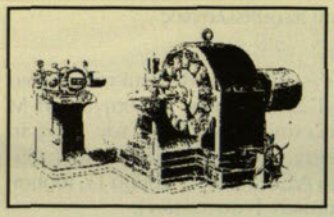


Βάσεις γνώσεων

της Φ. Αφράτη

1. Εισαγωγή

Τις τελευταίες δεκαετίες, η προσπάθεια για την ανάπτυξη ολοένα και εξυπνότερου λογισμικού (software), έχει οδηγήσει στην καθιέρωση μιας αντίστοιχα φιλόδοξης ορολογίας. Ο όρος «γνώση» συναντάται ολοένα συχνότερα σε κείμενα της περιοχής της τεχνητής νοημοσύνης και άλλων παρεμφερών περιοχών. Τις περισσότερες φορές, αναφέρεται στη δυνατότητα του προγράμματος να επεξεργάζεται ένα συνολο από προτάσεις της πρωτοβάθμιας ή (σπανιότερα) δευτεροβάθμιας λογικής, αλλά υπάρχουν επιστημονικές ομάδες που η αντίληψή τους και η παραπέρα επεξεργασία αυτής της βασικής ιδέας, διαφέρει σημαντικά.

Ένα σύστημα για βάσεις γνώσεων, περιμένουμε να προσφέρει τις ευκολίες που προσφέρει ένα σύστημα για βάσεις δεδομένων (δηλαδή, να διαχειρίζεται με αποδοτικό τρόπο μεγάλες ποσότητες δεδομένων), αλλά επίσης, να υποστηρίζει κάποια λογική γλώσσα επικοινωνίας με το χρήστη (σημείωση: η λέξη «λογική» σε αυτό το κείμενο αναφέρεται πάντα στη μαθηματική λογική). Στα επόμενα, θα αναφερθούμε σε γενικές περιοχές αλλά και σε συγκεκριμένα παραδείγματα, όπου ένα κλασικό σύστημα βάσεων δεδομένων δεν αρκεί. Θα περιγράψουμε ένα σύστημα το οποίο:

α) διαθέτει τις επί πλέον ευκολίες που αυτές οι εφαμογές απαιτούν, και

β) μπορεί να υλοποιηθεί «σχετικά εύκολα», δηλαδή, λαμβανομένων υπόψη των τεχνολογικών επιτευγμάτων, περιμένει κανείς ότι μπορούν να βρεθούν αποδοτικοί αλγόριθμοι για την επεξεργασία των ερωτήσεων που θα υποβάλει ο χρήστης. Τέλος, θα περιγράψουμε μερικά πειραματικά συστήματα τα οποία έχουν οικοδομηθεί με βάση αυτές τις αρχές.

Η Φ. Αφράτη είναι αναπληρώτρια καθηγήτρια στο Τομέα Πληροφορικής του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του ΕΜΠ.

2. Εφαρμογές

Τα κλασικά συστήματα βάσεων δεδομένων έχουν σχεδιαστεί για να διεκπεραιώνουν μερικές πολύ σημαντικές, αλλά περιορισμένου βεληνεκούς, εφαρμογές. Μερικές πολύ συνηθισμένες τέτοιες εφαρμογές είναι ο χειρισμός αρχείων υπαλλήλων, κρατήσεις θέσεων σε αεροπορικές εταιρείες, και ο χειρισμός αρχείων με τα οικονομικά στοιχεία μιας επιχείρησης. Το κοινό χαρακτηριστικό αυτών των εφαρμογών είναι ότι πρέπει να επεξεργαστούν μεγάλες ποσότητες δεδομένων, αλλά οι πράξεις που θέλουμε να εκτελεστούν, πάνω σε αυτά τα δεδομένα, είναι απλές. Σε τέτοια συστήματα, κυρίως ζητείται να γίνει εισαγωγή, διαγραφή η ανάκτηση ορισμένων εγγραφών, και ένα από τα πιο πολύπλοκα πράγματα που μπορεί να ζητηθεί από το σύστημα, είναι να συνδυάσει τιμές μεταξύ ενός μικρού αριθμού αρχείων. Εν όψει αυτών των εφαρμογών, έχει καθιερωθεί στα υπάρχοντα συστήματα ο διαχωρισμός μεταξύ της γλώσσας προσπέλασης (ή γλώσσας επικοινωνίας με το χρήστη - query language, user interface language) και της γλώσσας του συστήματος (η γλώσσα η οποία χρησιμοποιήθηκε για να κατασκευαστεί το σύστημα και η οποία θα χρησιμοποιηθεί για να γίνουν τυχόν μεταβολές - host language). Από αυτές, μόνον η γλώσσα προσπέλασης έχει ενσωματωμένες τεχνικές βελτιστοποίησης, οι οποίες χρησιμεύουν για τη γρήγορη επεξεργασία των ερωτήσεων.

Για να δείξουμε πιο καθαρά τους λόγους για τους οποίους χρειαζόμαστε πιο πλούσια γλώσσα προσπέλασης και, για να διασαφηνίσουμε τα χαρακτηριστικά των λογικών γλωσσών προσπέλασης (που θα ορίσουμε παρακάτω), ας θεωρήσουμε το ακόλουθο παράδειγμα: Υπάρχει ένα σύνολο από πόλεις-κόμβους, υπάρχει και ένα οδικό δίκτυο που τους συνδέει, το οποίο το περιγράφουμε ως ένα σύνολο από δρόμους που συνδέουν απ' ευθείας δυο πόλεις (δηλαδή, θεωρούμε ότι ένας δρόμος δεν διέρχεται από τρίτη πόλη). Είναι γνωστό ότι, από αυτή την πληροφορία μπορούμε, κατ' αρχήν, να απαντήσουμε στην εξής ερώτηση: «Υπάρχει διαδρομή που να συνδέει την πόλη Α με την πόλη Β;» Με τις γλώσσες προσπέλασης που διαθέτουν οι κλασικές βάσεις δεδομένων (QUEL, QBE, SQL), ο χρήστης δε μπορεί να κάνει αυτή την ερώτηση. Και αυτό, διότι οι περισσότερες γνωστές γλώσσες οι οποίες χρησιμοποιούνται ευρέως για να προσπελάσουμε μια σχεσιακή βάση δεδομένων έχει αποδειχθεί ότι διαθέτουν «περιορισμένη εκφραστική δύναμη». Οι κοινές γλώσσες προσπέλασης εν γένει, δε μπορούν να εκφράσουν ερωτήσεις (σαν την προηγούμενη) που αφορούν «μη τοπικές» ιδιότητες της αποθηκευμένης βά-

σης δεδομένων, δηλαδή ιδιότητες που συνδέουν δυο «μακρινές» σταθερές (ο αριθμός των ενδιάμεσων πόλεων της διαδρομής στο παράδειγμά μας, θα μπορούσε να ήταν πολύ μεγάλος, ανάλογος με τον αριθμό των πόλεων στη βάση δεδομένων). Οι γλώσσες αυτές έχουν μόνο *την εκφραστική δύναμη της πρωτοβάθμιας λογικής*.

