read))
 (if (eq ?answer y)
  then (assert (distortion ?distortion)))
 (printout t "Do you want to continue with the distortions?" crlf)
 (bind ?answer (read))
;Oso exoume diataraxes synexizoume tis erwtiseis
 (while (and (eq ?answer y) (<= ?i 18))
  do
  (printout t (nth$ ?i ?questions) crlf)
  (bind ?answer (read))
  (if (eq ?answer y)
   then (assert (distortion (nth$ (+ 1 ?i) ?questions))))
  (printout t "Do you want to mention more distortions?" crlf)
  (bind ?answer (read))
  (bind ?i (+ 2 ?i))
))


;;Synartisi gia thn eisagwgi twn symptomatwn tou eksetazomenou mesw erwthsewn

(deffunction get-symptoms (?question ?symptom)
 (printout t ?question " " crlf)
 (bind ?answer (read))
 (if (eq ?answer y)
  then (assert (symptom ?symptom)))
```

```
 (if (eq ?symptom weakness)
  then (printout t "Does the weakness appear in one of the below parts of
your body? If yes type carefully the specific part." crlf)
       (printout t "legs arms chest" crlf)
       (bind ?part (read))
       (assert (symptom ?part weakness))
  else (if (eq ?symptom enlarged_lymph)
       then (printout t "Where do the lymphs appear? If yes type
carefully one of the provided parts." crlf)
            (printout t "arms chest legs neck?" crlf)
            (bind ?part (read))
            (assert (symptom ?part lymph))))
)


(defrule start-get-distortions "initialize and get distortions"

=>
(set-strategy depth)
(printout t "Simple instructions you must follow!!!" crlf)
(printout t "In simple yes or no questions just answer with a y or n"
crlf)
(printout t "In questions that need more complicated answers type
whatever you want" crlf)
(printout t "What's your name patient?" crlf)
(bind ?name (read))
(assert (user-name ?name))
(printout t "Well " ?name " what's your sex?" crlf)
(bind ?sex (read))
(assert (user-sex ?sex))
(assert (symptoms nil))
(get-distortions "Have you got problems with urination?" urination)
)


;;Kanones gia thn eisagwgi symptomatwn twn diataraxwn

(defrule has-urination
"urination problems"
(declare (salience 100))
(distortion urination)
=>
(printout t "Let's take a look at your urination problems" crlf crlf)
(get-symptoms "Have you noticed blood in the urine?" blood_urine)
(get-symptoms "Are you feeling pain or burning upon urination?"
      pain_urination)
(get-symptoms "Have you cloudy urine?" cloudy_urine)
(get-symptoms "Have you frequent urination?" frequent_urination)
(get-symptoms "Does bladder retains urine?" bladder_with_urine)
(printout t crlf crlf)
)
```

```
(defrule has-focused-pain
"focused pain on some specific part of the body"
(declare (salience 100))
(distortion focused_pain)
=>
(printout t "Let's take a look at your focused pain problems" crlf crlf)
(get-symptoms "Have you ever felt bone pain?" bone_pain)
(get-symptoms "Have you ever had constipation or abdomen pain?"
      constipation)
(get-symptoms "Have you ever felt numbness in your legs?" numbness)
(get-symptoms "Have you ever felt severe headaches?" headache)
(get-symptoms "Have you ever felt breast pain?" breast_pain)
(get-symptoms "Are you often having cramps?" cramp)
(get-symptoms "Are you feeling pain in the back or the side?"
      back_side_pain)
(get-symptoms "Are you feeling pain in your chest?" chest_pain)
(get-symptoms "Are you feeling oral pain?" oral_pain)
(get-symptoms "Are you feeling pain in pelvic area?" pelvic_pain)
(printout t crlf crlf)
)


(defrule has-loss-of-control
"symptoms that exhibit loss of control"
(declare (salience 100))
(distortion loss_of_control)
=>
(printout t "Let's take a look at your loss of control problems" crlf
crlf)
(get-symptoms "Are you having problems with your vision?" loss_of_vision)
(get-symptoms "Are you having abnormal eye moves?" abnormal_eye_move)
(get-symptoms "Are you losing control of your body?" loss_of_body)
(printout t crlf crlf)
)


(defrule has-emotional-change
"symptoms that exhibit emotional change"
(declare (salience 100))
(distortion emotional_change)
=>
(printout t "Let's take a look at your indication of emotional change"
crlf crlf)
(get-symptoms "Have you noticed any pesonality change?"
personality_change)
(get-symptoms "Have you noticed changes in memory or speech?"
memory_speech_change)
(printout t crlf crlf)
)


