Πρόλογος στη δεύτερη έκδοση

Η πρακτική της κατασκευής μεταγλωττιστών μεταβάλλεται διαρκώς, εν μέρει διότι μεταβάλλεται η σχεδίαση των επεξεργαστών και των συστημάτων. Για παράδειγμα, όταν ξεκινήσαμε να γράφουμε τη *Σχεδίαση και κατασκευή μεταγλωττιστών* (ΣΚΜ) το 1998, μερικοί από τους συναδέλφους μας εξέφρασαν αμφιβολίες για το αν ήταν φρόνιμο να συμπεριλάβουμε ένα κεφαλαίο για τον χρονοπρογραμματισμό οδηγιών, διότι η εκτός σειράς εκτέλεση απειλούσε να ακυρώσει σε μεγάλο βαθμό την αξία του χρονοπρογραμματισμού. Σήμερα, καθώς η δεύτερη έκδοση βρίσκεται καθ' οδόν προς το τυπογραφείο, η εξάπλωση των πολυπύρηνων επεξεργαστών και η πίεση για περισσότερους πυρήνες έχει καταστήσει ξανά ελκυστικές τις εντός σειράς διοχετεύσεις εκτέλεσης, διότι τα μικρότερα αποτυπώματά τους επιτρέπουν στους σχεδιαστές να τοποθετούν περισσότερους πυρήνες στο ίδιο τσιπ. Ο χρονοπρογραμματισμός οδηγιών θα παραμείνει σημαντικός για το κοντινό μέλλον.

Την ίδια στιγμή, η κοινότητα όσων ασχολούνται με την κατασκευή μεταγλωττιστών συνεχίζει να αναπτύσσει νέες ιδέες και αλγορίθμους και να ανακαλύπτει εκ νέου παλαιότερες τεχνικές που ήταν αποτελεσματικές αλλά εν πολλοίς ξεχασμένες. Πρόσφατα ερευνητικά αποτελέσματα έχουν προκαλέσει ενθουσιασμό γύρω από τη χρήση έγχοδρων γραφημάτων στη δέσμευση καταχωρητών (βλ Ενότητα 13.5.2). Οι δουλειές αυτές υπόσχονται να απλουστεύσουν κάποιες πλευρές των εκχωρητών χρωματισμού γραφήματος. Ο αλγόριθμος του Brzozowski είναι μια τεχνική ελαχιστοποίησης ΑΠΑ που ανάγεται στις αρχές τις δεκαετίας του 1960, αλλά δεν διδασκόταν στα μαθήματα μεταγλωττιστών για δεκαετίες (βλ Ενότητα 2.6.2). Μας επιτρέπει να μεταβούμε με εύκολο τρόπο από μια υλοποίηση του αλγορίθμου υποσυνόλου σε μια υλοποίηση που ελαχιστοποιεί τα ΑΠΑ. Ένα σύγχρονο μάθημα πάνω στην κατασκευή μεταγλωττιστών θα μπορούσε να περιλαμβάνει και τις δύο αυτές ιδέες.

Με ποιον τρόπο θα πρέπει, επομένως, να δομήσουμε ένα μάθημα κατασκευής μεταγλωττιστών ώστε να προετοιμάζει τους φοιτητές για να εισέλθουν σε αυτό το συνεχώς μεταβαλλόμενο πεδίο; Πιστεύουμε ότι το μάθημα θα πρέπει να προσφέρει σε κάθε φοιτητή το σύνολο των βασικών δεξιοτήτων που θα χρειαστεί για να κατασκευάζει νέα μέρη ενός μεταγλωττιστή και για να τροποποιεί υπάρχοντα. Οι φοιτητές πρέπει να κατανοήσουν και γενικές έννοιες, όπως τη συνεργασία μεταξύ μεταγλωττιστή, συνδέτη, φορτωτή και λειτουργικού συστήματος που ενσωματώνονται στη σύμβαση σύνδεσης, και ψιλές λεπτομέρειες, όπως τον τρόπο με τον οποίο ο συγγραφέας του μεταγλωττιστή θα μπορούσε να μειώσει τον συνολικό χώρο που καταλαμβάνει ο κώδικας αποθήκευσης καταχωρητών σε κάθε κλήση διαδικασίας.

ΑΛΛΑΓΕΣ ΣΤΗ ΔΕΥΤΕΡΗ ΕΚΔΟΣΗ

Στη δεύτερη έκδοση της *Σχεδίασης και κατασκευής μεταγλωττιστών* (ΣΚΜ2ε) παρουσιάζονται και οι δύο οπτικές γωνίες: μια γενική, υψηλού επιπέδου εικόνα των προβλημάτων που εμφανίζονται στην κατασκευή μεταγλωττιστών και λεπτομερείς αναλύσεις εναλλακτικών αλγορίθμων. Καθώς προετοιμάζαμε την ΣΚΜ2ε, επικεντρωθήκαμε στη χρηστικότητα του βιβλίου, τόσο ως εγχειριδίου όσο και ως βιβλίου αναφοράς για επαγγελματίες. Ειδικότερα,

- Βελτιώσαμε τη ροή των ιδεών για να βοηθήσουμε τον φοιτητή που διαβάζει τα κεφάλαια του βιβλίου με τη σειρά. Οι εισαγωγές των κεφαλαίων εξηγούν τον σκοπό κάθε κεφαλαίου, παρουσιάζουν τις βασικές έννοιες και δίνουν μια υψηλού επιπέδου επισκόπηση της ύλης του κεφαλαίου. Επεξεργαστήκαμε ξανά τα παραδείγματα ώστε να υπάρχει συνέχεια μεταξύ των κεφαλαίων. Επιπλέον, κάθε κεφάλαιο ξεκινά με μια περίληψη και ένα σύνολο από λέξεις-κλειδιά ώστε να διευκολύνεται ο χρήστης που χρησιμοποιεί την ΣΚΜ2ε ως βιβλίο αναφοράς.

- Προσθέσαμε ανασκοπήσεις και ερωτήματα επανάληψης στο τέλος κάθε μεγάλης ενότητας. Τα ερωτήματα επανάληψης επιτρέπουν στον αναγνώστη να ελέγχει γρήγορα αν έχει καταλάβει ή όχι τα βασικά σημεία της ενότητας.
- Μετακινήσαμε τους ορισμούς των βασικών όρων στο περιθώριο, δίπλα στην παράγραφο όπου ορίζονται και αναλύονται για πρώτη φορά.
- Αναθεωρήσαμε εκτενώς την ύλη πάνω στη βελτιστοποίηση ώστε να καλύπτει εκτενέστερα τις δυνατότητες που υπάρχουν για την κατασκευή βελτιστοποιητικών μεταγλωττιστών.

Σήμερα, η ανάπτυξη μεταγλωττιστών επικεντρώνεται στη βελτιστοποίηση και την παραγωγή κώδικα. Ένας νεοπροσλαμβανόμενος συγγραφέας μεταγλωττιστή είναι πολύ πιο πιθανό να μεταφέρει μια γεννήτρια κώδικα σε έναν νέο επεξεργαστή ή να τροποποιήσει μια βελτιστοποιητική διέλευση, παρά να γράψει έναν σαρωτή ή έναν συντακτικό αναλυτή. Ένας επιτυχημένος συγγραφέας μεταγλωττιστών πρέπει να είναι εξοικειωμένος με τις τεχνικές που αποτελούν τις τρέχουσες βέλτιστες πρακτικές στη βελτιστοποίηση, όπως είναι η κατασκευή της μορφής στατικής μοναδικής τιμοδοσίας, και στην παραγωγή κώδικα, όπως είναι η λογισμική διοχέτευση. Πρέπει επίσης να διαθέτει το γνωστικό υπόβαθρο και την οξυδέρκεια που θα του επιτρέψουν να κατανοήσει τις νέες τεχνικές που θα εμφανιστούν στο μέλλον. Τέλος, πρέπει να έχει κατανοήσει αρκετά καλά τις τεχνικές της σάρωσης, της συντακτικής ανάλυσης και της σημασιολογικής επεξεργασίας, ώστε να μπορεί να κατασκευάσει ή να τροποποιήσει ένα μπροστινό άκρο.

Στόχος μας κατά τη συγγραφή της SKM2ε ήταν να φτιάξουμε ένα κείμενο και ένα μάθημα που να εκθέτει τους φοιτητές στα κρίσιμα ζητήματα των σύγχρονων μεταγλωττιστών και να τους παρέχει το γνωστικό υπόβαθρο που απαιτείται για την αντιμετώπιση αυτών των προβλημάτων. Από την πρώτη έκδοση, διατηρήσαμε τη βασική ισορροπία της ύλης. Τα μπροστινά άκρα είναι εμπορικά προϊόντα· μπορούν να αγοραστούν από κάποιον αξιόπιστο προμηθευτή ή να προσαρμοστούν από κάποιο από τα πολλά συστήματα ανοιχτού κώδικα. Την ίδια στιγμή, οι βελτιστοποιητές και οι γεννήτριες κώδικα κατασκευάζονται ειδικά για συγκεκριμένους επεξεργαστές και, μερικές φορές, για μεμονωμένα μοντέλα, διότι η επίδοση εξαρτάται σημαντικά από συγκεκριμένες χαμηλού επιπέδου λεπτομέρειες του παραγόμενου κώδικα. Αυτά τα πράγματα επηρεάζουν τον τρόπο με τον οποίο κατασκευάζουμε σήμερα τους μεταγλωττιστές· θα πρέπει να επηρεάζουν και τον τρόπο με τον οποίο διδάσκουμε την κατασκευή μεταγλωττιστών.

ΟΡΓΑΝΩΣΗ

Στην SKM2ε, η ύλη διαιρείται σε τέσσερα σχεδόν ίσα τμήματα:

- Στην πρώτη μεγάλη ενότητα, στα Κεφάλαια 2 ώς 4, καλύπτεται η σχεδίαση του μπροστινού άκρου ενός μεταγλωττιστή και η σχεδίαση και κατασκευή εργαλείων κατασκευής μπροστινών άκρων.
- Στη δεύτερη μεγάλη ενότητα, στα Κεφάλαια 5 ώς 7, εξετάζεται η απεικόνιση του πηγαίου κώδικα στην ενδιάμεση μορφή του μεταγλωττιστή – δηλαδή, σε αυτά τα κεφάλαια εξετάζεται το είδος κώδικα που παράγει το μπροστινό άκρο για τον βελτιστοποιητή και το πίσω άκρο.
- Στην τρίτη μεγάλη ενότητα, στα Κεφάλαια 8 ώς 10, εισάγεται το αντικείμενο της βελτιστοποίησης κώδικα. Στο Κεφάλαιο 8 παρουσιάζεται μια επισκόπηση της βελτιστοποίησης. Στα Κεφάλαια 9 και 10 εξετάζονται σε μεγαλύτερο βάθος η ανάλυση και ο μετασχηματισμός· αυτά τα δύο κεφάλαια συχνά παραλείπονται από ένα προπτυχιακό μάθημα.
- Στην τελευταία ενότητα, στα Κεφάλαια 11 ώς 13, επικεντρωνόμαστε στους αλγορίθμους που χρησιμοποιούνται στο πίσω άκρο του μεταγλωττιστή.

Η ΤΕΧΝΗ ΚΑΙ Η ΕΠΙΣΤΗΜΗ ΤΗΣ ΜΕΤΑΓΛΩΤΤΙΣΗΣ

Η παράδοση της κατασκευής μεταγλωττιστών περιλαμβάνει συναρπαστικές ιστορίες επιτυχίας σχετικά με την εφαρμογή της θεωρίας στην πράξη και ταπεινωτικές ιστορίες σχετικά με τα όρια των όσων μπορούμε να κάνουμε. Στην πλευρά των επιτυχιών, οι σύγχρονοι σαρωτές κατασκευάζονται με εφαρμογή της θεωρίας κανονικών γλωσσών στην αυτόματη κατασκευή αναγνωριστών. Οι συντακτικοί αναλυτές LR χρησιμοποιούν τις ίδιες τεχνικές για να υλοποιήσουν τη διαδικασία εντοπισμού λαβών που κατευθύνει τους συντακτικούς αναλυτές ολίσθησης-αναγωγής. Η ανάλυση ροής δεδομένων εφαρμόζει τη θεωρία δικτυωμάτων στην ανάλυση προγραμμάτων με έξυπνους και χρήσιμους τρόπους. Οι προσεγγιστικοί αλγόριθμοι που χρησιμοποιούνται στην παραγωγή κώδικα παράγουν καλές λύσεις σε πολλές εκφάνσεις πραγματικά δύσκολων προβλημάτων.

Από την άλλη πλευρά, η κατασκευή μεταγλωττιστών εμφανίζει σύνθετα προβλήματα για τα οποία δεν υπάρχουν καλές λύσεις. Το πίσω άκρο ενός μεταγλωττιστή για έναν σύγχρονο επεξεργαστή υπολογίζει μια προσεγγιστική λύση για δύο ή περισσότερα αλληλεπιδρώντα NP-πλήρη προβλήματα (χρονοπρογραμματισμός οδηγιών, δέσμευση καταχωρητών και, πιθανώς, τοποθέτηση οδηγιών και δεδομένων). Αυτά τα NP-πλήρη προβλήματα, ωστόσο, φαντάζουν εύκολα συγκρινόμενα με προβλήματα όπως η αλγεβρική επανασυσχέτιση εκφράσεων (βλ., για παράδειγμα, Σχήμα 7.1). Αυτό το πρόβλημα επιδέχεται τεράστιο αριθμό λύσεων· τα πράγματα γίνονται ακόμη χειρότερα, καθώς η επιθυμητή λύση εξαρτάται από τα συμφραζόμενα τόσο στον μεταγλωττιστή όσο και στον κώδικα της εφαρμογής. Καθώς ο μεταγλωττιστής υπολογίζει προσεγγιστικές λύσεις για τέτοιου είδους προβλήματα, αντιμετωπίζει περιορισμούς που αφορούν τον χρόνο μεταγλώττισης και τη διαθέσιμη μνήμη. Ένας καλός μεταγλωττιστής συνδυάζει επιδέξια τη θεωρία, την πρακτική γνώση, τις αρχές σχεδίασης και την εμπειρία.

Αν ανοίξετε έναν σύγχρονο βελτιστοποιητικό μεταγλωττιστή θα βρείτε μεγάλη ποικιλία τεχνικών. Οι μεταγλωττιστές χρησιμοποιούν ευρετικές τεχνικές άπληστης αναζήτησης που διερευνούν μεγάλους χώρους λύσεων και αιτιοκρατικά πεπερασμένα αυτόματα που αναγνωρίζουν λέξεις της εισόδου. Χρησιμοποιούν αλγορίθμους σταθερού σημείου για να εξάγουν συμπεράσματα σχετικά με τη συμπεριφορά του προγράμματος και απλούς αποδείκτες θεωρημάτων και αλγεβρικούς απλοποιητές για να προβλέπουν τις τιμές εκφράσεων. Οι μεταγλωττιστές εκμεταλλεύονται γρήγορους αλγορίθμους συμμόρφωσης μορφοτύπων για να απεικονίζουν αφηρημένους υπολογισμούς σε πράξεις επιπέδου μηχανής. Χρησιμοποιούν γραμμικές διοφαντικές εξισώσεις και αριθμητική Pressburger για να αναλύουν αριθμοδείκτες θέσης συστοιχιών. Τέλος, οι μεταγλωττιστές χρησιμοποιούν ένα μεγάλο σύνολο κλασικών αλγορίθμων και δομών δεδομένων, όπως πίνακες διασποράς, αλγόριθμους γραφημάτων και υλοποιήσεις αραιών συνόλων.

Στην SKM2ee, προσπαθήσαμε να αποτυπώσουμε τόσο την τέχνη όσο και την επιστήμη της κατασκευής μεταγλωττιστών. Το βιβλίο περιλαμβάνει μια αρκετά μεγάλη επιλογή ύλης για να δείξει στον αναγνώστη ότι κάθε απόφαση έχει πραγματικά πλεονεκτήματα και μειονεκτήματα και ότι οι επιπτώσεις κάθε σχεδιαστικής απόφασης μπορεί να είναι και δυσδιάκριτες και εκτενείς. Την ίδια στιγμή, από την SKM2e παραλείψαμε κάποιες τεχνικές που έχουν αποτελέσει επί μακρόν μέρος των προπτυχιακών μαθημάτων κατασκευής μεταγλωττιστών, αλλά η σημασία τους έχει περιοριστεί λόγω αλλαγών στην αγορά, στην τεχνολογία των γλωσσών και των μεταγλωττιστών και στη διαθεσιμότητα των εργαλείων.

ΠΡΟΣΕΓΓΙΣΗ

Η κατασκευή ενός μεταγλωττιστή είναι μια άσκηση σχεδίασης ενός ολοκληρωμένου συστήματος. Ο συγγραφέας του μεταγλωττιστή πρέπει να επιλέξει μια διαδρομή μέσα σε έναν χώρο σχεδίασης

γεμάτο διαφορετικές εναλλακτικές δυνατότητες, καθεμία από τις οποίες έχει τα δικά της κόστη, πλεονεκτήματα και πολυπλοκότητα. Κάθε απόφαση επηρεάζει τον μεταγλωττιστή που προκύπτει. Η ποιότητα του τελικού προϊόντος εξαρτάται από τη λήψη εμπεριστατωμένων αποφάσεων σε κάθε βήμα αυτής της διαδρομής.

Επομένως, για πολλές από τις σχεδιαστικές αποφάσεις που αφορούν έναν μεταγλωττιστή δεν υπάρχει μοναδική απάντηση. Ακόμα και για τα «καλά κατανοητά» και τα «λυμένα» προβλήματα, μικροδιαφορές στη σχεδίαση και την υλοποίηση επηρεάζουν τη συμπεριφορά του μεταγλωττιστή και την ποιότητα του κώδικα που παράγει. Κάθε απόφαση επηρεάζεται από πολλούς παράγοντες. Για παράδειγμα, η επιλογή της ενδιάμεσης αναπαράστασης ενός μεταγλωττιστή επηρεάζει ριζικά τον υπόλοιπο μεταγλωττιστή, από τις χρονικές και χωρικές απαιτήσεις έως την ευκολία εφαρμογής των διάφορων αλγορίθμων. Η απόφαση, ωστόσο, συχνά λαμβάνεται βιαστικά. Στο Κεφάλαιο 5 εξετάζεται ο χώρος των ενδιάμεσων αναπαραστάσεων και μερικά από τα ζητήματα που θα πρέπει να εξετάζονται κατά την επιλογή μίας εξ αυτών. Θίγουμε αυτό το ζήτημα σε διάφορα σημεία του βιβλίου – και άμεσα μέσα στο κείμενο και έμμεσα στις ασκήσεις.

Στην SKM2ε εξετάζεται ο χώρος σχεδίασης και διερευνάται το βάθος των προβλημάτων και το εύρος των δυνατών λύσεων. Παρουσιάζονται τρόποι με τους οποίους έχουν λυθεί αυτά τα προβλήματα, μαζί με τους περιορισμούς που έκαναν αυτές τις λύσεις ελκυστικές. Οι συγγραφείς μεταγλωττιστών πρέπει να κατανοούν και τα προβλήματα και τις λύσεις τους, καθώς και την επίδραση αυτών των αποφάσεων σε άλλες πλευρές της σχεδίασης ενός μεταγλωττιστή. Μόνο τότε μπορούν να κάνουν εμπεριστατωμένες και έξυπνες επιλογές.

■ ΦΙΛΟΣΟΦΙΑ

Σε αυτό το βιβλίο αποτυπώνεται η φιλοσοφία μας για την κατασκευή μεταγλωττιστών, όπως αναπτύχθηκε κατά τη διάρκεια εικοσιπέντε και πλέον ετών έρευνας, διδασκαλίας και πρακτικής εμπειρίας. Για παράδειγμα, οι ενδιάμεσες αναπαραστάσεις θα πρέπει να εκθέτουν εκείνες τις λεπτομέρειες που έχουν σημασία στον τελικό κώδικα· αυτή η πεποίθηση οδηγεί σε μεροληψία υπέρ των χαμηλού επιπέδου αναπαραστάσεων. Οι τιμές θα πρέπει να διαμένουν σε καταχωρητές μέχρι ο εκχωρητής να ανακαλύψει ότι δεν μπορεί να τις διατηρήσει εκεί· αυτή η πρακτική οδηγεί σε παραδείγματα που χρησιμοποιούν εικονικούς καταχωρητές και αποθηκεύουν τιμές στη μνήμη μόνο όταν αυτό δεν μπορεί να αποφευχθεί. Κάθε μεταγλωττιστής θα πρέπει να πραγματοποιεί βελτιστοποίηση· αυτό απλουστεύει τον υπόλοιπο μεταγλωττιστή. Επιλέξαμε την ύλη και την παρουσίασή της με βάση τις εμπειρίες μας όλα αυτά τα χρόνια.

■ ΛΙΓΑ ΛΟΓΙΑ ΓΙΑ ΤΙΣ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΑΣΚΗΣΕΙΣ

Ένα μάθημα κατασκευής μεταγλωττιστών προσφέρει την ευκαιρία να εξεταστούν ζητήματα σχεδίασης λογισμικού στα πλαίσια μιας συγκεκριμένης εφαρμογής – μιας εφαρμογής τις βασικές λειτουργίες της οποίας κατανοούν καλά όλοι οι φοιτητές με το απαραίτητο γνωστικό υπόβαθρο για ένα μάθημα κατασκευής μεταγλωττιστών. Στις περισσότερες εκδοχές αυτού του μαθήματος, οι προγραμματιστικές ασκήσεις παίζουν σημαντικό ρόλο.

Έχουμε διδάξει αυτό το μάθημα σε εκδοχές όπου οι φοιτητές κατασκευάζουν έναν απλό μεταγλωττιστή από την αρχή έως το τέλος – ξεκινώντας με έναν αυτόματα παραγόμενο σαρωτή και συντακτικό αναλυτή και τελειώνοντας με μια γεννήτρια κώδικα για κάποιο απλοποιημένο σύνολο οδηγιών RISC. Έχουμε διδάξει αυτό το μάθημα σε εκδοχές όπου οι φοιτητές γράφουν προγράμματα που αντιμετωπίζουν αυτόνομα επιμέρους προβλήματα, όπως είναι η δέσμευση καταχωρητών και

ο χρονοπρογραμματισμός οδηγιών. Η επιλογή των προγραμματιστικών ασκήσεων εξαρτάται σε μεγάλο βαθμό από τον ρόλο που παίζει το μάθημα στο πρόγραμμα σπουδών όπου εντάσσεται.

