Το Σύστημα FLEX

Στο παράρτημα αυτό περιγράφεται το κέλυφος έμπειρου συστήματος (expert system shell) FLEX της εταιρείας Logic Programming Associates. Το FLEX στηρίζεται πάνω στη γλώσσα LPA WinProlog της ίδιας εταιρείας, η οποία είναι ένας interpreter/ compiler της γλώσσας PROLOG για περιβάλλον MS-Windows, με κατάλληλες επεκτάσεις για να υποστηρίζει όλα τα χαρακτηριστικά του λειτουργικού αυτού συστήματος, όπως παράθυρα, μενού, μηχανισμούς ελέγχου, κλπ. Το FLEX αποτελείται ουσιαστικά από έναν αριθμό πρόσθετων κατηγορημάτων στην PROLOG, τη γλώσσα αναπαράστασης γνώσης KSL (*Knowledge Specification Language*) και ένα μεταδιερμηνέα που μεταφράζει τις προτάσεις KSL σε προτάσεις PROLOG.

Το FLEX είναι ένα ισχυρό εργαλείο ανάπτυξης εμπείρων συστημάτων. Υποστηρίζει συλλογιστική που βασίζεται σε προγραμματισμό με κανόνες, πλαίσια με χρήση κληρονομικότητας και διαδικασίες οδηγούμενες από τα δεδομένα (data-driven). Όλα τα παραπάνω είναι πλήρως ενσωματωμένα μέσα σε περιβάλλον λογικού προγραμματισμού. Το FLEX υπερισχύει έναντι πολλών κελυφών εμπείρων συστημάτων στο γεγονός ότι χρησιμοποιεί μια ανοιχτή αρχιτεκτονική, με την έννοια ότι επιτρέπει στον προγραμματιστή να προσπελάσει, να αυξήσει και να τροποποιήσει τη συμπεριφορά του εργαλείου μέσα από ένα πλαίσιο λειτουργιών. Για αυτό το λόγο άλλωστε συχνά αναφέρεται σαν εργαλείο TN. Ο συνδυασμός του FLEX με την PROLOG έχει ως αποτέλεσμα ένα λειτουργικά πλούσιο και πολλαπλά χρήσιμο περιβάλλον ανάπτυξης εμπείρων συστημάτων, στο οποίο οι προγραμματιστές μπορούν να ρυθμίσουν καθώς και να επεκτείνουν τους μηχανισμούς συμπεριφοράς που υπάρχουν προκειμένου να ικανοποιήσουν τις δικές τους συγκεκριμένες απαιτήσεις.

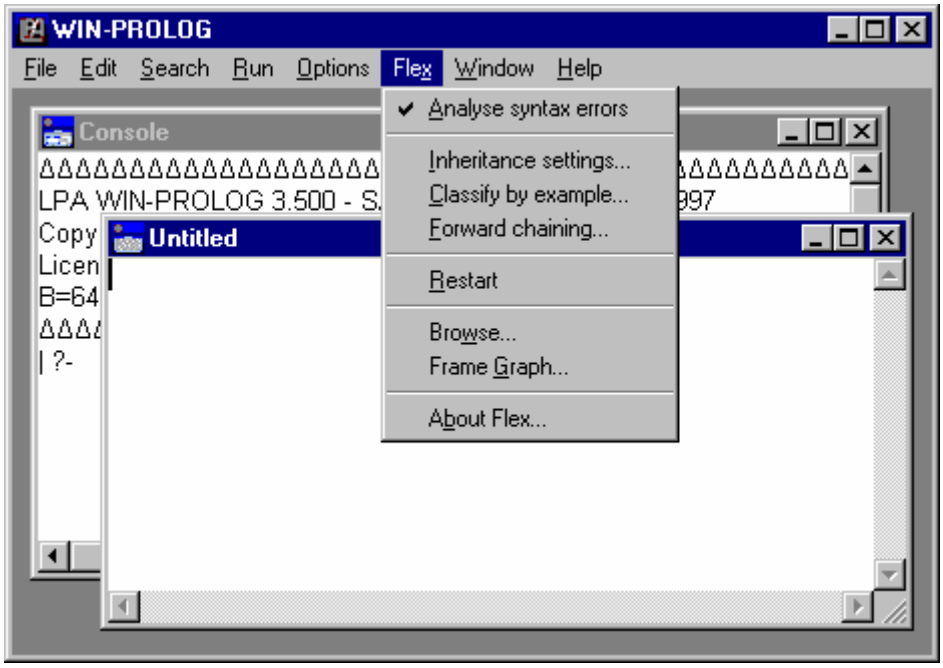
Περιεχόμενα

Π3.1 Το Περιβάλλον

Το περιβάλλον του FLEX βασίζεται στο περιβάλλον της LPA WinProlog, επαυξημένο με ένα επιπλέον μενού επιλογών, που υποστηρίζει τις νέες λειτουργίες. Το περιβάλλον του FLEX με το ομώνυμο πρόσθετο μενού ανοικτό φαίνεται στο Σχήμα Π3.1.

Για τη συγγραφή προγραμμάτων στη γλώσσα KSL μπορεί να χρησιμοποιηθεί ο συντάκτης κειμένου του περιβάλλοντος της LPA WinProlog ή οποιοσδήποτε απλός κειμενογράφος (ASCII text editor). Ένα πρόγραμμα μπορεί να διαχωριστεί σε περισσότερα από ένα παράθυρα και να αποθηκευτεί στη συνέχεια σε αντίστοιχο αριθμό αρχείων κειμένου. Η επέκταση των ονομάτων των αρχείων που περιέχουν κώδικα FLEX είναι **`.ksl`**, ενώ των αντίστοιχων που περιέχουν κώδικα PROLOG είναι **`.pl`**.

Ένα πρόγραμμα σε FLEX μπορεί, εκτός από τον κώδικα σε KSL, να περιλαμβάνει και κώδικα σε PROLOG. Ο μόνος περιορισμός είναι ότι δε θα πρέπει να αναμιγνύονται στο ίδιο αρχείο κώδικας σε KSL με κώδικα σε PROLOG.



Σχήμα Π3.1: Το περιβάλλον ανάπτυξης του FLEX.

Στο σημείο αυτό πρέπει να σημειωθεί ότι ένα παράθυρο που περιέχει κώδικα FLEX δεν μπορεί να μεταγλωττισθεί πριν αποθηκευτεί σε αρχείο, αφού αλλιώς δεν έχει ακόμη προσδιορισθεί ο τύπος του, με αποτέλεσμα η LPA να επιχειρεί να το μεταγλωττίσει σαν PROLOG, παράγοντας έτσι λάθη μεταγλώττισης.

Π3.2 Βασικά Χαρακτηριστικά

Στην ενότητα αυτή αναφέρονται εν συντομία τα βασικά χαρακτηριστικά του FLEX. Σε επόμενες ενότητες τα χαρακτηριστικά αυτά θα παρουσιασθούν εκτενέστερα.

Ορθή ακολουθία εκτέλεσης

Το FLEX υποστηρίζει την ορθή ακολουθία εκτέλεσης (forward chaining), όπου οι κανόνες παραγωγής ακολουθούν την κλασσική μορφή "εάν-τότε" (IF-THEN). Παρέχει επίσης τη δυνατότητα επεξήγησης και ένα μηχανισμό δυναμικής βαθμολόγησης των κανόνων (scoring mechanism) για την επίλυση συγκρούσεων κατά την πυροδότησή τους. Οι κανόνες μπορούν να έχουν πολλαπλά συμπεράσματα ή ενέργειες στο THEN μέρος τους. Επιπλέον υποστηρίζονται πολλοί αλγόριθμοι επίλυσης σύγκρουσεων (conflict resolution) κανόνων, ενώ παράλληλα υπάρχει και η δυνατότητα ορισμού νέων αλγορίθμων.

Ανάστροφη ακολουθία εκτέλεσης

Το FLEX υποστηρίζει και ανάστροφη ακολουθία εκτέλεσης, χρησιμοποιώντας κανόνες ανάλογους των κανόνων της PROLOG. Οι κανόνες αυτοί ονομάζονται *συσχετίσεις* (*relations*) και έχουν ένα μόνο συμπέρασμα το οποίο αληθεύει αν όλες οι συνθήκες είναι δυνατό να αποδειχθούν.

Πλαίσια και κληρονομικότητα

Το FLEX υποστηρίζει ιεραρχίες πλαισίων επιτρέποντας στις ιδιότητες τους να κληρονομούνται. Κάθε *πλαίσιο* (frame) ή *στιγμιότυπο* (instance) έχει ένα σύνολο από *ιδιότητες* (slots) που περιγράφουν τα χαρακτηριστικά των πλαισίων. Τα πλαίσια κληρονομούν τις τιμές των ιδιοτήτων άλλων πλαισίων σύμφωνα με τη θέση τους στην ιεραρχία. Με τον τρόπο αυτό αποφεύγεται η επανάληψη πληροφοριών και απλοποιείται σημαντικά ο κώδικας. Η κληρονομικότητα γίνεται αυτόματα και είναι δυνατό να ελεγχθεί με διάφορες οδηγίες (directives).

Ερωτήσεις και απαντήσεις

Το FLEX διαθέτει ένα ενσωματωμένο υποσύστημα ερωταπαντήσεων. Με τη χρήση αυτού τα προγράμματα που αναπτύσσονται με το FLEX μπορούν να ζητήσουν από το χρήστη να εισάγει δεδομένα μέσω παραθύρων διαλόγου.