Η γλώσσα του συστήματος, όμως, αφού είναι μια γενική γλώσσα προγραμματισμού, μπορεί να εκφράσει τα πάντα, άρα και ερωτήσεις σαν την παραπάνω. Αυτό, όμως, δεν είναι χρήσιμο διότι:

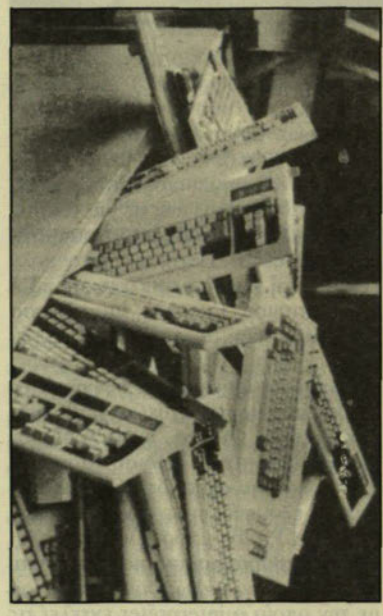
α) ο χρήστης θα ήθελε να χρησιμοποιεί μια γλώσσα στην οποία να μπορεί να εκφράσει εύκολα και απλά τις ερωτήσεις του (θυμηθείτε ότι ο χρήσης δεν είναι ειδικός), και

β) η γλώσσα του συστήματος δε διαθέτει μηχανισμούς βελτιστοποίησης για να επεξεργάζεται ερωτήσεις.

Φυσικά, αυτές οι ερωτήσεις θα ήταν πάλι από μια περιορισμένη (σε εκφραστικότητα) κατηγορία αλλά ευρύτερη από ό,τι συνήθως έχουν την δυνατότητα να επεξεργάζονται τα κλασικά συστήματα. Ο διαχωρισμός της γλώσσας προσπέλασης από τη γλώσσα του συστήματος θεωρείται, εν γένει, πλεονέκτημα στα κλασικά συστήματα, και τούτο, διότι η περιορισμένη εκφραστική ικανότητα της γλώσσας προσπέλασης, είναι το χαρακτηριστικό που επιτρέπει την ανάπτυξη και ενσωμάτωση στο σύστημα μηχανισμών βελτιστοποίησης, που κάνουν δυνατή τη γρήγορη επεξεργασία των ερωτήσεων. Ωστόσο, για κάποιες καινούργιες εφαρμογές, φαίνεται ότι είναι πολύ σημαντική η θεώρηση ενιαίας γλώσσας, η οποία να μπορεί να χρησιμοποιηθεί, τόσο ως γλώσσα προσπέλασης, όσο και ως γλώσσα του συστήματος. Χαρακτηριστικά, τέτοιες εφαρμογές περιλαμβάνουν τη χρήση βάσεων δεδομένων στην αυτόματη σχεδίαση VLSI, στην αυτόματη ανάπτυξη software (CASE) και στην αυτόματη σχεδίαση (CAD). Αυτές οι εφαρμογές απαιτούν πολύ πιο ισχυρή γλώσσα επικοινωνίας με το χρήστη.

Στις βάσεις δεδομένων που χρησιμοποιούνται για σχεδίαση αντικειμένων τα οποία συντίθενται με ιεραρχικό τρόπο, όπως συμβαίνει στην αυτόματη σχεδίαση VLSI, είναι πολύ σημαντικό να έχουμε στη διάθεσή μας μια πιο ισχυρή γλώσσα, για πολλούς λόγους, μεταξύ των οποίων οι ακόλουθοι. Στην περίπτωση των VLSI, τα chips σχεδιάζονται ως αποτελούμενα από cells, τα cells ως αποτελούμενα από subcells κ.ο.κ., μέχρι κάποιο επίπεδο, πεπερασμένο μεν, αλλά το βάθος του οποίου εξαρτάται από την εφαρμογή. Αν θέλουμε να αναπτύξουμε μια σχεδίαση, είτε για να παρατηρήσουμε ή για να επεξεργαστούμε ολόκληρο το αναπαριστώμενο αντικείμενο, τότε θα πρέπει να έχουμε στη διάθεσή μας έναν αναδρομικό τελεστή, όπως και στην περίπτωση του παραδείγματός μας με το οδικό δίκτυο.

Στις βάσεις δεδομένων που χρησιμοποιούνται για την αυτόματη σχέδιαση software (CASE), επίσης, τα προ-



γράμματα, συνήθως, παριστώνται ως αναδρομικές δομές, όπως, π.χ. τα parse trees. Ένας άλλος τρόπος παράστασης, είναι υπό μορφή γράφου, οπότε απαιτείται η δυνατότητα επεξεργασίας ενός τέτοιου γράφου, του οποίου το μέγεθος εξαρτάται από την εφαρμογή. Αυτός επίσης, δε μπορεί να γίνει με τις γλώσσες που έχουν οι κλασικές βάσεις δεδομένων. Παράδειγμα ενός τέτοιου γράφου είναι το διάγραμμα ροής. Ένα άλλο παράδειγμα είναι ο γράφος που παριστά τις αλληλεξαρτήσεις μεταξύ των διαφόρων κομματιών του προγράμματος, και δείχνει τον τρόπο με τον οποίο, μεταβολές σε ένα κομμάτι του προγράμματος επηρεάζουν άλλα κομμάτια του προγράμματος.