Σε μερικές σχολές, το μάθημα των μεταγλωττιστών αποτελεί ένα επιστέγασμα για τους προχωρημένους φοιτητές, όπου έννοιες από πολλά άλλα μαθήματα συνδυάζονται σε μια μεγάλη, πρακτική εργασία σχεδίασης και υλοποίησης. Σε ένα τέτοιο μάθημα, οι φοιτητές μπορούν να γράψουν έναν πλήρη μεταγλωττιστή για μια απλή γλώσσα ή να τροποποιήσουν έναν μεταγλωττιστή ανοιχτού κώδικα ώστε να υποστηρίζει κάποιο νέο χαρακτηριστικό της γλώσσας ή κάποιο νέο χαρακτηριστικό της αρχιτεκτονικής. Σε ένα τέτοιο μάθημα, η ύλη μπορεί να παρουσιαστεί με γραμμική σειρά που να ακολουθεί στενά την οργάνωση του βιβλίου.

Σε άλλες σχολές, η εργασία επιστέγασμα γίνεται σε άλλα μαθήματα ή με άλλους τρόπους. Σε ένα τέτοιο μάθημα, ο καθηγητής θα μπορούσε να εστιάσει τις προγραμματιστικές ασκήσεις στους αλγορίθμους και τις υλοποιήσεις τους, χρησιμοποιώντας εργαστηριακές ασκήσεις όπως την κατασκευή ενός τοπικού εκχωρητή καταχωρητών ή μιας διέλευσης επανισοστάθμισης ύψους δένδρου. Σε ένα τέτοιο μάθημα, η παρουσίαση της ύλης μπορεί να μη γίνει γραμμικά, αλλά με κάποια άλλη σειρά προσαρμοσμένη στις ανάγκες των εργαστηριακών ασκήσεων. Για παράδειγμα, στο Rice, έχουμε χρησιμοποιήσει συχνά την κατασκευή ενός απλού τοπικού εκχωρητή καταχωρητών ως πρώτη εργαστηριακή άσκηση· κάθε φοιτητής με εμπειρία στον προγραμματισμό συμβολογλώσσας κατανοεί τα βασικά στοιχεία του προβλήματος. Αυτή η στρατηγική, ωστόσο, εκθέτει τους φοιτητές στην ύλη του Κεφαλαίου 13 πριν δουν το Κεφάλαιο 2.

Και στις δύο περιπτώσεις, το μάθημα θα πρέπει να χρησιμοποιήσει ύλη από άλλα μαθήματα. Το μάθημα συνδέεται προφανώς με την οργάνωση υπολογιστών, τον προγραμματισμό συμβολογλώσσας, τα λειτουργικά συστήματα, την αρχιτεκτονική υπολογιστών, τους αλγορίθμους και τις τυπικές γλώσσες. Παρότι η σχέση της κατασκευής μεταγλωττιστών με άλλα μαθήματα μπορεί να είναι λιγότερο προφανής, δεν είναι λιγότερο σημαντική. Η αντιγραφή χαρακτήρων, όπως αναλύεται στο Κεφάλαιο 7, παίζει κρίσιμο ρόλο στην επίδοση εφαρμογών που περιλαμβάνουν πρωτόκολλα δικτύων, εξυπηρετητές αρχείων και ιστοεξυπηρετητές. Οι τεχνικές που αναπτύσσονται στο Κεφάλαιο 2 για τη σάρωση εφαρμόζονται από την επεξεργασία κειμένου μέχρι το φιλτράρισμα URL. Ο αναβιβαστικός τοπικός εκχωρητής καταχωρητών του Κεφαλαίου 13 είναι εξάδελφος του βέλτιστου εκχρονικού αλγόριθμου αντικατάστασης σελίδων MIN.

■ ΕΠΙΠΛΕΟΝ ΥΛΙΚΟ

Υπάρχει διαθέσιμο και επιπλέον υλικό που μπορεί να σας βοηθήσει να προσαρμόσετε την ύλη που παρουσιάζεται στην SKM2ε στο δικό σας μάθημα. Σε αυτό περιλαμβάνεται ένα πλήρες σύνολο διαλέξεων από την εκδοχή του μαθήματος που πραγματοποιούν οι συγγραφείς στο Rice University και ένα σύνολο λύσεων για τις ασκήσεις. Ο τοπικός αντιπρόσωπος της Elsevier μπορεί να σας δώσει πρόσβαση.

Ευχαριστίες

Στην προετοιμασία της πρώτης έκδοσης της SKM ενεπλάκησαν πολλοί άνθρωποι. Η συνεισφορά τους είναι παρούσα και σε αυτή τη δεύτερη έκδοση. Πολλοί είναι αυτοί που επισήμαναν προβλήματα στην πρώτη έκδοση, μεταξύ άλλων οι Amit Saha, Andrew Waters, Anna Youssefi, Ayal Zachs, Daniel Salce, David Peixotto, Fengmei Zhao, Greg Malecha, Hwansoo Han, Jason Eckhardt, Jeffrey Sandoval, John Elliot, Kamal Sharma, Kim Hazelwood, Max Hailperin, Peter Froehlich, Ryan Stinnett, Sachin Rehki, Sağnak Taşırlar, Timothy Harvey και Xipeng Shen. Θέλουμε επίσης να ευχαριστήσουμε τους κριτές

της δεύτερης έκδοσης, που ήταν οι Jeffery von Ronne, Carl Offner, David Orleans, K. Stuart Smith, John Mallozzi, Elizabeth White και Paul C. Anagnostopoulos. Η ομάδα παραγωγής της Elsevier, ειδικότερα οι Alisa Andreola, Andre Cuello και Megan Guiney, έπαιξαν κρίσιμο ρόλο στη μετατροπή του πρόχειρου χειρογράφου στην τελική του μορφή. Όλοι αυτοί οι άνθρωποι βελτίωσαν σημαντικά αυτό το βιβλίο με τις γνώσεις και τη βοήθειά τους.

Τέλος, πολλοί είναι οι άνθρωποι που μας στήριξαν πνευματικά και συναισθηματικά τα τελευταία πέντε χρόνια. Πρώτα και κύρια, οι οικογένειές μας και οι συνάδελφοί μας στο Rice μάς ενθάρρυναν σε κάθε βήμα της διαδρομής. Ειδικότερα, η Christine και η Carolyn ανέχθηκαν χιλιάδες μακρές συζητήσεις πάνω σε ζητήματα κατασκευής μεταγλωττιστών. Ο Nate McFadden κατηύθυνε αυτή την έκδοση από τη σύλληψή της έως την έκδοσή της με υπομονή και χιούμορ. Η Penny Anderson παρείχε διοικητική και οργανωτική στήριξη που ήταν κρίσιμη για την ολοκλήρωση της δεύτερης έκδοσης. Σε όλους αυτούς τους ανθρώπους εκφράζουμε τις εγκάρδιες ευχαριστίες μας.

Κατάλογος συντομογραφιών

ΑΑΔΣ	αραιή απλή διάδοση σταθερών (sparse simple constant propagation)
ΑΕΓΣ	αντικατάσταση ελέγχων γραμμικής συνάρτησης (linear-function test replacement)
ΑΙΤ	ανάλυση ιδιόμορφων τιμών (singular value decomposition)
ΑΜΔ	αντίστροφη μεταδιατακτική (reverse postorder)
ΑΠΑ	αιτιοκρατικό πεπερασμένο αυτόματο (deterministic finite automaton)
ΑΣΑ	αναβιβαστικά συστήματα αναδιατύπωσης (bottom-up rewrite systems)
ΑΣΓ	αντικειμενοστρεφείς γλώσσες (object-oriented language)
ΑΣΓ	ασυμφραστική γραμματική (context-free grammar)
ΑΣΔ	αφηρημένο συντακτικό δένδρο (abstract syntax tree)
ΑΣΚ	αντίστροφο σύνορο κυριαρχίας (reverse dominance frontier)
ΑΣΟ	αρχιτεκτονική συνόλου οδηγιών (instruction set architecture)
ΑΤΒΚ	αρίθμηση τιμών βάσει κυριάρχων (dominator-based value numbering)
ΑΥΣΔΣ	αραιή υπό συνθήκη διάδοση σταθερών (sparse conditional constant propagation)
ΓΡΕ	γράφημα ροής ελέγχου (control-flow graph)
ΓΤΑ	γλώσσες τύπου Algol (Algol-like languages)
ΔΑ	δελτίο αντικειμένου (object record)
ΔΔ	δραστήριο διάστημα (live range)
ΔΔΕ	δείκτης δελτίου ενεργοποίησης (activation record pointer)
ΔΕ	δελτίο ενεργοποίησης (activation record)
ΕΑ	ενδιάμεση αναπαράσταση (intermediate representation)
ΕΒΜΠ	επεκτεταμένο βασικό μπλοκ (extended basic block)
ΕΙ/Ε	εισόδου/εξόδου (input/output)
ΙΣΣ	ισχυρά συνεκτική συνιστώσα (strongly connected component)
ΚΑΓ	κατευθυντό άκυκλο γράφημα (directed acyclic graph)
ΚΓ	κανονική γραμματική (regular grammar)
ΚΕ	κανονική έκφραση (regular expression)
ΚΣ	κορυφή στοίβας (top-of-stack)
μΑΠΑ	μη αιτιοκρατικό πεπερασμένο αυτόματο (nondeterministic finite automaton)
ΜΤΣ	μεταγλωττιστής τελευταίας στιγμής (just-in-time compiler)
ΟΜΚ	οκνηρή μετακίνηση κώδικα (lazy code motion)
ΟΠΑ	ολοκληρωμένο περιβάλλον ανάπτυξης (integrated development environment)
ΠΑ	πεπερασμένο αυτόματο (finite automaton)
ΠΕΟ	παραλληλισμός επίπεδου οδηγίας (instruction-level parallelism)
ΠΜΛΟ	πολύ μακράς λέξης οδηγίας (very long instruction word)
ΣΚΜ	Σχεδίαση και κατασκευή μεταγλωττιστών
ΣΜΒ	συμβολοκώδικας (assembly code)
ΣΜΤ	στατική μοναδική τιμοδοσία (static single-assignment)
ΤΑΤ	τοπική αρίθμηση τιμών (local value numbering)
ΥΑΤ	υπερτοπική αρίθμηση τιμών (superlocal value numbering)
ΧΕΕΑ	χαμηλού επιπέδου ενδιάμεση αναπαράσταση (low-level intermediate representation)
RTL	register transfer language (γλώσσα μεταφοράς καταχωρητών)

Κεφάλαιο 1

Επισκόπηση της μεταγλώττισης

ΕΠΙΣΚΟΠΗΣΗ ΚΕΦΑΛΑΙΟΥ

Οι μεταγλωττιστές είναι προγράμματα που μεταφράζουν άλλα προγράμματα, γραμμένα σε κάποια γλώσσα, σε προγράμματα γραμμένα σε κάποια άλλη γλώσσα. Την ίδια στιγμή, ένας μεταγλωττιστής είναι ένα μεγάλο σύστημα λογισμικού, με πολλά εσωτερικά μέρη και αλγορίθμους που αλληλεπιδρούν με σύνθετο τρόπο μεταξύ τους. Γι' αυτό, η μελέτη της κατασκευής μεταγλωττιστών είναι μια εισαγωγή στις τεχνικές της μετάφρασης και της βελτίωσης προγραμμάτων, αλλά και μια πρακτική άσκηση τεχνολογίας λογισμικού. Σε αυτό το κεφάλαιο παρουσιάζεται μια αφηρημένη επισκόπηση όλων των βασικών τμημάτων ενός σύγχρονου μεταγλωττιστή.

Λέξεις-κλειδιά: μεταγλωττιστής, διερμηνευτής, αυτόματη μετάφραση

1.1 ΕΙΣΑΓΩΓΗ

Ο ρόλος των υπολογιστών στην καθημερινή ζωή μεγαλώνει κάθε χρόνο. Με την εξάπλωση του Διαδικτύου, οι υπολογιστές και το λογισμικό που εκτελείται σε αυτούς παρέχουν επικοινωνία, ειδήσεις, διασκέδαση και ασφάλεια. Οι ενσωματωμένοι υπολογιστές έχουν αλλάξει τους τρόπους με τους οποίους κατασκευάζουμε αυτοκίνητα, τηλέφωνα, τηλεοράσεις και ραδιόφωνα. Η χρήση των υπολογιστών έχει δημιουργήσει εντελώς νέες κατηγορίες δραστηριοτήτων, από τα βιντεοπαιχνίδια μέχρι τα κοινωνικά δίκτυα. Οι υπερυπολογιστές προβλέπουν τον καιρό κάθε ημέρας και την εξέλιξη των σφοδρών καταιγίδων. Οι ενσωματωμένοι υπολογιστές συγχρονίζουν τους φωτεινούς σηματοδότες και παραδίδουν την ηλεκτρονική αλληλογραφία στην τσέπη μας.

Η ύπαρξη όλων αυτών των εφαρμογών υπολογιστών στηρίζεται σε λογισμικό το οποίο κατασκευάζει εικονικά εργαλεία οικοδομημένα πάνω στις χαμηλού επιπέδου αφηρημένες λειτουργίες που παρέχει το υποκείμενο υλισμικό του υπολογιστή. Σχεδόν όλο αυτό το λογισμικό μεταφράζεται από ένα εργαλείο που καλείται *μεταγλωττιστής*. Ένας μεταγλωττιστής είναι απλώς ένα πρόγραμμα που μεταφράζει άλλα προγράμματα ώστε να είναι έτοιμα για εκτέλεση. Σε αυτό το βιβλίο παρουσιάζονται οι θεμελιώδεις τεχνικές αυτόματης μετάφρασης που χρησιμοποιούνται για την κατασκευή μεταγλωττιστών. Περιγράφονται πολλές από τις προκλήσεις που παρουσιάζονται στην κατασκευή ενός μεταγλωττιστή και οι αλγόριθμοι που χρησιμοποιούν οι κατασκευαστές μεταγλωττιστών για να τις αντιμετωπίσουν.

Μεταγλωττιστής
Ένα πρόγραμμα που μεταφράζει άλλα προγράμματα.

Δομή του κεφαλαίου

Ένας μεταγλωττιστής είναι ένα εργαλείο που μεταφράζει λογισμικό γραμμένο σε μια γλώσσα σε κάποια άλλη γλώσσα. Για να μεταφράσει κείμενα από μια γλώσσα σε κάποια άλλη, το εργαλείο πρέπει να κατανοεί τόσο τη μορφή, ή αλλιώς τη σύνταξη, όσο και το περιεχόμενο, ή αλλιώς τη σημασία, της γλώσσας εισόδου. Πρέπει να κατανοεί τους κανόνες που διέπουν τη σύνταξη και τη σημασία της γλώσσας εξόδου. Τέλος, χρειάζεται ένας μηχανισμός για την απεικόνιση του περιεχομένου της γλώσσας πηγής στη γλώσσα στόχου.

Η δομή ενός τυπικού μεταγλωττιστή απορρέει από αυτές τις απλές παρατηρήσεις. Ο μεταγλωττιστής έχει ένα μπροστινό άκρο που ασχολείται με τη γλώσσα πηγής και ένα πίσω άκρο που ασχολείται με τη γλώσσα στόχου. Η σύνδεση του μπροστινού με το πίσω άκρο γίνεται μέσω μιας τυπικής δομής, την οποία ο μεταγλωττιστής χρησιμοποιεί για την αναπαράσταση του προγράμματος σε μια ενδιάμεση μορφή, της οποίας το νόημα είναι εν πολλοίς ανεξάρτητο από τις δύο γλώσσες. Για τη βελτίωση της μετάφρασης, οι μεταγλωττιστές συχνά περιλαμβάνουν έναν βελτιστοποιητή, ο οποίος αναλύει και αναπροσαρμόζει την ενδιάμεση αυτή μορφή.

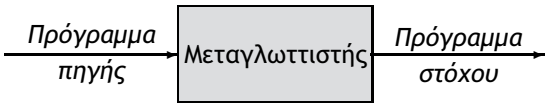
Επισκόπηση

Τα προγράμματα είναι απλώς ακολουθίες αφηρημένων πράξεων γραμμένες σε κάποια *γλώσσα προγραμματισμού* – μια τυπική γλώσσα σχεδιασμένη για τη διατύπωση μιας υπολογιστικής διαδικασίας. Οι γλώσσες προγραμματισμού έχουν αυστηρές ιδιότητες και σημασίες – σε αντίθεση με τις φυσικές γλώσσες, όπως τα κινεζικά και τα πορτογαλικά. Οι γλώσσες προγραμματισμού είναι σχεδιασμένες ώστε να είναι εκφραστικές, συνοπτικές και σαφείς. Οι φυσικές γλώσσες επιτρέπουν αμφισημίες. Οι γλώσσες προγραμματισμού είναι σχεδιασμένες έτσι ώστε να αποφεύγονται οι αμφισημίες· ένα αμφίσημο πρόγραμμα δεν έχει κανένα νόημα. Οι γλώσσες προγραμματισμού είναι σχεδιασμένες έτσι ώστε να επιτρέπουν τον ορισμό υπολογιστικών διαδικασιών – την καταγραφή της ακολουθίας ενεργειών που απαιτούνται για την πραγματοποίηση κάποιας εργασίας ή την παραγωγή κάποιου αποτελέσματος.

Οι γλώσσες προγραμματισμού είναι, εν γένει, σχεδιασμένες έτσι ώστε να επιτρέπουν στους ανθρώπους να διατυπώνουν τις υπολογιστικές διαδικασίες ως ακολουθίες πράξεων. Οι επεξεργαστές των υπολογιστών, τους οποίους στο εξής θα ονομάζουμε επεξεργαστές, μικροεπεξεργαστές ή μηχανές, είναι σχεδιασμένοι έτσι ώστε να εκτελούν ακολουθίες πράξεων. Το επίπεδο αφαίρεσης των πράξεων που υλοποιεί ένας επεξεργαστής είναι, κατά το μεγαλύτερο μέρος, πολύ χαμηλότερο από το επίπεδο αφαίρεσης των πράξεων που ορίζονται σε μια γλώσσα προγραμματισμού. Για παράδειγμα, μια γλώσσα προγραμματισμού περιλαμβάνει συνήθως έναν συνοπτικό τρόπο για την εκτύπωση ενός αριθμού σε ένα αρχείο. Για να μπορέσει να εκτελεστεί η απλή αυτή εντολή της γλώσσας προγραμματισμού, πρέπει να μεταφραστεί στην κυριολεξία σε εκατοντάδες πράξεις μηχανής.

Το εργαλείο που πραγματοποιεί αυτού του είδους τις μεταφράσεις ονομάζεται μεταγλωττιστής. Ο μεταγλωττιστής παίρνει ως είσοδο ένα πρόγραμμα γραμμένο σε κάποια γλώσσα και παράγει ως έξοδο ένα ισοδύναμο πρόγραμμα. Ένας κλασικός μεταγλωττιστής διατυπώνει το πρόγραμμα εξόδου χρησιμοποιώντας τις πράξεις

που διαθέτει ένας συγκεκριμένος επεξεργαστής, ο οποίος συχνά καλείται μηχανή στόχου. Ιδωμένος ως μαύρο κουτί, ένας μεταγλωττιστής θα είχε την εξής μορφή:



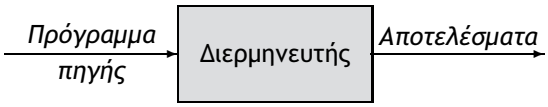
Συνήθεις γλώσσες-«πηγές» είναι η C, η C++, η FORTRAN, η JAVA και η ML. Η γλώσσα-«στόχος» είναι συνήθως το σύνολο οδηγιών ενός επεξεργαστή.

Σύνολο οδηγιών
Το σύνολο των πράξεων που υποστηρίζει ένας επεξεργαστής· η γενική αρχιτεκτονική ενός συνόλου οδηγιών συχνά ονομάζεται *αρχιτεκτονική συνόλου οδηγιών* ή ΑΣΟ.

Μερικοί μεταγλωττιστές παράγουν ένα πρόγραμμα στόχου γραμμένο σε κάποια ανθρωποστρεφή γλώσσα προγραμματισμού αντί για τη συμβολογλώσσα κάποιου υπολογιστή. Για να μπορέσουν να εκτελεστούν απευθείας σε κάποιον υπολογιστή, τα προγράμματα που παράγουν αυτοί οι μεταγλωττιστές χρειάζονται περαιτέρω μετάφραση. Πολλοί ερευνητικοί μεταγλωττιστές παράγουν ως έξοδό τους προγράμματα C. Δεδομένου ότι οι περισσότεροι υπολογιστές διαθέτουν μεταγλωττιστές για την C, το πρόγραμμα στόχου είναι εκτελέσιμο σε όλα αυτά τα συστήματα, με κόστος μια επιπλέον μεταγλώττιση προς τον τελικό στόχο. Οι μεταγλωττιστές που έχουν ως γλώσσα στόχου κάποια γλώσσα προγραμματισμού αντί του συνόλου οδηγιών κάποιου υπολογιστή συνήθως ονομάζονται *μεταφραστές πηγής-προς-πηγή*.

Υπάρχουν επίσης πολλά ακόμη συστήματα που μπορούν να χαρακτηριστούν μεταγλωττιστές. Για παράδειγμα, ένα πρόγραμμα στοιχειοθεσίας που παράγει POSTSCRIPT μπορεί να θεωρηθεί μεταγλωττιστής. Παίρνει ως είσοδο μια προδιαγραφή της όψης που θα πρέπει να έχουν οι εκτυπωμένες σελίδες ενός εγγράφου και παράγει ως έξοδο ένα αρχείο POSTSCRIPT. Η POSTSCRIPT είναι απλώς μια γλώσσα περιγραφής εικόνων. Αφού το πρόγραμμα στοιχειοθεσίας παίρνει μια εκτελέσιμη προδιαγραφή και παράγει μια άλλη εκτελέσιμη προδιαγραφή, είναι ένας μεταγλωττιστής.

Ο κώδικας που μετατρέπει την POSTSCRIPT σε εικονοστοιχεία είναι συνήθως ένας *διερμηνευτής*, και όχι ένας μεταγλωττιστής. Ένας διερμηνευτής παίρνει ως είσοδο μια εκτελέσιμη προδιαγραφή και παράγει ως έξοδο το αποτέλεσμα της εκτέλεσης της προδιαγραφής.