Επεξηγήσεις

Το FLEX διαθέτει ένα ενσωματωμένο σύστημα επεξήγησης το οποίο υποστηρίζει τις επεξηγήσεις *πώς* (*how*) και *γιατί* (*why*). Οι επεξηγήσεις μπορούν να ενσωματωθούν στους κανόνες και στις ερωτήσεις.

Προγραμματισμός οδηγούμενος από τα δεδομένα

Το FLEX προσφέρει ειδικές διαδικασίες οι οποίες μπορούν να προσαρτηθούν σε συλλογές πλαισίων, σε ανεξάρτητα πλαίσια ή σε ιδιότητες πλαισίων. Αυτές οι διαδικασίες παραμένουν αδρανείς έως ότου ενεργοποιηθούν από την προσπέλαση ή την τροποποίηση συγκεκριμένων δομών στις οποίες έχουν προσαρτηθεί. Υπάρχουν τέσσερις

τύποι διαδικασιών διαθέσιμες στο FLEX: οι *διαδικασίες αρχικοποίησης* (*launches*), οι *δαίμονες* (*demons*), οι *φύλακες* (*watchdogs*), και οι *περιορισμοί* (*constraints*).

Π3.3 Η Γλώσσα Καθορισμού Γνώσης KSL

Το FLEX υποστηρίζει τη *Γλώσσα Καθορισμού Γνώσης* KSL (*Knowledge Specification Language*) για τον ορισμό κανόνων, πλαισίων, διαδικασιών κλπ. Αυτή η γλώσσα υποστηρίζει μαθηματικές, λογικές και υποθετικές εκφράσεις και είναι επεκτάσιμη με τη χρήση των συνωνύμων και των προτύπων. Τα βασικά στοιχεία της KSL, τα οποία θα περιγραφούν στις επόμενες παραγράφους, είναι τα ακόλουθα:

- *όροι* (*terms*)
- *αντικείμενα* (*objects*)
- *αριθμητικές εκφράσεις* (*numerical expressions*)
- *λογικές και διαδικαστικές εκφράσεις* (*formulae*)
- *δομές ελέγχου* (*control structures*)
- *προτάσεις* (*sentences*)

Π3.3.1 'Οροι

Οι *όροι* (*terms*) της KSL είναι τα *σχόλια* (*comments*), τα *σημεία στίξης* (*punctuation marks*), οι *αριθμοί* (*numbers*), τα *άτομα* (*atoms*), οι *συμβολοσειρές* (*strings*), οι *μεταβλητές* (*variables*), τα *ονόματα* (*names*) και οι *τιμές* (*values*).

- *Σχόλια.* Κάθε κείμενο που περικλείεται από τα σύμβολα **/*** και ***/** θεωρείται σχόλιο και αγνοείται από το μεταφραστή του FLEX. Επίσης το σύμβολο **%** δηλώνει την αρχή ενός σχολίου το οποίο τελειώνει στο τέλος της γραμμής στην οποία βρίσκεται.
- *Σημεία στίξης*. Τα σημεία στίξης θεωρούνται τα ακόλουθα σύμβολα:

  ( ) [ ] { } | ! ; , .
- *Αριθμοί.* Οι αριθμοί είναι είτε ακέραιοι είτε κινητής υποδιαστολής. Για παράδειγμα, αριθμοί είναι οι: **211327**, **-32768**, **0**, **2.34**, **10.3e99**, **-0.81** κλπ.
- *Άτομα.* Υπάρχουν τρεις τύποι ατόμων: τα *αλφαριθμητικά* (*alphanumeric*), τα *συμβολικά* (*symbolic*) και τα άτομα που περικλείονται σε μονά εισαγωγικά (quoted). Τα αλφαριθμητικά άτομα αρχίζουν με μικρό γράμμα (**a-z**) και είτε δεν ακολουθούνται από τίποτα άλλο είτε ακολουθούνται από μια σειρά αλφαβητικών χαρακτήρων (**a-z**, **A-Z** ή **_**) ή ψηφίων (**0-9**). Για παράδειγμα, τα άτομα **apple**, **aPPLE**, **h45j**, **apple_cart**. Τα συμβολικά άτομα είναι συνεχόμενη ακολουθία συμβόλων όπως τα *, **>** ή **#**. Συμβολικοί χαρακτήρες είναι όλοι οι χαρακτήρες εκτός από τα ψηφία (**0-9**), τα σημεία στίξης και τους αλφαβητικούς χαρακτήρες (**a-z**, **A-Z** ή **_**). Για παράδειγμα, συμβολικά άτομα είναι: **&**, **>=**, **#@&**, ***/**. Τέλος, άτομο θεωρείται και κάθε ακολουθία χαρακτήρων που περικλείεται σε μονά εισαγωγικά. Για παράδειγμα, τέτοια άτομα είναι τα **'Apple'**, **'The green ***'**, **'^h' 'ht'**.

- *Συμβολοσειρές*. Συμβολοσειρά θεωρείται κάθε ακολουθία χαρακτήρων που περικλείεται από διπλά εισαγωγικά. Για παράδειγμα, οι "**A boy**" και " ".
- *Μεταβλητές*. Μεταβλητές θεωρούνται οι ακολουθίες χαρακτήρων που αρχίζουν με κεφαλαίο γράμμα (**A-Z**) ή με το σύμβολο **_** και ακολουθούνται από μια ακολουθία κανενός ή περισσότερων αλφαβητικών χαρακτήρων (**a-z**, **A-Z** ή **_**) ή ψηφίων (**0-9**). Για παράδειγμα, μεταβλητές είναι οι **X**, **Apple**, **X_Apple_23**.
- *Ονόματα*. Όνομα θεωρείται κάθε άτομο μέσα σε απλά εισαγωγικά ή κάθε άτομο που δεν είναι δεσμευμένη λέξη. Για παράδειγμα, ονόματα είναι τα **brick**, **brick32**, '**The**', '**the brick**'.
- *Τιμές*. Τιμή θεωρείται κάθε αριθμός, κάθε συμβολοσειρά και κάθε όνομα. Για παράδειγμα, τιμές είναι οι **9821**, **-0.81**, '**the**', "**the empty string follows**", " ".

Π3.3.2 Αντικείμενα

Τα *αντικείμενα* (*objects*) της KSL είναι οι *μεταβλητές δομές* (*variants*), οι *σύνθετες μεταβλητές δομές* (*complex variants*), τα *σύνολα* (*sets*) και οι *γενικοί όροι* (*general terms*).

Οι *μεταβλητές δομές* έχουν τη δυνατότητα να αλλάζουν τιμή (όπως γίνεται και με τις μεταβλητές). Υπάρχουν δύο τύποι μεταβλητών δομών: οι *καθολικές μεταβλητές* (*global variables*) και τα *ιδιότητες* (*slots*). Μια καθολική μεταβλητή μπορεί να είναι οποιοδήποτε όνομα από το οποίο προαιρετικά προηγείται η προσδιοριστική λέξη **the** και μπορεί να θεωρηθεί σαν ιδιότητα ενός ειδικού πλαισίου που λέγεται **global**. Μία ιδιότητα προσδιορίζεται από το όνομά της και το όνομα του *πλαισίου* (ή του στιγμιότυπου) στο οποίο ανήκει, χρησιμοποιώντας τη λέξη-κλειδί της KSL **of** ή τον τελεστή **`s**. Στην περίπτωση αυτή, η προσδιοριστική λέξη **the** μπορεί προαιρετικά να προηγείται της ιδιότητας. Για παράδειγμα, η παρακάτω έκφραση προσδιορίζει την ιδιότητα **size** του πλαισίου (ή στιγμιοτύπου) **box**.:

```
the box `s size
```

Στις *σύνθετες μεταβλητές δομές* η τιμή που έχει ανατεθεί σε μια ιδιότητα ενός πλαισίου μπορεί να είναι το όνομα μιας άλλης ιδιότητας ή πλαισίου. Για παράδειγμα, αν έχουμε κάποιο πλαίσιο **employee** με την ιδιότητα **employee_address** και η τιμή αυτής είναι **address**, τότε αυτή η τιμή μπορεί να είναι και το όνομα κάποιου άλλου πλαισίου.

Υπάρχουν δύο τύποι συνόλων στην KSL: τα *σαφή σύνολα* (*explicit sets*) και τα *επαγόμενα σύνολα* (*implicit sets*). Ένα σαφές σύνολο ορίζεται παραθέτοντας μια περιγραφή καθενός από τα στοιχεία του. Τα στοιχεία του συνόλου μπορεί να χωρίζονται με κόμμα ή με τις λέξεις-κλειδιά της KSL **or** ή **and**. Για παράδειγμα:

```
{ fred, angela, john and mary }
{ the staff or fred and mary }
```

Ένα επαγόμενο σύνολο δηλώνει το πώς θα βρεθεί ή θα υπολογιστεί κάθε ξεχωριστό στοιχείο, χωρίς απαραίτητα να γίνεται σαφής αναφορά στο καθένα. Εναλλακτι-

κά, ένα επαγόμενο σύνολο μπορεί να αποτελείται από όλα τα μέλη μιας συγκεκριμένης ομάδας (group) με χρήση των λέξεων-κλειδιών της KSL **all**, **every** ή **each**. Τέλος, ένα επαγόμενο σύνολο μπορεί να είναι μια συλλογή από στιγμιότυπα πλαισίου που έχουν συγκεκριμένες ιδιότητες. Η λέξη κλειδί **whose** μπορεί να χρησιμοποιηθεί για να γίνει αναφορά στις ιδιότητες του πλαισίου. Για παράδειγμα:

```
{ X : bird(X) or reptile(X) and can_fly(X) }
all colours
every dessert
each flower
every reptile whose skin is smooth
```

Τέλος, οι *γενικοί όροι* περιλαμβάνουν ονόματα, τιμές, μεταβλητές, μεταβλητές δομές, σχήματα μεταβλητής δομής και σύνολα.