Ένα τρίτο παράδειγμα εφαρμογής των βάσεων γνώσεων, είναι στην αναγνώριση προτύπων (pattern recognition). Σ' αυτή την περίπτωση, «μη επεξεργασμένα» δεδομένα, όπως π.χ., bits, ομαδοποιούνται σταδιακά σε πιο πολύπλοκες δομές. Ένα παράδειγμα είναι μια βάση δεδομένων που έχει αποθηκευμένες φωτογραφίες. Τότε, μπορεί να θέλουμε να ομαδοποιήσουμε τα bits σε γραμμές και καμπύλες, μετά να ομαδοποιήσουμε τις γραμμές και καμπύλες σε άλλα πρότυπα (patterns), στη συνέχεια, να ομαδοποιήσουμε τα απλά πρότυπα σε πιο σύνθετα, π.χ., σε περιοχές στη φωτογραφία, κ.ο.κ. Ένα τέτοιο πρόβλημα αναγνώρισης προτύπων, είναι το πρόβλημα αναγνώρισης των πρωτεϊνικών αλυσίδων που αποτελούν τα γονίδια. Σε πολλές περιπτώσεις επίσης, χρειάζεται να περιγράψουμε πολύπλοκες δομές με γραμματικές, δηλαδή να δώσουμε έναν ανδρομικό ορισμό. Οι γλώσσες προσπέλασης των κλασικών βάσεων δεδομένων, δε μπορούν να χειριστούν πρότυπα που έχουμε ορίσει με αναδρομικό τρόπο.

3. Γλώσσα Επικοινωνίας με το Χρήστη με Λογικούς Κανόνες

Υπάρχουν δύο επικρατέστερες προσεγγίσεις στο πρόβλημα της προηγούμενης παραγράφου, της ενοποίησης, δηλαδή, της γλώσσας επικοινωνίας με το χρήστη με τη γλώσσα του συστήματος. Η μια είναι αντικειμενοστρεφής (object-oriented) προσέγγιση, όπου χρησιμοποιείται μια γλώσσα με την δυνατότητα να ορίζει τύπους ή κλάσεις (κατηγορίες) αντικειμένων. Έτσι, το σύστημα επιτρέπει στο χρήστη να ενσωματώσει σε κάθε κλάση διαφορετικές δομές δεδομένων, οι οποίες χρησιμοποιούνται για γρήγορη προσπέλαση αντικειμένων αυτού του τύπου ή αντικειμένων που ανήκουν σ' αυτή την κατηγορία. Η άλλη είναι η προσέγγιση της λογικής. Στην περίπτωση αυτή, χρησιμοποιείται μια γλώσσα με λογικούς κανόνες, όπου μερικά κατηγορήματα θεωρούνται μέρος της αποθηκευμένης βάσης δεδομένων (του conceptual scheme), ενώ άλλα κατηγορήματα χρησιμοποιούνται για να εκφράσουν «όψεις» (views), σαν να ήταν μέρος του subscheme στις κλασικές βάσεις δεδομένων.

Ας θεωρήσουμε, πάλι, το πρόβλημα του οδικού δικτύου. Φανταστείτε ότι η πληροφορία για τους απ' ευθείας δρόμους που συνδέουν δύο πόλεις, είναι αποθηκευμένη στη βάση δεδομένων ως δυαδική σχέση *δρόμος*, η οποία αντιστοιχεί στο κατηγόρημα *δρόμος*, ως εξής: Το *δρόμος* (A, B) είναι αληθές, τότε και μόνον τότε, αν υπάρχει δρόμος από την πόλη *A* στην πόλη *B*. Τότε, η ερώτηση για την ύπαρξη διαδρομής θα εκφρασθεί με το ακόλουθο «λογικό πρόγραμμα» (έτσι ονομάζεται ένα σύνολο από λογικούς κανόνες):

διαδρομή (x, y): – διαδρομή (x, z), δρόμος (z, y).

διαδρομή (x, y): – δρόμος (x, y).

:– διαδρομή (A, B)

Αυτή τη γλώσσα μπορεί να τη φανταστεί κανείς ως Prolog χωρίς συναρτησιακά σύμβολα. Κάθε κανόνας περιγράφει έναν από τους τρόπους με τους οποίους μπορούμε να συνδέσουμε τις μεταβλητές *x* και *y* (δύο οποιεσδήποτε πόλεις, αν θέλετε) με τη σχέση *διαδρομή*. (Ο τρίτος κανόνας είναι απλώς ένας ενιαίος τρόπος να διατυπώσουμε την ερώτηση, αν υπάρχει διαδρομή από την πόλη *A* στην πόλη *B*). Η περιγραφή αυτή χρησιμοποιεί άλλες σχέσεις και, ίσως, την ίδια τη σχέση *διαδρομή*. Όταν ένας κανόνας χρησιμοποιεί την ίδια σχέση που περιγράφει, τότε λέγεται αναδρομικός κανόνας. Έτσι, μπορεί να φανταστεί κανείς ότι, το παραπάνω πρόγραμμα ορίζει μια καινούργια σχέση (τη σχέση *διαδρομή*) συναρτήσει της σχέσης *δρόμος*. Δεδομένης της σχέσης *δρόμος*, το αποτέλεσμα της συνάρτησης αυτής είναι η μικρότερη σχέση *διαδρομή*, για την οποία συμβαίνει το εξής: Για όλους τους κανόνες του προγράμματος και για όλες τις αντικαταστάσεις των μεταβλητών με σταθερές: αν κάθε κατηγόρημα στο δεξιό μέρος κάθε κανόνα είναι αληθές τότε είναι αληθές, και το αριστερό μέρος του κανόνα. Αυτή η *μοναδική* σχέση ονομάζεται το *ελάχιστο σταθερό σημείο* (least fixed point) του προγράμματος δεδομένης της σχέσης *δρόμος*.

Ένα άλλο πρόγραμμα, που εκφράζει, ισοδύναμα, την ίδια ερώτηση, είναι:

διαδρομή (x, y): – διαδρομή (x, z), διαδρομή (z, y).

διαδρομή (x, y): – δρόμος (x, y).