Μερικές γλώσσες, όπως η PERL, η SCHEME και η APL, υλοποιούνται συχνότερα με διερμηνευτές παρά με μεταγλωττιστές.

Μερικές γλώσσες υιοθετούν μεταφραστικούς μηχανισμούς που περιλαμβάνουν και μεταγλώττιση και διερμήνευση. Ο πηγαίος κώδικας της JAVA μεταγλωττίζεται σε μια μορφή που καλείται *οκταδυφιακός κώδικας* (byte code), μια συμπιεσμένη αναπαράσταση που έχει στόχο τη μείωση των χρόνων καταφόρτωσης των εφαρμογών JAVA. Οι εφαρμογές JAVA εκτελούνται μέσω της εκτέλεσης του οκταδυφιακού κώδικα στην αντίστοιχη εικονική μηχανή της JAVA (Java Virtual Machine, JVM), η οποία είναι ένας διερμηνευτής οκταδυφιακού κώδικα. Τα πράγματα περιπλέκονται ακόμη περισσότερο καθώς πολλές υλοποιήσεις της JVM περιλαμβάνουν έναν μεταγλωττιστή που εκτελείται κατά την εκτέλεση του προγράμματος, ο

Εικονική μηχανή
Μια εικονική μηχανή είναι ένας προσομοιωτής κάποιου επεξεργαστή. Είναι ένας *διερμηνευτής* για το σύνολο οδηγιών της συγκεκριμένης μηχανής.

οποίος μερικές φορές καλείται *μεταγλωττιστής τελευταίας στιγμής* ή ΜΤΣ και μεταφράζει τις πολύ συχνά χρησιμοποιούμενες ακολουθίες οκταδυφιακού κώδικα σε εγγενή κώδικα του υποκείμενου υπολογιστή.

Οι διερμηνευτές και οι μεταγλωττιστές έχουν πολλά κοινά. Εκτελούν πάνω κάτω τις ίδιες εργασίες. Και οι δύο αναλύουν το πρόγραμμα εισόδου και αποφασίζουν αν είναι έγκυρο ή όχι. Και οι δύο κατασκευάζουν ένα εσωτερικό μοντέλο της δομής και της σημασίας του προγράμματος. Και οι δύο αποφασίζουν πού θα αποθηκεύονται οι τιμές κατά την εκτέλεση. Ωστόσο, η διερμήνευση του κώδικα με σκοπό την παραγωγή ενός αποτελέσματος είναι αρκετά διαφορετικό πράγμα από την παραγωγή ενός μεταφρασμένου προγράμματος που να μπορεί να εκτελείται ώστε να παράγει το αποτέλεσμα. Στο βιβλίο αυτό επικεντρωνόμαστε στα προβλήματα που ανακύπτουν στην κατασκευή μεταγλωττιστών. Ωστόσο, μεγάλο μέρος της ύλης μπορεί να φανεί χρήσιμο και στους κατασκευαστές διερμηνευτών.

Γιατί να μελετήσει κανείς την κατασκευή μεταγλωττιστών;

Οι μεταγλωττιστές είναι μεγάλα, σύνθετα προγράμματα. Οι μεταγλωττιστές συχνά περιλαμβάνουν εκατοντάδες χιλιάδες, αν όχι εκατομμύρια, γραμμές κώδικα, οργανωμένες σε πολλά υποσυστήματα και τμήματα. Τα διάφορα τμήματα ενός μεταγλωττιστή αλληλεπιδρούν μεταξύ τους με σύνθετους τρόπους. Οι σχεδιαστικές αποφάσεις που λαμβάνονται για κάποιο τμήμα ενός μεταγλωττιστή έχουν σημαντικό αντίκτυπο στα άλλα τμήματά του. Επομένως, η σχεδίαση και η υλοποίηση ενός μεταγλωττιστή είναι μια σημαντική άσκηση τεχνολογίας λογισμικού.

Ένας καλός μεταγλωττιστής περιέχει έναν μικρόκοσμο της επιστήμης υπολογιστών. Χρησιμοποιεί στην πράξη άπληστους αλγόριθμους (για τη δέσμευση καταχωρητών), ευρετικές τεχνικές αναζήτησης (για τον χρονοπρογραμματισμό λίστας), αλγορίθμους γραφημάτων (για την εξάλειψη νεκρού κώδικα), δυναμικό προγραμματισμό (για την επιλογή οδηγιών), πεπερασμένα αυτόματα και αυτόματα στοίβας (για τη σάρωση και τη συντακτική ανάλυση) και αλγορίθμους σταθερού σημείου (για την ανάλυση ροής δεδομένων). Ασχολείται με προβλήματα όπως η δυναμική δέσμευση, ο συγχρονισμός, η ονοματοδοσία, η τοπικότητα, η διαχείριση της ιεραρχίας μνήμης και ο χρονοπρογραμματισμός διοχετεύσεων. Λίγα συστήματα λογισμικού συνδυάζουν τόσα πολλά σύνθετα και διαφορετικά τμήματα. Η ενασχόληση με τα ενδότερα ενός μεταγλωττιστή προσφέρει πρακτική εμπειρία στην τεχνολογία λογισμικού που είναι δύσκολο να αποκτήσει κανείς ασχολούμενος με μικρότερα, λιγότερο περίπλοκα συστήματα.

Οι μεταγλωττιστές παίζουν θεμελιώδη ρόλο στη βασική δραστηριότητα της επιστήμης υπολογιστών: την προπαρασκευή προβλημάτων προκειμένου να μπορούν να επιλυθούν από υπολογιστές. Το μεγαλύτερο μέρος του λογισμικού μεταγλωττίζεται· η ορθότητα αυτής της διαδικασίας και η αποδοτικότητα του παραγόμενου κώδικα επηρεάζουν άμεσα την ικανότητά μας να κατασκευάζουμε μεγάλα συστήματα. Οι περισσότεροι φοιτητές δεν μένουν ικανοποιημένοι διαβάζοντας απλώς για αυτές τις ιδέες· πολλές από αυτές θα πρέπει κανείς να τις υλοποιήσει για να τις εκτιμήσει. Γι' αυτό, η μελέτη της κατασκευής μεταγλωττιστών αποτελεί σημαντικό μέρος της εκπαίδευσης στην επιστήμη υπολογιστών.

Οι μεταγλωττιστές καταδεικνύουν την επιτυχή εφαρμογή της θεωρίας σε πρακτικά προβλήματα. Τα εργαλεία που αυτοματοποιούν την παραγωγή σαρωτών και συντακτικών αναλυτών εφαρμόζουν αποτελέσματα της θεωρίας τυπικών γλωσσών. Τα ίδια εργαλεία χρησιμοποιούνται στην αναζήτηση κειμένου, το φιλτράρισμα ιστοτόπων, την επεξεργασία κειμένου και τους διερμηνευτές των γλωσσών εντολών. Στον έλεγχο τύπων και τη στατική ανάλυση εφαρμόζονται αποτελέσματα της θεωρίας δικτυωμάτων, της αριθμοθεωρίας και άλλων κλάδων των μαθηματικών προκειμένου τα προγράμματα να γίνουν κατανοητά και να βελτιωθούν. Οι γεννήτριες κώδικα χρησιμοποιούν αλγορίθμους συμμόρφωσης δενδρικών μορφοτύπων, συντακτικής ανάλυσης, δυναμικού προγραμματισμού και συμμόρφωσης κειμένου για να αυτοματοποιήσουν την επιλογή οδηγιών.

Παρόλα αυτά, μερικά από τα προβλήματα που ανακύπτουν στην κατασκευή ενός μεταγλωττιστή παραμένουν ανοικτά – δηλαδή, οι τρέχουσες καλύτερες λύσεις επιδέχονται βελτίωση. Οι απόπειρες να σχεδιαστούν υψηλού επιπέδου, καθολικές, ενδιάμεσες αναπαραστάσεις έχουν προσκρούσει στην υψηλή πολυπλοκότητα. Η επικρατούσα μέθοδος χρονοπρογραμματισμού οδηγιών είναι ένας άπληστος αλγόριθμος που περιλαμβάνει διάφορα επίπεδα ευρετικών τεχνικών άρσης ισοβαθμιών. Παρότι είναι προφανές ότι οι μεταγλωττιστές θα πρέπει να χρησιμοποιούν αντιμεταθετικότητα και προσεταιριστικότητα για τη βελτίωση του κώδικα, οι περισσότεροι μεταγλωττιστές που προσπαθούν να το κάνουν απλώς αναδιατάσσουν τις εκφράσεις βάσει κάποιας κανονικής διάταξης.

Για την κατασκευή ενός επιτυχημένου μεταγλωττιστή απαιτούνται γνώσεις αλγορίθμων, μηχανικής και σχεδιασμού. Οι καλοί μεταγλωττιστές υπολογίζουν προσεγγιστικές λύσεις για δύσκολα προβλήματα. Δίνουν έμφαση στην αποδοτικότητα, τόσο της υλοποίησης των ιδίων όσο και του κώδικα που παράγουν. Διαθέτουν εσωτερικές δομές δεδομένων και αναπαραστάσεις γνώσης που αποκαλύπτουν το κατάλληλο επίπεδο λεπτομέρειας – αρκετό για να επιτρέπει ισχυρή βελτιστοποίηση, αλλά όχι τόσο ώστε να αναγκάζει τον μεταγλωττιστή να χάνεται στις λεπτομέρειες. Στην κατασκευή μεταγλωττιστών συνδυάζονται ιδέες και τεχνικές από όλο το εύρος της επιστήμης υπολογιστών, οι οποίες εφαρμόζονται σε ένα περιβάλλον με πολλούς περιορισμούς προκειμένου να επιλυθούν μερικά πραγματικά δύσκολα προβλήματα.

Οι θεμελιώδεις αρχές της μεταγλώττισης

Οι μεταγλωττιστές είναι μεγάλα, σύνθετα, προσεκτικά σχεδιασμένα αντικείμενα. Παρότι πολλά από τα ζητήματα που ανακύπτουν στη σχεδίαση ενός μεταγλωττιστή επιδέχονται διάφορες λύσεις και ερμηνείες, υπάρχουν δύο θεμελιώδεις αρχές που πρέπει να έχει στο νου του συνεχώς ένας κατασκευαστής μεταγλωττιστών. Η πρώτη αρχή είναι απαράβατη:

> *Ο μεταγλωττιστής πρέπει να διατηρεί το νόημα του μεταγλωττιζόμενου προγράμματος.*

Η ορθότητα αποτελεί θεμελιώδες ζήτημα στον προγραμματισμό. Ο μεταγλωττιστής πρέπει να διατηρεί την ορθότητα υλοποιώντας πιστά το «νόημα» του προγράμματος εισόδου του. Η αρχή αυτή είναι το κομβικό σημείο του κοινωνικού συμβολαίου μεταξύ του κατασκευαστή του μεταγλωττιστή και του χρήστη

του μεταγλωττιστή. Αν ο μεταγλωττιστής μπορεί να μεταβάλλει το νόημα κατά βούληση, γιατί να μην παράγει απλώς μια εντολή `nop` ή `return`; Αν είναι αποδεκτή μια εσφαλμένη μετάφραση, γιατί να κοπιάσουμε ώστε να πετύχουμε τη σωστή;

Η δεύτερη αρχή που πρέπει να τηρεί ένας μεταγλωττιστής είναι πρακτικής φύσης:

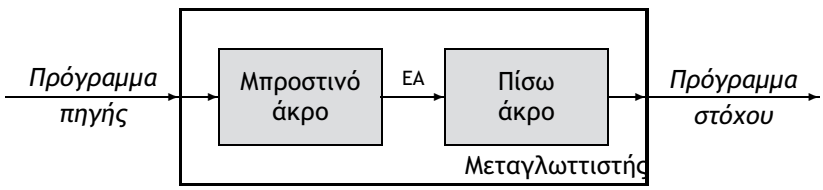
> *Ο μεταγλωττιστής πρέπει να βελτιώνει το πρόγραμμα εισόδου με κάποιον ευδιάκριτο τρόπο.*

Ένας παραδοσιακός μεταγλωττιστής βελτιώνει το πρόγραμμα εισόδου καθιστώντας το άμεσα εκτελέσιμο σε κάποια μηχανή στόχου. Άλλοι «μεταγλωττιστές» βελτιώνουν την είσοδό τους με διαφορετικούς τρόπους. Για παράδειγμα, το `tpic` είναι ένα πρόγραμμα που παίρνει την προδιαγραφή ενός σχεδίου γραμμένου στη γλώσσα γραφικών `pic` και τη μετατρέπει σε LaTeX· η «βελτίωση» έγκειται στο γεγονός ότι το LaTeX είναι ευρύτερα διαθέσιμο και πιο γενικό. Ένας μεταφραστής πηγής-προς-πηγή για την C πρέπει να παράγει κώδικα που να είναι, κατά κάποιο μέτρο, καλύτερος από το πρόγραμμα εισόδου· αν δεν είναι, γιατί να τον καλέσει κανείς;

1.2 ΔΟΜΗ ΕΝΟΣ ΜΕΤΑΓΛΩΤΤΙΣΤΗ

Οι μεταγλωττιστές είναι μεγάλα, σύνθετα συστήματα λογισμικού. Μεταγλωττιστές κατασκευάζονται από το 1955· με την πάροδο των χρόνων έχουμε διδαχτεί πολλά για τον ενδεδειγμένο τρόπο δόμησης ενός μεταγλωττιστή. Προηγουμένως, αναπαραστήσαμε έναν μεταγλωττιστή σαν ένα απλό κουτί που μεταφράζει ένα πρόγραμμα πηγής σε ένα πρόγραμμα στόχου. Η πραγματικότητα είναι, ασφαλώς, πιο σύνθετη από αυτή την απλή εικόνα.

Σύμφωνα με το μοντέλο του απλού κουτιού, ένας μεταγλωττιστής πρέπει να κατανοήσει το πρόγραμμα πηγής που παίρνει ως είσοδο και να απεικονίσει τη λειτουργία του στη μηχανή στόχου. Η διαφορετική φύση αυτών των δύο δραστηριοτήτων επιτρέπει τον καταμερισμό εργασίας και οδηγεί σε μια σχεδίαση όπου η μεταγλώττιση διασπάται σε δύο κύρια τμήματα: στο *μπροστινό άκρο* και στο *πίσω άκρο*.



Το μπροστινό άκρο επικεντρώνεται στην κατανόηση του προγράμματος της γλώσσας πηγής. Το πίσω άκρο επικεντρώνεται στην απεικόνιση προγραμμάτων στη μηχανή στόχου. Αυτός ο διαχωρισμός αρμοδιοτήτων έχει αρκετές σημαντικές συνέπειες για τη σχεδίαση και την υλοποίηση των μεταγλωττιστών.

ΕΑ
Για να αναπαραστήσει τον κώδικα που επεξεργάζεται, ο μεταγλωττιστής χρησιμοποιεί ένα σύνολο δομών δεδομένων. Η μορφή αυτή ονομάζεται *ενδιάμεση αναπαράσταση* ή ΕΑ.

Το μπροστινό άκρο πρέπει να κωδικοποιήσει τη γνώση που έχει για το πρόγραμμα πηγής σε κάποια δομή που θα χρησιμοποιηθεί αργότερα από το πίσω άκρο. Αυτή η *ενδιάμεση αναπαράσταση* (ΕΑ) αποτελεί για τον μεταγλωττιστή την

οριστική αναπαράσταση του κώδικα που μεταφράζει. Σε κάθε σημείο της μεταγλώττισης, ο μεταγλωττιστής πρέπει να έχει μια οριστική αναπαράσταση. Στην πραγματικότητα, μπορεί να χρησιμοποιεί πολλές διαφορετικές ΕΑ καθώς προχωρά η μεταγλώττιση, αλλά κάθε στιγμή μία αναπαράσταση θα είναι η οριστική ΕΑ. Μπορούμε να φανταστούμε την οριστική ΕΑ ως την εκδοχή του προγράμματος που μεταβιβάζεται μεταξύ των ανεξάρτητων φάσεων του μεταγλωττιστή, όπως είναι η ΕΑ που μεταβιβάζεται από το μπροστινό άκρο στο πίσω άκρο στο παραπάνω σχέδιο.

Το μπροστινό άκρο ενός διδιελευσιακού μεταγλωττιστή πρέπει να διασφαλίσει ότι το πρόγραμμα πηγής είναι καλοσχηματισμένο και να απεικονίσει τον κώδικα στην ΕΑ. Το πίσω άκρο πρέπει να απεικονίσει το πρόγραμμα ΕΑ στο σύνολο οδηγιών και τους πεπερασμένους πόρους της μηχανής στόχου. Επειδή το πίσω άκρο επεξεργάζεται μόνο ΕΑ που έχουν παραχθεί από το μπροστινό άκρο, μπορεί να θεωρήσει δεδομένο ότι η ΕΑ δεν περιέχει συντακτικά ή σημασιολογικά σφάλματα.

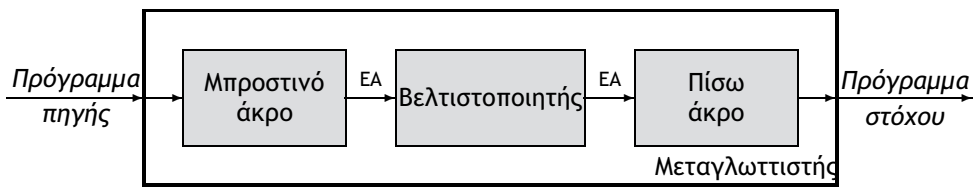
Ο μεταγλωττιστής μπορεί να πραγματοποιήσει πολλές διελεύσεις επί της μορφής ΕΑ του κώδικα μέχρι να παραγάγει το πρόγραμμα στόχου. Αυτό θα πρέπει να οδηγεί σε καλύτερο κώδικα, καθώς ο μεταγλωττιστής μπορεί, ουσιαστικά, να μελετήσει τον κώδικα σε μία φάση και να καταγράψει τις σχετικές λεπτομέρειες. Κατόπιν, σε επόμενες φάσεις, μπορεί να χρησιμοποιήσει τα καταγεγραμμένα αυτά γεγονότα ώστε να βελτιώσει την ποιότητα της μετάφρασης. Σε αυτή η στρατηγική, η γνώση που συνάγεται στην πρώτη διέλευση πρέπει να καταγραφεί στην ΕΑ, όπου μπορούν να τη βρουν και να τη χρησιμοποιήσουν οι επόμενες διελεύσεις.

Τέλος, η διδιελευσιακή δομή μπορεί να απλουστεύσει τη διαδικασία *αναστόχευσης* του μεταγλωττιστή. Μπορούμε εύκολα να φανταστούμε την κατασκευή πολλών πίσω άκρων για το ίδιο μπροστινό άκρο, ώστε να παραγάγουμε μεταγλωττιστές που να δέχονται την ίδια γλώσσα αλλά να παράγουν κώδικα για διαφορετικές μηχανές στόχου. Με αντίστοιχο τρόπο, μπορούμε να φανταστούμε μπροστινά άκρα για διαφορετικές γλώσσες που να παράγουν την ίδια ΕΑ και να χρησιμοποιούν ένα κοινό πίσω άκρο. Και στις δύο περιπτώσεις γίνεται η παραδοχή ότι η ίδια ΕΑ μπορεί να εξυπηρετήσει διάφορους συνδυασμούς πηγής και στόχου· στην πράξη, στην ΕΑ ενσωματώνονται συνήθως λεπτομέρειες που σχετίζονται τόσο με τη γλώσσα όσο και με τη μηχανή.

Αναστόχευση
Η εργασία τροποποίησης ενός μεταγλωττιστή ώστε να παράγει κώδικα για έναν νέο επεξεργαστή συχνά ονομάζεται *αναστόχευση* του μεταγλωττιστή.

Η εισαγωγή της ΕΑ επιτρέπει την προσθήκη περισσότερων φάσεων στη μεταγλώττιση. Ο κατασκευαστής του μεταγλωττιστή μπορεί να εισαγάγει μια τρίτη φάση μεταξύ του μπροστινού άκρου και του πίσω άκρου. Αυτό το μεσαίο τμήμα, ή *βελτιστοποιητής*, παίρνει ως είσοδο ένα πρόγραμμα ΕΑ και παράγει ως έξοδο ένα σημασιολογικά ισοδύναμο πρόγραμμα ΕΑ. Χρησιμοποιώντας την ΕΑ ως διεπαφή, ο κατασκευαστής του μεταγλωττιστή μπορεί να εισαγάγει αυτή την τρίτη φάση παρεμβαίνοντας ελάχιστα στο μπροστινό και το πίσω άκρο. Αυτό οδηγεί στην ακόλουθη δομή μεταγλωττιστή, που ονομάζεται *τριδιελευσιακός μεταγλωττιστής*:

Βελτιστοποιητής
Το μεσαίο τμήμα ενός μεταγλωττιστή, που ονομάζεται *βελτιστοποιητής*, αναλύει και μετασχηματίζει την ΕΑ ώστε να τη βελτιώσει.



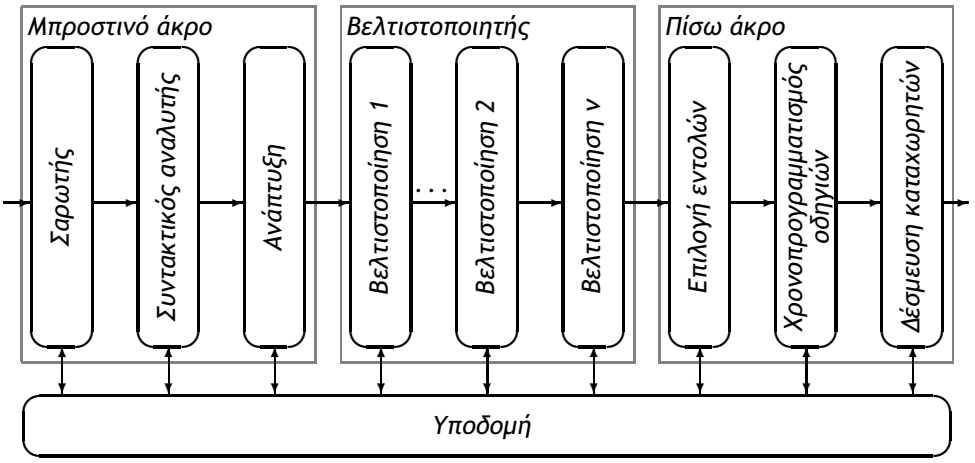
ΕΙΘΕ ΝΑ ΣΠΟΥΔΑΣΕΙΣ ΣΕ ΕΝΔΙΑΦΕΡΟΝΤΕΣ ΚΑΙΡΟΥΣ

Η σημερινή εποχή είναι μια συναρπαστική εποχή για τη σχεδίαση και την υλοποίηση μεταγλωττιστών. Τη δεκαετία του 1980, σχεδόν όλοι οι μεταγλωττιστές ήταν μεγάλα, μονολιθικά συστήματα. Έπαιρναν ως είσοδο κάποια γλώσσα από ένα μικρό σύνολο γλωσσών και παρήγαγαν συμβολοκώδικα για κάποιον συγκεκριμένο υπολογιστή. Ο συμβολοκώδικας συνενωνόταν με τον κώδικα που είχε παραχθεί από άλλες μεταγλωττίσεις -μεταξύ των οποίων βιβλιοθήκες συστήματος και βιβλιοθήκες εφαρμογών- ώστε να σχηματιστεί ένα εκτελέσιμο αρχείο. Το εκτελέσιμο αρχείο αποθηκευόταν σε έναν δίσκο, και την κατάλληλη στιγμή ο τελικός κώδικας μεταφερόταν από τον δίσκο στην κύρια μνήμη και εκτελούνταν.