(defrule has-noisy-feelings
"examine the symptoms that may be linked to noisy feelings"
(declare (salience 100))
(distortion noisy_feelings)
```

```
=>
(printout t "Let's take a look at your noisy feelings" crlf crlf)
(get-symptoms "Are you often having nausea?" nausea)
(get-symptoms "Are you feeling like vomiting?" vomiting)
(get-symptoms "Are you having feelings of heat?" heat)
(get-symptoms "Are you often having diarrhea?" diarrhea)
(get-symptoms "Do you have constipation?" constipation)
(get-symptoms "Are you having digestive discomfort?"
digestive_discomfort)
(get-symptoms "Are you sweating at night?" night_sweat)
(get-symptoms "Are you having difficulty in breathing?" chest_congestion)
(get-symptoms "Are you having abdominal pains?" abdominal_pain)
(printout t crlf crlf)
)


(defrule has-bleeding
"exhamine where the bleeding comes from"
(declare (salience 100))
(distortion bleeding)
(user-sex ?sex)
=>
(printout t "Let's take a look at your bleeding problems" crlf crlf)
(get-symptoms "Are you having rectal bledding?" rectal_bleeding)
(get-symptoms "Are you experiencing prolonged bleeding?"
prolonged_bleeding)
(get-symptoms "Are you experiencing oral bleeding?" oral_bleeding)
(if (eq ?sex f)
then (get-symptoms "Have you ever had abnormal vaginal bleeding?"
vaginal_bleeding))
(printout t crlf crlf)
)


(defrule has-weakness
"try to find what is the weakness'es cause of"
(declare (salience 100))
(distortion weakness)
=>
(printout t "Let's take a look at your weakness problems" crlf crlf)
(get-symptoms "Are you feeling fatigue?" fatigue)
(get-symptoms "Are you feeling general weakness?" weakness)
(get-symptoms "Are you getting pale?" paleness)
(get-symptoms "Are you experiencing loss of appetite?" loss_of_appetite)
(printout t crlf crlf)
)


(defrule has-ill-like-symptoms
"try to exploit such symptoms"
(declare (salience 100))
(distortion ill_like_symptoms)
=>
(printout t "Let's take a look at the ill-like-symptoms you have noticed"
      crlf crlf)
```

```
(get-symptoms "Are you feeling like having some kind of flu?" flu)
(get-symptoms "Are you having fever?" fever)
(get-symptoms "Are you having persistent cough?" cough)
(printout t crlf crlf)
)


(defrule has-change-of-shape
"what exactly change is there"
(declare (salience 100))
(distortion change_of_shape)
=>
(printout t "Let's take a look at the shape changes you have noticed"
crlf crlf)
(get-symptoms "Have you got fractures in bones?" bone_fracture)
(get-symptoms "Have you noticed enlarged lymph nodes?" enlarged_lymph)
(get-symptoms "Have you got any lumps or infections inside your mouth?"
      mouth_infection)
(get-symptoms "Have you noticed some weird abdominal swelling?"
      abdominal_swelling)
(get-symptoms "Have you noticed unexplained weight loss lately?"
      weight_loss)
(printout t crlf crlf)
)


(defrule has-change-of-skin
"distortions in skin"
(declare (salience 100))
(distortion change_of_skin)
=>
(printout t "Let's take a look at your skin changes that have occured"
crlf crlf)
(get-symptoms "Have you noticed some change in the skin of your chest?"
skin_chest)
(get-symptoms "Have you noticed change in mole on the skin?"
skin_mole_change)
(get-symptoms "Have you got any small lumps on your skin?" skin_lumps)
(get-symptoms "Have you noticed yellowing of the skin?"
yellowing_of_skin)
(printout t crlf crlf)
)


;;Kanones bebaihs diagnosis

(defrule bladder-cancer
"Karkinos tis ourodoxou kistis"
(declare (salience 60))
(symptom blood_urine)
(symptom pain_urination)
(symptom frequent_urination)
(symptom cloudy_urine)
=>
(assert (cancer bladder_cancer))
```

```
)


(defrule bone-cancer
"karkinos ostwn"
(declare (salience 60))
(symptom bone_pain)
(symptom bone_fracture)
(symptom fatigue)
(symptom weight_loss)
(symptom nausea)
(symptom constipation)
(or (symptom legs weakness) (symptom arms weakness))
=>
(assert (cancer bone_cancer))
)