: – διαδρομή (A, B)

Η γλώσσα με τους λογικούς κανόνες, που μόλις περιγράψαμε, «προσθέτει την ισχύ της αναδρομής στην πρωτοβάθμια λογική». Με άλλα λόγια, επιτρέπει να γίνουν ερωτήσεις, όπως η παραπάνω, οι οποίες συνδέουν «μακρινές» σταθερές στη βάση (στο γράφο, στο παράδειγμά μας). Γλώσσες σαν αυτή, ονομάζονται *δηλωτικές* (declarative) γλώσσες, εν αντιθέσει με τις *διαδικασιακές* (procedural) γλώσσες.

Ένα χαρακτηριστικό τους (που εύκολα παρατηρεί ο χρήστης), είναι ότι επιτρέπουν την διατύπωση πολύπλοκων ερωτήσεων με πολύ απλό τρόπο. Παρατηρείστε ότι δεν είναι εύκολο να εκφράσει ένας μη ειδικός την ερώτηση του παραδείγματός μας σε μια από τις συνηθισμένες γλώσσες προγραμματισμού (Pascal, C, Lisp), ενώ ήταν πολύ εύκολο στη γλώσσα του λογικού προγραμματισμού. Εν γένει, μια γλώσσα είναι δηλωτική, αν ο χρήστης μπορεί να εκφράσει την ερώτησή του χωρίς να είναι απαραίτητο να εξηγήσει ακριβώς πως θα παραχθεί το επιθυμητό αποτέλεσμα. Φυσικά, ο διαχωρισμός μεταξύ δηλωτικών ή διαδικασιακών γλωσσών είναι σχετικός. Όποιος έχει κάποια εξοικείωση με την Prolog, ξέρει ότι η Prolog μπορεί να θεωρηθεί ως μια καθαρά διαδικασιακή γλώσσα, καθώς έχει μια καλώς ορισμένη σειρά με την οποία ο interpreter εκτελεί τις απαραίτητες πράξεις για να απαντήσει σε μια ερώτηση η οποία υποβάλλεται υπό μορφή λογικού κανόνα. Όμως, σύμφωνα με τα παραπάνω, μπορεί κανείς να δει την Prolog και ως μια καθαρά δηλωτική γλώσσα.

Τα συστήματα βάσεων γνώσεων που υποστηρίζουν μια δηλωτική γλώσσα, συνηθίζεται να ονομάζονται και *βάσεις δεδομένων με κανόνες παραγωγής* (deductive databases).

4. Ερευνητικά προβλήματα

Τα κυριότερα ερευνητικά προβλήματα, έχουν σχέση με τη βελτιστοποίηση ερωτήσεων από τον interpreter. Ο τρόπος που αυτό επιτυγχάνεται είναι συνήθως ο ακόλουθος: Ο interpreter μεταφράζει το πρόγραμμα σε ένα άλλο ισοδύναμο, για το οποίο, όμως, ξέρουμε κάποιον αποδοτικό αλγόριθμο επεξεργασίας του και, κατόπιν, επεξεργάζεται το δεύτερο αυτό πρόγραμμα. Με κεντρικό στόχο αυτή τη βελτιστοποίηση ερωτήσεων, αναφέρουμε μερικά από τα θέματα που εγείρονται προς μελέτη.

Δεδομένου ενός λογικού προγράμματος και της αντίστοιχης βάσης δεδομένων, υπάρχει ένας προφανής τρόπος (αλλά όχι κατ' ανάγκη γρήγορος) για να υπολογίσει κανείς τις σχέσεις τις οποίες ορίζει το πρόγραμμα: Ξεκινάμε θεωρώντας κενές τις προηγούμενες σχέσεις και εφαρμόζουμε τους κανόνες πολλές φορές, αλλά κάθε φορά προσθέτουμε στις παραγόμενες σχέσεις τα καινούργια γεγονότα που προέκυψαν κατά την τελευταία εφαρμογή των κανόνων. Όταν δε μπορούμε να προσθέσουμε καινούργια γεγονότα, σταματάμε. Έχουμε υπολογίσει τις παραγόμενες σχέσεις.

Τα δύο λογικά προγράμματα της προηγούμενης παραγράφου είναι ισοδύναμα με την έννοια ότι, αν εφαρμοστούν στην ίδια βάση δεδομένων (οποιαδήποτε κι αν είναι), δίνουν το ίδιο αποτέλεσμα. Έχουν όμως την εξής διαφορά ως προς τον τρόπο επεξεργασίας τους από τον interpreter: Με τον απλό αλγόριθμο που αναφέραμε, για τον τρόπο υπολογισμού τέτοιων προγραμμάτων, το δεύτερο πρόγραμμα απαιτεί χρόνο λογαριθμικό στο μέγεθος της βάσης δεδομένων, ενώ το πρώτο απαιτεί χρόνο γραμμικό στο μέγεθος της βάσης δεδομένων. Θα ήταν επιθυμητό να είχαμε έναν τρόπο να λέμε ποιο από τα δύο ή περισσότερα προγράμματα είναι το «καλύτερο» από άποψη ταχύτητας επεξεργασίας. Αυτό το πρόβλημα δεν είναι λυμμένο στη γενική του περίπτωση, αλλά έχει γίνει σημαντική πρόοδος σε ειδικές περιπτώσεις, ενώ εξακολουθεί να είναι ένα από τα κυριότερα προβλήματα που απασχολεί τη σημερινή έρευνα στην περιοχή. Σ' αυτό το πλαίσιο, αναφέρουμε τις παρακάτω μεγάλες περιοχές γύρω από τις οποίες, κυρίως, κινείται η έρευνα σ' αυτό το πεδίο:

1) Μας ενδιαφέρει να ανακαλύψουμε συντακτικούς κανόνες (ή οτιδήποτε άλλους που να ελέγχονται εύκολα) οι οποίοι να μπορούν:

α) να βρούν αν δύο προγράμματα είναι ισοδύναμα,ή

β) να βρουν αν ένα δεδομένο πρόγραμμα είναι ισοδύναμο με κάποιο πρόγραμμα από μια κατηγορία προγραμμάτων (π.χ., αν είναι ισοδύναμο με ένα γραμμικό πρόγραμμα) [AC89], [ACY91] (Από την εκτενή βιβλιογραφία στο θέμα, θα αναφέρουμε μόνο μερικές από τις εργασίες που έχουν γίνει στο ΕΜΠ - για μια πλήρη βιβλιογραφία βλέπε [Uℓ89]).