Σήμερα, η τεχνολογία των μεταγλωττιστών εφαρμόζεται σε πολλά διαφορετικά περιβάλλοντα. Καθώς οι υπολογιστές βρίσκουν εφαρμογή σε πολλά περιβάλλοντα, οι μεταγλωττιστές καλούνται να ανταποκριθούν σε νέους και διαφορετικούς περιορισμούς. Η ταχύτητα δεν είναι πλέον το μοναδικό κριτήριο αξιολόγησης του μεταγλωττισμένου κώδικα. Σήμερα, ο κώδικας μπορεί να αξιολογηθεί με βάση το πόσο μικρός είναι, πόση ενέργεια καταναλώνει, πόσο καλά συμπιέζεται ή πόσα σφάλματα σελίδας παράγει όταν εκτελείται.

Την ίδια στιγμή, οι τεχνικές μεταγλώττισης έχουν ξεφύγει από το όρια των μονολιθικών συστημάτων της δεκαετίας του 1980. Εμφανίζονται σε πολλά νέα μέρη. Οι μεταγλωττιστές της JAVA παίρνουν μερικώς μεταγλωττισμένα προγράμματα (στη μορφή οκταδυφιακού κώδικα JAVA) και τα μεταφράζουν σε εγγενή κώδικα της μηχανής στόχου. Σε αυτό το περιβάλλον, επιτυχία είναι το άθροισμα του χρόνου μεταγλώττισης και του χρόνου εκτέλεσης να είναι μικρότερο από το κόστος της διερμήνευσης. Οι τεχνικές ανάλυσης ολόκληρων προγραμμάτων μετατοπίζονται από τον χρόνο μεταγλώττισης στον χρόνο σύνδεσης, όπου ο συνδέτης μπορεί να αναλύσει τον συμβολοκώδικα ολόκληρης της εφαρμογής και να χρησιμοποιήσει αυτή τη γνώση για να βελτιώσει το πρόγραμμα. Τέλος, οι μεταγλωττιστές καλούνται κατά την εκτέλεση, ώστε να παραγάγουν εξατομικευμένο κώδικα που να εκμεταλλεύεται γεγονότα που δεν μπορούσαν να είναι γνωστά νωρίτερα. Αν ο χρόνος μεταγλώττισης μπορεί να διατηρηθεί μικρός και τα πλεονεκτήματα είναι μεγάλα, αυτή η στρατηγική μπορεί να αποφέρει αισθητές βελτιώσεις.

Ο βελτιστοποιητής είναι ένας μετασχηματιστής ΕΑ-προς-ΕΑ που προσπαθεί να βελτιώσει με κάποιον τρόπο το πρόγραμμα ΕΑ. (Σημειωτέον ότι σύμφωνα με τον ορισμό που δώσαμε στην Ενότητα 1.1 οι μετασχηματιστές αυτοί είναι, και οι ίδιοι, μεταγλωττιστές.) Ο βελτιστοποιητής μπορεί να πραγματοποιήσει μία ή περισσότερες διελεύσεις επί της ΕΑ, να την αναλύσει και να την αναδιατυπώσει. Ο βελτιστοποιητής μπορεί να αναδιατυπώσει την ΕΑ έτσι ώστε το πίσω άκρο να είναι πιθανότερο να παραγάγει ένα ταχύτερο ή ένα μικρότερο πρόγραμμα στόχου. Μπορεί να έχει άλλους στόχους, όπως ένα πρόγραμμα που να παράγει λιγότερα σφάλματα σελίδας ή να χρησιμοποιεί λιγότερη ενέργεια.



ΣΧΗΜΑ 1.1 Δομή ενός τυπικού μεταγλωττιστή.

Εννοιολογικά, η τριδιελευσιακή δομή αντιπροσωπεύει τον κλασικό βελτιστοποιητικό μεταγλωττιστή. Στην πράξη, κάθε φάση διαιρείται εσωτερικά σε μια σειρά διελεύσεων. Το μπροστινό άκρο αποτελείται από δύο ή τρεις διελεύσεις που χειρίζονται τις λεπτομέρειες της αναγνώρισης των έγκυρων προγραμμάτων της γλώσσας πηγής και της παραγωγής της αρχικής μορφής ΕΑ του προγράμματος. Το μεσαίο τμήμα περιλαμβάνει διελεύσεις που πραγματοποιούν διαφορετικές βελτιστοποιήσεις. Το πλήθος και ο σκοπός αυτών των διελεύσεων ποικίλλει από μεταγλωττιστή σε μεταγλωττιστή. Το πίσω άκρο αποτελείται από μια σειρά διελεύσεων καθεμιά από τις οποίες φέρνει το πρόγραμμα ΕΑ ένα βήμα πλησιέστερα στο σύνολο οδηγιών της μηχανής στόχου. Οι τρεις φάσεις και οι επιμέρους διελεύσεις που πραγματοποιούν χρησιμοποιούν κοινή υποδομή. Η δομή αυτή παρουσιάζεται στο Σχήμα 1.1.

Στην πράξη, η εννοιολογική διαίρεση ενός μεταγλωττιστή σε τρεις φάσεις, ένα μπροστινό άκρο, ένα μεσαίο τμήμα ή βελτιστοποιητής και ένα πίσω άκρο, είναι χρήσιμη. Τα προβλήματα που καλούνται να αντιμετωπίσουν αυτές οι φάσεις είναι διαφορετικά. Το μπροστινό άκρο ασχολείται με την κατανόηση του προγράμματος πηγής και την καταγραφή των αποτελεσμάτων της ανάλυσής του στη μορφή ΕΑ. Το τμήμα του βελτιστοποιητή επικεντρώνεται στη βελτίωση της μορφής ΕΑ. Το πίσω άκρο πρέπει να απεικονίσει το μετασχηματισμένο πρόγραμμα ΕΑ στους περιορισμένους πόρους της μηχανής στόχου έτσι ώστε αυτοί οι πόροι να χρησιμοποιούνται αποδοτικά.

Από τις τρεις αυτές φάσεις, ο βελτιστοποιητής έχει την πιο αόριστη περιγραφή. Ο χρήση του όρου *βελτιστοποίηση* υποδηλώνει ότι ο μεταγλωττιστής ανακαλύπτει μια βέλτιστη λύση σε κάποιο πρόβλημα. Τα ζητήματα και τα προβλήματα που ανακύπτουν κατά τη βελτιστοποίηση είναι τόσο σύνθετα και τόσο αλληλεξαρτώμενα που, στην πράξη, δεν μπορούν να επιλυθούν με βέλτιστο τρόπο. Επιπλέον, η πραγματική συμπεριφορά του μεταγλωττισμένου κώδικα εξαρτάται από τις αλληλεπιδράσεις μεταξύ όλων των τεχνικών που εφαρμόζονται από τον βελτιστοποιητή και το πίσω άκρο. Επομένως, ακόμα και αν μπορεί να αποδειχθεί ότι κάποια συγκεκριμένη τεχνική είναι βέλτιστη, οι αλληλεπιδράσεις της με

τις άλλες τεχνικές ενδέχεται να οδηγήσουν σε υποβέλτιστα αποτελέσματα. Επομένως, ένας καλός βελτιστοποιητικός μεταγλωττιστής μπορεί να βελτιώσει την ποιότητα του κώδικα σε σχέση με τη μη βελτιστοποιημένη εκδοχή. Ωστόσο, ένας βελτιστοποιητικός μεταγλωττιστής δεν θα καταφέρει σχεδόν ποτέ να παραγάγει βέλτιστο κώδικα.

Το μεσαίο τμήμα μπορεί είτε να αποτελείται από μία μονολιθική διέλευση που εφαρμόζει μία ή περισσότερες βελτιστοποιήσεις για να βελτιώσει τον κώδικα, είτε να είναι δομημένο ως μια σειρά μικρότερων διελεύσεων, καθεμία από τις οποίες διαβάζει και γράφει μια ΕΑ. Η μονολιθική δομή μπορεί να είναι πιο αποδοτική. Η πολυδιελευσιακή δομή μπορεί να είναι προσφορότερη για μια λιγότερο σύνθετη υλοποίηση και μια απλούστερη στρατηγική αποσφαλμάτωσης του μεταγλωττιστή. Προσφέρει επίσης την ευελιξία χρήσης διαφορετικών συνόλων βελτιστοποιήσεων σε διαφορετικές περιστάσεις. Η επιλογή μεταξύ αυτών των δύο προσεγγίσεων εξαρτάται από τους περιορισμούς υπό τους οποίους κατασκευάζεται και λειτουργεί ο μεταγλωττιστής.

1.3 ΕΠΙΣΚΟΠΗΣΗ ΤΗΣ ΜΕΤΑΦΡΑΣΗΣ

Για να μεταφράσει κώδικα γραμμένο σε μια γλώσσα προγραμματισμού σε κώδικα κατάλληλο για εκτέλεση σε κάποια μηχανή στόχου, ο μεταγλωττιστής πραγματοποιεί πολλά βήματα. Για να συγκεκριμενοποιήσουμε λίγο αυτή την αφηρημένη διαδικασία, ας εξετάσουμε τα βήματα που απαιτούνται για την παραγωγή εκτελέσιμου κώδικα για την ακόλουθη έκφραση:

```
a ← a × 2 × b × c × d
```

όπου a, b, c και d είναι μεταβλητές, το σύμβολο ← δηλώνει τιμοδοσία και × είναι ο τελεστής του πολλαπλασιασμού. Στις υποενότητες που ακολουθούν θα παρακολουθήσουμε την πορεία που ακολουθεί ένας μεταγλωττιστής για να μετατρέψει αυτή την απλή έκφραση σε εκτελέσιμο κώδικα.

1.3.1 Το μπροστινό άκρο

Προκειμένου να μπορέσει να μεταφράσει μια έκφραση σε εκτελέσιμο κώδικα της μηχανής στόχου, ο μεταγλωττιστής πρέπει να κατανοήσει τόσο τη μορφή της, ή αλλιώς τη *σύνταξη* της, όσο και το νόημά της, ή αλλιώς τη *σημασιολογία* της. Το μπροστινό άκρο ελέγχει αν ο κώδικας εισόδου είναι καλοσχηματισμένος ως προς τη σύνταξη και τη σημασιολογία. Αν διαπιστώσει ότι ο κώδικας είναι έγκυρος, δημιουργεί μια αναπαράσταση του κώδικα στην ενδιάμεση αναπαράσταση του μεταγλωττιστή· αν όχι, ενημερώνει τον χρήστη με διαγνωστικά μηνύματα σφάλματος ώστε να μπορέσει να προσδιορίσει τα προβλήματα που έχει ο κώδικας.

Έλεγχος της σύνταξης

Για να ελέγξει τη σύνταξη του προγράμματος εισόδου, ο μεταγλωττιστής πρέπει να συγκρίνει τη δομή του προγράμματος με έναν ορισμό της γλώσσας. Για αυτό απαιτείται ένας κατάλληλος τυπικός ορισμός, ένας αποδοτικός μηχανισμός ελέγχου για το αν η είσοδος συμφωνεί ή όχι με τον ορισμό, και ένα σχέδιο δράσης για την περίπτωση μη επιτρεπτής εισόδου.

ΘΕΜΑΤΑ ΣΥΜΒΟΛΙΣΜΟΥ

Τα βιβλία μεταγλωττιστών ασχολούνται, ουσιαστικά, με θέματα συμβολισμού. Άλλωστε, ένας μεταγλωττιστής μεταφράζει ένα πρόγραμμα γραμμένο με τη χρήση ενός συμβολισμού σε ένα ισοδύναμο πρόγραμμα γραμμένο με τη χρήση κάποιου άλλου συμβολισμού. Κατά την ανάγνωση του βιβλίου θα προκύψουν διάφορα ζητήματα συμβολισμού. Σε μερικές περιπτώσεις, τα ζητήματα αυτά επηρεάζουν άμεσα την κατανόηση της ύλης.

Διατύπωση των αλγορίθμων Προσπαθήσαμε να κρατήσουμε τους αλγορίθμους σύντομους και γραμμένους σε σχετικά υψηλό επίπεδο, θεωρώντας ότι ο αναγνώστης μπορεί να προσθέσει τις λεπτομέρειες της υλοποίησης. Είναι γραμμένοι με `pl'agia stoiqe'ia, qwr'is akrem'ones (our'es)`. Η χρήση εσοχών είναι σκόπιμη και σημαντική· έχει περισσότερη σημασία στην περίπτωση των δομών `αν-τότε-άλλως`. Ο κώδικας με ενιαία εσοχή μετά το `τότε` ή το `άλλως` αποτελεί ένα μπλοκ. Στο ακόλουθο τμήμα κώδικα

```
αν Ενέργεια[s,λέξη] = «ολίσθηση sᵢ» τότε
   αποθέτουμε την λέξη
   αποθέτουμε το sᵢ
   λέξη ← ΕπόμενηΛέξη()
άλλως αν ...
```

όλες οι εντολές μεταξύ του `τότε` και του `άλλως` είναι μέρος της φράσης `τότε` της δομής `αν-τότε-άλλως`. Όταν μια φράση σε μια δομή `αν-τότε-άλλως` περιέχει μόνο μία εντολή, γράφουμε το κλειδωνύμιο `τότε` ή `άλλως` στην ίδια γραμμή με την εντολή.

Κείμενο κώδικα Σε μερικά παραδείγματα, παρουσιάζουμε κείμενο πραγματικού προγράμματος γραμμένων σε κάποια γλώσσα, επιλεγμένο έτσι ώστε να καταδεικνύεται κάποιο ιδιαίτερο σημείο. Το κείμενο πραγματικού προγράμματος είναι στοιχειοθετημένο με γραμματοσειρά `σταθερού πλάτους`.

Αριθμητικοί τελεστές Τέλος, σε αντίθεση με ό,τι συνηθίζεται, δεν χρησιμοποιούμε το * αντί του $\times$ και το / αντί του $\div$, παρά μόνο σε κείμενο πραγματικού προγράμματος. Το νόημα θα πρέπει να είναι σαφές στον αναγνώστη.

Μαθηματικά, η γλώσσα πηγής είναι ένα, συνήθως άπειρο, σύνολο συμβολοσειρών ορισμένων βάσει κάποιου πεπερασμένου συνόλου κανόνων, οι οποίοι ονομάζονται *γραμματική*. Με δύο ξεχωριστές διελεύσεις στο μπροστινό άκρο, οι οποίες ονομάζονται σαρωτής και συντακτικός αναλυτής, διαπιστώνεται αν ο κώδικας εισόδου είναι πράγματι ή όχι μέλος του συνόλου των έγκυρων προγραμμάτων που ορίζει η γραμματική.

Οι γραμματικές των γλωσσών προγραμματισμού συνήθως αναφέρονται στις λέξεις με βάση τα μέρη του λόγου στα οποία ανήκουν, τα οποία μερικές φορές ονομάζονται συντακτικές κατηγορίες. Η διατύπωση των γραμματικών κανόνων με βάση τα μέρη του λόγου επιτρέπει την περιγραφή πολλών προτάσεων με έναν

κανόνα. Για παράδειγμα, σε διάφορες φυσικές γλώσσες πολλές προτάσεις έχουν τη μορφή

Πρόταση → *Υποκείμενο* ρήμα *Αντικείμενο* τελεία

όπου ρήμα και τελεία είναι μέρη του λόγου, και *Πρόταση*, *Υποκείμενο* και *Αντικείμενο* είναι συντακτικές μεταβλητές. Η *Πρόταση* αναπαριστά όλες τις συμβολοσειρές που έχουν τη μορφή που περιγράφει ο συγκεκριμένος κανόνας. Το σύμβολο «→» διαβάζεται «παράγει» και σημαίνει ότι ένα στιγμιότυπο του δεξιού σκέλους μπορεί να αναπαρασταθεί αφηρημένα από τη συντακτική μεταβλητή του αριστερού σκέλους.

Ας θεωρήσουμε την πρόταση «Οι μεταγλωττιστές είναι καλοσχεδιασμένα αντικείμενα.» Το πρώτο βήμα για την κατανόηση της σύνταξης αυτής της πρότασης είναι η αναγνώριση των διαφορετικών λέξεων του προγράμματος εισόδου και η κατάταξη κάθε λέξης σε κάποιο μέρος του λόγου. Σε έναν μεταγλωττιστή, την εργασία αυτή αναλαμβάνει μια διέλευση που ονομάζεται *σαρωτής*. Ο σαρωτής παίρνει ένα ρεύμα χαρακτήρων και το μετατρέπει σε ένα ρεύμα ταξινομημένων λέξεων – δηλαδή σε ζεύγη της μορφής (*p*,*s*), όπου *p* είναι το *μέρος του λόγου* της λέξης και *s* η ορθογραφία της. Ένας σαρωτής θα μετέτρεπε την πρόταση του παραδείγματος στο ακόλουθο ρεύμα ταξινομημένων λέξεων:

Σαρωτής
Η διέλευση του μεταγλωττιστή που μετατρέπει μια συμβολοσειρά χαρακτήρων σε ένα ρεύμα λέξεων.

(άρθρο,«Οι»),(ουσιαστικό,«μεταγλωττιστές»),
(ρήμα,«είναι»),(επίθετο,«καλοσχεδιασμένα»),
(ουσιαστικό,«αντικείμενα»),(τελεία,«.»)

Στην πράξη, η πραγματική ορθογραφία των λέξεων μπορεί να αποθηκευτεί σε έναν πίνακα διασποράς και να αναπαρασταθεί στα διάφορα ζεύγη με έναν ακέραιο αριθμοδείκτη ώστε να απλουστευθούν οι έλεγχοι ισότητας. Η θεωρία και η πρακτική της κατασκευής σαρωτών εξετάζονται στο Κεφάλαιο 2.

Στο επόμενο βήμα, ο μεταγλωττιστής προσπαθεί να συμμορφώσει (δηλαδή να «συνταιριάξει») το ρεύμα των ταξινομημένων λέξεων με τους κανόνες που καθορίζουν τη σύνταξη της γλώσσας εισόδου. Για παράδειγμα, η λειτουργία κάποιας φυσικής γλώσσας θα μπορούσε να περιλαμβάνει τους εξής γραμματικούς κανόνες (όπου η έννοια *Αντικείμενο* περιλαμβάνει και το *κατηγορούμενο*):

1	*Πρόταση*	→	*Υποκείμενο* ρήμα *Αντικείμενο* τελεία
2	*Υποκείμενο*	→	άρθρο ουσιαστικό
3	*Υποκείμενο*	→	άρθρο *Τροποποιητής* ουσιαστικό
4	*Αντικείμενο*	→	ουσιαστικό
5	*Αντικείμενο*	→	*Τροποποιητής* ουσιαστικό
6	*Τροποποιητής*	→	επίθετο
	...		

Εποπτικά, για την πρόταση του παραδείγματός μας μπορούμε να κατασκευάσουμε την ακόλουθη *παραγωγή*:

Κανόνας	Πρωτότυπη πρόταση
—	*Πρόταση*
1	*Υποκείμενο* ρήμα *Αντικείμενο* τελεία
2	άρθρο ουσιαστικό ρήμα *Αντικείμενο* τελεία
5	άρθρο ουσιαστικό ρήμα *Τροποποιητής* ουσιαστικό τελεία
6	άρθρο ουσιαστικό ρήμα επίθετο ουσιαστικό τελεία

Η παραγωγή ξεκινά με τη συντακτική μεταβλητή *Πρόταση*. Σε κάθε βήμα, ένας όρος της πρωτότυπης πρότασης αναδιατυπώνεται και αντικαθίσταται με το δεξιό σκέλος ενός κανόνα από τον οποίο μπορεί να παραχθεί. Στο πρώτο βήμα αντικαθίσταται η *Πρόταση* με χρήση του Κανόνα 1. Στο δεύτερο αντικαθίσταται το *Υποκείμενο* με χρήση του Κανόνα 2. Στο τρίτο βήμα αντικαθίστανται το *Αντικείμενο* με χρήση του Κανόνα 5, ενώ στο τελευταίο βήμα ο *Τροποποιητής* αναδιατυπώνεται ως επίθετο βάσει του Κανόνα 6. Σε αυτό το σημείο, η πρωτότυπη πρόταση που κατασκευάστηκε μέσω αυτής της διαδικασίας παραγωγής συμμορφώνεται με το ρεύμα των ταξινομημένων λέξεων που παρήγαγε ο σαρωτής.

Η παραπάνω παραγωγή αποδεικνύει ότι η πρόταση «Οι μεταγλωττιστές είναι καλοσχεδιασμένα αντικείμενα.» ανήκει στη γλώσσα που περιγράφεται από τους Κανόνες 1 ώς 6. Η πρόταση είναι γραμματικά σωστή. Η διαδικασία αυτόματης εύρεσης παραγωγών ονομάζεται *συντακτική ανάλυση*. Στο Κεφάλαιο 3 παρουσιάζονται οι τεχνικές που χρησιμοποιούν οι μεταγλωττιστές για τη συντακτική ανάλυση του προγράμματος εισόδου.

Συντακτικός αναλυτής
Η διέλευση του μεταγλωττιστή που διαπιστώνει αν το ρεύμα εισόδου είναι πρόταση της γλώσσας πηγής.