(defrule brain-cancer
"symptoms of brain cancer"
(declare (salience 60))
(or (symptom loss_of_vision) (symptom abnormal_eye_move))
(symptom weakness)
(symptom loss_of_body)
(symptom personality_change)
(symptom memory_speech_change)
(or (symptom nausea) (symptom headache))
=>
(assert (cancer brain_cancer))
)


(defrule breast-cancer
"symptoms of the breast cancer"
(declare (salience 60))
(symptom skin_chest)
(symptom heat)
(or (symptom arms lymph) (symptom chest lymph))
=>
(assert (cancer breast_cancer))
)


(defrule colorectal-cancer
(declare (salience 60))
(symptom rectal_bleeding)
(symptom cramp)
(symptom constipation)
(symptom weight_loss)
(symptom weakness)
=>
(assert (cancer colorectal_cancer))
)
```

```
(defrule kidney-cancer
(declare (salience 60))
(symptom blood_urine)
(symptom back_side_pain)
=>
(assert (cancer kidney_cancer))
)


(defrule leukemia
(declare (salience 60))
(symptom weakness)
(symptom paleness)
(symptom fever)
(symptom flu)
(symptom prolonged_bleeding)
(symptom enlarged_lymph)
(symptom bone_pain)
(symptom night_sweat)
(symptom weight_loss)
=>
(assert (cancer leukemia))
)


(defrule lung-cancer
(declare (salience 60))
(symptom cough)
(symptom chest_pain)
(symptom chest_congestion)
(symptom neck lymph)
=>
(assert (cancer lung_cancer))
)


(defrule melanoma
(declare (salience 60))
(symptom skin_mole_change)
(distortion bleeding)
=>
(assert (cancer melanoma))
)


(defrule hodgkin-lymphoma
(declare (salience 60))
(symptom neck lymph)
(symptom fever)
(symptom weight_loss)
(symptom bone_pain)
=>
(assert (cancer hodgkin_lymphoma))
)
```

```
(defrule oral-cancer
(declare (salience 60))
(symptom mouth_infection)
(symptom oral_pain)
(symptom oral_bleeding)
=>
(assert (cancer oral_cancer))
)


(defrule ovarian-cancer
(declare (salience 60))
(user-sex f)
(symptom abdominal_swelling)
(symptom vaginal_bleeding)
(symptom digestive_discomfort)
=>
(assert (cancer ovarian_cancer))
)


(defrule pancreatic-cancer
(declare (salience 60))
(symptom weight_loss)
(symptom back_side_pain)
(symptom digestive_discomfort)
(symptom yellowing_of_skin)
=>
(assert (cancer pancreatic_cancer))
)


(defrule prostate-cancer
(declare (salience 60))
(symptom bladder_with_urine)
(symptom pain_urination)
(symptom blood_urine)
(symptom back_side_pain)
=>
(assert (cancer prostate_cancer))
)


(defrule stomach-cancer
(declare (salience 60))
(symptom digestive_discomfort)
(symptom abdominal_pain)
(or (symptom nausea) (symptom vomiting))
(or (symptom diarrhea) (symptom constipation))
(or (symptom fatigue) (symptom weakness))
(distortion bleeding)
=>
(assert (cancer stomach_cancer))
)
```

```
(defrule uterine-cancer
(declare (salience 60))
(user-sex f)
(symptom vaginal_bleeding)
(symptom pain_urination)
(symptom pelvic_pain)
=>
(assert (cancer uterine_cancer))
)


;;Kanones pithanis diagnosis

(defrule bladder-cancer-possibility
"Choose the possibility for that cancer to occur with the provided
symptoms"
(declare (salience 50))
(end_of_test true)
(not (cancer bladder_cancer))
(distortion urination)
=>
(printout t "There is a small indication of bladder cancer to occur"
crlf)
(printout t "I will list a few symptoms that are stricly linked with that
      type of cancer" crlf crlf)
(printout t "blood_urine" crlf "pain_urination" crlf "frequent_urination"
      crlf "cloudy_urine" crlf crlf)
(printout t "Give me the number of the symptoms above that you have
      experienced:")
(bind ?num (read))
(bind ?poss (* (/ (- ?num 1) 4) 100))
(assert (cancer_possibility bladder_cancer ?poss))
)


