Κεφάλαιο 6

Υπολογισιμότητα

6.1 Υπολογίσιμες συναρτήσεις και αναδρομικά σύνολα

Μέχρι στιγμής έχουμε δει ουσιαστικά δύο κατηγορίες γλωσσών: τις κανονικές (regular) γλώσσες και τις context free γλώσσες. Η πρώτη κατηγορία περιέχεται στη δεύτερη. Από τη μέχρι στιγμής παρουσίαση πρέπει να έχει φανεί καθαρά ότι για κάθε μια από τις δύο κατηγορίες υπάρχει και ένα αντίστοιχο υπολογιστικό μοντέλο: τα πεπερασμένα αυτόματα (ντετερμινιστικά ή μη) για τις κανονικές γλώσσες και τα αυτόματα με στοίβα (push-down automata) για τις context free γλωσσες. Η αντιστοιχία στην οποία αναφερθήκαμε είναι ότι μια γλώσσα είναι στην κατηγορία που εξετάζουμε (κανονική ή context-free) αν και μόνο αν υπάρχει μια *μηχανή* του αντίστοιχου τύπου (πεπερασμένο αυτόματο ή αυτόματο με στοίβα) που αναγνωρίζει τη γλώσσα αυτή.

Είδαμε ότι το να υπάρχει ένα DFA για μια γλώσσα L είναι ισοδύναμο με το μπορεί κανείς να γράψει ένα πρόγραμμα σε μια γλώσσα προγραμματισμού (ας πούμε σε C) που να αναγνωρίζει τις λέξεις της L (να επιστρέφει δηλ. το πρόγραμμα αυτό 1 αν η λέξη που του δώσαμε ανήκει στην L και 0 αλλιώς), το οποίο δε χρησιμοποιεί απεριόριστη μνήμη αλλά μνήμη μεγέθους σταθερού και προκαθορισμένου, το ίδιο για όλες τις λέξεις της L. Ισοδύναμα, μιλάμε για ένα πρόγραμμα σε C που χρησιμοποιεί ένα στεθερό αριθμό μεταβλητών (όχι πίνακες) η καθεμία από τις οποίες χρησιμοποιεί προκαθορισμένη μνήμη (π.χ. ακέραιοι των 32 bits – δυαδικών ψηφίων).

Αντίστοιχα, ένα *ντετερμινιστικό* αυτόματο με στοίβα είναι ακριβώς ένα πρόγραμμα σε C που χρησιμοποιεί μεν πεπερασμένη μνήμη μόνο, αλλά έχει και πρόσβαση σε μια άπειρη στοίβα με ένα περιορισμένο όμως τρόπο, που του επιτρέπει να διαβάζει και να τροποποιεί πάντα μόνο την κορυφή της στοίβας. Τα *ντετερμινιστικά* αυτόματα με στοίβα δεν αρκούν για να αναγνωρίσουν όλες τις context free γλώσσες όμως, αλλά μπορεί κανείς με λίγη προσοχή να δείξει ότι και τα *μη ντετερμιστικά* αυτόματα με στοίβα έχουν ισοδύναμα C προγράμματα.

Είναι πλέον φυσιολογικός ο ακόλουθος ορισμός.

Ορισμός 37. Έστω Σ ένα πεπερασμένο αλφάβητο. Μια συνάρτηση $f : \Sigma^* \to \Sigma^*$ λέγεται *υπολογίσιμη συνάρτηση* αν υπάρχει ένα C πρόγραμμα `P` που με είσοδο $x \in \Sigma^*$ επιστρέφει τη λέξη $f(x)$.

Τονίζουμε εδώ ότι δε μας ενδιαφέρει πόσο χρόνο θα χρειαστεί να *τρέξει* αυτό το πρόγραμμα για να υπολογίσει την απάντησή του, αλλά απαιτούμε πάντα από αυτό να μας βρεί τη σωστή την απάντηση.

Ας παρατηρήσουμε εδώ ότι δεν υπάρχει κάτι μαγικό που να μας επιβάλλει το Σ^* σα πεδίο ορισμού και πεδίο τιμών στον παραπάνω ορισμό. Στη θέση του θα μπορούσε να είναι οποιοδήποτε σύνολο που τα στοιχεία του να μπορούν να παρασταθούν, με κάποιο τρόπο, στον υπολογιστή (ένα τέτοιο σύνολο *δεν* είναι το σύνολο $\mathbf{R}$ των πραγματικών αριθμών). Αυτό όμως σημαίνει ουσιαστικά ότι μπορούμε να γράψουμε κάτω μια λέξη από 0 ή 1 που να αντστοιχεί μοναδικά σε ένα στοιχείο του συνόλου αυτού. Πολλές φορές βλέπει κανείς τον ορισμό της υπολογίσιμης συνάρτησης να δίνεται π.χ. για συναρτήσεις από τους φυσικούς αριθμούς στους φυσικούς. Εμείς με τον όρο υπολογίσιμη συνάρτηση θα καταλαβαίνουμε οποιαδήποτε απεικόνιση την οποία μπορούμε να υπολογίζουμε χρησιμοποιώντας ένα πρόγραμμα.