2) Μία άλλη κατεύθυνση ερευνητικής προσπάθειας είναι η εύρεση απο-

δοτικότερων αλγορίθμων για τον interpreter, οι οποίοι μπορούν να επεξεργαστούν σε λογαριθμικό χρόνο μια περιορισμένη κατηγορία προγραμμάτων (π.χ. το πρώτο πρόγραμμα του παραδείγματός μας). Για να γίνει αυτό, πρέπει να εκμεταλλευτούμε την ειδικότερη δομή του προγράμματος. Στο παράδειγμά μας, το πρώτο πρόγραμμα εμπίπτει στην κατηγορία *γραμμικών* προγραμμάτων (δηλαδή, έχουν το πολύ μια αναδρομικά ορισμένη σχέση στο δεξιό μέρος κάθε κανόνα). Γνωρίζουμε αλγόριθμο (βλέπε και [AP87]), ο οποίος υπολογίζει σε λογαριθμικό χρόνο κάθε γραμμικό πρόγραμμα. Επίσης, γνωρίζουμε αλγόριθμους, οι οποίοι υπολογίζουν σε λογαριθμικό χρόνο μεγαλύτερες κατηγορίες προγραμμάτων (βλέπε [AP87]).

3) Οι δύο περισσότερο γνωστές τεχνικές επεξεργασίας λογικών προγραμμάτων, είναι:

α) Η τεχνική top-down, σύμφωνα με την οποία ξεκινάμε με το ζητούμενο και εξερευνούμε πιθανούς τρόπους υπολογισμού του. Στο παράδειγμά μας, πάλι, ο υπολογισμός του δεύτερου προγράμματος, θα γινόταν περίπου ως εξής: Για να υπάρχει διαδρομή από *το a* στο *b*, αρκεί να υπάρχει κάποιος κόμβος *c* ,τέτοιος ώστε να υπάρχει διαδρομή από το *a* στο *c* να υπάρχει διαδρομή από το *c* στο *b*. Στη συνέχεια, για να υπάρχει διαδρομή από το *a* στο *c*, αρκεί... κ.ο.κ.

β) Η τεχνική bottom-up, σύμφωνα με την οποία αρχίζουμε και υπολογίζουμε με τη σειρά όλα τα δυνατά «γεγονότα» τα οποία προκύπτουν από την εφαρμογή του προγράμματος στη βάση δεδομένων, και μετά ελέγχουμε αν το ζητούμενο είναι μέσα στα υπολογισθέντα. Στο παράδειγμά μας, αυτός ο τρόπος υπολογισμού ισοδυναμεί με τον υπολογισμό του μεταβατικού κλεισίματος του γράφου, και για τα δύο προγράμματα. Φαίνεται ότι οι δύο τεχνικές είναι το ίδιο αποδοτικές, αν υποθέσουμε ότι, προτού αυτές εφαρμοστούν, γίνεται μια αρχική επεξεργασία στα λογικά προγράμματα, η οποία μετατρέπει, συνήθως, τα προγράμματα σε άλλα, τα οποία δεν είναι εν γένει ισοδύναμα, αλλά δίνουν την ίδια απάντηση στη συγκεκριμένη ερώτηση. Χρειάζεται, ωστόσο, να αναπτύξουμε κριτήρια, τα οποία να αποφασίζουν ποια τεχνική επεξεργασίας είναι περισσότερο αποδοτική για ένα συγκεκριμένο πρόγραμμα. Επίσης, χρειάζεται να αναπτύξουμε αλγόριθμους, οι οποίοι να μετατρέπουν ένα λογικό πρόγραμμα σε ένα ισοδύναμο, στο οποίο θα εφαρμόσουμε μια από τις προηγούμενες τεχνικές, με τέτοιο τρόπο, ώστε η όλη διαδικασία να γίνει με αποδοτικό τρόπο [BR86].

5. Πειραματικά Συστήματα

Εμπορικά συστήματα με τα λειτουργικά χαρακτηριστικά που περιγράψαμε, δεν υπάρχουν σήμερα. Υπάρχουν, όμως, αρκετά «συστήματα γνώσεων», δηλαδή συστήματα τα οποία υποστηρίζουν μια «σχετικά» δηλωτική (declarative) γλώσσα, δηλαδή μια γλώσσα βασισμένη στη λογική (logic-based) ή βασισμένη σε κανόνες παραγωγής (rule-based). Αυτά τα συστήματα είναι γνωστά ως πεπειραμένα συστήματα (expert system shell) ή γλώσσες με κανόνες παραγωγής (production system languages) ή γλώσσες λογικού προγραμματισμού (logic programming languages). Συστήματα για βάσεις γνώσεων, δηλαδή, τα οποία παρέχουν επί πλέον τις ευκολίες που παρέχει ένα σύστημα για βάσεις δεδομένων, υπάρχουν αρκετά στο πειραματικό στάδιο. Θα αναφερθούμε σε τρία από αυτά στο υπόλοιπο αυτής της παραγράφου.

5.1. Το σύστημα NAIL!

Το σύστημα NAIL! (Not Another Implementation of Logic!), έχει αναπτυχθεί στο πανεπιστήμιο του Stanford [U+89]. Ο κύριος σκοπός του προγράμματος είναι να μελετηθούν τεχνικές βελτιστοποίησης για λογικές ερωτήσεις (logical queries). Αποτέλεσμα αυτής της μελέτης είναι ένα σύστημα, του οποίου η αρχιτεκτονική στηρίζεται σε ένα κλασικό σύστημα βάσεων δεδομένων, το SQL, το οποίο χρησιμοποιείται για να διεκπεραιώνει τις καθιερωμένες λειτουργίες ενός DBMS, όπως διατήρηση δεικτών (index maintenance). Αναμφίβολα, σε ένα εμπορικό σύστημα, αυτός δεν είναι ο καταλληλότερος τρόπος για να προσπελάσει κανείς μια βάση δεδομένων. Χρειάζεται ένα σύστημα με πιο ολοκληρωμένη και ενιαία αρχιτεκτονική.