Π3.3.3 Αριθμητικές Εκφράσεις

Οι αριθμητικές εκφράσεις σχηματίζονται με τους δυαδικούς τελεστές *****, **+**, **-**, **/** και **^**. Επίσης μπορούν να χρησιμοποιηθούν οι λέξεις **plus**, **minus**, **times**, **divided by** και **to the power of**. Τη μεγαλύτερη προτεραιότητα έχει ο τελεστής **^**, στη συνέχεια ακολουθούν οι τελεστές ***** και **/**, ενώ τελευταίοι στη σειρά προτεραιότητας είναι οι τελεστές **+** και **-**. Η προτεραιότητα των τελεστών μπορεί να αλλάξει με χρήση παρενθέσεων. Για παράδειγμα:

```
1 + 2 * 3 - 4 / 5 ^ 6
(1 - 2) * (3+4)
the box`s size to the power of 3
```

Π3.3.4 Λογικές και Διαδικαστικές Εκφράσεις

Οι λογικές και οι διαδικαστικές εκφράσεις χρησιμοποιούνται για να δημιουργήσουν σχέσεις ανάμεσα στα αντικείμενα της KSL. Χωρίζονται σε δύο κατηγορίες:

- στις *συνθήκες* (*conditions*), οι οποίες ελέγχουν αν κάτι είναι αληθές ή όχι.
- στις *εντολές* (*directives*), οι οποίες αλλάζουν την κατάσταση ενός αντικειμένου σε μια νέα κατάσταση.

Συνθήκες

Οι συνθήκες χρησιμοποιούνται για να ελέγξουν την τρέχουσα κατάσταση των καθολικών μεταβλητών, των πλαισίων ή των στιγμιότυπών τους. Επίσης μπορούν να ελέγξουν για την ύπαρξη μιας μεταβλητής δομής ή να συγκρίνουν την τιμή δύο εκφράσεων. Τέλος, μπορούν να αποτελούνται από την κλήση μιας διαδικασίας.

Έλεγχος ισότητας

Για τον έλεγχο της ισότητας χρησιμοποιείται ο αριθμητικός τελεστής "**=**" ή οι:

is, **are**, **is equal to**

Οι παραπάνω τελεστές μπορούν επίσης να χρησιμοποιηθούν με τον τελεστή **not**. Για παράδειγμα:

```
alpha=beta/2
the size of some brick is equal to 4
not X is an elephant
```

Για τον έλεγχο της ανισότητας χρησιμοποιείται ο τελεστής **\=** ή οι:

not equal to, **different from**

Έλεγχος ύπαρξης

Ένας έλεγχος για την ύπαρξη μιας μεταβλητής μπορεί να γίνει ελέγχοντας εάν αυτή έχει τιμή ή όχι. Η περίπτωση μη ύπαρξης τιμής ανιχνεύεται με τις δεσμευμένες λέξεις **known** και **unknown**. Για παράδειγμα:

```
the starter of the meal is unknown
the temperature is known
```

Απευθείας σύγκριση

Για τη σύγκριση δύο όρων χρησιμοποιούνται οι αριθμητικοί τελεστές **>**, **<**, **=<**, **>=** ή οι τελεστές:

```
greater than, greater than or equal to, above, at or above, less than, less than
or equal to, below, at or below
```

Όλοι οι τελεστές μπορούν να δεχθούν το πρόθεμα **not**. Για παράδειγμα:

```
alpha>beta/2
the pupil`s mark is not below 50
the food`s calories is less than or equal to 400
```

Σχετική σύγκριση

Η σχετική σύγκριση δύο όρων καθορίζεται από τις θέσεις τους μέσα σε μια *ομάδα* (*group*). Αυτή η ομάδα μπορεί να καθοριστεί χρησιμοποιώντας τον τελεστή **according to**. Για παράδειγμα:

```
its color is at or above the colour of money according to {red, blue, white, green}
group fuzzy_ordering
     certain, probable, possible, unlikely, impossible.
the likelihood of frost is less than probable
     according to fuzzy_ordering
```

Μέλος συνόλου

Για να ελέγξουμε τα μέλη συνόλου χρησιμοποιούμε τους τελεστές **include**, **includes**, **included in**, **does not include**, **do not include**. Για παράδειγμα:

```
the staff includes {john and mary}
the Rodent`s tail does not include bushy
```

Κλήσεις διαδικασιών

Μια συνθήκη μπορεί να είναι μια κλήση σε μια διαδικασία, δηλαδή μια συσχέτιση του FLEX, μια ενέργεια του FLEX ή μια κλήση της PROLOG. Με αυτό τον τρόπο επιτυγχάνεται η σύνδεση με το μηχανισμό της ανάστροφης ακολουθίας εκτέλεσης της PROLOG.

Συζεύξεις και διαζεύξεις

Οι συνθήκες μπορούν να συνδεθούν χρησιμοποιώντας τους τελεστές **and** και **or**. Ο τελεστής **not** έχει μεγαλύτερη προτεραιότητα από τον τελεστή **and**, ο οποίος με τη σειρά του έχει μεγαλύτερη προτεραιότητα από τον τελεστή **or**. Η σειρά αυτή μπορεί να τροποποιηθεί με τη χρήση των αγκυλών [ ]. Για παράδειγμα:

```
c is some cat and M is C`s meal
test1 and [ test2(X) or alpha > 10 ]
not [ test1 and test2 ]
```

Χρήση συνθηκών σε θέση εντολών

Για την εισαγωγή μιας συνθήκης σε θέση όπου απαιτείται η εισαγωγή εντολής (π.χ. στο δεξιό μέρος ενός κανόνα), μπορούν να χρησιμοποιηθούν οι δεσμευμένες λέξεις **check that** ακολουθούμενες από τη συνθήκη. Στο παρακάτω παράδειγμα, η *συσχέτιση* (*relation*) πρέπει να ακολουθείται από συνθήκες, ενώ η *ενέργεια* (*action*) πρέπει να ακολουθείται από εντολές. Έτσι, για να εισαχθεί μια συνθήκη στην ενέργεια χρησιμοποιήθηκαν οι δεσμευμένες λέξεις **check that**.

```
relation emp_name (Emp, Name)
    if Name is Emp`s name.
action emp_name (Emp, Name) ;
    do check that Name is Emp`s name.
```

Εντολές

Οι εντολές χρησιμοποιούνται για να αλλάξουν την παρούσα κατάσταση σε μια νέα. Η τρέχουσα κατάσταση προσδιορίζεται από τις καθολικές μεταβλητές, τα πλαίσια και τα στιγμιότυπα.

Αναθέσεις τιμών

Μία ανάθεση αποτελείται από μία μεταβλητή δομή στην αριστερή πλευρά και από μία έκφραση στη δεξιά. Η ανάθεση αντικαθιστά την υπάρχουσα τιμή της μεταβλητής δομής με την τιμή της έκφρασης. Η γενική μορφή μιας ανάθεσης είναι η ακόλουθη:

```
Variant := Expression
Variant become Expression
Variant becomes Expression
```

Στους παραπάνω ορισμούς οι λέξεις που είναι με πλάγια γράμματα υποδηλώνουν μεταβλητές ή άλλα αντικείμενα ενός προγράμματος, ενώ οι λέξεις που είναι με κανονικά γράμματα υποδηλώνουν δεσμευμένες λέξεις της γλώσσας KSL. Η παραπάνω σύμβαση υιοθετείται στη συνέχεια του παραρτήματος, οποτεδήποτε δίνεται η σύνταξη μιας δομής της γλώσσας KSL. Για παράδειγμα:

```
methane_level := high
the Kettle`s temperature becomes 45
```

Αύξηση και μείωση

Οι τιμές των αριθμητικών μεταβλητών (ιδιοτήτων) μπορούν αυξηθούν ή να μειωθούν χρησιμοποιώντας τις ακόλουθες εντολές:

```
add Expression to Variant
subtract Expression from Variant
```

Για παράδειγμα:

```
add 100 to the car`s mileage
subtract 1 from the total
```

Μέλη συνόλων

Για την προσθήκη ή την αφαίρεση στοιχείων από σύνολα χρησιμοποιούνται οι παρακάτω εντολές:

```
include Item(s) in Set
remove Item(s) from Set
```

Αν δεν υπάρχει τρέχουσα τιμή στο σύνολο, τότε πραγματοποιείται ανάθεση τιμής. Αν επιχειρηθεί να απομακρυνθεί ένα αντικείμενο, όταν δεν υπάρχει τρέχουσα τιμή για το σύνολο, η προσπάθεια αποτυχαίνει. Για παράδειγμα:

```
include lemon_sole in the entree of set_meal
remove {david and liz} from the staff
```