Για παράδειγμα οι παρακάτω συναρτήσεις είναι υπολογίσιμες αφού εύκολα βλέπει κανείς ότι υπάρχουν προγράμματα που τις υπολογίζουν.

$$\begin{aligned} f_1(x) &= x+1 \quad (\mathbf{N} \to \mathbf{N}) \\ f_2(x) &= x^2 \quad (\mathbf{N} \to \mathbf{N}) \\ f_3(x) &= |x| \quad (\Sigma^* \to \mathbf{N} \end{aligned}$$

Ορισμός 38. Έστω Σ ένα πεπερασμένο αλφάβητο. Ένα σύνολο $L \subseteq \Sigma^*$ λέγεται *αναδρομικό* αν η χαρακτηριστική του συνάρτηση

$$\chi_L(x) = \begin{cases} 1 & x \in L \\ 0 & x \notin L \end{cases}$$

είναι υπολογίσιμη.

Και πάλι τη θέση του Σ^* μπορεί να πάρει οποιοδήποτε σύνολο του οποίου τα στοιχεία μπορούν να παρασταθούν στον υπολογιστή, π.χ. το σύνολο $\mathbf{N}$ των φυσικών αριθμών. Ένα σύνολο λοιπόν λέγεται αναδρομικό αν μπορούμε να γράψουμε ένα πρόγραμμα που να αποφασίζει πότε ένα στοιχείο x ανήκει στο σύνολο ή όχι. Για παράδειγμα, το σύνολο των πρώτων αριθμών (εκείνοι οι φυσικοί αριθμοί που δεν έχουν κανένα φυσικό αριθμό ως διαιρέτη, εκτός από τον εαυτό τους και τη μονάδα) είναι ένα αναδρομικό σύνολο, αφού μπορούμε να γράψουμε ένα πρόγραμμα πουα να αποφασίζει αν ένας δοσμένος φυσικός αριθμός είναι πρώτος ή όχι, εξετάζοντας κάθε μικρότερό του φυσικό αριθμό για το αν διαιρεί το δοσμένο φυσικό ή όχι. Αυτό σίγουρα δεν είναι το 'καλύτερο' πρόγραμμα που μπορεί κανείς να γράψει για αυτό το σκοπό, αλλά πάντως δουλεύει.

Είναι τώρα φανερό ότι οι κανονικές και οι context free γλώσσες είναι αναδρομικά σύνολα, αφού υπάρχουν προγράμματα που τις 'αποφασίζουν'. Η ιεραρχία των γλωσσών που έχουμε μέχρι τώρα δεί λοιπόν είναι η

$$(\text{κανονικές γλώσσες}) \subset (\text{context free γλώσσες}) \subset (\text{αναδρομικές γλώσσες}).$$

Και οι δύο παραπάνω εγκλεισμοί είναι γνήσιοι, υπάρχει δηλ. context free γλώσσα που δεν είναι κανονική (π.χ. η γλώσσα των παλίνδρομων λέξεων xx^R πάνω από ένα αλφάβητο) και αναδρομική γλώσσα που δεν είναι context free. Ένα παράδειγμα για το τελευταίο αποτελεί η γλώσσα $\{a^nb^nc^n : n \in \mathbf{N}\}$. Γι' αυτήν έχουμε δει στην §5.7 ότι δεν είναι context free γλώσσα, ενώ είναι αρκετά απλό να γράψει κανείς ένα C πρόγραμμα που να αναγνωρίζει πότε μια λέξη είναι της μορφής $a^nb^nc^n$. Ένα τέτοιο πρόγραμμα απλά μετράει το πλήθος των a, b και c και ελέγχει ότι είναι ίδια, καθώς και ότι όλα τα b έρχονται μετά από όλα τα a κι όλα τα c μετά τα b.