Η κύρια ερευνητική προσπάθεια στο NAIL! εστιάστηκε στη βελτιστοποίηση του τρόπου με τον οποίο ο interpreter επεξεργάζεται μια ερώτηση. Αποτέλεσμα αυτής της έρευνας υπήρξε μια αρκετά πολύπλοκη διαδικασία, σύμφωνα με την οποία, η επεξεργασία μιας ερώτησης γίνεται ως εξής: Πρώτα το σύστημα διαλέγει μια *στρατηγική αποτίμησης, η οποία εξαρτάται με ειδικό τρόπο από τη μόρφη της ερώτησης. Π.χ., η ερώτηση S(5, x)* και η ερώτηση *S(18, x)* θα έχουν την ίδια στρατηγική αποτίμησης, διότι και στις δύο περιπτώσεις το κατηγόρημα της ερώτησης έχει στο πρώτο όρισμα, δεσμευμένη μεταβλητή και στο δεύτερο όρισμα ελεύθερη μεταβλητή. Το σύστημα αποφασίζει ποιά στρατηγική θα διαλέξει, θεωρώντας ένα σύνολο από «κανόνες σύλληψης» (capture rules). Κάθε τέτοιος κανόνας, αντιστοιχεί σε κάποιο διαφορετικό αλγόριθμο αποτίμησης. Η βασική ιδέα είναι ότι σε διαφορετικές μορφές ερωτήσεων, διαφορετικές τεχνικές είναι πιο αποδοτικές [APP89]. Η γλώσσα προσπέλασης στο NAIL! είναι η Datalog, που μπορεί να τη φανταστεί κανείς, συντακτικά, ως μια Prolog χωρίς συναρτησιακά σύμβολα, αλλά η σημαντική της (διαφορετική από αυτήν της Prolog) είναι η σημαντική του σταθερού σημείου (fixpoint). Στην παράγραφο 3, τα λογικά προγράμματα, που διατυπώνουν την ερώτηση στο παράδειγμά μας, είναι γραμμένα σε Datalog.

5.2. Το σύστημα LDL

Το σύστημα LDL (Logic Data Language), έχει αναπτυχθεί στο MCC στο Austin, Texas [NT88]. Η γενική δομή και οι επιδιώξεις του LDL είναι παρόμοιες με αυτές του NAIL!. Το LDL είναι μάλλον πιο φιλόδοξο, καθώς επιδιώκει να κατασκευάσει, τόσο ένα πλήρες σύστημα διαχείρισης βάσεων δεδομένων, όσο και ένα καλό επεξεργαστή λογικών προγραμμάτων.

Η γλώσσα προσπέλασης, η οποία επίσης ονομάζεται LDL, είναι παρόμοια με την Datalog του NAIL!, μόνο που η LDL επιδέχεται συναρτησιακά σύμβολα και επιτρέπει στις μεταβλητές και στα ορίσματα να παίρνουν σύνολα ως τιμές. Έτσι, η LDL υποστηρίζει πράξεις πάνω σε σύνολα ως εξής: Αν μια μεταβλητή, στο αριστερό μέρος ενός κανόνα (όπως αυτοί της παραγράφου 3) περικλείεται σε < >, τότε το όρισμα στη θέση του οποίου εμφανίζεται το <*X*>, παίρνει ως τιμή το σύνολο. Το σύνολο αυτό περιέχει όλες τις τιμές που αποδίδονται στο *X*, καθώς οι μεταβολές του κανόνα παίρνουν ό-

λες τις δυνατές τιμές οι οποίες κάνουν το δεξιό μέρος του κανόνα αληθές.

Επίσης, υπάρχουν ενσωματωμένες πράξεις, όπως:

1) Ένωση συνόλων, η οποία εκφράζεται με το κατηγόρημα *union* (A, B, C), το οποίο γίνεται αληθές μόνο τότε, όταν το σύνολο *C* είναι ίσο με την ένωση των συνόλων *A* και *B*.
2) Το κατηγόρημα *member (X, S)*, το οποίο γίνεται αληθές, μόνο τότε όταν το στοιχείο *X* ανήκει στο σύνολο *S*.
3) Η συνάρτηση *scons (X, S)*, η οποία δίνει ως αποτέλεσμα το σύνολο $S \cup \{X\}$.

Η αρχιτεκτονική του τμήματος του interpreter που κάνει βελτιστοποίηση στις ερωτήσεις του LDL, διαφέρει ουσιαστικά από την αντίστοιχη του NAIL!, αν και, πολύ συχνά, το αποτέλεσμα είναι το ίδιο. Οι τεχνικές βελτιστοποίησης που χρησιμοποιούνται (οι περισσότερες είναι παρόμοιες με αυτές του NAIL!) αντλούνται, γενικά, από τις ίδιες βιβλιογραφικές αναφορές.

5.3. Το σύστημα POSTGRES

Το σύστημα POSTGRES έχει αναπτυχθεί στο Berkeley και είναι διάδοχος του INGRES [We88]. Σκοπός του είναι να κατασκευάσει ένα σύστημα

για βάσεις γνώσεων, κάνοντας απλές και φυσικές προεκτάσεις στο υπάρχον σύστημα INGERS. Η γλώσσα του, POSTQUEL, είναι σε μεγάλο βαθμό προέκταση της QUEL, της γλώσσας του INGRES. Η POSTQUEL, για να υποστηρίξει αναδρομή, επιτρέπει στον χρήστη να καθορίσει να επαναληφθεί μια εντολή, τόσες φορές όσες είναι απαραίτητο για να φθάσει η σχέση στο σταθερό σημείο (να μην υπάρχει, δηλαδή, καμία αλλαγή στο σύνολο των κ-αδων που αποτελούν τη σχέση, από τη μια επανάληψη στην επόμενη).