Δημιουργία στιγμιοτύπων

Οι εντολές μπορούν να δημιουργήσουν νέα στιγμιότυπα πλαισίων με τοπικές τιμές στις ιδιότητες. Όλες οι ιδιότητες από τα πλαίσια-γονείς κληρονομούνται αυτόματα σε κάθε στιγμιότυπο. Ακολουθεί η γενική μορφή:

```
Instance is a new Frame
Instance is another Frame
```

Η λέξη **whose** χρησιμοποιείται για να δώσει τιμές σε τοπικές ιδιότητες. Για παράδειγμα:

```
house_2 is a new house whose bedrooms is 4
```

Διαχείριση βάσης δεδομένων

Η διαχείριση μιας βάσης δεδομένων επιτυγχάνεται με εντολές οι οποίες προσθέτουν ή αφαιρούν γεγονότα, χρησιμοποιώντας τις ακόλουθες εντολές:

```
remember, remember that , forget, forget that
```

Για παράδειγμα:

```
remember that pregnant(P)
forget that not boiling
```

Ερωτήσεις

Οι ερωτήσεις τίθενται χρησιμοποιώντας τη λέξη **ask**.

```
ask Question
```

Για παράδειγμα:

```
ask password
```

Π3.3.5 Δομές Ελέγχου

Οι δομές ελέγχου στο FLEX μπορούν να χωριστούν σε δύο κατηγορίες: στις δομές λήψης απόφασης και στους βρόχους. Στις πρώτες περιλαμβάνεται η διακλάδωση υπό

συνθήκη (δομή **if-then-else**), ενώ στις δεύτερες περιλαμβάνονται οι βρόχοι **repeat-until, while-do και for.**

Διακλάδωση υπό συνθήκη

Για αλλαγή της ροής του προγράμματος υπό συνθήκη χρησιμοποιείται η κλασσική δομή ελέγχου **if-then-else**, η οποία λειτουργεί όπως και στον κλασσικό προγραμματισμό: γίνεται έλεγχος στη συνθήκη (που βρίσκεται στο **if** μέρος) και αν αυτή ικανοποιείται τότε εκτελούνται οι εντολές που βρίσκονται στο **then** μέρος, ειδάλλως εκτελούνται οι εντολές που βρίσκονται στο **else** μέρος. Η σύνταξη μιας **if-then-else** δομής είναι η εξής:

```
if condition(s)
    then directive(s)
    else directive(s)
end if
```

Για παράδειγμα:

```
action print_result ;
do if the result of test is greater than 50
    then write( 'You have passed' )
    else write( 'You have failed' )
end if .
```

Βρόχος Repeat-Until

Ένας **repeat-until** βρόχος επαναλαμβάνει μια δεδομένη εντολή μέχρι να ικανοποιηθεί κάποια συνθήκη. Η σύνταξη ενός **repeat-until** βρόχου είναι η εξής:

```
repeat directive(s)
    until condition(s)
end repeat
```

Για παράδειγμα:

```
action balance ;
    do repeat add 1 to the up of count
        and subtract 1 from the down of count
        until the up of count >= down of count
    end repeat
    and write( 'Balance' )
    and write( up of count ) and tab( 1 )
    and write( down of count ) and nl .
```

Βρόχος While-Do

Οι βρόχοι **while** επαναλαμβάνουν μια εντολή για όσο διάστημα ικανοποιείται μια συνθήκη. Η εντολή εκτελείται αφού πρώτα γίνει ο έλεγχος. Η σύνταξη ενός **while-do** βρόχου είναι η πιο κάτω:

```
while condition(s)
```

```
        do directive(s)
    end while
```

Για παράδειγμα:

```
    action remove_paid_customer ;
        do while Customer is some customer
            and the loan of Customer is paid
            do remove_instance( Customer )
        end while .
```

Βρόχος For

Οι απλοί βρόχοι **for** επαναλαμβάνουν εντολές όσο ικανοποιούνται κάποιες συνθήκες. Πρακτικά ένας απλός βρόχος **for** είναι λειτουργικά ισοδύναμος με έναν βρόχο **while-do**. Η σύνταξη ενός βρόχου **for** είναι η εξής:

```
    for condition(s)
        do directive(s)
    end for
```

Για παράδειγμα:

```
action print_employees ;
    do for every Name is some instance of employee
        do write( Name ) and nl
    end for .
```

Εκτεταμένος βρόχος For

Οι εκτεταμένοι βρόχοι **for** επαναλαμβάνουν εντολές για όσο διάστημα η τιμή ενός δείκτη κυμαίνεται από ένα όριο ως κάποιο άλλο με κάποιο συγκεκριμένο βήμα μεταβολής. Η σύνταξη ενός **for-from-to-step** βρόχου είναι η πιο κάτω:

```
    for counter from expression1 to expression2 step expression3
        do directive(s)
    end for
```

Εάν παραλειφθεί η δήλωση **step expression3**, θεωρείται ότι το βήμα είναι 1. Για παράδειγμα:

```
    action multiply ;
    do for X from -20 to 20 step 5
            do for Y from -20 to 20
                do write( X * Y ) and nl
            end for
        end for .
```

Π3.3.6 Προτάσεις

Στην προηγούμενες ενότητες παρουσιάστηκε η σύνταξη για τα αντικείμενα της KSL και τις μεταξύ τους σχέσεις. Σε αυτή την ενότητα θα παρουσιαστεί η σύνταξη των KSL προτάσεων μέσα στις οποίες εμφανίζονται τα παραπάνω αντικείμενα. Οι προτάσεις καθορίζουν τι μπορεί και τι δεν μπορεί να δηλωθεί σε ένα πρόγραμμα γραμμένο

σε KSL. Μια *πρόταση* (sentence) στην KSL αρχίζει με μια από τις ακόλουθες λέξεις-κλειδιά:

```
action, constraint, data, demon, do, frame, function, group, instance, launch,
        question, relation, rule, ruleset, synonym, template, watchdog.
```

και κάθε πρόταση τερματίζει με ένα διάστημα ακολουθούμενο από μια τελεία.

Πλαίσια και στιγμιότυπα

Υπάρχουν τρία μέρη στον καθορισμό ενός πλαισίου. Το πρώτο καθορίζει σε ποιο σημείο της ιεραρχίας βρίσκεται το πλαίσιο. Αυτό επιτυγχάνεται με τον καθορισμό των πλαισίων-γονέων (αν υπάρχουν). Το δεύτερο μέρος είναι προαιρετικό και καθορίζει ποιες είναι οι ιδιότητες του πλαισίου και οι προκαθορισμένες τιμές τους. Στο τρίτο μέρος, που είναι επίσης προαιρετικό, επαναπροσδιορίζεται η ιεραρχία της κληρονομικότητας. Ειδικότερα, ορισμένα πλαίσια μπορούν να γίνουν πλαίσια-γονείς για κάποιο άλλο πλαίσιο προκειμένου να κληρονομήσει ορισμένες ιδιότητές τους, ή συγκεκριμένες ιδιότητες μπορούν να δηλωθούν έτσι ώστε να μην κληρονομηθούν. Η γενική σύνταξη ενός πλαισίου είναι η ακόλουθη:

```
frame Name is a kind of Frame1, Frame2, … ;
    default Attribute1 is Value ;
    inherit Attribute2 from Frame ;
    do not inherit Attribute3 from Frame .
```

Για να καθοριστούν τα πλαίσια-γονείς χρησιμοποιούνται οι λέξεις-κλειδιά **is a** και **is a kind of**. Για να καθοριστεί η κληρονομικότητα κάποιων συγκεκριμένων ιδιοτήτων από πλαίσια που δεν είναι γονείς χρησιμοποιούνται οι λέξεις-κλειδιά **inherit ... from** . Τέλος, για να μην κληρονομηθούν ιδιότητες από πλαίσια που είναι γονείς χρησιμοποιούνται οι δεσμευμένες λέξεις **do not inherit**. Για παράδειγμα:

```
frame wedge is a kind of block , toy ;
    do not inherit shape and
    inherit volume from block and
    inherit density from pyramid, toy .
```

Τα στιγμιότυπα διαφέρουν από τα πλαίσια στο ότι μπορούν να έχουν μόνο ένα πλαίσιο-γονέα, δεν μπορούν να έχουν παιδιά, κληρονομούν όλες τις ιδιότητες του πλαισίου-γονέας, χωρίς να υπάρχει δυνατότητα ορισμού νέων ιδιοτήτων. Όλες οι τιμές των ιδιοτήτων τους θεωρούνται τρέχουσες (current values). Η δήλωση ενός στιγμιότυπου γίνεται με την παρακάτω σύνταξη:

```
instance Instance is a Frame ;
    Attribute is Value .
```

Για παράδειγμα:

```
instance tweety_pie is a kind of bird ;
    habitat is a cage and
    predator is sylvester;
    do not inherit motions .
```