(defrule bone-cancer-possibility
"Choose the possibility for that cancer to occur with the provided
symptoms"
(declare (salience 50))
(end_of_test true)
(not (cancer bone_cancer))
(or (symptom bone_pain) (symptom bone_fracture))
(distortion weakness)
=>
(printout t "There is a small indication of bone cancer to occur" crlf)
(printout t "I will list a few symptoms that are stricly linked with that
      type of cancer" crlf crlf)
(printout t "bone_pain" crlf "bone_fracture" crlf "nausea weakness" crlf
      crlf)
(printout t "Give me the number of the symptoms above that you have
      experienced:")
(bind ?num (read))
(bind ?poss (* (/ (- ?num 1) 4) 100))
```

```
(assert (cancer_possibility bone_cancer ?poss))
)


(defrule prostate-cancer-possibility
"Choose the possibility for that cancer to occur with the provided
symptoms"
(declare (salience 50))
(end_of_test true)
(not (cancer prostate_cancer))
(distortion urination)
=>
(printout t "There is a small indication of prostate cancer to occur."
      crlf)
(printout t "I will list a few symptoms that are stricly linked with that
      type of cancer" crlf crlf)
(printout t "bladder_with_urine" crlf "pain_urination" crlf "blood_urine"
      crlf "back_side_pain" crlf crlf)
(printout t "Give me the number of the symptoms above that you have
      experienced:")
(bind ?num (read))
(bind ?poss (* (/ (- ?num 1) 4) 100))
(assert (cancer_possibility prostate_cancer ?poss))
)


(defrule brain-cancer-possibility
"Choose the possibility for that cancer to occur with the provided
symptoms"
(declare (salience 50))
(end_of_test true)
(not (cancer brain_cancer))
(distortion loss_of_control)
(distortion emotional_change)
=>
(printout t "There is a small indication of brain cancer to occur" crlf)
(printout t "I will list a few symptoms that are stricly linked with that
      type of cancer" crlf crlf)
(printout t "nausea" crlf "problem_with_speaking" crlf  "low_vision" crlf
      "dizziness" crlf  "personality_change" crlf crlf)
(printout t "Give me the number of the symptoms above that you have
      experienced:")
(bind ?num (read))
(bind ?poss (* (/ (- ?num 1) 5) 100))
(assert (cancer_possibility brain_cancer ?poss))
)


(defrule breast-cancer-possibility
"Choose the possibility for that cancer to occur with the provided
      symptoms"
(declare (salience 50))
(end_of_test true)
(not (cancer breast_cancer))
(or (symptom skin_chest)
```

```
(symptom heat)
(symptom arms lymph)
(symptom chest lymph))
=>
(printout t "There is a small indication of breast cancer to occur" crlf)
(printout t "I will list a few symptoms that are stricly linked with that
      type of cancer" crlf crlf)
(printout t "change_in_the_skin_chest" crlf  "feeling_of_heat" crlf
      "lymphs" crlf crlf)
(printout t "Give me the number of the symptoms above that you have
      experienced:")
(bind ?num (read))
(bind ?poss (* (/ (- ?num 1) 3) 100))
(assert (cancer_possibility breast_cancer ?poss))
)


(defrule colorectal-cancer-possibility
"Choose the possibility for that cancer to occur with the provided
symptoms"
(declare (salience 50))
(end_of_test true)
(not (cancer colorectal_cancer))
(or (distortion weakness) (symptom rectal_bleeding))
=>
(printout t "There is a small indication of colorectal cancer to occur"
      crlf)
(printout t "I will list a few symptoms that are stricly linked with that
      type of cancer" crlf crlf)
(printout t "weakness" crlf "rectal bleeding" crlf "constipation" crlf
      "abdominal problems" crlf crlf)
(printout t "Give me the number of the symptoms above that you have
      experienced:")
(bind ?num (read))
(bind ?poss (* (/ (- ?num 1) 3) 100))
(assert (cancer_possibility colorectal_cancer ?poss))
)


(defrule leukemia-possibility
"Choose the possibility for that cancer to occur with the provided
      symptoms"
(declare (salience 50))
(end_of_test true)
(not (cancer leukemia_cancer))
(distortion weakness)
(or
(symptom weight_loss)
(symptom bone_pain)
(symptom enlarges_lymph))
(symptom flu)
=>
(printout t "There is a small indication of leukemia to occur" crlf)
(printout t "I will list a few symptoms that are stricly linked with that
      type of cancer" crlf crlf)
```

```
(printout t "lymphs" crlf "weight_loss" crlf "weakness" crlf "bone_pain"
      crlf "flu_like_symptoms" crlf crlf)
(printout t "Give me the number of the symptoms above that you have
      experienced:")
(bind ?num (read))
(bind ?poss (* (/ (- ?num 1) 5) 100))
(assert (cancer_possibility leukemia ?poss))
)