Η POSTQUEL επιτρέπει να ορίζουμε τους συνηθισμένους τύπους, δηλαδή ακέραιους, πραγματικούς, συμβολοσειρές σταθερού μήκους. Επίσης, επιτρέπει να ορίσουμε πίνακες με οποιονδήποτε αριθμό διαστάσεων και με οποιοδήποτε μήκος στην κάθε διάσταση. Το πιο σημαντικό χαρακτηριστικό της POSTQUEL, όμως, είναι το ότι επιτρέπει να ορίσουμε τύπους που παίρνουν ως τιμές, ολόκληρες διαδικασίες. Έτσι, όταν ένα πεδίο έχει τύπο postquel, έχει τιμή μια σειρά από εντολές της POSTQUEL. Όταν χρειαστεί ο interpreter να αποκτήσει την τιμή μιας συνιστώσας τύπου postquel, εκτελεί αυτές τις εντολές και το σύνολο των κ-αδων που δημιουργούνται, είναι η τιμή αυτής της συνιστώσας. Μπορεί, δηλαδή, να θεωρηθεί ότι η τιμή της συνιστώσας είναι μια ολόκληρη σχέση. Ακόμα πιο ισχυρό είναι ένα πεδίο με τύπο procedure. Η τιμή του είναι μια procedure γραμμένη σε μια γενική γλώσσα, π.χ. την C. Εδώ, θα παρατηρήσει κανείς ότι αυτοί οι πολύ ισχυροί τύποι δεν αφήνουν πολλά περιθώρια για βελτιστοποίηση, καθώς ένα πεδίο με τύπο postquel ή procedure μπορεί να έχει διαφορετικό κομμάτι κώδικα σε κάθε κ-αδα. (κ-αδα ονομάζουμε ένα στοιχείο μιας σχέσης. Μια σχέση μπορούμε να τη φανταστούμε, σαν ένα υποσύνολο του καρτεσιανού γινομένου κ συνόλων. Πεδίο ονομάζουμε μια από τις κ συνιστώσες της κ-αδας). Ένα πιο περιοριστικό, αλλά ακόμα αρκετά ισχυρό σχήμα είναι να χρησιμοποιούμε το ίδιο κομμάτι κώδικα σε κάθε κ-αδα με διαφορετικές μόνο παραμέτρους, σε διαφορετικές κ-αδες (το παράδειγμα που ακολουθεί, διασαφηνίζει αυτήν την ιδέα).

Μια πολύ σημαντική χρησιμοποίηση του τύπου postquel είναι να παρουσιάζει δομές που υπάρχουν σε αντικειμενοστρεφή συστήματα, τις οποίες η POSTQUEL δεν υποστηρίζει, αφού βασίζεται στο σχεσιακό μοντέλο. Για παράδειγμα, ας υποθέσουμε ότι θέλουμε να παραστήσουμε ένα σχήμα το οποίο περιέχει *είδη, ζευγάρια είδος-ποσότητα* και *παραγγελίες*. Ένα είδος αποτελείται από το όνομά του και τον αύξοντα αριθμό του - αυτή η κλάση είναι στοιχειώδης και μοιάζει σαν μια συνηθισμένη σχέση. Ένα ζευγάρι είδος-ποσότητα, είναι ένα αντικείμενο με δύο συνιστώσες, η πρώτη είναι ένα αντικείμενο από την κατηγορία *είδος* και η δεύτερη είναι ένας ακέραιος αριθμός, που δίνει την ποσότητα. Μια παραγγελία είναι ένα σύνολο από *ζευγάρια είδος-ποσότητα* μαζί με έναν αριθμό παραγγελίας. Η σχέση που αντιστοιχεί στη κατηγορία *είδος*, δηλώνεται στην POSTQUEL με τον κλασικό τρόπο ως σχέση, χωρίς να απαιτεί κανένα από τους ισχυρούς τύπους. Γράφουμε:

```
create ΕΙΔΟΣ(ΟΝΟΜΑ = char[20],
             ΑΡΙΘΜΟΣ = integer)
```

Στη συνέχεια, μπορούμε να ορίσουμε τη σχέση ΖΕΥΓΗ-ΕΙΔΟΣΠΟΣΟΤΗΤΑ (ΑΑΡΙΘΜΟΣ, ΕΙΔΟΣΠΟΣΟΤΗΤΑ) με την ακόλουθη εντολή της POSTQUEL.:

```
create ΖΕΥΓΗ-ΕΙΔΟΣΠΟΣΟΤΗΤΑ

(ΑΑΡΙΘΜΟΣ = integer,
ΕΙΔΟΣ = postquel,  ΠΟΣΟΤΗΤΑ =
integer)
```

Η τιμή του πεδίου ΕΙΔΟΣ είναι ένα κομμάτι κώδικα POSTQUEL. Κατ' αρχήν, οποιοδήποτε κομμάτι κώδικα θα μπορούσε να υπάρχει σε κάθε διαφο-

Παράδειγμα: Ένα χρήσιμο κατηγόρημα θα ήταν το *disjoint (S, T)*, το οποίο θα εκφράζει το γεγονός ότι δύο σύνολα είναι ξένα μεταξύ τους. Αυτό ορίζεται με τον ακόλουθο τρόπο, βάσει των ενσωματωμένων κατηγορημάτων και συναρτήσεων:

disjoint ({ }, S)

disjoint (scons (X, T), S):

– ~member(X,S),disjoint(T,S)

Το σύμβολο ~ υποδηλώνει άρνηση δήλαδη *~member (X, S)* σημαίνει ότι το *X* δεν ανήκει στο *S*. Ένα άλλο χρήσιμο κατηγόρημα είναι το *inter (S, T, U)* ,το οποίο γίνεται αληθές μόνο όταν το*U* είναι η τομή των *S* και *T*. Αυτό ορίζεται ως εξής:

inter (S, T, { }): – disjoint (S, T)

inter (scons (X, S), scons (X, T), scons(X, U))

: – inter (S, T, U)

Εδώ, ο πρώτος κανόνας λέει ότι η τομή δύο ξένων μεταξύ τους συνόλων είναι το κενό σύνολο. Ο δεύτερος κανόνας λέει ότι μπορούμε να υπολογίσουμε την τομή δύο συνόλων, αν βρούμε ένα κοινό στοιχείο, το απαλείψουμε και από τα δύο σύνολα, υπολογίσουμε την τομή των υπολοίπων και, τέλος, προσθέσουμε το απαληφθέν στοιχείο σ' αυτή την τομή.