Μια γραμματικά σωστή πρόταση μπορεί να μην έχει νόημα. Για παράδειγμα, η πρόταση «Οι βράχοι είναι πράσινα λαχανικά.» έχει τα ίδια μέρη του λόγου με την ίδια σειρά με την πρόταση «Οι μεταγλωττιστές είναι καλοσχεδιασμένα αντικείμενα.» αλλά δεν έχει λογικό νόημα. Για να κατανοήσουμε τη διαφορά μεταξύ αυτών των δύο προτάσεων πρέπει να γνωρίζουμε τα συμφραζόμενα των λογισμικών συστημάτων, των βράχων και των λαχανικών.

Τα σημασιολογικά μοντέλα που χρησιμοποιούν οι μεταγλωττιστές για να εξάγουν συμπεράσματα σχετικά με τις γλώσσες προγραμματισμού είναι απλούστερα από τα μοντέλα που απαιτούνται για την κατανόηση της φυσικής γλώσσας. Οι μεταγλωττιστές κατασκευάζουν μαθηματικά μοντέλα που εντοπίζουν συγκεκριμένα είδη ασυνέπειας σε ένα πρόγραμμα. Οι μεταγλωττιστές ελέγχουν τη συνέπεια των τύπων· για παράδειγμα, η έκφραση

Έλεγχος τύπων
Η διέλευση του μεταγλωττιστή που ελέγχει αν η χρήση των ονομάτων στο πρόγραμμα εισόδου είναι συνεπής ως προς τους τύπους.

```
a ← a × 2 × b × c × d
```

μπορεί να είναι συντακτικά καλοσχηματισμένη, αλλά αν οι `b` και `d` είναι συμβολοσειρές χαρακτήρων, η πρόταση μπορεί παρόλα αυτά να μην είναι έγκυρη. Σε συγκεκριμένες περιστάσεις, οι μεταγλωττιστές ελέγχουν επίσης τη συνεπή χρήση των αριθμών· για παράδειγμα, μια παραπομπή σε μια συστοιχία πρέπει να έχει το ίδιο πλήθος διαστάσεων με τη δηλωμένη τάξη της συστοιχίας, ενώ όταν καλείται μια διαδικασία πρέπει να δηλώνεται το ίδιο πλήθος ορισμάτων με αυτό που περιλαμβάνει ο ορισμός της διαδικασίας. Στο Κεφάλαιο 4 διερευνώνται μερικά

από τα ζητήματα που ανακύπτουν στον έλεγχο τύπων και στη σημασιολογική επεξεργασία που πραγματοποιούν οι μεταγλωττιστές.

Ενδιάμεσες αναπαραστάσεις

Το τελικό ζήτημα που διαχειρίζεται το μπροστινό άκρο ενός μεταγλωττιστή είναι η δημιουργία μιας μορφής ΕΑ του κώδικα. Οι μεταγλωττιστές χρησιμοποιούν πληθώρα διαφορετικών ειδών ΕΑ, ανάλογα με τη γλώσσα πηγής, τη γλώσσα στόχου και τους συγκεκριμένους μετασχηματισμούς που εφαρμόζει ο μεταγλωττιστής. Μερικές ΕΑ αναπαριστούν το πρόγραμμα ως γράφημα. Άλλες μοιάζουν με ακολουθιακά προγράμματα συμβολοκώδικα. Ο κώδικας που βλέπουμε στο περιθώριο δείχνει τη μορφή που θα μπορούσε να έχει η έκφραση του παραδείγματός μας σε μια χαμηλού επιπέδου, ακολουθιακή ΕΑ. Στο Κεφάλαιο 5 παρουσιάζεται μια επισκόπηση των διάφορων ειδών ΕΑ που χρησιμοποιούν οι μεταγλωττιστές.

```
t₀ ← a × 2
t₁ ← t₀ × b
t₂ ← t₁ × c
t₃ ← t₂ × d
a  ← t₃
```

Για κάθε δομή της γλώσσας πηγής, ο μεταγλωττιστής χρειάζεται μια στρατηγική για το πώς θα υλοποιήσει τη συγκεκριμένη δομή στη μορφή ΕΑ του κώδικα. Οι επιλογές που γίνονται επηρεάζουν τη δυνατότητα του μεταγλωττιστή να μετασχηματίσει και να βελτιώσει τον κώδικα. Γι' αυτό, αφιερώνουμε δύο κεφάλαια στα ζητήματα που ανακύπτουν στη δημιουργία μιας ΕΑ για τις διάφορες δομές του πηγαίου κώδικα. Η σύνδεση διαδικασιών είναι μια πηγή επιβάρυνσης της αποδοτικότητας στον τελικό κώδικα, και συνάμα η βασική κόλλα που συνενώνει τα διαφορετικά αρχεία πηγής σε μια εφαρμογή. Γι' αυτό, αφιερώνουμε το Κεφάλαιο 6 στα ζητήματα που αφορούν τις κλήσεις διαδικασιών. Στο Κεφάλαιο 7 παρουσιάζονται στρατηγικές υλοποίησης για τις περισσότερες από τις υπόλοιπες δομές των γλωσσών προγραμματισμού.

1.3.2 Ο βελτιστοποιητής

Όταν το μπροστινό άκρο παράγει την ΕΑ του προγράμματος εισόδου, επεξεργάζεται μία-μία τις εντολές, με τη σειρά με την οποία τις συναντά. Έτσι, το αρχικό πρόγραμμα ΕΑ περιέχει γενικές στρατηγικές υλοποίησης που δουλεύουν για κάθε είδους περιβάλλοντα συμφραζόμενα που ενδέχεται να παραγάγει ο μεταγλωττιστής. Κατά την εκτέλεση, ο κώδικας θα εκτελεστεί με πιο περιορισμένα και προβλέψιμα συμφραζόμενα. Ο βελτιστοποιητής αναλύει τη μορφή ΕΑ του κώδικα για να ανακαλύψει γεγονότα σχετικά με αυτά τα συμφραζόμενα και κατόπιν χρησιμοποιεί τη συμφραστική γνώση ώστε να αναδιατυπώσει τον κώδικα για να υπολογίζει την ίδια απάντηση με αποδοτικότερο τρόπο.

Αποδοτικότητα μπορεί να σημαίνει πολλά πράγματα. Υπό την κλασική έννοια, βελτιστοποίηση σημαίνει μείωση του χρόνου εκτέλεσης της εφαρμογής. Σε άλλες περιπτώσεις, ο βελτιστοποιητής μπορεί να προσπαθήσει να μειώσει το μέγεθος του μεταγλωττισμένου κώδικα ή άλλα πράγματα, όπως την ενέργεια που καταναλώνει ο επεξεργαστής κατά την αποτίμηση του κώδικα. Όλες αυτές οι στρατηγικές έχουν στόχο την αύξηση της αποδοτικότητας.

Επανερχόμενοι στο παράδειγμά μας, ας το εξετάσουμε στα συμφραζόμενα του Σχήματος 1.2α. Η εντολή εμφανίζεται μέσα σε έναν βρόχο. Από τις τιμές που χρησιμοποιεί, μόνο οι `a` και `d` μεταβάλλονται μέσα στον βρόχο. Οι τιμές των `2`, `b`

ΟΡΟΛΟΓΙΑ

Ο προσεκτικός αναγνώστης θα παρατηρήσει ότι χρησιμοποιούμε τη λέξη *κώδικας* σε πολλά σημεία όπου θα ταίριαζε ίσως είτε η λέξη *πρόγραμμα* είτε η λέξη *διαδικασία*. Οι μεταγλωττιστές μπορούν να κληθούν για να μεταφράσουν διάφορα τμήματα κώδικα, από μια απλή παραπομπή μέχρι ένα ολόκληρο σύστημα προγραμμάτων. Αντί να ορίζουμε ρητά κάποια εμβέλεια μεταγλώττισης, θα συνεχίσουμε να χρησιμοποιούμε τον αμφίσημο, αλλά γενικότερο, όρο *κώδικα*.

```
b ← ...
c ← ...
a ← 1
for i = 1 to n
  read d
  a ← a × 2 × b × c × d
  end
```

(α) Αρχικός κώδικας εντός συμφραζομένων

```
b ← ...
c ← ...
a ← 1
t ← 2 × b × c
for i = 1 to n
  read d
  a ← a × d × t
  end
```

(β) Βελτιωμένος κώδικας

ΣΧΗΜΑ 1.2 Τα συμφραζόμενα κάνουν τη διαφορά.

και c παραμένουν αμετάβλητες μέσα στον βρόχο. Αν ο βελτιστοποιητής ανακαλύψει αυτό το γεγονός, μπορεί να αναδιατυπώσει τον κώδικα όπως στο Σχήμα 1.2β. Σε αυτή την εκδοχή, το πλήθος των πολλαπλασιασμών έχει μειωθεί από $4 \cdot n$ σε $2 \cdot n + 2$. Για $n > 1$, ο αναδιατυπωμένος βρόχος θα πρέπει να εκτελείται ταχύτερα. Αυτού του είδους η βελτιστοποίηση εξετάζεται στα Κεφάλαια 8, 9 και 10.

Ανάλυση

Οι περισσότερες βελτιστοποιήσεις αποτελούνται από μια ανάλυση και έναν μετασχηματισμό. Η ανάλυση προσδιορίζει τα σημεία όπου ο μεταγλωττιστής μπορεί να εφαρμόσει ασφαλώς και επωφελώς τη συγκεκριμένη τεχνική. Για την υποστήριξη των μετασχηματισμών, οι μεταγλωττιστές χρησιμοποιούν διάφορα είδη ανάλυσης. Η *ανάλυση ροής δεδομένων* εξάγει, κατά τη μεταγλώττιση, συμπεράσματα σχετικά με τη ροή τιμών κατά την εκτέλεση. Οι αναλυτές ροής δεδομένων συνήθως επιλύουν ένα σύστημα εξισώσεων μεταξύ συνόλων που προκύπτουν από τη δομή του μεταφραζόμενου κώδικα. Η *ανάλυση εξαρτήσεων* χρησιμοποιεί αριθμοθεωρητικούς ελέγχους για να εξαγάγει συμπεράσματα σχετικά με τις τιμές που μπορούν να πάρουν οι εκφράσεις που υπολογίζουν αριθμοδείκτες θέσης συστοιχιών. Χρησιμοποιείται για την αποσαφήνιση των παραπομπών σε στοιχεία συστοιχιών. Στο Κεφάλαιο 9 παρουσιάζεται λεπτομερώς η ανάλυση ροής δεδομένων και η εφαρμογή της, καθώς και η κατασκευή της μορφής στατικής μοναδικής τιμοδοσίας, μιας ΕΑ που κωδικοποιεί πληροφορία σχετικά με τη ροή τιμών και ελέγχου απευθείας στην ΕΑ.

Ανάλυση ροής δεδομένων
Μια διαδικασία εξαγωγής συμπερασμάτων κατά τη μεταγλώττιση σχετικά με τη ροή τιμών κατά την εκτέλεση.

Μετασχηματισμός

Για να βελτιώσει τον κώδικα, ο μεταγλωττιστής δεν πρέπει απλώς να τον αναλύσει. Ο μεταγλωττιστής πρέπει να χρησιμοποιήσει τα αποτελέσματα της ανάλυσης για να αναδιατυπώσει τον κώδικα σε αποδοτικότερη μορφή. Για τη βελτίωση των χρονικών ή χωρικών απαιτήσεων του εκτελέσιμου κώδικα έχουν επινοηθεί χιλιάδες μετασχηματισμοί. Μερικοί, όπως ο εντοπισμός βροχοαναλλοίωτων υπολογισμών και η μετακίνησή τους σε λιγότερο συχνά εκτελούμενες τοποθεσίες, βελτιώνουν τον χρόνο εκτέλεσης του προγράμματος. Άλλοι συμπτύσσουν τον κώδικα. Οι μετασχηματισμοί ποικίλλουν ως προς το αποτέλεσμά τους, την εμβέλεια εφαρμογής τους και την ανάλυση που απαιτείται για να υποστηριχθούν. Η βιβλιογραφία πάνω στους μετασχηματισμούς είναι πλούσια· επειδή το θέμα είναι αρκετά μεγάλο και μπορεί να διερευνηθεί σε βάθος, του αξίζουν ένα ή περισσότερα ξεχωριστά βιβλία. Στο Κεφάλαιο 10 καλύπτεται το θέμα των βαθμωτών μετασχηματισμών – δηλαδή των μετασχηματισμών που έχουν στόχο τη βελτίωση της επίδοσης του κώδικα σε έναν μόνο επεξεργαστή. Η παρουσίαση είναι διαρθρωμένη με βάση μια ταξινομία των μετασχηματισμών η οποία συνοδεύεται από παραδείγματα.

1.3.3 Το πίσω άκρο

Το πίσω άκρο του μεταγλωττιστή διασχίζει τη μορφή EA του κώδικα και παράγει κώδικα για τη μηχανή στόχου. Για να υλοποιήσει κάθε πράξη της EA, επιλέγει πράξεις της μηχανής στόχου. Επιλέγει μια αποδοτική σειρά εκτέλεσης των πράξεων. Αποφασίζει ποιες τιμές θα διαμένουν σε καταχωρητές και ποιες τιμές θα διαμένουν στη μνήμη, και εισάγει κώδικα για να επιβάλει την εφαρμογή αυτών των αποφάσεων.

Επιλογή οδηγιών

Στο πρώτο στάδιο της παραγωγής κώδικα, οι πράξεις της EA αναδιατυπώνονται ως πράξεις της μηχανής στόχου· η διαδικασία αυτή ονομάζεται *επιλογή οδηγιών*. Η επιλογή οδηγιών απεικονίζει κάθε πράξη της EA, στα συγκεκριμένα συμφραζόμενά της, σε μία ή περισσότερες πράξεις της μηχανής στόχου. Για να εξηγήσουμε τη διαδικασία, ας εξετάσουμε την αναδιατύπωση της έκφρασης $a \leftarrow a \times 2 \times b \times c \times d$ του παραδείγματός μας σε κώδικα για την εικονική μηχανή ILOC. (Θα χρησιμοποιήσουμε την ILOC σε όλο το βιβλίο.) Η μορφή EA της έκφρασης παρουσιάζεται εκ νέου στο περιθώριο. Ο μεταγλωττιστής ενδέχεται να επιλέξει τις πράξεις που παρουσιάζονται στο Σχήμα 1.3. Στον συγκεκριμένο κώδικα, γίνεται η παραδοχή ότι οι a, b, c και d βρίσκονται σε σχετικές θέσεις @a, @b, @c και @d από μια διεύθυνση που περιέχεται στον καταχωρητή r_{arp}.

$t_0 \leftarrow a \times 2$
$t_1 \leftarrow t_0 \times b$
$t_2 \leftarrow t_1 \times c$
$t_3 \leftarrow t_2 \times d$
$a \leftarrow t_3$

Ο μεταγλωττιστής έχει επιλέξει μια απλή ακολουθία πράξεων. Φορτώνει όλες τις αναγκαίες τιμές στους καταχωρητές, εκτελεί τους πολλαπλασιασμούς με τη σειρά και αποθηκεύει το αποτέλεσμα στη θέση μνήμης της a. Θεωρεί ότι υπάρχει απεριόριστο απόθεμα καταχωρητών και τους ονοματίζει με συμβολικά ονόματα: ο r_a προορίζεται για την αποθήκευση της a και ο r_{arp} για την αποθήκευση της διεύθυνσης όπου ξεκινά η περιοχή αποθήκευσης δεδομένων των ονοματισμένων τιμών μας. Για να απεικονίσει αυτά τα συμβολικά ονόματα καταχωρητών, ή αλλιώς *ει-*

Εικονικός καταχωρητής
Ένα συμβολικό όνομα καταχωρητή που χρησιμοποιεί ο μεταγλωττιστής για να δηλώσει ότι μια τιμή μπορεί να αποθηκευτεί σε έναν καταχωρητή.

ΣΧΕΤΙΚΑ ΜΕ ΤΗΝ ILOC

Σε όλο το βιβλίο, τα χαμηλού επιπέδου παραδείγματα είναι γραμμένα σε έναν συμβολισμό τον οποίο ονομάζουμε ILOC – το ακρωνύμιο της φράσης «intermediate language for an optimizing compiler» (ενδιάμεση γλώσσα ενός βελτιστοποιητικού μεταγλωττιστή). Με τα χρόνια, ο συγκεκριμένος συμβολισμός έχει υποστεί πολλές αλλαγές. Η εκδοχή που χρησιμοποιείται σε αυτό το βιβλίο περιγράφεται λεπτομερώς στο Παράρτημα Α.

Φανταστείτε την ILOC σαν τη συμβολογλώσσα μιας απλής μηχανής RISC. Διαθέτει ένα τυπικό σύνολο πράξεων. Οι περισσότερες πράξεις παίρνουν ορίσματα που είναι καταχωρητές. Οι πράξεις μνήμης, οι `load` και `store`, μεταφέρουν τιμές μεταξύ της μνήμης και των καταχωρητών. Για να απλουστεύσουμε την παρουσίαση, στα περισσότερα παραδείγματα κάνουμε την παραδοχή ότι όλα τα δεδομένα αποτελούνται από ακέραιους αριθμούς.

Κάθε πράξη έχει ένα σύνολο τελεστέων και έναν στόχο. Η πράξη γράφεται σε πέντε μέρη: ένα όνομα πράξης, μια λίστα τελεστέων, ένα διαχωριστικό, μια λίστα στόχων και ένα προαιρετικό σχόλιο. Έτσι, για να προσθέσει τους καταχωρητές 1 και 2 και να τοποθετήσει το αποτέλεσμα στον καταχωρητή 3, ο προγραμματιστής θα έγραφε

```
add r1,r2 ⇒ r3  // ενδεικτική οδηγία
```

Το διαχωριστικό ⇒ προηγείται της λίστας στόχων. Μας υπενθυμίζει σχηματικά ότι η πληροφορία ρέει από τα αριστερά προς τα δεξιά. Ειδικότερα, αποτρέπει τη σύγχυση σε περιπτώσεις όπου ένας άνθρωπος που διαβάζει ένα κείμενο γραμμένο σε συμβολογλώσσα θα μπορούσε να μπερδέψει εύκολα τους τελεστέους με τους στόχους. (Βλ. `loadAI` και `storeAI` στον ακόλουθο πίνακα.)

Στο παράδειγμα του Σχήματος 1.3 χρησιμοποιούνται μόνο τέσσερεις πράξεις ILOC:

Πράξη ILOC		Σημασία
`loadAI`	$r_1, c_2 \Rightarrow r_3$	Μνήμη $(r_1 + c_2) \rightarrow r_3$
`loadI`	$c_1 \Rightarrow r_2$	$c_1 \rightarrow r_2$
`mult`	$r_1, r_2 \Rightarrow r_3$	$r_1 \times r_2 \rightarrow r_3$
`storeAI`	$r_1 \Rightarrow r_2, c_3$	$r_1 \rightarrow$ Μνήμη $(r_2 + c_3)$

Στο Παράρτημα Α παρουσιάζεται μια λεπτομερέστερη περιγραφή της ILOC. Στα παραδείγματα, ο r_{arp} είναι πάντα ο καταχωρητής που περιέχει την αρχή της περιοχής αποθήκευσης δεδομένων της τρέχουσας διαδικασίας, που είναι γνωστός και ως *δείκτης δελτίου ενεργοποίησης*.

κονικούς καταχωρητές, στους πραγματικούς καταχωρητές της μηχανής στόχου, ο επιλογέας οδηγιών στηρίζεται, εμμέσως, στον εκχωρητή καταχωρητών.

```
loadAI  r_arp, @a ⇒ r_a        // φορτώνουμε την «a»
loadI   2         ⇒ r_2        // η σταθερά 2 στον  r_2
loadAI  r_arp, @b ⇒ r_b        // φορτώνουμε την «b»
loadAI  r_arp, @c ⇒ r_c        // φορτώνουμε την «c»
loadAI  r_arp, @d ⇒ r_d        // φορτώνουμε την «d»
mult    r_a, r_2  ⇒ r_a        // r_a ← a × 2
mult    r_a, r_b  ⇒ r_a        // r_a ← (a × 2) × b
mult    r_a, r_c  ⇒ r_a        // r_a ← (a × 2 × b) × c
mult    r_a, r_d  ⇒ r_a        // r_a ← (a × 2 × b × c) × d
storeAI r_a       ⇒ r_arp, @a  // γράφουμε τον r_a στην «a»
```

ΣΧΗΜΑ 1.3 Κώδικας ILOC για την $a \leftarrow a \times 2 \times b \times c \times d$.

Ο επιλογέας οδηγιών μπορεί να εκμεταλλευτεί τις ιδιαίτερες πράξεις της μηχανής στόχου. Για παράδειγμα, αν υποστηρίζεται η πράξη πολλαπλασιασμού άμεσης τιμής (`multI`), μπορεί να αντικαταστήσει την πράξη `mult` $r_a, r_2 \Rightarrow r_a$ με την `multI` $r_a, 2 \Rightarrow r_a$, εξαλείφοντας την ανάγκη για την πράξη `loadI` $2 \Rightarrow r_2$ και μειώνοντας τη ζήτηση για καταχωρητές. Αν η πρόσθεση είναι ταχύτερη από τον πολλαπλασιασμό, θα μπορούσε να αντικαταστήσει την `mult` $r_a, r_2 \Rightarrow r_a$ με την `add` $r_a, r_a \Rightarrow r_a$, αποφεύγοντας αφ' ενός την `loadI` και τη χρήση του r_2 σε αυτή, και αντικαθιστώντας παράλληλα την `mult` με την ταχύτερη `add`. Στο Κεφάλαιο 11 παρουσιάζονται δύο διαφορετικές τεχνικές επιλογής οδηγιών που επιλέγουν αποδοτικές υλοποιήσεις για τις πράξεις της ΕΑ χρησιμοποιώντας συμμόρφωση μορφοτύπων.