(defrule hodgkin-lymphoma-possibility
"Choose the possibility for that cancer to occur with the provided
      symptoms"
(declare (salience 50))
(end_of_test true)
(not (cancer hodgkin_lymphoma))
(symptom neck lymph)
(symptom bone_pain)
(or (symptom fever) (symptom weight_loss))
=>
(printout t "There is a small indication of hodgkin's lymphoma to occur"
      crlf)
(printout t "I will list a few symptoms that are stricly linked with that
      type of cancer" crlf crlf)
(printout t "neck_lymphs" crlf "bone_pain" crlf "fever" crlf
      "weight_loss" crlf crlf)
(printout t "Give me the number of the symptoms above that you have
      experienced:")
(bind ?num (read))
(bind ?poss (* (/ (- ?num 1) 4) 100))
(assert (cancer_possibility hodgkin_lymphoma ?poss))
)


(defrule oral-cancer-possibility
"Choose the possibility for that cancer to occur with the provided
      symptoms"
(declare (salience 50))
(end_of_test true)
(not (cancer oral_cancer))
(symptom mouth_bleeding)
=>
(printout t "There is a small indication of oral cancer to occur" crlf)
(printout t "I will list a few symptoms that are stricly linked with that
      type of cancer" crlf crlf)
(printout t "infections_that_do_not_heal" crlf "bleeding" crlf
      "loose_teeth" crlf "changes_in_speech" crlf crlf)
(printout t "Give me the number of the symptoms above that you have
      experienced:")
(bind ?num (read))
(bind ?poss (* (/ (- ?num 1) 4) 100))
(assert (cancer_possibility oral_cancer ?poss))
)
```

```
(defrule pancreatic-cancer-possibility
"Choose the possibility for that cancer to occur with the provided
      symptoms"
(declare (salience 50))
(end_of_test true)
(not (cancer pancreatic_cancer))
(distortion noisy_feelings)
=>
(printout t "There is a small indication of pancreatic cancer to occur"
      crlf)
(printout t "I will list a few symptoms that are stricly linked with that
      type of cancer" crlf crlf)
(printout t "abdominal_pains" crlf "diarrhea" crlf "weight_loss" crlf
      "digestive_discomfort" crlf crlf)
(printout t "Give me the number of the symptoms above that you have
      experienced:")
(bind ?num (read))
(bind ?poss (* (/ (- ?num 1) 4) 100))
(assert (cancer_possibility pancreatic_cancer ?poss))
)


(defrule stomach-cancer-possibility
"Choose the possibility for that cancer to occur with the provided
      symptoms"
(declare (salience 50))
(end_of_test true)
(not (cancer stomach_cancer))
(distortion noisy_feelings)
=>
(printout t "There is a small indication of stomach cancer to occur"
      crlf)
(printout t "I will list a few symptoms that are stricly linked with that
      type of cancer" crlf crlf)
(printout t "abdominal_pains" crlf "diarrhea" crlf "weight_loss" crlf
      "digestive_discomfort" crlf  "weakness" crlf "bloating_food" crlf
      crlf)
(printout t "Give me the number of the symptoms above that you have
      experienced:")
(bind ?num (read))
(bind ?poss (* (/ (- ?num 1) 6) 100))
(assert (cancer_possibility stomach_cancer ?poss))
)


;;Kanones ektypwshs apotelesmatwn

(defrule cancer-results
"prints the results of the run"
(declare (salience 55))
(cancer ?x)
=>
(printout t "With the symptoms provided so far I have come to a
conclusion" crlf)
```

```
(printout t "Grade of danger: severe. Possible cancer type: " ?x crlf)
(printout t "Type a key to continue" crlf)
(read)
(assert (end_of_test true))
)


(defrule no-results
"not some particular cancer found"
(declare (salience 45))
(and (not (cancer ?x)) (user-name ?name))
=>
(printout t ?name " there is no big indication of a specific cancer
      type." crlf "Nevertheless we will provide below a list of possible
      cancer types that correspond to the symptoms provided." crlf)
(assert (end_of_test true))
)


(defrule possibility-of-cancer
"finds all the possibilities for the cancers"
(declare (salience 455))
(cancer_possibility ?cancer ?poss)
=>
(printout t "The possibility for " ?cancer " is " ?poss "%." crlf crlf
crlf))
```