ρετική κ-αδα της σχέση ΖΕΥΓΗ-ΕΙΔΟΣΠΟΣΟΤΗΤΑ (δηλαδή, τελείως διαφορετικό σε κάθε κ-αδα). Στην πράξη, θέλουμε, σε κάθε κ-αδα να υπάρχει το ίδιο κομμάτι κώδικα, το οποίο θα επιστρέφει μια συγκεκριμένη κ-αδα από τη σχέση ΕΙΔΟΣ. Αν υποθέσουμε ότι το ΟΝΟΜΑ είναι ένα κλειδί για τη σχέση ΕΙΔΟΣ, θα μπορούσαμε σε κάθε κ-αδα της σχέσης ΖΕΥΓΗ-ΕΙΔΟΣΠΟΣΟΤΗΤΑ, να τοποθετήσουμε το εξής κομμάτι κώδικα (αυτός θα χρησιμοποιηθεί σε κ-αδες όπου το είδος είναι «ψωμί», στις άλλες κ-αδες θα έχει αντικατασταθεί το «ψωμί» με το αντίστοιχο όνομα):

```
retreive (i.ΟΝΟΜΑ, i.ΑΡΙΘΜΟΣ)
    from i in ΕΙΔΟΣ
    where i.ΟΝΟΜΑ = «ψωμί»
```

Έτσι, η τιμή της συνιστώστας ΕΙΔΟΣ στη σχέση ΖΕΥΓΗ-ΕΙΔΟΣΠΟΣΟΤΗΤΑ, σε μια συγκεκριμένη κ-αδα, θα είναι (ψωμί, 47) (αν υποθέσουμε ότι ο αριθμός είναι 47). Τώρα, αν κάποτε αλλάξει ο αριθμός και γίνει 48, στην επόμενη κλήση της ίδιας συνιστώσας της ίδιας κ-αδας, η τιμή της θα είναι (ψωμί, 48), δηλαδή η αλλαγή διαδίδεται αυτόματα, χωρίς να χρειάζεται ιδιαίτερη ενημέρωση και σ' αυτό το σημείο της σχέσης ΖΕΥΓΗ-ΕΙΔΟΣΠΟΣΟΤΗΤΑ.

Στην υλοποίηση της POSTQUEL, χρησιμοποιούνται, με έξυπνο τρόπο, μερικές τεχνικές από τις κλασικές βάσεις δεδομένων, για να υποστηρίξουν μια γλώσσα προσπέλασης με λογικούς κανόνες. Για παράδειγμα, ένα πολύ σημαντικό χαρακτηριστικό στην αρχιτεκτονική της POSTQUEL είναι ότι οι μηχανισμοί «κλειδώματος» του INGERS, που χρησιμοποιούνται για να υποστηρίζουν «συγχρονισμό» (concurrency), μπορούν πολύ φυσικά να επεκταθούν, ώστε να υποστηρίζουν τις καινούργιες πράξεις που εμφανίζονται στην POSTQUEL.

6. Συμπεράσματα

Τα πρώτα συστήματα βάσεων δεδομένων, που εμφανίστηκαν τη δεκαετία του '60, ήταν βασισμένα, είτε στο ιεραρχικό μοντέλο ή στο μοντέλο δικτύου, και οι γλώσσες επικοινωνίας με το χρήστη ήταν «διαδικασιακές» (procedural). Στην δεκαετία του '70, εμφανίστηκαν τα πρώτα συστήματα που ήταν βασισμένα στο σχεσιακό μοντέλο. Αυτά τα τελευταία, χρησιμοποιούσαν «δηλωτικές» (declarative) γλώσσες επικοινωνίας με το χρήστη, και αυτή την εποχή αναπτύχθησαν καλές τεχνικές βελτιστοποιήσης του τρόπου επεξεργασίας των ερωτήσεων από τον interpreter, αλλά, ακόμη, η γλώσσα προσπέλασης ήταν διαφορετική από την γλώσσα του συστήματος. Στη δεκαετία του '80, εμφανίστηκαν τα αντικειμενοστρεφή (object-oriented) συστήματα, τα πρώτα συστήματα που διέθεταν ενιαία γλώσσα προσπέλασης και γλώσσα συστήματος, αλλά αυτή η γλώσσα ήταν «λιγότερο» δηλωτική από τη γλώσσα προσπέλασης των σχεσιακών συστημάτων. Τα συστήματα που αναφέραμε ως βάσεις γνώσεων, διαθέτουν ενιαία γλώσσα συστήματος και προσπέλασης, η οποία συγχρόνως είναι δηλωτική. Πολλοί πιστεύουν ότι, τη δεκαετία του '90, αυτά τα συστήματα θα είναι το αντίπαλο δέος στα αντικειμενοστρεφή (object-oriented) συσήματα. Ωστόσο, φαίνεται ότι υπάρχει πολύ δουλειά που χρειάζεται να γίνει προς την κατεύθυνση της εύρεσης τεχνικών βελτιστοποίησης του τρόπου επεξεργασίας των ερωτήσεων, προτού καταστεί δυνατό να υλοποιηθεί ένα τέτοιο εμπορικό σύστημα.

Βιβλιογραφία

[ACY91] Afrati F., Cosmadakis S., Yannakakis M., «Datalog vs. Polynomial Time», *Proceedings of the 10th ACM PODS, Symposium on the Principles of Database Systems,* 1991, pp. 13-25.

[AC89] F. Afrati and S.S. Cosmadakis, «Expressiveness of restricted recursive queries», *Proc. 21st ACM Symp. on Theory of Computing,* 1989, pp. 113-126.

[AP87] Afrati F., Papadimitriou C.H.: The parallel complexity of simple chain queries. *Proc. 6th ACM Symp. on Principles of Database Systems,* San Diego, 1987, pp. 210-213, και πρόκειται να δημοσιευθεί στο *J. ACM.*

[APP89] Afrati F., Papadimitriou C.H., Papageorgiou, G., Roussou A., Sagiv Y., Ullman, J.D., «On the Convergence of Query Evaluation», *Journal of Computer and Systems Sciences,* Vol. 38 No 2, April 1989 (special issue), pp. 341-359.

[BR86] Bancilhon F., Ramakrishnan R.: An amateur's introduction to recursive query processing strategies. *Proc. ACM Conf. on Management of Data,* Washington, 1986, pp. 16-52.

[NT88] Naqvi S.A. and Tsur S.: *A Logical Language for Data and Knowledge ases,* Computer science press, Rckville, Md 1988.

[Ul89] Ullman J.D.: *Database and Knowledge-Base Systems.* Volumes I and II, Computer Science Press, 1989.

[We88] Wensel S. : *The Postgres reference manual,* UCB/ERL M88/20, Dept. of EECS, Univ. of California, Berkekey 1988.