Δέσμευση καταχωρητών

Κατά την επιλογή οδηγιών, ο μεταγλωττιστής σκοπίμως αγνόησε το γεγονός ότι η μηχανή στόχου διαθέτει περιορισμένο σύνολο καταχωρητών και χρησιμοποίησε εικονικούς καταχωρητές κάνοντας την παραδοχή ότι υπάρχουν «αρκετοί» καταχωρητές. Στην πράξη, στα αρχικά στάδια της μεταγλώττισης μπορεί να δημιουργηθεί μεγαλύτερη ζήτηση για καταχωρητές από όση μπορεί να υποστηρίξει το υλισμικό. Ο εκχωρητής καταχωρητών πρέπει να απεικονίσει αυτούς τους εικονικούς καταχωρητές σε πραγματικούς καταχωρητές της μηχανής στόχου. Έτσι, ο εκχωρητής καταχωρητών αποφασίζει, σε κάθε σημείο του κώδικα, ποιες τιμές πρέπει να διαμείνουν στους καταχωρητές της μηχανής στόχου και στη συνέχεια αναδιατυπώνει τον κώδικα ώστε να συμμορφώνεται με τις αποφάσεις του. Για παράδειγμα, ένας εκχωρητής καταχωρητών θα μπορούσε να ελαχιστοποιήσει τη χρήση καταχωρητών αναδιατυπώνοντας τον κώδικα του Σχήματος 1.3 ως εξής:

```
loadAI  r_arp, @a ⇒ r_1        // φορτώνουμε την «a»
add     r_1, r_1  ⇒ r_1        // r_1 ← a × 2
loadAI  r_arp, @b ⇒ r_2        // φορτώνουμε την «b»
mult    r_1, r_2  ⇒ r_1        // r_1 ← (a × 2) × b
loadAI  r_arp, @c ⇒ r_2        // φορτώνουμε την «c»
mult    r_1, r_2  ⇒ r_1        // r_1 ← (a × 2 × b) × c
loadAI  r_arp, @d ⇒ r_2        // φορτώνουμε την «d»
mult    r_1, r_2  ⇒ r_1        // r_1 ← (a × 2 × b × c) × d
storeAI r_1       ⇒ r_arp, @a  // γράφουμε τον r_a στην «a»
```

Αυτή η ακολουθία χρησιμοποιεί τρεις καταχωρητές αντί για έξι.

Η ελαχιστοποίηση της χρήσης καταχωρητών μπορεί να είναι αντιπαραγωγική. Αν, για παράδειγμα, κάποιες από τις ονοματισμένες τιμές `a`, `b`, `c` ή `d` βρίσκονται ήδη σε καταχωρητές, ο κώδικας θα πρέπει να παραπέμψει απευθείας σε αυτούς τους καταχωρητές. Αν όλες οι τιμές βρίσκονται σε καταχωρητές, η ακολουθία θα μπορούσε να υλοποιηθεί με τέτοιο τρόπο ώστε να μην απαιτηθούν επιπλέον καταχωρητές. Επίσης, αν κάποια γειτονική έκφραση υπολογίζει επίσης το $a \times 2$, πιθανόν να είναι καλύτερο να διατηρηθεί αυτή η τιμή σε έναν καταχωρητή αντί να υπολογιστεί εκ νέου αργότερα. Αυτή η βελτιστοποίηση θα είχε ως αποτέλεσμα την αύξηση της ζήτησης για καταχωρητές, αλλά θα εξάλειφε μια μετέπειτα οδηγία. Στο Κεφάλαιο 13 μελετώνται τα προβλήματα που ανακύπτουν κατά τη δέσμευση καταχωρητών και οι τεχνικές που χρησιμοποιούν οι κατασκευαστές μεταγλωττιστών για να τα επιλύουν.

Χρονοπρογραμματισμός οδηγιών

Για να παραγάγει κώδικα που να εκτελείται γρήγορα, η γεννήτρια κώδικα μπορεί να χρειαστεί να αναδιατάξει κάποιες πράξεις ώστε να συμμορφώνονται με τους ιδιαίτερους περιορισμούς επίδοσης της μηχανής στόχου. Ο χρόνος εκτέλεσης των διάφορων πράξεων μπορεί να ποικίλλει. Οι πράξεις προσπέλασης της μνήμης μπορεί να απαιτούν δεκάδες ή εκατοντάδες κύκλους, ενώ κάποιες αριθμητικές πράξεις, ιδιαίτερα η διαίρεση, απαιτεί αρκετούς κύκλους. Η επίδραση αυτών των πράξεων μεγάλου χρόνου απόκρισης στην επίδοση του μεταγλωττισμένου κώδικα μπορεί να είναι σημαντικότατη.

Ας θεωρήσουμε, προς στιγμήν, ότι οι πράξεις `loadAI` και `storeAI` απαιτούν τρεις κύκλους, ότι η πράξη `mult` απαιτεί δύο κύκλους και ότι όλες οι άλλες πράξεις απαιτούν έναν κύκλο. Στον πίνακα που ακολουθεί παρουσιάζεται η επίδοση του προηγούμενου αποσπάσματος κώδικα υπό αυτές τις παραδοχές. Στη στήλη **Έναρξη** παρουσιάζεται ο κύκλος στον οποίο ξεκινά η εκτέλεση κάθε πράξης και στη στήλη **Λήξη** ο κύκλος στον οποίο ολοκληρώνεται.

Έναρξη	Λήξη			
1	3	loadAI	r_{arp}, @a ⇒ r_1	// φορτώνουμε την «a»
4	4	add	r_1, r_1 ⇒ r_1	// $r_1 \leftarrow a \times 2$
5	7	loadAI	r_{arp}, @b ⇒ r_2	// φορτώνουμε την «b»
8	9	mult	r_1, r_2 ⇒ r_1	// $r_1 \leftarrow (a \times 2) \times b$
10	12	loadAI	r_{arp}, @c ⇒ r_2	// φορτώνουμε την «c»
13	14	mult	r_1, r_2 ⇒ r_1	// $r_1 \leftarrow (a \times 2 \times b) \times c$
15	17	loadAI	r_{arp}, @d ⇒ r_2	// φορτώνουμε την «d»
18	19	mult	r_1, r_2 ⇒ r_1	// $r_1 \leftarrow (a \times 2 \times b \times c) \times d$
20	22	storeAI	r_1 ⇒ r_{arp}, @a	// γράφουμε τον r_a στην «a»

Για την εκτέλεση αυτής της ακολουθίας εννέα πράξεων απαιτούνται 22 κύκλοι. Η ελαχιστοποίηση της χρήσης καταχωρητών δεν απέφερε γρήγορη εκτέλεση.

Πολλοί επεξεργαστές έχουν μια ιδιότητα που τους επιτρέπει να εκκινούν νέες πράξεις ενόσω εκτελείται κάποια πράξη μεγάλου χρόνου απόκρισης. Αν δεν υπάρξει καμία παραπομπή στα αποτελέσματα της πράξης μεγάλου χρόνου απόκρισης μέχρι την ολοκλήρωση της πράξης, η εκτέλεση προχωρά κανονικά. Αν, όμως, κάποια παρεμβαλλόμενη πράξη προσπαθήσει να διαβάσει πρόωρα το αποτέλεσμα της πράξης μεγάλου χρόνου απόκρισης, ο επεξεργαστής καθυστερεί την πράξη που χρειάζεται την τιμή, μέχρι να ολοκληρωθεί η πράξη μεγάλου χρόνου απόκρισης. Μια πράξη δεν μπορεί να ξεκινήσει να εκτελείται προτού να είναι έτοιμοι οι τελεστέοι της, και τα αποτελέσματά της δεν είναι έτοιμα πριν από τον τερματισμό της πράξης.

Ο χρονοπρογραμματιστής οδηγιών αναδιατάσσει τις πράξεις του κώδικα. Προσπαθεί να ελαχιστοποιήσει το πλήθος των κύκλων που αναλώνονται σε αναμονή τελεστέων. Ασφαλώς, ο χρονοπρογραμματιστής πρέπει να διασφαλίζει ότι η νέα ακολουθία παράγει το ίδιο αποτέλεσμα με την αρχική. Σε πολλές περιπτώσεις, ο χρονοπρογραμματιστής μπορεί να βελτιώσει δραστικά την επίδοση «απλοϊκού» κώδικα. Για το παράδειγμά μας, ένας καλός χρονοπρογραμματιστής θα μπορούσε να παραγάγει την εξής ακολουθία:

Έναρξη	Λήξη			
1	3	loadAI	r_{arp}, @a ⇒ r_1	// φορτώνουμε την «a»
2	4	loadAI	r_{arp}, @b ⇒ r_2	// φορτώνουμε την «b»
3	5	loadAI	r_{arp}, @c ⇒ r_3	// φορτώνουμε την «c»
4	4	add	r_1, r_1 ⇒ r_1	// $r_1 \leftarrow a \times 2$
5	6	mult	r_1, r_2 ⇒ r_1	// $r_1 \leftarrow (a \times 2) \times b$
6	8	loadAI	r_{arp}, @d ⇒ r_2	// φορτώνουμε την «d»
7	8	mult	r_1, r_3 ⇒ r_1	// $r_1 \leftarrow (a \times 2 \times b) \times c$
9	10	mult	r_1, r_2 ⇒ r_1	// $r_1 \leftarrow (a \times 2 \times b \times c) \times d$
11	13	storeAI	r_1 ⇒ r_{arp}, @a	// γράφουμε τον r_a στην «a»

Για την εκτέλεση αυτής της εκδοχής του κώδικα απαιτούνται μόλις 13 κύκλοι. Ο κώδικας χρησιμοποιεί έναν ακόμη καταχωρητή επιπλέον του ελάχιστου πλήθους. Εκκινεί μια πράξη σε κάθε κύκλο εκτός των 8, 10 και 12. Υπάρχουν και άλλα δυνατά ισοδύναμα χρονοδιαγράμματα, όπως και χρονοδιαγράμματα ίδιου μήκους που χρησιμοποιούν περισσότερους καταχωρητές. Στο Κεφάλαιο 12 παρουσιάζονται διάφορες ευρέως χρησιμοποιούμενες τεχνικές χρονοπρογραμματισμού.

Αλληλεπιδράσεις μεταξύ των τμημάτων παραγωγής κώδικα

Τα περισσότερα από τα πραγματικά δύσκολα προβλήματα της μεταγλώττισης ανακύπτουν στην παραγωγή κώδικα. Τα πράγματα περιπλέκονται ακόμη περισσότερο, καθώς τα προβλήματα αυτά αλληλεπιδρούν μεταξύ τους. Για παράδειγμα, ο χρονοπρογραμματισμός οδηγιών απομακρύνει τις πράξεις `load` από τις αριθμητικές πράξεις που εξαρτώνται από αυτές. Αυτό μπορεί να επιμηκύνει την περίοδο κατά την οποία χρειάζονται οι τιμές, και συνεπώς να αυξήσει το πλήθος των καταχωρητών που απαιτούνται στη διάρκεια αυτής της περιόδου. Με αντίστοιχο τρόπο, η τιμοδοσία συγκεκριμένων καταχωρητών με συγκεκριμένες τιμές

Η ΚΑΤΑΣΚΕΥΗ ΕΝΟΣ ΜΕΤΑΓΛΩΤΤΙΣΤΗ ΑΠΑΙΤΕΙ ΜΕΛΕΤΗ ΚΑΙ ΣΧΕΔΙΑΣΗ

Ένας τυπικός μεταγλωττιστής περιλαμβάνει μια σειρά διελεύσεων που, όλες μαζί, μεταφράζουν κώδικα από κάποια γλώσσα πηγής σε κάποια γλώσσα στόχου. Σε αυτή την πορεία, ο μεταγλωττιστής χρησιμοποιεί δεκάδες αλγορίθμους και δομές δεδομένων. Για κάθε βήμα της διαδικασίας, ο κατασκευαστής του μεταγλωττιστή πρέπει να επιλέξει μια κατάλληλη λύση.

Ένας επιτυχημένος μεταγλωττιστής εκτελείται απίστευτα πολλές φορές. Σκεφτείτε πόσες φορές έχει εκτελεστεί συνολικά ο μεταγλωττιστής GCC. Αν αθροίσουμε ακόμη και τις μικρές υστερήσεις αποδοτικότητας του GCC καθ' όλη τη διάρκεια της ζωής του, προκύπτει ένα σημαντικό χρονικό διάστημα. Με τον χρόνο, τα κέρδη που αποκομίζονται λόγω καλής σχεδίασης και καλής υλοποίησης μεγεθύνονται. Γι' αυτό, ο κατασκευαστής ενός μεταγλωττιστή πρέπει να δείξει προσοχή στα κόστη της μεταγλώττισης, όπως στην ασυμπτωτική πολυπλοκότητα των αλγορίθμων, στον πραγματικό χρόνο εκτέλεσης της υλοποίησης και στον χώρο που χρησιμοποιούν οι δομές δεδομένων. Ο κατασκευαστής του μεταγλωττιστή πρέπει να έχει κατά νου μια εκτίμηση για τον χρόνο που πρόκειται να δαπανά ο μεταγλωττιστής στις διάφορες εργασίες του.

Για παράδειγμα, η σάρωση και η συντακτική ανάλυση είναι δύο προβλήματα για τα οποία υπάρχουν άφθονοι αποδοτικοί αλγόριθμοι. Οι σαρωτές αναγνωρίζουν και ταξινομούν λέξεις σε χρόνο ανάλογο με το πλήθος των χαρακτήρων του προγράμματος εισόδου. Για μια συνήθη γλώσσα προγραμματισμού, ένας συντακτικός αναλυτής μπορεί να κατασκευάσει μια παραγωγή σε χρόνο ανάλογο με το μήκος της. (Η περιορισμένη δομή των γλωσσών προγραμματισμού επιτρέπει αποδοτική συντακτική ανάλυση.) Δεδομένου ότι για τη σάρωση και τη συντακτική ανάλυση υπάρχουν αποδοτικές και αποτελεσματικές τεχνικές, ο κατασκευαστής ενός μεταγλωττιστή θα πρέπει να υπολογίζει πως σε αυτές τις εργασίες θα αναλωθεί ένα μικρό τμήμα του χρόνου μεταγλώττισης.

Αντιθέτως, η βελτιστοποίηση και η παραγωγή κώδικα ενέχουν διάφορα προβλήματα που απαιτούν περισσότερο χρόνο. Πολλοί από τους αλγορίθμους ανάλυσης προγραμμάτων και βελτιστοποίησης που θα εξετάσουμε θα έχουν πολυπλοκότητα μεγαλύτερη από $O(n)$. Συνεπώς, οι αλγόριθμοι που επιλέγονται για τον βελτιστοποιητή και τη γεννήτρια κώδικα επηρεάζουν περισσότερο τον χρόνο μεταγλώττισης απ' ό,τι οι αλγόριθμοι που επιλέγονται για το μπροστινό άκρο του μεταγλωττιστή. Ο κατασκευαστής ενός μεταγλωττιστή μπορεί να χρειαστεί να κάνει συμβιβασμούς μεταξύ της ακρίβειας της ανάλυσης και της αποτελεσματικότητας της βελτιστοποίησης από τη μία και της αύξησης του χρόνου μεταγλώττισης από την άλλη. Θα πρέπει να παίρνει αυτού του είδους τις αποφάσεις συνειδητά και προσεκτικά.

μπορεί να επιβάλει περιορισμούς στον χρονοπρογραμματισμό οδηγιών δημιουργώντας μια «εσφαλμένη» εξάρτηση μεταξύ δύο πράξεων. (Η δεύτερη πράξη δεν μπορεί να χρονοπρογραμματιστεί μέχρι να ολοκληρωθεί η πρώτη, παρότι οι τιμές στον κοινό καταχωρητή είναι ανεξάρτητες. Η μετονομασία των τιμών μπορεί να εξαλείψει αυτή την εσφαλμένη εξάρτηση, με κόστος τη χρήση περισσότερων καταχωρητών.)

1.4 ΠΕΡΙΛΗΨΗ ΚΑΙ ΠΡΟΟΠΤΙΚΗ

Η κατασκευή μεταγλωττιστών είναι σύνθετη δουλειά. Ένας καλός μεταγλωττιστής συνδυάζει ιδέες από τη θεωρία τυπικών γλωσσών, τη μελέτη αλγορίθμων, την τεχνητή νοημοσύνη, τη σχεδίαση συστημάτων, την αρχιτεκτονική υπολογιστών και τη θεωρία γλωσσών προγραμματισμού, και τις εφαρμόζει στο πρόβλημα της μετάφρασης ενός προγράμματος. Ένας μεταγλωττιστής συνδυάζει άπληστους αλγορίθμους, ευρετικές τεχνικές, αλγορίθμους γραφημάτων, δυναμικό προγραμματισμό, αιτιοκρατικά και μη αιτιοκρατικά πεπερασμένα αυτόματα, αλγορίθμους σταθερού σημείου, θέματα συγχρονισμού και τοπικότητας, δέσμευση καταχωρητών και ονοματοδοσία, και διαχείριση διοχετεύσεων. Πολλά από τα προβλήματα που καλείται να λύσει ο μεταγλωττιστής είναι τόσο δύσκολα που δεν μπορούν να λυθούν με βέλτιστο τρόπο· γι' αυτό, οι μεταγλωττιστές χρησιμοποιούν προσεγγιστικές λύσεις, ευρετικές τεχνικές και εμπειρικούς κανόνες. Αυτό προκαλεί σύνθετες αλληλεπιδράσεις που μπορεί να οδηγήσουν σε αναπάντεχα αποτελέσματα – καλά και κακά.

Για να εντάξουν τις παραπάνω δραστηριότητες σε ένα δομημένο πλαίσιο, οι περισσότεροι μεταγλωττιστές συγκροτούνται από τρεις βασικές φάσεις: ένα μπροστινό άκρο, έναν βελτιστοποιητή και ένα πίσω άκρο. Κάθε φάση καλείται να αντιμετωπίσει ένα διαφορετικό σύνολο προβλημάτων· οι προσεγγίσεις που χρησιμοποιούνται για την επίλυση αυτών των προβλημάτων επίσης διαφέρουν. Το μπροστινό άκρο επικεντρώνεται στη μετάφραση του πηγαίου κώδικα σε κάποια ΕΑ. Τα μπροστινά άκρα στηρίζονται στα αποτελέσματα της θεωρίας τυπικών γλωσσών και της θεωρίας τύπων, με γενναίες δόσεις αλγορίθμων και δομών δεδομένων. Το μεσαίο τμήμα, ή αλλιώς βελτιστοποιητής, μεταφράζει ένα πρόγραμμα ΕΑ σε ένα άλλο, με στόχο να παραγάγει ένα πρόγραμμα ΕΑ που να εκτελείται αποδοτικά. Οι βελτιστοποιητές αναλύουν τα προγράμματα για να εξαγάγουν γνώση σχετικά με τη συμπεριφορά των προγραμμάτων κατά την εκτέλεση και κατόπιν χρησιμοποιούν αυτή τη γνώση για να μετασχηματίσουν τον κώδικα και να βελτιώσουν τη συμπεριφορά του. Το πίσω άκρο απεικονίζει ένα πρόγραμμα ΕΑ στο σύνολο οδηγιών ενός συγκεκριμένου επεξεργαστή. Το πίσω άκρο υπολογίζει προσεγγιστικές λύσεις σε δύσκολα προβλήματα δέσμευσης και χρονοπρογραμματισμού, και η ποιότητα των προσεγγίσεων που υπολογίζει επηρεάζει άμεσα την ταχύτητα και το μέγεθος του μεταγλωττισμένου κώδικα.

Σε αυτό το βιβλίο μελετώνται όλες αυτές οι φάσεις. Στα Κεφάλαια 2 ώς 4 μελετώνται οι αλγόριθμοι που χρησιμοποιεί το μπροστινό άκρο ενός μεταγλωττιστή. Στα Κεφάλαια 5 ώς 7 παρουσιάζεται το γνωστικό υπόβαθρο που απαιτείται για τη μελέτη της βελτιστοποίησης και της παραγωγής κώδικα. Το Κεφάλαιο 8 αποτελεί μια εισαγωγή στη βελτιστοποίηση κώδικα. Στα Κεφάλαια 9 και 10 εξετάζεται με μεγαλύτερη λεπτομέρεια η ανάλυση και η βελτιστοποίηση για τον ενδιαφερόμενο

αναγνώστη. Τέλος, στα Κεφάλαια 11 ώς 13 καλύπτονται οι τεχνικές που χρησιμοποιούνται από τα πίσω άκρα για την επιλογή οδηγιών, τον χρονοπρογραμματισμό και τη δέσμευση καταχωρητών.

ΣΗΜΕΙΩΣΕΙΣ ΚΕΦΑΛΑΙΟΥ

Οι πρώτοι μεταγλωττιστές εμφανίστηκαν τη δεκαετία του 1950. Τα πρώτα αυτά συστήματα ήταν εξαιρετικά σύνθετα. Ο αρχικός μεταγλωττιστής της FORTRAN ήταν ένα πολυδιελευσιακό σύστημα που περιλάμβανε ξεχωριστό σαρωτή, συντακτικό αναλυτή και εκχωρητή καταχωρητών, μαζί με κάποιες βελτιστοποιήσεις [27, 26]. Το σύστημα ALPHA, που κατασκευάστηκε από τον Ershov και τους συνεργάτες του, πραγματοποιούσε τοπική βελτιστοποίηση [139] και χρησιμοποιούσε χρωματισμό γραφημάτων για να ελαττώνει την ποσότητα της απαιτούμενης μνήμης για τα στοιχεία των δεδομένων [140, 141].

Ο Knuth μάς θυμίζει μερικά ενδιαφέροντα πράγματα από την κατασκευή μεταγλωττιστών στις αρχές τις δεκαετίας του 1960 [227]. Οι Randell και Russell περιγράφουν τις πρώτες προσπάθειες υλοποίησης της ALGOL 60 [293]. Ο Allen περιγράφει την ιστορία της ανάπτυξης μεταγλωττιστών μέσα στην IBM δίνοντας έμφαση στην αλληλεπίδραση θεωρίας και πράξης [14].

Πολλοί σημαντικοί μεταγλωττιστές κατασκευάστηκαν τις δεκαετίες του 1960 και 1970. Σε αυτούς περιλαμβάνονται ο κλασικός βελτιστοποιητικός μεταγλωττιστής FORTRAN H [252, 307], οι μεταγλωττιστές της BLISS-11 και BLISS-32 [356, 72], και ο φορητός μεταγλωττιστής της BCPL [300]. Αυτοί οι μεταγλωττιστές παρήγαν υψηλής ποιότητας κώδικα για διάφορες μηχανές CISC. Από την άλλη, οι μεταγλωττιστές που απευθύνονταν σε φοιτητές έδιναν έμφαση στη γρήγορη μεταγλώττιση, την παραγωγή καλών διαγνωστικών μηνυμάτων και τη διόρθωση σφαλμάτων [97, 146].