Κανόνες - σύνολα κανόνων

Οι *κανόνες* στην KSL είναι κανόνες ορθής ακολουθίας εκτέλεσης. Το μέρος **if** ενός κανόνα περιλαμβάνει τις συνθήκες που πρέπει να ικανοποιούνται προκειμένου να πυροδοτηθεί ο κανόνας, ενώ το μέρος **then** περιλαμβάνει τις εντολές που θα εκτελεστούν όταν πυροδοτηθεί ο κανόνας. Επιπλέον είναι δυνατόν να υπάρχει επεξήγηση η οποία θα συσχετίζεται με τον κανόνα. Τέλος, στον κανόνα μπορεί να υπάρχει ένας *βαθμός σημαντικότητας* (*score*), ο οποίος χρησιμοποιείται για την επίλυση συγκρούσεων που συμβαίνουν όταν περισσότεροι από ένας κανόνες είναι σε θέση να πυροδοτηθούν στον ίδιο κύκλο εκτέλεσης, δίνοντας προτεραιότητα στον κανόνα με το μεγαλύτερο βαθμό. Οι βαθμοί μπορεί να είναι σταθερές αριθμητικές τιμές ή αριθμητικές εκφράσεις που υπολογίζονται κατά την προσπάθεια επίλυσης της σύγκρουσης. Η σύνταξη ενός κανόνα είναι η ακόλουθη:

```
rule rule_name
    if condition(s)
    then directive(s) ;
    because explanation ;
    score score .
```

όπου τα μέρη **because** και **score** είναι προαιρετικά. Για παράδειγμα:

```
rule feed_cat
    if      C is some cat
            and C`s condition is hungry
    then    feed ( C, C`s food )
            and C`s condition becomes fed ;
    because We need to feed hungry cats ;
    score 5 .
```

Το *σύνολο κανόνων* (*ruleset*) είναι η δομή που κατευθύνει το μηχανισμό της ορθής ακολουθίας εκτέλεσης Καθορίζει ποιοι κανόνες θα ληφθούν υπόψη, πώς θα επιλύονται οι συγκρούσεις, πότε η ορθή ακολουθία εκτέλεσης θα τερματίσει, κλπ. Ένα πρόγραμμα KSL μπορεί να περιλαμβάνει πολλά σύνολα κανόνων, κάθε φορά όμως ενεργοποιείται μόνο ένα από αυτά. Ένας κανόνας ωστόσο μπορεί να ανήκει σε περισσότερα του ενός σύνολα κανόνων. Η σύνταξη δήλωσης ενός συνόλου κανόνων είναι η εξής:

```
ruleset Name
    contains rule(s);
    initiate by doing directive(s);
    terminate when condition(s);
    select rule using rule_selection;
    update ruleset agenda_update;
    when a rule misfires do directive(s).
```

Η δήλωση **contains** περιλαμβάνει τα ονόματα των κανόνων που απαρτίζουν το σύνολο κανόνων. Η δήλωση **initiate by doing** περιλαμβάνει κάποιες αρχικές εντο-

λές που εκτελούνται πριν την έναρξη της ορθής ακολουθίας εκτέλεσης. Η δήλωση **terminate when** περιλαμβάνει κάποιες συνθήκες που όταν ικανοποιηθούν τερματίζει η ορθή ακολουθία εκτέλεσης. Να σημειωθεί ωστόσο ότι η εκτέλεση τερματίζει επίσης εφόσον δεν υπάρχει κανένας κανόνας για πυροδότηση.

Η δήλωση **select rule using** καθορίζει τον τρόπο με τον οποίο θα επιλύονται οι συγκρούσεις κανόνων. Οι δυνατές επιλογές είναι οι εξής:

- *first come first served*: επιλέγει τον πρώτο κανόνα από αυτούς που μπορούν να ενεργοποιηθούν, με βάση τη σειρά δήλωσής τους στο σύνολο κανόνων.
- *conflict resolution*: επιλέγει τον κανόνα με το μεγαλύτερο βαθμό.
- *conflict resolution with threshold N*: επιλέγει τον κανόνα με το μεγαλύτερο βαθμό, ανάμεσα σε αυτούς που έχουν βαθμό μεγαλύτερο από κάποιον αριθμό N.

Η δήλωση **agenda_update** καθορίζει πώς θα αναδιατάσσονται οι κανόνες στην ατζέντα, ύστερα από κάθε πυροδότηση. Η ατζέντα (agenda) είναι μια ειδική περιοχή μνήμης, όπου φυλάσσεται ένα αντίγραφο του συνόλου των κανόνων. Κάθε φορά που εκτελείται ένας κανόνας, είναι δυνατόν οι κανόνες στην ατζέντα να αναδιαταχθούν, γεγονός που θα επηρεάσει την επόμενη επιλογή κανόνα, εφόσον συμβεί σύγκρουση. Οι δυνατές τιμές που μπορεί να πάρει η συγκεκριμένη δήλωση είναι οι εξής:

- *removing each selected rule*: διαγράφει από την ατζέντα κάθε κανόνα που εκτελέσθηκε.
- *promoting each selected rule*: μεταφέρει στην αρχή της ατζέντας τον κανόνα που εκτελέσθηκε.
- *demoting each selected rule*: μεταφέρει στο τέλος της ατζέντας τον κανόνα που εκτελέσθηκε.
- *cyclic rotation of rules*: θεωρώντας την ατζέντα σαν μία κυκλική διάταξη κανόνων, γίνεται πρώτος ο κανόνας που ακολουθεί στη σειρά της ατζέντας αυτόν που εκτελέσθηκε.
- *removing any unsatisfied rules*: σε κάθε κύκλο διαγράφει από την ατζέντα όλους τους κανόνες που δεν ενεργοποιούνται και διατηρεί τη σειρά των υπολοίπων.

Κατά τη διαγραφή των κανόνων από την ατζέντα, εάν η ατζέντα παραμείνει κενή τερματίζει η ορθή ακολουθία εκτέλεσης των κανόνων.

Θα πρέπει να τονισθεί στο σημείο αυτό ο διαχωρισμός ανάμεσα στο σύνολο κανόνων, στην ατζέντα και στο σύνολο σύγκρουσης. Το σύνολο κανόνων είναι αυτό που δηλώνεται με τη σχετική πρόταση σε KSL. Οι κανόνες σε αυτό μπορούν να περιέχουν μεταβλητές μη δεσμευμένες. Η ατζέντα αρχικά περιλαμβάνει όλους τους κανόνες του συνόλου κανόνων, χωρίς δέσμευση των μεταβλητών. Στη συνέχεια όμως ενδέχεται κάποιοι κανόνες να διαγραφούν από την ατζέντα, ανάλογα και με την στρατηγική ανανέωσής της. Τέλος, το σύνολο σύγκρουσης υπολογίζεται πριν από κάθε εκτέλεση κανόνα και περιέχει τους κανόνες εκείνους της ατζέντας των οποίων οι προϋποθέσεις ικανοποιούνται. Οι κανόνες στο σύνολο σύγκρουσης έχουν δεσμευμένες όλες τις μεταβλητές τους. Ακολουθεί ένα παράδειγμα ορισμού ενός συνόλου κανόνων:

```
ruleset example_ruleset
```

```
contains rule1, rule2 ;
initiate by doing write( 'Starting' ) and action1 and nl ;
terminate when condition1 and condition2 ;
select rule using first come first served ;
update ruleset by removing each selected rule ;
when a rule misfires do fail .
```

Η έναρξη της επεξεργασίας ενός συνόλου κανόνων γίνεται με την εντολή:

```
invoke ruleset ruleset_name
```

Συσχετίσεις

Οι *συσχετίσεις* (*relations*) είναι κανόνες ανάστροφης ακολουθίας εκτέλεσης, αντίστοιχοι των κανόνων της PROLOG. Μια συσχέτιση καθορίζεται από μια συλλογή προτάσεων. Η βασική σύνταξη δήλωσης μιας συσχέτισης είναι η εξής:

```
relation relation_name
    if condition(s) .
```

Για παράδειγμα:

```
relation is_leaf( Node )
    if the links_out of Node is empty .
```

Ενέργειες - συναρτήσεις

Μια *ενέργεια* είναι μια συλλογή από εντολές, αντίστοιχη με τις διαδικασίες (procedures) του διαδικαστικού προγραμματισμού. Μια ενέργεια μπορεί να έχει οποιονδήποτε αριθμό ορισμάτων. Η σύνταξη δήλωσης μιας ενέργειας είναι η ακόλουθη:

```
action action_name ;
    do directive(s) .
```

Για παράδειγμα:

```
action empty_into( X, Y ) ;
    do Y`s contents := Y`s contents + X`s contents
    and X`s contents := 0 .
```

Μια *συνάρτηση* είναι ένας όρος που αποτιμάται κατά την εκτέλεση του προγράμματος, κατά τρόπο αντίστοιχο με τις συναρτήσεις του διαδικαστικού προγραμματισμού. Οι γενικές μορφές δήλωσης μιας συνάρτησης είναι οι ακόλουθες:

```
function function_name = expression .
function function_name = Variable
    where condition(s) .
    function function_name =
    if condition(s)
    then expression1
    else expression2 .
```

Όταν κατά τη διάρκεια της εκτέλεσης συναντηθεί το όνομα μιας συνάρτησης μέσα σε κάποια εντολή, αυτό αντικαθίσταται με το δεξιό μέρος του ορισμού της. Σημειώνεται ότι το δεξιό μέρος της συνάρτησης μπορεί να περιέχει υποθετικές δηλώσεις (εντολές **if**), όπως φαίνεται παραπάνω στην τελευταία σύνταξη. Ακολουθούν μερικά παραδείγματα δήλωσης συναρτήσεων.

```
function father( X ) = Y
    where parent( Y, X )
    and male( Y ) .
function maximum( A, B ) =
    if A > B then A
    else B .
```