Η έλευση της αρχιτεκτονικής RISC τη δεκαετία του 1980 οδήγησε σε μια νέα γενιά μεταγλωττιστών, οι οποίοι έδιναν έμφαση στην ισχυρή βελτιστοποίηση και την παραγωγή κώδικα [89, 24, 81, 204]. Αυτοί οι μεταγλωττιστές περιλάμβαναν μεγάλης κλίμακας βελτιστοποιητές δομημένους όπως στο Σχήμα 1.1. Οι σύγχρονοι μεταγλωττιστές RISC ακολουθούν ακόμη αυτό το μοντέλο.

Τη δεκαετία του 1990, η έρευνα πάνω στην κατασκευή μεταγλωττιστών επικεντρώθηκε στην προσαρμογή στις γρήγορες αλλαγές που συντελούνταν στην αρχιτεκτονική των μικροεπεξεργαστών. Η δεκαετία ξεκίνησε με τον επεξεργαστή *i*860 της Intel, που υποχρέωσε τους κατασκευαστές μεταγλωττιστών να διαχειριστούν άμεσα τις διοχετεύσεις και τους χρόνους απόκρισης της μνήμης. Στο τέλος της δεκαετίας, οι μεταγλωττιστές αντιμετώπισαν προκλήσεις όπως οι πολλές λειτουργικές μονάδες, οι μεγάλοι χρόνοι απόκρισης της μνήμης και η παραγωγή παράλληλου κώδικα. Η δομή και η οργάνωση των μεταγλωττιστών RISC της δεκαετίας του 1980 αποδείχθηκε αρκετά ευέλικτη για την αντιμετώπιση των νέων αυτών προκλήσεων, και έτσι οι ερευνητές εισήγαγαν νέες διελεύσεις τις οποίες συμπεριέλαβαν στους βελτιστοποιητές και στις γεννήτριες κώδικα των μεταγλωττιστών τους.

Παρότι τα συστήματα JAVA χρησιμοποιούν έναν συνδυασμό μεταγλώττισης και διερμήνευσης [63, 279], η JAVA δεν είναι η πρώτη γλώσσα που χρησιμοποιεί

έναν τέτοιο συνδυασμό. Τα συστήματα LISP είχαν περιλάβει προ πολλού μεταγλωττιστές εγγενούς κώδικα και στρατηγικές υλοποίησης για εικονικές μηχανές [324, 266]. Το σύστημα SMALLTALK 80 χρησιμοποιούσε μια διανομή οκταδυφιακού κώδικα και μια εικονική μηχανή [233]· διάφορες υλοποιήσεις προσέθεσαν μεταγλωττιστές τελευταίας στιγμής [126].

ΑΣΚΗΣΕΙΣ

1. Θεωρήστε ένα απλό πρόγραμμα ιστοπεριήγησης που παίρνει ως είσοδο μια συμβολοσειρά κειμένου σε μορφή HTML και απεικονίζει τα περιγραφόμενα γραφικά στην οθόνη. Η διεργασία απεικόνισης είναι μεταγλώττιση ή διερμήνευση;
2. Στη σχεδίαση ενός μεταγλωττιστή θα πρέπει να κάνετε πολλούς συμβιβασμούς. Ποιες είναι οι πέντε ιδιότητες που εσείς, ως χρήστης, θεωρείτε πιο σημαντικές σε έναν μεταγλωττιστή που αγοράζετε; Αλλάζει αυτή η λίστα αν κατασκευαστής του μεταγλωττιστή είστε εσείς; Σε τι συμπεράσματα σας οδηγεί αυτή η λίστα για κάποιον μεταγλωττιστή που πρόκειται να υλοποιήσετε;
3. Οι μεταγλωττιστές χρησιμοποιούνται σε πολλές διαφορετικές περιστάσεις. Ποιες διαφορές θα περιμένατε να έχουν οι μεταγλωττιστές που είναι σχεδιασμένοι για τις παρακάτω εφαρμογές;
   α. Ένας μεταγλωττιστής τελευταίας στιγμής που χρησιμοποιείται για τη μετάφραση κώδικα διεπαφής εργασίας που καταφορτώνεται μέσω ενός δικτύου.
   β. Ένας μεταγλωττιστής που προορίζεται για τον ενσωματωμένο επεξεργαστή ενός κινητού τηλεφώνου.
   γ. Ένας μεταγλωττιστής που χρησιμοποιείται σε ένα εισαγωγικό μάθημα προγραμματισμού στο λύκειο.
   δ. Ένας μεταγλωττιστής που χρησιμοποιείται για την κατασκευή προσομοιώσεων αιολικών σηράγγων οι οποίες εκτελούνται σε έναν επεξεργαστή υψηλής παραλληλίας (αποτελούμενο από όμοιους επεξεργαστές).
   ε. Ένας μεταγλωττιστής που προορίζεται για προγράμματα που εκτελούν πολλούς αριθμητικούς υπολογισμούς σε μεγάλο πλήθος διαφορετικών μηχανών.

Κεφάλαιο 2

Σαρωτές

ΕΠΙΣΚΟΠΗΣΗ ΚΕΦΑΛΑΙΟΥ

Δουλειά του σαρωτή είναι να μετασχηματίζει ένα ρεύμα χαρακτήρων σε ένα ρεύμα λέξεων της γλώσσας εισόδου. Κάθε λέξη πρέπει να ταξινομηθεί σε κάποια συντακτική κατηγορία, ή αλλιώς «μέρος του λόγου». Ο σαρωτής είναι η μόνη διέλευση του μεταγλωττιστή που έρχεται σε άμεση επαφή με κάθε χαρακτήρα του προγράμματος εισόδου. Οι συγγραφείς μεταγλωττιστών θεωρούν ιδιαίτερα σημαντική την ταχύτητα της σάρωσης, εν μέρει διότι η είσοδος του σαρωτή είναι μεγαλύτερη, σε κάποιο βαθμό, από την είσοδο οποιασδήποτε άλλης διέλευσης, και εν μέρει διότι υπάρχουν ιδιαίτερα αποδοτικές τεχνικές που κανείς μπορεί να κατανοήσει και να υλοποιήσει εύκολα.

Σε αυτό το κεφάλαιο εισάγονται οι κανονικές εκφράσεις (ένας συμβολισμός που χρησιμοποιείται για την περιγραφή των έγκυρων λέξεων μιας γλώσσας προγραμματισμού), και αναπτύσσονται οι τυπικοί μηχανισμοί δημιουργίας σαρωτών από κανονικές εκφράσεις, είτε χειρωνακτικά είτε αυτόματα.

Λέξεις-κλειδιά: σαρωτής, πεπερασμένο αυτόματο, κανονική έκφραση, σταθερό σημείο

2.1 ΕΙΣΑΓΩΓΗ

Η σάρωση είναι το πρώτο στάδιο μιας διαδικασίας τριών τμημάτων που χρησιμοποιεί ο μεταγλωττιστής για να κατανοήσει το πρόγραμμα εισόδου. Ο σαρωτής, ή λεκτικός αναλυτής, διαβάζει ένα ρεύμα χαρακτήρων και παράγει ένα ρεύμα λέξεων. Σωρεύει χαρακτήρες για να σχηματίσει λέξεις και εφαρμόζει ένα σύνολο κανόνων για να διαπιστώσει αν κάθε λέξη της γλώσσας πηγής είναι έγκυρη ή όχι. Αν η λέξη είναι έγκυρη, ο σαρωτής την ταξινομεί σε μια συντακτική κατηγορία, ή αλλιώς μέρος του λόγου.

Ο σαρωτής είναι η μόνη διέλευση του μεταγλωττιστή που ασχολείται με κάθε χαρακτήρα του προγράμματος εισόδου. Επειδή οι σαρωτές εκτελούν μια σχετικά απλή εργασία, την ομαδοποίηση χαρακτήρων για τον σχηματισμό λέξεων και σημείων στίξης της γλώσσας πηγής, επιδέχονται γρήγορες υλοποιήσεις. Υπάρχουν πολλά αυτόματα εργαλεία δημιουργίας σαρωτών. Τα εργαλεία αυτά επεξεργάζονται μια μαθηματική περιγραφή της λεκτικής σύνταξης της γλώσσας και παράγουν έναν γρήγορο αναγνωριστή. Από την άλλη, πολλοί μεταγλωττιστές χρησιμοποιούν χειρωνακτικά κατασκευασμένους σαρωτές· επειδή πρόκειται για απλή εργασία, αυτού του είδους οι σαρωτές μπορούν να είναι γρήγοροι και ευσταθείς.

Δομή του κεφαλαίου

Σε αυτό το κεφάλαιο περιγράφονται τα μαθηματικά εργαλεία και οι προγραμματιστικές τεχνικές που χρησιμοποιούνται συνήθως για την κατασκευή σαρωτών – τόσο των αυτόματα κατασκευαζόμενων όσο και εκείνων που κατασκευάζονται χειρωνακτικά. Το κεφάλαιο ξεκινά με την Ενότητα 2.2, όπου εισάγεται ένα μοντέλο για τους *αναγνωριστές*, οι οποίοι είναι προγράμματα που αναγνωρίζουν λέξεις σε ένα ρεύμα χαρακτήρων. Στην Ενότητα 2.3 περιγράφονται οι *κανονικές εκφράσεις*, ένας τυπικός συμβολισμός για τον ορισμό της σύνταξης. Στην Ενότητα 2.4 παρουσιάζουμε ένα σύνολο αλγορίθμων για τη μετατροπή μιας κανονικής έκφρασης σε αναγνωριστή. Τέλος, στην Ενότητα 2.5 παρουσιάζουμε τρεις διαφορετικούς τρόπους υλοποίησης ενός σαρωτή: τον πινακοκατευθυνόμενο σαρωτή, τον απευθείας κωδικογραφούμενο σαρωτή και τον χειρωνακτικά κωδικογραφούμενο σαρωτή.

Αναγνωριστής
Ένα πρόγραμμα που αναγνωρίζει συγκεκριμένες λέξεις σε ένα ρεύμα χαρακτήρων.

Τόσο οι αυτόματα όσο και οι χειρωνακτικά κατασκευαζόμενοι σαρωτές στηρίζονται στις ίδιες γενικές τεχνικές. Παρότι τα περισσότερα εγχειρίδια και μαθήματα συνιστούν τη χρήση αυτόματα κατασκευασμένων σαρωτών, οι περισσότεροι εμπορικοί μεταγλωττιστές και μεταγλωττιστές ανοιχτού κώδικα χρησιμοποιούν χειρωνακτικά κατασκευασμένους σαρωτές. Ένας χειρωνακτικά κατασκευασμένος σαρωτής μπορεί να είναι ταχύτερος από έναν αυτόματα κατασκευασμένο, διότι η υλοποίηση μπορεί να βελτιστοποιηθεί ώστε να εξαλειφθεί μέρος των επιβαρύνσεων που δεν μπορούν να αποφευχθούν στους αυτόματα κατασκευαζόμενους σαρωτές. Επειδή οι σαρωτές είναι απλοί και δεν αλλάζουν συχνά, πολλοί συγγραφείς μεταγλωττιστών θεωρούν ότι η καλύτερη επίδοση που προσφέρει ένας χειρωνακτικά κατασκευασμένος σαρωτής είναι σημαντικότερη από την ευκολία που προσφέρει η αυτόματη δημιουργία του σαρωτή. Θα εξετάσουμε και τις δύο εναλλακτικές προσεγγίσεις.

Επισκόπηση

Ο σαρωτής ενός μεταγλωττιστή διαβάζει ένα ρεύμα εισόδου που αποτελείται από χαρακτήρες και παράγει ένα ρεύμα εξόδου που περιέχει λέξεις, καθεμιά από τις οποίες επιγράφεται με τη *συντακτική κατηγορία* της – το ισοδύναμο του μέρους του λόγου μιας λέξης στα ελληνικά. Για να πραγματοποιήσει τη σώρευση των χαρακτήρων και την ταξινόμηση των λέξεων σε κατηγορίες, ο σαρωτής εφαρμόζει ένα σύνολο κανόνων που περιγράφουν τη λεκτική δομή της γλώσσας προγραμματισμού εισόδου, το οποίο μερικές φορές ονομάζεται *μικροσύνταξη*. Η μικροσύνταξη μιας γλώσσας προγραμματισμού καθορίζει με ποιον τρόπο μπορούν να ομαδοποιηθούν οι χαρακτήρες για να σχηματιστούν λέξεις και, αντιστρόφως, με ποιον τρόπο διαχωρίζονται οι λέξεις που εμφανίζονται η μία μετά την άλλη. (Στα πλαίσια της σάρωσης, θεωρούμε τα σημεία στίξης και τα άλλα σύμβολα λέξεις.)

Συντακτική κατηγορία
Η κατηγορία στην οποία ταξινομείται μια λέξη με βάση τη γραμματική της χρήση.

Μικροσύνταξη
Η λεκτική δομή μιας γλώσσας.

Οι δυτικές γλώσσες, όπως τα αγγλικά, έχουν απλή μικροσύνταξη. Οι παρακείμενοι αλφαβητικοί χαρακτήρες (γράμματα) ομαδοποιούνται, από τα αριστερά προς τα δεξιά, ώστε να σχηματιστεί μια λέξη. Το κενό σημαίνει το τέλος μιας λέξης, όπως και τα περισσότερα μη αλφαβητικά σύμβολα. (Ο αλγόριθμος κατασκευής λέξεων μπορεί να μεταχειριστεί το ενωτικό στο μέσο μιας λέξης σαν να ήταν αλφαβητικός χαρακτήρας.) Μόλις σωρευτεί μια ομάδα χαρακτήρων και σχηματιστεί

μια εν δυνάμει λέξη, ο αλγόριθμος κατασκευής λέξεων μπορεί να διαπιστώσει την εγκυρότητά της εκτελώντας αναζήτηση σε ένα λεξικό.

Οι περισσότερες γλώσσες προγραμματισμού έχουν εξίσου απλή μικροσύνταξη. Οι χαρακτήρες σωρεύονται σε λέξεις. Στις περισσότερες γλώσσες, τα κενά και τα σημεία στίξης σημαίνουν το τέλος μιας λέξης. Για παράδειγμα, η ALGOL και οι απόγονοί της ορίζουν ως *αναγνωριστικό* έναν αλφαβητικό χαρακτήρα ακολουθούμενο από μηδέν ή περισσότερους αλφαριθμητικούς χαρακτήρες. Το αναγνωριστικό τελειώνει με τον πρώτο μη αλφαριθμητικό χαρακτήρα. Συνεπώς, τα `fee` και `f1e` είναι έγκυρα αναγνωριστικά, ενώ το `12fum` δεν είναι. Προσέξτε ότι το σύνολο των έγκυρων λέξεων καθορίζεται μέσω κανόνων και όχι μέσω απαρίθμησης σε κάποιο λεξικό.

Σε μια τυπική γλώσσα προγραμματισμού, μερικές λέξεις, που ονομάζονται *κλειδωνύμια* ή *δεσμευμένες λέξεις*, συμμορφώνονται με τον κανόνα ορισμού των αναγνωριστικών αλλά έχουν ειδικές σημασίες. Τα `while` και `static` είναι αμφότερα κλειδωνύμια και στη C και την JAVA. Τα κλειδωνύμια (και τα σημεία στίξης) σχηματίζουν τις δικές τους ιδιαίτερες συντακτικές κατηγορίες. Παρότι το `static` συμμορφώνεται με τον κανόνα ορισμού των αναγνωριστικών, ο σαρωτής ενός μεταγλωττιστή της C ή της JAVA θα το ταξινομήσει αναμφίβολα σε μια κατηγορία που έχει μόνο ένα στοιχείο, το κλειδωνύμιο `static`. Για να αναγνωρίσει τα κλειδωνύμια, ο σαρωτής μπορεί είτε να εκτελέσει αναζήτηση σε λεξικό είτε να κωδικοποιήσει απευθείας τα κλειδωνύμια στους κανόνες της μικροσύνταξης.

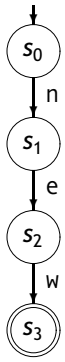
Κλειδωνύμιο
Μια λέξη που είναι δεσμευμένη για έναν συγκεκριμένο συντακτικό σκοπό, και συνεπώς δεν μπορεί να χρησιμοποιηθεί ως αναγνωριστικό.

Η απλή λεκτική δομή των γλωσσών προγραμματισμού προσφέρεται για την κατασκευή αποδοτικών σαρωτών. Ο συγγραφέας ενός μεταγλωττιστή ξεκινά από έναν ορισμό της μικροσύνταξης της γλώσσας και είτε κωδικοποιεί τη μικροσύνταξη χρησιμοποιώντας κάποιον συμβολισμό αποδεκτό από μια γεννήτρια σαρωτών, η οποία στη συνέχεια κατασκευάζει έναν εκτελέσιμο σαρωτή, είτε χρησιμοποιεί τον συγκεκριμένο ορισμό για να κατασκευάσει χειρωνακτικά έναν σαρωτή. Τόσο οι αυτόματα όσο και οι χειρωνακτικά κατασκευαζόμενοι σαρωτές μπορούν να υλοποιηθούν έτσι ώστε να απαιτούν χρόνο μόλις $\mathbf{O}(1)$ ανά χαρακτήρα· συνεπώς εκτελούνται σε χρόνο ανάλογο με το πλήθος των χαρακτήρων του ρεύματος εισόδου.

2.2 ΑΝΑΓΝΩΡΙΣΗ ΛΕΞΕΩΝ

Ο απλούστερος τρόπος να εξηγηθεί ένας αλγόριθμος αναγνώρισης λέξεων συνήθως είναι να παρουσιαστεί χαρακτήρα-προς-χαρακτήρα. Η δομή του κώδικα μπορεί να δώσει μια εικόνα του προβλήματος που κρύβεται από πίσω. Ας θεωρήσουμε το πρόβλημα αναγνώρισης του κλειδωνυμίου `new`. Αν υποθέσουμε ότι υπάρχει μια ρουτίνα `ΕπόμενοςΧαρακτήρας` που επιστρέφει τον επόμενο χαρακτήρα, ο κώδικας μπορεί να έχει τη μορφή του αποσπάσματος που παρουσιάζεται στο Σχήμα 2.1. Ο κώδικας ελέγχει αν υπάρχει κάποιο `n` ακολουθούμενο από ένα `e` ακολουθούμενο από ένα `w`. Σε κάθε βήμα, αν η συμμόρφωση με τον κατάλληλο χαρακτήρα δεν καταστεί δυνατή, ο κώδικας απορρίπτει τη συμβολοσειρά και «δοκιμάζει κάτι άλλο». Αν μόνος σκοπός του προγράμματος ήταν να αναγνωρίσει τη λέξη `new`, θα έπρεπε να τυπώσει ένα μήνυμα σφάλματος ή να επιστρέψει αποτυχία. Επειδή οι σαρωτές σπανίως αναγνωρίζουν μόνο μία λέξη, θα αφήσουμε αυτή τη «διαδρομή σφάλματος» σκοπίμως αόριστη σε αυτό το σημείο.

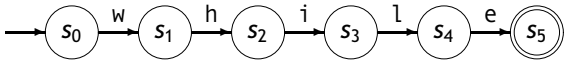
```
c ← ΕπόμενοςΧαρακτήρας()
αν (c = «n»)
   τότε αρχή
      c ← ΕπόμενοςΧαρακτήρας()
      αν (c = «e»)
         τότε αρχή
            c ← ΕπόμενοςΧαρακτήρας()
            αν (c = «w»)
               τότε αναφέρουμε επιτυχία
               άλλως δοκιμάζουμε κάτι άλλο
         τέλος
         άλλως δοκιμάζουμε κάτι άλλο
   τέλος
   άλλως δοκιμάζουμε κάτι άλλο
```



ΣΧΗΜΑ 2.1 Απόσπασμα κώδικα για την αναγνώριση της «new».

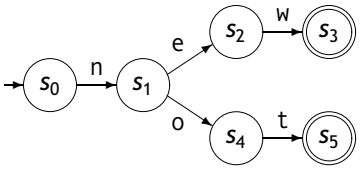
Το συγκεκριμένο απόσπασμα κώδικα πραγματοποιεί έναν έλεγχο ανά χαρακτήρα. Το απόσπασμα κώδικα μπορεί να αναπαρασταθεί με το απλό διάγραμμα μεταβάσεων που παρουσιάζεται στα δεξιά του κώδικα. Το διάγραμμα μεταβάσεων αναπαριστά έναν αναγνωριστή. Κάθε κύκλος αναπαριστά μια αφηρημένη κατάσταση του υπολογισμού. Για ευκολία, κάθε κατάσταση φέρει μια επιγραφή. Η αρχική, ή εναρκτήρια κατάσταση είναι η s_0. Θα επιγράφουμε πάντα την ενακτήρια κατάσταση με s_0. Η κατάσταση s_3 είναι κατάσταση αποδοχής· ο αναγνωριστής φτάνει στην s_3 μόνο όταν η είσοδος είναι new. Όπως φαίνεται στο περιθώριο, οι καταστάσεις αποδοχής σχεδιάζονται με διπλούς κύκλους. Τα βέλη αναπαριστούν μεταβάσεις από κατάσταση σε κατάσταση με βάση τον χαρακτήρα εισόδου. Αν ο αναγνωριστής ξεκινήσει από την s_0 και διαβάσει τους χαρακτήρες n, e και w, οι μεταβάσεις τον μεταφέρουν στην s_3. Τι θα συμβεί με οποιαδήποτε άλλη είσοδο, όπως με την n, o και t; Το n μεταφέρει τον αναγνωριστή στην s_1. Το o δεν συμμορφώνεται με την ακμή που εξέρχεται από την s_1, άρα η λέξη εισόδου δεν είναι η new. Στον κώδικα, στις περιπτώσεις όπου η είσοδος δεν συμμορφώνεται με το new *δοκιμάζεται κάτι άλλο*. Στον αναγνωριστή, μπορούμε θα θεωρήσουμε αυτή την ενέργεια ως μετάβαση σε κάποια κατάσταση σφάλματος. Όταν σχεδιάζουμε το διάγραμμα μεταβάσεων ενός αναγνωριστή, συνήθως παραλείπουμε τις μεταβάσεις προς την κατάσταση σφάλματος. Κάθε κατάσταση έχει μια μετάβαση προς την κατάσταση σφάλματος για κάθε μη προσδιοριζόμενη είσοδο.

Αν χρησιμοποιήσουμε την ίδια προσέγγιση για να κατασκευάσουμε έναν αναγνωριστή για την while, θα προκύψει το ακόλουθο διάγραμμα μεταβάσεων:



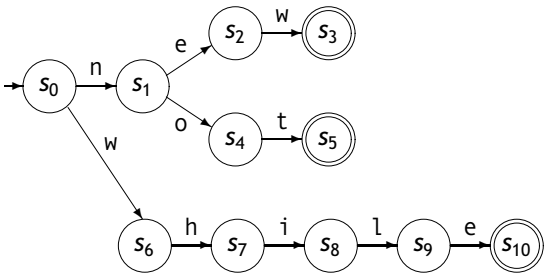
Αν ξεκινήσει από την s_0 και φτάσει στην s_5, θα έχει αναγνωρίσει τη λέξη while. Το αντίστοιχο απόσπασμα κώδικα θα περιλάμβανε πέντε ένθετες δομές αν-τότε-άλλως.

Για να αναγνωρίσουμε πολλές λέξεις, μπορούμε να κατασκευάσουμε πολλές ακμές που να εξέρχονται από μια δεδομένη κατάσταση. (Στον κώδικα, θα αρχίζαμε να αναλύουμε τις διαδρομές *δοκιμάζουμε κάτι άλλο*.) Ένας αναγνωριστής για τις `new` και `not` θα μπορούσε να είναι ο



Ο αναγνωριστής πραγματοποιεί έναν κοινό έλεγχο για το `n`, ο οποίος τον μεταφέρει από την s_0 στην s_1· συμβολίζουμε αυτή τη μετάβαση με $s_0 \xrightarrow{\texttt{n}} s_1$. Αν ο επόμενος χαρακτήρας είναι το `e`, πραγματοποιεί τη μετάβαση $s_1 \xrightarrow{\texttt{e}} s_2$. Αν, αντιθέτως, ο επόμενος χαρακτήρας είναι το `o`, πραγματοποιεί τη μετάβαση $s_1 \xrightarrow{\texttt{o}} s_4$. Τέλος, ένα `w` στην s_2 έχει ως αποτέλεσμα τη μετάβαση $s_2 \xrightarrow{\texttt{w}} s_3$, ενώ ένα `t` στην s_4 προκαλεί την $s_4 \xrightarrow{\texttt{t}} s_5$. Η κατάσταση s_3 υποδηλώνει ότι η είσοδος ήταν `new`, ενώ η s_5 υποδηλώνει ότι ήταν `not`. Ο αναγνωριστής πραγματοποιεί μία μετάβαση ανά χαρακτήρα εισόδου.

Μπορούμε να συνδυάσουμε τον αναγνωριστή για τις `new` και `not` με τον αναγνωριστή για την `while` συγχωνεύοντας τις αρχικές τους καταστάσεις και δίνοντας νέες επιγραφές σε όλες τις καταστάσεις.



Η κατάσταση s_0 διαθέτει μεταβάσεις για τα `n` και `w`. Ο αναγνωριστής έχει τρεις καταστάσεις αποδοχής, τις s_3, s_5 και s_{10}. Αν σε οποιαδήποτε κατάσταση συναντήσει έναν χαρακτήρα εισόδου που δεν συμμορφώνεται με κάποια από τις μεταβάσεις της, ο αναγνωριστής μεταβαίνει σε μια κατάσταση σφάλματος.

2.2.1 Ένας φορμαλισμός για αναγνωριστές

Τα διαγράμματα μεταβάσεων αποτελούν αφηρημένες αναπαραστάσεις του κώδικα που απαιτείται για την υλοποίησή τους. Μπορούν να ιδωθούν και ως τυπικά μαθηματικά αντικείμενα ορισμού αναγνωριστών, τα οποία ονομάζονται *πεπερασμένα αυτόματα*. Τυπικά, ένα πεπερασμένο αυτόματο (ΠΑ) είναι μια πεντάδα (S, Σ, δ, s_0, S_A), όπου:

Πεπερασμένο αυτόματο
Ένας φορμαλισμός για αναγνωριστές που διαθέτει ένα πεπερασμένο σύνολο καταστάσεων, ένα αλφάβητο, μια συνάρτηση μεταβάσεων, μια εναρκτήρια κατάσταση και μία ή περισσότερες καταστάσεις αποδοχής.

- S είναι το πεπερασμένο σύνολο καταστάσεων του αναγνωριστή, μαζί με μια κατάσταση σφάλματος s_e.

- Σ είναι το πεπερασμένο αλφάβητο που χρησιμοποιεί ο αναγνωριστής. Συνήθως, το Σ είναι η ένωση των επιγραφών των ακμών του διαγράμματος μεταβάσεων.
- $\delta(s, c)$ είναι η συνάρτηση μεταβάσεων του αναγνωριστή. Απεικονίζει κάθε κατάσταση $s \in S$ και κάθε χαρακτήρα $c \in \Sigma$ σε μια επόμενη κατάσταση. Στην κατάσταση s_i με χαρακτήρα εισόδου το c, το ΠΑ πραγματοποιεί τη μετάβαση $s_i \xrightarrow{c} \delta(s_i, c)$.
- $s_0 \in S$ είναι η προκαθορισμένη εναρκτήρια κατάσταση.
- S_A είναι το σύνολο των καταστάσεων αποδοχής, με $S_A \subseteq S$. Κάθε κατάσταση του S_A εμφανίζεται στο διάγραμμα μεταβάσεων ως διπλός κύκλος.

Για παράδειγμα, σε αυτό τον φορμαλισμό μπορούμε να αναπαραστήσουμε το ΠΑ για τις *new*, *not* και *while* ως εξής:

$$S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_e\}$$

$$\Sigma = \{\mathtt{e}, \mathtt{h}, \mathtt{i}, \mathtt{l}, \mathtt{n}, \mathtt{o}, \mathtt{t}, \mathtt{w}\}$$

$$\delta = \begin{Bmatrix} s_0 \xrightarrow{\mathtt{n}} s_1, & s_0 \xrightarrow{\mathtt{w}} s_6, & s_1 \xrightarrow{\mathtt{e}} s_2, & s_1 \xrightarrow{\mathtt{o}} s_4, & s_2 \xrightarrow{\mathtt{w}} s_3, \\ s_4 \xrightarrow{\mathtt{t}} s_5, & s_6 \xrightarrow{\mathtt{h}} s_7, & s_7 \xrightarrow{\mathtt{i}} s_8, & s_8 \xrightarrow{\mathtt{l}} s_9, & s_9 \xrightarrow{\mathtt{e}} s_{10} \end{Bmatrix}$$

$$s_0 = s_0$$

$$S_A = \{s_3, s_5, s_{10}\}$$

Για όλους τους υπόλοιπους συνδυασμούς κατάστασης s_i και χαρακτήρα εισόδου c, ορίζουμε ότι $\delta(s_i, c) = s_e$, όπου s_e είναι η προκαθορισμένη κατάσταση σφάλματος. Αυτή η πεντάδα είναι ισοδύναμη με το διάγραμμα μεταβάσεων· όταν γνωρίζουμε το ένα από τα δύο, μπορούμε εύκολα να ανακατασκευάσουμε το άλλο. Το διάγραμμα μεταβάσεων είναι μια απεικόνιση του αντίστοιχου ΠΑ.

Ένα ΠΑ αποδέχεται μια συμβολοσειρά x αν και μόνο αν, με αφετηρία την s_0, η ακολουθία χαρακτήρων της συμβολοσειράς αναγκάσει το ΠΑ να πραγματοποιήσει μια σειρά μεταβάσεων οι οποίες, όταν εξαντληθεί ολόκληρη η συμβολοσειρά, το αφήνουν σε μια κατάσταση αποδοχής. Αυτό συμφωνεί με τον τρόπο με τον οποίο αντιλαμβανόμαστε διαισθητικά το διάγραμμα μεταβάσεων. Για τη συμβολοσειρά `new`, ο αναγνωριστής του παραδείγματός μας πραγματοποιεί τις μεταβάσεις $s_0 \xrightarrow{\mathtt{n}} s_1$, $s_1 \xrightarrow{\mathtt{e}} s_2$ και $s_2 \xrightarrow{\mathtt{w}} s_3$. Δεδομένου ότι $s_3 \in S_A$ και δεν απομένει άλλη είσοδος, το ΠΑ αποδέχεται την `new`. Για τη συμβολοσειρά εισόδου `nut`, η συμπεριφορά είναι διαφορετική. Λόγω του `n`, το ΠΑ πραγματοποιεί τη μετάβαση $s_0 \xrightarrow{\mathtt{n}} s_1$. Λόγω του `u`, πραγματοποιεί την $s_1 \xrightarrow{\mathtt{u}} s_e$. Άπαξ και εισέλθει στην s_e, το ΠΑ παραμένει εκεί μέχρι να εξαντληθεί το ρεύμα εισόδου.

Πιο τυπικά, αν η συμβολοσειρά x αποτελείται από τους χαρακτήρες $x_1\ x_2\ x_3 \ldots x_n$, τότε το ΠΑ $(S, \Sigma, \delta, s_0, S_A)$ αποδέχεται την x αν και μόνο αν

$$\delta(\delta(\ldots \delta(\delta(\delta(s_0, x_1), x_2), x_3) \ldots, x_{n\text{-}1}), x_n) \in S_A.$$

Διαισθητικά, αυτός ο ορισμός αντιστοιχεί στην επαναληπτική εφαρμογή της δ σε ένα ζεύγος αποτελούμενο από μια κατάσταση $s \in S$ και ένα σύμβολο εισόδου x_i. Η πρώτη εφαρμογή, η $\delta(s_0, x_1)$, αντιπροσωπεύει την αρχική μετάβαση του ΠΑ από την εναρκτήρια κατάσταση s_0 για τον χαρακτήρα x_1. Η κατάσταση που προκύπτει από την $\delta(s_0, x_1)$ χρησιμοποιείται στη συνέχεια, μαζί με τον x_2, ως είσοδος

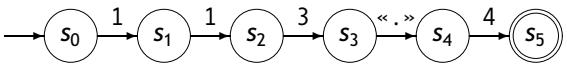
της δ για να προκύψει η επόμενη κατάσταση, κ.ο.κ. μέχρι να εξαντληθεί όλη η είσοδος. Το αποτέλεσμα της τελικής εφαρμογής της δ είναι, και πάλι, μια κατάσταση. Αν αυτή η κατάσταση είναι κατάσταση αποδοχής, τότε το ΠΑ αποδέχεται την $x_1\, x_2\, x_3 \ldots x_n$.

Υπάρχουν δύο ακόμη δυνατές περιπτώσεις. Το ΠΑ μπορεί να συναντήσει κάποιο σφάλμα κατά την επεξεργασία της συμβολοσειράς – δηλαδή, κάποιος χαρακτήρας x_j μπορεί να το οδηγήσει στην κατάσταση σφάλματος s_e. Αυτή η κατάσταση υποδηλώνει λεκτικό σφάλμα· η συμβολοσειρά $x_1\, x_2\, x_3 \ldots x_j$ δεν είναι έγκυρο πρόθημα καμίας λέξης της γλώσσας που αποδέχεται το ΠΑ. Το ΠΑ μπορεί επίσης να ανακαλύψει κάποιο σφάλμα εξαντλώντας την είσοδό του και τερματίζοντας σε μια μη αποδεκτική κατάσταση διαφορετική της s_e. Σε αυτή την περίπτωση, η συμβολοσειρά εισόδου είναι πρόθημα κάποιας λέξης που αποδέχεται το ΠΑ. Και πάλι, αυτό υποδηλώνει σφάλμα. Και τα δύο είδη σφάλματος πρέπει να αναφερθούν στον τελικό χρήστη.

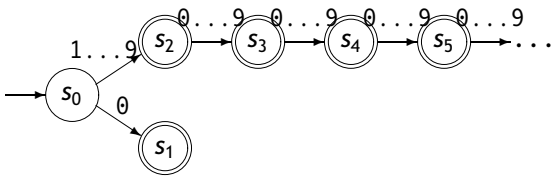
Σε κάθε περίπτωση, παρατηρήστε ότι το ΠΑ πραγματοποιεί μια μετάβαση για κάθε χαρακτήρα εισόδου. Αν υποθέσουμε ότι μπορούμε να υλοποιήσουμε το ΠΑ αποδοτικά, ο αναγνωριστής θα πρέπει να εκτελείται σε χρόνο ανάλογο με το μήκος της συμβολοσειράς εισόδου.

2.2.2 Αναγνώριση πιο σύνθετων λέξεων

Το μοντέλο χαρακτήρα-προς-χαρακτήρα που παρουσιάσαμε με τον αρχικό αναγνωριστή για την `not` μπορεί να επεκταθεί εύκολα ώστε να μπορεί να χειριστεί οποιεσδήποτε συλλογές πλήρως καθορισμένων λέξεων. Πώς θα μπορούσαμε να αναγνωρίσουμε έναν αριθμό με έναν τέτοιο αναγνωριστή; Για έναν συγκεκριμένο αριθμό, όπως για τον 113.4, αυτό είναι εύκολο.



Για να φτιάξουμε κάτι χρήσιμο, ωστόσο, χρειαζόμαστε ένα διάγραμμα μεταβάσεων (και το αντίστοιχο τμήμα κώδικα) που να μπορεί να αναγνωρίζει οποιονδήποτε αριθμό. Χάριν απλότητας, ας περιορίσουμε την ανάλυση στους απρόσημους ακέραιους. Εν γένει, ένας ακέραιος είναι είτε το μηδέν είτε μια ακολουθία ενός ή περισσότερων ψηφίων όπου το πρώτο ψηφίο είναι από το ένα έως το εννέα και τα επόμενα ψηφία από το μηδέν έως το εννέα. (Αυτός ο ορισμός αποκλείει τα αρχικά μηδενικά.) Πώς θα σχεδιάζαμε ένα διάγραμμα μεταβάσεων για αυτό τον ορισμό;



Η μετάβαση $s_0 \xrightarrow{0} s_1$ αφορά την περίπτωση του μηδενός. Η άλλη διαδρομή, από την s_0 στην s_2, στην s_3, κ.ο.κ., αφορά την περίπτωση ακεραίων που είναι μεγαλύτεροι του μηδενός. Αυτή η διαδρομή, ωστόσο, παρουσιάζει διάφορα προβλήματα. Πρώτον, δεν τελειώνει, παραβιάζοντας την υπόθεση ότι το S είναι πεπερασμένο.

```
χαρακτήρας ← ΕπόμενοςΧαρακτήρας( )
κατάσταση ← s0
ενόσω (χαρακτήρας ≠ eof και κατάσταση ≠ se)
   κατάσταση ← δ(κατάσταση,χαρακτήρας)
   χαρακτήρας ← ΕπόμενοςΧαρακτήρας( )
τέλος
αν (κατάσταση ∈ SA)
   τότε αναφέρουμε αποδοχή
   άλλως αναφέρουμε αποτυχία
```

$$S = \{s_0, s_1, s_2, s_e\}$$

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$\delta = \left\{ s_0 \xrightarrow{0} s_1, \quad s_0 \xrightarrow{1\text{-}9} s_2 \right\}$$
$$\left\{ s_2 \xrightarrow{0\text{-}9} s_2, \quad s_2 \xrightarrow{0\text{-}9} s_2 \right\}$$

$$S_A = \{s_1, s_2\}$$

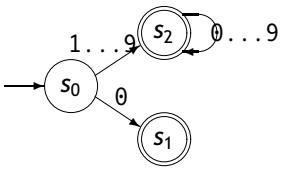
ΣΧΗΜΑ 2.2 Αναγνωριστής απρόσημων ακεραίων.

Δεύτερον, όλες οι καταστάσεις της διαδρομής που ξεκινά από την s_2 είναι ισοδύναμες, δηλαδή οι μεταβάσεις εξόδου τους έχουν τις ίδιες επιγραφές και είναι όλες καταστάσεις αποδοχής.

Αυτό το ΠΑ αναγνωρίζει μια κλάση συμβολοσειρών με μια κοινή ιδιότητα: είναι όλες απρόσημοι ακέραιοι. Αυτό αναδεικνύει το ζήτημα της διάκρισης μεταξύ των κλάσεων συμβολοσειρών και του κειμένου μιας συγκεκριμένης συμβολοσειράς. Η κλάση «απρόσημος ακέραιος» είναι μια συντακτική κατηγορία, ή αλλιώς ένα μέρος του λόγου. Το κείμενο ενός συγκεκριμένου απρόσημου ακέραιου, όπως του 113, είναι το *λέξημα* του.

Λέξημα
Το πραγματικό κείμενο μιας λέξης που αναγνωρίζει ένα ΠΑ.

Μπορούμε να απλουστεύσουμε σημαντικά το ΠΑ επιτρέποντας στο διάγραμμα μεταβάσεων να περιέχει κύκλους. Μπορούμε να αντικαταστήσουμε ολόκληρη την αλυσίδα καταστάσεων που ξεκινά από την s_2 με μια απλή μετάβαση από την s_2 στον εαυτό της:

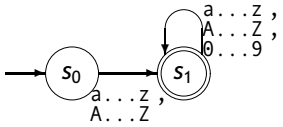


Αυτό το κυκλικό διάγραμμα μεταβάσεων έχει νόημα ως ΠΑ. Από άποψη υλοποίησης, ωστόσο, είναι πιο σύνθετο από τα άκυκλα διαγράμματα μεταβάσεων που παρουσιάσαμε προηγουμένως. Δεν μπορούμε να το μεταφράσουμε απευθείας σε ένα σύνολο ενθετών δομών αν-τότε-άλλως. Η εισαγωγή του κύκλου στο γράφημα μεταβάσεων δημιουργεί την ανάγκη για κυκλική ροή ελέγχου. Αυτό μπορεί να υλοποιηθεί με έναν βρόχο ενόσω, όπως στο Σχήμα 2.2. Η δ μπορεί να οριστεί αποδοτικά με τη βοήθεια ενός πίνακα:

δ	0	1	2	3	4	5	6	7	8	9	Άλλο
s_0	s_1	s_2	s_2	s_2	s_2	s_2	s_2	s_2	s_2	s_2	s_e
s_1	s_e	s_e	s_e	s_e	s_e	s_e	s_e	s_e	s_e	s_e	s_e
s_2	s_2	s_2	s_2	s_2	s_2	s_2	s_2	s_2	s_2	s_2	s_e
s_e	s_e	s_e	s_e	s_e	s_e	s_e	s_e	s_e	s_e	s_e	s_e

Αλλάζοντας τον πίνακα μπορούμε με τον ίδιο βασικό σκελετό του κώδικα να υλοποιήσουμε και άλλους αναγνωριστές. Παρατηρήστε ότι ο συγκεκριμένος πίνακας έχει μεγάλα περιθώρια συμπίεσης. Οι στήλες για τα ψηφία `1` έως `9` είναι πανομοιότυπες και θα μπορούσαν να αναπαρασταθούν μία φορά. Αν το κάνουμε αυτό, απομένει ένας πίνακας με τρεις στήλες: τις `0`, `1 ... 9` και *άλλο*. Αν εξετάσουμε προσεκτικά τον σκελετό του κώδικα θα διαπιστώσουμε ότι μόλις εισέλθει στην s_e αναφέρει αποτυχία, κι έτσι δεν παραπέμπει ποτέ σε αυτή τη γραμμή του πίνακα. Στην υλοποίηση μπορεί να παραλειφθεί ολόκληρη η γραμμή, ώστε να απομείνει ένας πίνακας με μόλις τρεις γραμμές και τρεις στήλες.

Μπορούμε να αναπτύξουμε παρόμοια ΠΑ για τους προσημασμένους ακέραιους, τους πραγματικούς αριθμούς και τους μιγαδικούς αριθμούς. Μια απλουστευμένη εκδοχή του κανόνα ορισμού ονομάτων αναγνωριστικών στις γλώσσες τύπου ALGOL, όπως είναι η C και η JAVA, θα μπορούσε να είναι ο εξής: *ένα αναγνωριστικό αποτελείται από έναν αλφαβητικό χαρακτήρα ακολουθούμενο από μηδέν ή περισσότερους αλφαριθμητικούς χαρακτήρες*. Το σύνολο των αναγνωριστικών που επιτρέπει αυτός ο ορισμός είναι άπειρο, αλλά μπορεί να οριστεί με το απλό ΠΑ δύο καταστάσεων που παρουσιάζεται στο περιθώριο. Σε πολλές γλώσσες προγραμματισμού η έννοια του «αλφαβητικού χαρακτήρα» επεκτείνεται, ώστε να περιλαμβάνει συγκεκριμένους ειδικούς χαρακτήρες, όπως την κάτω παύλα.



Τα ΠΑ μπορούν να ιδωθούν ως ορισμοί αναγνωριστών. Ωστόσο, δεν είναι ιδιαίτερα συνοπτικοί ορισμοί. Για να απλουστεύσουμε την υλοποίηση των σαρωτών, χρειαζόμαστε έναν συνοπτικό συμβολισμό για να ορίζουμε τη λεκτική δομή των λέξεων και έναν τρόπο για να μετατρέπουμε αυτούς τους ορισμούς σε ΠΑ και σε κώδικα που να υλοποιεί τα ΠΑ. Στις υπόλοιπες ενότητες αυτού του κεφαλαίου αναπτύσσονται αυτές ακριβώς οι ιδέες.

ΑΝΑΣΚΟΠΗΣΗ ΕΝΟΤΗΤΑΣ

Η προσέγγιση του προβλήματος της σάρωσης με χρήση του μοντέλου χαρακτήρα-προς-χαρακτήρα προσφέρει αλγοριθμική καθαρότητα. Οι σαρωτές χαρακτήρα-προς-χαρακτήρα μπορούν να αναπαρασταθούν με ένα διάγραμμα μεταβάσεων· αυτό το διάγραμμα, με τη σειρά του, αντιστοιχεί σε ένα πεπερασμένο αυτόματο. Τα μικρά σύνολα λέξεων μπορούν να κωδικοποιηθούν εύκολα ως άκυκλα διαγράμματα μεταβάσεων. Για τα απειροσύνολα, όπως είναι το σύνολο των ακεραίων ή το σύνολο των αναγνωριστικών μιας γλώσσας τύπου Algol, απαιτούνται κυκλικά διαγράμματα μεταβάσεων.

Ερωτήσεις επανάληψης

Κατασκευάστε ένα ΠΑ που να αποδέχεται τις ακόλουθες γλώσσες:

1. Ένα αναγνωριστικό έξι χαρακτήρων που αποτελείται από έναν αλφαβητικό χαρακτήρα ακολουθούμενο από μηδέν έως πέντε αλφαριθμητικούς χαρακτήρες.
2. Μια συμβολοσειρά ενός ή περισσότερων ζευγών, όπου κάθε ζεύγος αποτελείται από μια αριστερή παρένθεση ακολουθούμενη από μια δεξιά παρένθεση.