Περιορισμοί και διαδικασίες ελέγχου

Μία *διαδικασία αρχικοποίησης* (*launch*) μπορεί να προσαρτηθεί σε πλαίσια και καλείται αυτόματα όταν δημιουργηθεί ένα καινούργιο στιγμιότυπο του πλαισίου. Η κύρια χρήση του είναι η ανάθεση αρχικών τιμών στις ιδιότητες του στιγμιοτύπου του πλαισίου. Οι εντολές της διαδικασίας αρχικοποίησης εκτελούνται αμέσως μετά τη δημιουργία του στιγμιοτύπου. Τέλος, μια τέτοια διαδικασία μπορεί να περιλαμβάνει συνθήκες που πρέπει να πληρούνται για να εκτελεστεί. Η σύνταξη δήλωσης διαδικασιών αρχικοποίησης είναι η ακόλουθη:

```
launch launch_name
    when Instance is a new Frame
    and condition (s)
    then directive (s)
```

Για παράδειγμα:

```
launch pick_up_new_carrier_bag
    when bag is a new instance of carrier
    then write('I need another carrier bag')
    and write (Bag)
    and nl.
```

Ο *περιορισμός* (*constraint*) είναι ένας έλεγχος ο οποίος προσαρτάται σε μία ιδιότητα ενός πλαισίου και καλείται αυτόματα οποτεδήποτε η τιμή της ιδιότητας αλλάζει. Ο έλεγχος γίνεται πριν την αλλαγή μιας τιμής και η τιμή αλλάζει μόνο αν ο έλεγχος πετύχει. Αν ο έλεγχος αποτύχει, η αλλαγή δε γίνεται. Η σύνταξη δήλωσης ενός περιορισμού είναι η εξής:

```
constraint constraint_name
    when attribute changes
    from Expression1
    to  Expression2
    and condition1(s)
    then check that condition2(s)
    otherwise directive(s)
```

Για παράδειγμα:

```
constraint minimum_widget
    when the widget of Container changes to N
    and Container is some container
    then check that N>0
    otherwise Container`s widgets becomes 0.
```

Ένας *δαίμονας* (*demon*) είναι μία διαδικασία που προσαρτάται σε μία ιδιότητα ενός πλαισίου και καλείται αυτόματα κάθε φορά που η τιμή της ιδιότητας αλλάζει. Οι

εντολές ενός δαίμονα εκτελούνται αμέσως μετά την αλλαγή της τιμής της ιδιότητας. Ένας δαίμονας προσαρμόζεται έτσι ώστε να πυροδοτείται μόνο για δεδομένες τιμές και μόνο κάτω από συγκεκριμένες περιστάσεις. Η σύνταξη της δήλωσης ενός δαίμονα είναι η εξής:

```
demon demon_name
    when Attribute changes
    from Expression1
    to Expression2
    and condition(s)
    then directive(s)
```

όπου τα μέρη **from**, **to** και οι συνθήκες **conditions** είναι προαιρετικά. Για παράδειγμα:

```
demon check_for_melt_down_of_core
    when the temperature changes to T
    and T is above boiling_point
    then remember that danger_level(red)
    and shut down
```

Ένας *φύλακας* (*watchdog*) ελέγχει τα δικαιώματα πρόσβασης σε μία ιδιότητα ενός πλαισίου και καλείται αυτόματα οποτεδήποτε υπάρχει μια αίτηση για την τρέχουσα τιμή της ιδιότητας. Οι έλεγχοι αυτοί εκτελούνται λίγο πριν να προσπελαστεί η τιμή. Αν ο έλεγχος αποτύχει, αποτυγχάνει και η προσπέλαση. Η σύνταξη δήλωσης ενός φύλακα είναι η εξής:

```
watchdog watchdog_name
    when Attribute is requested
    and condition1(s)
    then check that condition2(s)
    otherwise directive(s)
```

Για παράδειγμα:

```
watchdog account_sequrity
    when the contents of account is requested
    and outside_office_hours
    then check that the user`s classification is above 99
    otherwise report_illegal_entry
```

Ερωτήσεις

Μια *ερώτηση* περιγράφει τον τρόπο με τον οποίο κάποιες πληροφορίες θα εισαχθούν από το χρήστη. Η απόκριση λαμβάνεται μέσω ενός καθορισμένου μηχανισμού απαντήσεων που διαθέτει το FLEX. Μια ερώτηση αποτελείται από το όνομά της, ακολουθούμενο από κάποιο κείμενο που εμφανίζεται στην οθόνη, από τον τύπο της ερώτησης και από μια επεξήγηση που δηλώνεται προαιρετικά με τη λέξη **because**.

Υπάρχουν διαφορετικές μορφές σύνταξης μιας ερώτησης, ανάλογα με το αν ζητείται από το χρήστη να επιλέξει την απάντηση από ένα μενού ή εάν ζητείται να την πληκτρολογήσει. Οι ερωτήσεις που ακολουθούν χρησιμοποιούν μενού επιλογών για τον καθορισμό της απάντησης.

```
question question_name
```

```
    text of question;
    choose from menu items;
    because explanation.

question question_name
    text of question;
    choose one of menu items.

question question_name
    text of question;
    choose some of menu items.
```

Οι δύο πρώτες από τις παραπάνω ερωτήσεις ζητούν από το χρήστη να επιλέξει μια από τις επιλογές του μενού, ενώ η τρίτη ερώτηση επιτρέπει πολλαπλές επιλογές.

Οι παρακάτω ερωτήσεις ζητούν από το χρήστη να εισάγει την απάντηση μέσω του πληκτρολογίου. Ο τύπος της απάντησης μπορεί να είναι κάποιο όνομα, αριθμός, ακέραιος ή σύνολο.

```
question question_name
    text of question;
    input datatype;
    because explanation .
```

Ο τύπος **datatype** μπορεί να είναι **name**, **number**, **integer** ή **set**, με τις προφανείς ερμηνείες. Μπορεί μάλιστα να τεθεί επιπλέον περιορισμός στις δυνατές τιμές της απάντησης, χρησιμοποιώντας τις δεσμευμένες λέξεις **such that** ως εξής:

```
question age_of_applicant
    Please enter your age;
   input X such that integer(X) and X>18 .
```

Τέλος μπορεί να μην τεθεί βασικός τύπος απάντησης, παρά μόνο ειδικός περιορισμός, όπως φαίνεται στην παρακάτω σύνταξη:

```
question question_name
    answer is value
    such that directive.
```

Για παράδειγμα:

```
question check_printing_right
       answer is A such that
       msgbox('NOVELL EXPERT ASSISTANT',
       'Does the user have the right to print?',36,A).
```

Στο τελευταίο παράδειγμα δε χρησιμοποιείται ο ενσωματωμένος μηχανισμός απαντήσεων του FLEX, αλλά καλείται ένα κοινό πλαίσιο διαλόγου των Windows, το οποίο επιστρέφει κάποια τιμή. Μπορεί μάλιστα ο χρήστης να σχεδιάσει δικά του πλαίσια διαλόγου, τα οποία και θα εμφανίζονται όταν θα τίθεται η ερώτηση.

Δεδομένα και δηλώσεις

Τα *δεδομένα* (*data*) είναι μία σειρά από εντολές οι οποίες αντιστοιχίζουν τιμές σε ιδιότητες ή τροποποιούν τα γεγονότα μιας βάση δεδομένων. Χρησιμοποιούνται για

την αρχικοποίηση του προβλήματος, αφού οι προτάσεις αυτές θα εκτελεστούν κατά τη στιγμή της μεταγλώττισης και θα εκτελούνται κάθε φορά που καλείται η ρουτίνα του FLEX **restart/0**. Η σύνταξη μιας πρότασης δεδομένων είναι:

```
data data_name
    do directive(s)
```

Για παράδειγμα:

```
data start_up_configuration
    do the contents of jugA becomes 2
    and jugB`s contents becomes jugB`s capacity-2
    and remember that danger_level(yellow)
```

Μια πρόταση **do** είναι παρόμοια με μια πρόταση **data**, με τη διαφορά όμως ότι η πρόταση **do** εκτελεί τις εντολές της μόνο κατά τη στιγμή της μεταγλώττισης. Η σύνταξή της είναι η ακόλουθη:

```
do directive(s).
```

Για παράδειγμα:

```
do c1 is a new car whose engine_size is 1100
    and c1`s model := 'Whizzo'
    and c1`s colour := metallic_blue
```

Ομάδες

Μια *ομάδα* (*group*) είναι μια συλλογή αντικειμένων, στην οποία μπορεί κανείς να αναφέρεται με το όνομά της. Η σύνταξη δήλωσης μιας ομάδας είναι:

```
group group_name
    first_item, second_item, …, last_item;
```

Μια συνηθισμένη χρήση των ομάδων είναι για την απαρίθμηση των απαντήσεων σε ερωτήσεις που εμφανίζουν μενού επιλογών. Για παράδειγμα:

```
group wall_colours
    magnolia, coffee, apple_white, barley, buttermilk.

question wall_colour
    Please choose a colour for your room;
    choose from wall_colours.
```

Τα αντικείμενα μιας ομάδας μπορεί επίσης να είναι ονόματα κανόνων, οπότε η ομάδα αναφέρεται από ένα σύνολο κανόνων (ruleset). Για παράδειγμα:

```
group food_rules
    ask_for_the_starter, ask_for_the_entree.

ruleset dinner
    contains food_rules.
```

Τέλος, μία ομάδα μπορεί να περιέχει μια διατεταγμένη λίστα αντικειμένων η οποία μπορεί να χρησιμοποιηθεί για διάφορες συγκρίσεις. Για παράδειγμα:

```
group fuzzy_ordering
    impossible, improbable, possible, probable, definite.
```

```
relation less_likeky(A,B)
   if A is less than B
   according to fuzzy_ordering.
```

Συνώνυμα και πρότυπα

Τα *συνώνυμα* (*synonyms*) χρησιμοποιούνται για να απλοποιήσουν τον κώδικα, αντικαθιστώντας κάποιους συχνά εμφανιζόμενους όρους ή εκφράσεις με ένα όνομα. Ένα από τα πλεονεκτήματά τους είναι ότι κάνουν τον κώδικα ευανάγνωστο. Ένα άλλο είναι ότι όταν αλλάζει η τιμή μιας σταθεράς είναι ευκολότερο να αλλάξει στη δήλωση του συνωνύμου παρά σε κάθε σημείο του κώδικα που αυτή απαντάται. Η σύνταξη της δήλωσης ενός συνωνύμου είναι η εξής:

```
synonym name_to_replace replacement_expression.
```

Για παράδειγμα:

```
synonym  pi 3.14159
```

Ενώ τα συνώνυμα εκτελούν απλή αντικατάσταση κειμένου, τα *πρότυπα* (*templates*) μπορούν να περιλαμβάνουν και μεταβλητές μέσα στην έκφρασή τους. Η σύνταξη της δήλωσής τους είναι:

```
template replacement
   positive_template;
   negative_template.
```

Κατά τη στιγμή της μεταγλώττισης, όλοι οι όροι του **positive_template** αντικαθίστανται από την **replacement**, ενώ όλοι οι όροι του **negative_replacement** αντικαθίστανται από την **not replacement**. Το σύμβολο ^ χρησιμοποιείται για να υποδείξει τη θέση κάποιου ορίσματος. Για παράδειγμα, έστω το ακόλουθο πρότυπο:

```
template on_top_of
   block ^ is on top of  ^ ;
   block ^ is not on top of ^.
```

Τότε η ακόλουθη πρόταση:

```
block p is on top of the table
```

γίνεται:

```
on_top_of( p, table )
```

Π3.3.7 Παράδειγμα Έμπειρου Συστήματος σε FLEX

Στην ενότητα αυτή περιγράφεται η υλοποίηση ενός έμπειρου συστήματος στην πλατφόρμα LPA-FLEX. Το σύστημα είναι μια περιορισμένη εκδοχή του γνωστού έμπειρου συστήματος MYCIN για διάγνωση ασθενειών από παρατηρούμενα συμπτώματα με χρήση συντελεστών βεβαιότητας. Μια πιο ολοκληρωμένη περιγραφή του MYCIN υπάρχει στο σχετικό κεφάλαιο.

Στο MYCIN, τα δεδομένα αποθηκεύονται σε τριάδες *Αντικείμενο-Ιδιότητα-Τιμή* (*Object-Attribute-Value* ή *OAV*). Συνήθως τα αντικείμενα είναι οι ασθενείς, ενώ οι

ιδιότητες είναι τα διάφορα συμπτώματα, ασθένειες, χαρακτηριστικά, κλπ, του ασθενή. Για παράδειγμα, μερικές τριάδες θα μπορούσε να είναι οι εξής:

```
<Γιάννης, ηλικία, 30>
<Νίκος, ηλικία, 31>
<Γιάννης, σύμπτωμα, πυρετός>
<Γιάννης, σύμπτωμα, πονόλαιμος>
<Γιάννης, ασθένεια, ίωση>
```

κλπ.

Όπως φαίνεται από τις παραπάνω τριάδες, είναι δυνατό αυτές να αναφέρονται στο ίδιο αντικείμενο και στην ίδια ιδιότητα. Ωστόσο δεν είναι δυνατό δύο τριάδες να είναι εντελώς ίδιες, δηλαδή να έχουν ίδια τιμή και στις τρεις ιδιότητες. Τέλος, όπως έχει ήδη ειπωθεί, κάθε τριάδα χαρακτηρίζεται από ένα συντελεστή βεβαιότητας, ο οποίος κυμαίνεται από -1 μέχρι 1, όπου η τιμή 1 σημαίνει απόλυτη βεβαιότητα για την αλήθεια της πρότασης που περιγράφει η τριάδα, η τιμή -1 δηλώνει απόλυτη βεβαιότητα για το ψευδές της τριάδας και η τιμή 0 σημαίνει πλήρη άγνοια. Οι ενδιάμεσες τιμές ορίζουν ενδιάμεσες καταστάσεις βεβαιότητας. Στο FLEX οι τριπλέτες περιγράφονται με το παρακάτω πλαίσιο:

```
frame oav;
    default obj is unknown and
    default attribute is unknown and
    default value is unknown and
    default cf is 0 .
```

όπου οι τέσσερις ιδιότητες του πλαισίου είναι το αντικείμενο, η ιδιότητα, η τιμή και ο συντελεστής βεβαιότητας της τριάδας.

Αρχικά το έμπειρο σύστημα εμφανίζει ένα μενού επιλογών, από όπου ο χρήστης επιλέγει την προσθήκη ενός νέου ασθενή, τη δήλωση ενός συμπτώματος ή τον τερματισμό του προγράμματος. Η σχετική ερώτηση που εμφανίζει το μενού επιλογών είναι η παρακάτω:

```
question main_menu
    Select one of the following functions;
    choose one of 'Add new person', 'Add new symptom', 'End program' ;
    because 'Click on one of the three choices and then click OK'.
```

Εάν ο χρήστης επιλέξει να προσθέσει ένα νέο ασθενή, το σύστημα ζητά το όνομα του ασθενή, την ηλικία και το φύλο του. Οι σχετικές ερωτήσεις είναι οι παρακάτω:

```
question get_name
    Give a new unique person name;
    input Name such that unique_name(Name);
    because 'Type the name of the person and then click OK'.

question get_age
    Give the age of the person;
    input X such that integer(X) and X>=0;
    because 'Type the age of the person. Age must be a non negative integer. '.
```

```
question get_sex
    Choose the sex of the person;
    choose one of 'Male', 'Female';
    because 'Click on one of the two choices and then click OK'.
```

Η ερώτηση **get_name** ελέγχει εάν το όνομα ασθενούς που εισάγεται είναι νέο. Αυτό γίνεται απαιτώντας το όνομα να ικανοποιεί τη σχέση **unique_name**:

```
relation unique_name(Name)
    if X is an OAV and X`s obj is Name and
    write('ERROR: Name ') and write(Name) and
    write(' already exists. Give a new unique name.') and nl and
    ttyflush and ! and fail.
relation unique_name(Name).
```

Στην περίπτωση που ο χρήστης ζητήσει να προσθέσει ένα σύμπτωμα, το σύστημα ζητά από το χρήστη να επιλέξει έναν ασθενή από τη λίστα των ασθενών, ένα σύμπτωμα από τον κατάλογο των συμπτωμάτων και να δώσει το βαθμό βεβαιότητας με τον οποίο είναι γνωστό το σύμπτωμα. Οι σχετικές ερωτήσεις είναι οι ακόλουθες:

```
question choose_a_person
    Select one of the following persons;
    choose one of persons;
    because 'Click on one person`s name and then click OK'.

question choose_a_symptom
    Select one of the following symptoms;
    choose one of symptoms;
    because 'Click on one symptom and then click OK'.

question give_cf
    Give the confidence factor for the previous symptom;
    input CF such that check_cf(CF);
    because 'Give the confidence factor and click OK. Confidence factor
    can be any real number between -1 and 1'.
```

Η ερώτηση **choose_a_symptom** ζητά από το χρήστη να διαλέξει ένα σύμπτωμα από μια ομάδα (group) συμπτωμάτων. Ο ορισμός της ομάδας αυτής είναι στατικός:

```
group symptoms
    fever, throat_ache, toothache.
```

Παρόμοια, η ερώτηση **choose_a_person** ζητά από το χρήστη να διαλέξει έναν ασθενή από την ομάδα των ασθενών. Η ομάδα αυτή, που ονομάζεται **persons**, δημιουργείται δυναμικά κάθε φορά που καλείται η ερώτηση **choose_a_person**, όπως θα φανεί παρακάτω.

Η ερώτηση **give_cf** ελέγχει εάν ο συντελεστής βεβαιότητας που δόθηκε είναι μέσα στα επιτρεπτά όρια. Ο έλεγχος γίνεται με τη σχέση **check_cf**:

```
relation check_cf(CF) if not number(CF) and
    write('WRONG CF: Confidence factor must be a real number between -1
        and 1') and nl and ttyflush and
    ! and fail.
relation check_cf(CF) if CF < -1 and
    write('WRONG CF: Confidence factor must be a real number between -1
        and 1') and nl and ttyflush and
    ! and fail.
relation check_cf(CF) if CF > 1 and
    write('WRONG CF: Confidence factor must be a real number between -1
        and 1') and nl and ttyflush and
    ! and fail.
relation check_cf(CF).
```

Η εκκίνηση του έμπειρου συστήματος γίνεται με την παρακάτω ενέργεια:

```
action start;
do  ask main_menu and
    while main_menu\='End program' do
        if main_menu='Add new person' then
            add_new_person
        else
            add_new_symptom
        end if
            and
        ask main_menu
    end while.
```

Η ενέργεια αυτή εμφανίζει το αρχικό μενού επιλογών και ανάλογα με την επιλογή του χρήστη καλεί μια από τις ενέργειες **add_new_person** και **add_new_symtom** . Η ενέργεια **add_new_person** είναι υπεύθυνη για τη δημιουργία ενός νέου ασθενή.

```
action add_new_person;
do  write('Adding new person...') and nl and ttyflush and
    ask get_name and
    ask get_age and
    ask get_sex and
    create_oav(get_name, age, get_age, 1) and
    create_oav(get_name, sex, get_sex, 1) and
    write('New person ') and write(get_name) and write(' has created successfully.')
and nl and
    write(get_name) and write('`s age = ') and write(get_age) and nl and
    write(get_name) and write('`s sex = ') and write(get_sex) and nl and nl and
ttyflush.
```

Αφού ζητηθούν τα στοιχεία του ασθενή, δημιουργούνται δύο τριάδες, μία για την ηλικία του και μια για το φύλο του, με συντελεστή βεβαιότητας 1. Η δημιουργία των νέων τριπλετών γίνεται με την ενέργεια **create_oav**, η οποία δέχεται σαν παραμέτρους τις τιμές της τριάδας. Οι νέες τριάδες κατασκευάζονται σαν στιγμιότυπα του πλαισίου **oav**.

```
action create_oav(Object, Attribute, Value, CF);
do if X is some oav and
   X`s obj is Object and
   X`s attribute is Attribute and
   X`s value is Value
then
   check that CF1 is X`s cf and
   if CF>0 and CF1 > 0 then
      X`s cf becomes CF+CF1 - CF*CF1   % CF>0, CF1>0
      and write_oav(Object, Attribute, Value, X`s cf)
   else
      if CF<0 and CF1<0 then
      X`s cf becomes CF+CF1 + CF*CF1   % CF<0, CF1<0
      and write_oav(Object, Attribute, Value, X`s cf)
      else
      abs_min(CF, CF1, Min) and
      X`s cf becomes (CF+CF1)/(1- Min) % CF*CF1<0
      and write_oav(Object, Attribute, Value, X`s cf)
      end if
   end if
else
   get_new_oav(OAV) and
   OAV is a new oav and
   OAV`s obj becomes Object and
   OAV`s attribute becomes Attribute and
   OAV`s value becomes Value and
   OAV`s cf becomes CF and
   write_oav(Object, Attribute, Value, CF)
end if.
```

Η ενέργεια **create_oav** ελέγχει εάν υπάρχει ήδη όμοια τριάδα. Εάν δεν υπάρχει, δημιουργεί μια νέα με τις τιμές των παραμέτρων εισόδου της. Εάν υπάρχει όμοια τριάδα δεν δημιουργείται νέα, αλλά ο συντελεστής βεβαιότητας της υπάρχουσας τριάδας συνδυάζεται με το συντελεστή βεβαιότητας που δόθηκε ως παράμετρος εισόδου, βάσει των τύπων που έχουν παρουσιαστεί στο σχετικό κεφάλαιο. Έτσι προκύπτει ο νέος συντελεστής βεβαιότητας της υπάρχουσας τριάδας. Ορισμένες βοηθητικές ενέργειες και σχέσεις που χρησιμοποιούνται από την **create_oav** είναι οι ακόλουθες:

```
action write_oav(Object, Attribute, Value, CF);
do  write('OAV changed succesfully') and nl and
   write('Object ') and write(Object) and write(' has attribute ') and
write(Attribute) and
   write(' equal to ') and write(Value) and write(' with probability ') and
write(CF) and nl and nl and ttyflush.

relation abs_min(CF1, CF2, MIN)
   if abs(CF1, ABS1) and
   abs(CF2, ABS2) and
   min(ABS1, ABS2, MIN).
```

```
relation abs(X, X) if X>=0.
relation abs(X, Y) if X<0 and Y is -X.

relation min(X, Y, X) if X=<Y.
relation min(X, Y, Y) if Y<X.


action get_new_oav(OAV);
do  if counter(X) then
        check that X1 is X+1 and
        retract(counter(X)) and ! and
        assert(counter(X1))
    else
        assert(counter(1))
        and X1=1
    end if and
    number_atom(X1, C_atom) and
    cat({'oav', C_atom}, OAV, _ ) and !.
```

Η ενέργεια **write_oav** εμφανίζει τα στοιχεία μιας τριάδας στην οθόνη. Η σχέση **relation_abs** επιστρέφει την απόλυτη τιμή ενός αριθμού, ενώ η σχέση **relation_min** επιστρέφει το μικρότερο από δύο αριθμούς. Τέλος, η ενέργεια **get_new_oav** επιστρέφει ένα μοναδικό όνομα για τη νέα τριάδα.

Εάν στο αρχικό μενού επιλογών ο χρήστης επιλέξει να δηλώσει ένα σύμπτωμα, τότε εκτελείται η ενέργεια **add_new_symptom**.

```
action add_new_symptom;
do  write('Adding new symptom...') and nl and ttyflush and
    findall(Person, isa_person(Person), Persons) and
    new_group(persons, Persons) and
    ask choose_a_person and
    ask choose_a_symptom and
    ask give_cf and
    create_oav(choose_a_person, symptom, choose_a_symptom, give_cf) and
    invoke ruleset mycin.
```

Η ενέργεια **add_new_symptom** δημιουργεί δυναμικά μια ομάδα με τα ονόματα όλων των ασθενών. Για τη δημιουργία αυτής της ομάδας χρησιμοποιείται η σχέση **isa_person**.

```
relation isa_person(Person) if X is an oav and
    X`s obj is Person and
    X`s attribute is age.
```

Στη συνέχεια, η ενέργεια **add_new_symptom** ζητά τα απαραίτητα στοιχεία και δημιουργεί τη σχετική τριάδα. Τέλος, εκτελεί τους κανόνες του έμπειρου συστήματος,

ώστε να εξαχθούν, αν είναι δυνατόν, οι σχετικές διαγνώσεις. Οι κανόνες δηλώνονται στην ομάδα κανόνων **mycin**:

```
ruleset mycin
   contains fever, throat_ache.
```

και είναι οι ακόλουθοι:

```
rule fever
if
   X is an instance of oav and
   X`s attribute is symptom and
   X`s value is fever and
   Object is X`s obj and
   cat({Object, '_fever'}, NewLog, _) and
   not NewLog is an instance of log_fever and
   CF is X`s cf
then
   write('rule fever has fired') and nl and
   check that VirusCF is 0.6* CF times X`s cf and
   write('Patient ') and write(Object) and
   write(' has been infected by a virus with certainty ') and
   write(VirusCF) and nl and nl and ttyflush and
   NewLog is a new log_fever and
   create_oav(Object, disease, virus, VirusCF).

rule throat_ache
if
   X is an instance of oav and
   X`s attribute is symptom and
   X`s value is throat_ache and
   Object is X`s obj and
   cat({Object, '_throat_ache'}, NewLog, _) and
   not NewLog is an instance of log_throat_ache and
   CF is X`s cf
then
   write('rule throat_ache has fired') and nl and
   check that VirusCF is 0.8* CF times X`s cf and
   write('Patient ') and write(Object) and
   write(' has been infected by a virus with certainty ') and
   write(VirusCF) and nl and nl and ttyflush and
   NewLog is a new log_throat_ache and
   create_oav(Object, disease, virus, VirusCF).
```

Κάθε ένας από τους δυο κανόνες αναζητά εάν υπάρχει σχετική τριάδα, για την οποία ο κανόνας δεν έχει εκτελεστεί ξανά. Εάν υπάρχει, ο κανόνας εκτελείται, εμφανίζει στην οθόνη σχετικά μηνύματα, και δημιουργεί μια νέα τριάδα με την διάγνωση, μαζί με τον αντίστοιχο συντελεστή βεβαιότητας. Για να μην εκτελεστεί ξανά ένας κανόνας με ίδια δεδομένα (δηλ. με ίδιες ακριβώς τριάδες), κάθε φορά που εκτελείται δημιουργεί ένα στιγμιότυπο ενός πλαισίου, με το όνομα του κανόνα και του ασθενή. Τα δύο πλαίσια που αντιστοιχούν στους παραπάνω κανόνες είναι τα:

```
frame log_fever.
frame log_throat_ache.
```

Έτσι για παράδειγμα, εάν ο κανόνας **fever** εκτελεστεί για τον ασθενή **john**, δημιουργείται ένα στιγμιότυπο του πλαισίου **log_fever** με όνομα **john_fever**. Έτσι, την επόμενη φορά που θα ενεργοποιηθούν οι κανόνες, ο κανόνας **fever** θα ανιχνεύσει το στιγμιότυπο **john_fever** και δεν θα εκτελεστεί ξανά για τον ασθενή **john**. Εάν αυτός ο έλεγχος δεν γινόταν, η επαναληπτική εκτέλεση ενός κανόνα στα ίδια δεδομένα θα είχε σαν αποτέλεσμα συντελεστές βεβαιότητας 1 σε όλες τις διαγνώσεις.