Είναι σημαντικό να τονίσουμε εδώ ότι η μεταβλητή του C προγράμματος όπου κρατιέται, π.χ., το πλήθος των a, είναι μια μεταβλητή για την οποία χρειαζόμαστε απεριόριστη μνήμη. Δεν είναι δηλ. δυνατόν να προκαθορίσουμε ένα άνω όριο για το πλήθος αυτό. Και επειδή ο φυσικός αριθμός x χρειάζεται περίπου $\log_2 x$ δυαδικά ψηφία για να παρασταθεί στον υπολογιστή, κι επειδή η συνάρτηση $\log_2 x$ αυξάνει (αν και αργά) χωρίς όριο όταν $x \to \infty$, αυτό το C πρόγραμμα δεν χρησιμοποιεί σταθερή μνήμη (αν αυτό συνέβαινε τότε η γλώσσα αυτή θα ήταν κανονική, ενώ δεν είναι καν context free).

Άσκηση 64. Αν τα σύνολα $A, B \subseteq \mathbf{N}$ είναι αναδρομικά δείξτε ότι το ίδιο ισχύει και για τα σύνολα $A \cap B, A \cup B, A^c$.

6.2 Μη υπολογίσιμες συναρτήσεις

Είναι πολύ φυσιολογικό το ερώτημα αν υπάρχουν συναρτήσεις που να μην είναι υπολογίσιμες. Για να συγκεκριμενοποιήσουμε τα πράγματα, ένα φυσιολογικό ερώτημα είναι αν υπάρχει συνάρτηση $f : \mathbf{N} \to \mathbf{N}$ για την οποία να μη μπορούμε να γράψουμε ένα πρόγραμμα που να την υπολογίζει.

Θεώρημα 15. *Υπάρχει συνάρτηση $f : \mathbf{N} \to \mathbf{N}$ για την οποία δεν υπάρχει πρόγραμμα που να την υπολογίζει.*

Απόδειξη: Η πρώτη παρατήρηση που κάνουμε είναι ότι το σύνολο των υπολογισίμων συναρτήσεων είναι αριθμήσιμο, μπορεί δηλ. κάποιος να γράψει κάτω όλες τις υπολογίσιμες συναρτήσεις $\mathbf{N} \to \mathbf{N}$ σε μια ακολουθία

$$f_1, f_2, \ldots, f_n, \ldots$$

(δεν υποθέτουμε εδώ ότι $f_i \neq f_j$ για $i \neq j$, αλλά επιτρέπουμε τις επαναλήψεις στην λίστα αυτή όλων των υπολογισίμων συναρτήσεων). Αυτό είναι άμεση συνέπεια του ότι το σύνολο όλων των προγραμμάτων είναι αριθμήσιμο. Πράγματι, αν το υποθέσουμε αυτό ως γνωστό δεν έχουμε παρά να απαριθμήσουμε όλα τα προγράμματα και για κάθε ένα από αυτά γράφουμε κάτω ποια συνάρτηση υπολογίζει (αν υπολογίζει συνάρτηση, αλλιώς δε γράφουμε τίποτα). Κάθε υπολογίσιμη συνάρτηση θα εμφανιστεί τότε στη λίστα μια και κάποια στιγμή θα εξετάσουμε ένα από τα προγράμματα που την υπολογίζουν.

Γιατί είναι τώρα το σύνολο όλων των προγραμμάτων αριθμήσιμο;Απλούστατα, κάθε πρόγραμμα δεν είναι παρά μια, συνήθως μεγάλη, λέξη πάνω από ένα συγκεκριμένο αλφάβητο Σ. Για παράδειγμα, όλα τα προγράμματα σε C γράφονται στο αλφάβητο που χρησιμοποιούν οι σύγχρονοι υπολογιστές και που περιλαμβάνει όλα τα λατινικά γράμματα, μικρά και κεφαλαία, διάφορα σημεία στίξης, αριθμητικούς και άλλους τελεστές, ψηφία, κλπ. Απαριθμούμε πρώτα λοιπόν όλες τις λέξεις μήκους 1, μετά όλες τις λέξεις μήκους 2, όλες τις λέξεις μήκους 3, κ.ο.κ., και για κάθε λέξη που παράγουμε την αναφέρουμε στη λίστα των προγραμμάτων, αν αυτή είναι πρόγραμμα (πληροί δηλ. τους συντακτικούς κανόνες της γλώσσας προγραμματισμού που χρησιμοποιούμε). Μπορούμε να το κάνουμε αυτό ακριβώς επειδή όλες οι λέξεις πάνω από το Σ μήκους k είναι φυσικά πεπερασμένες το πλήθος, και άρα κάποια στιγμή τελειώνουμε την απαρίθμηση των λέξεων μήκους k και προχωράμε στην απαρίθμηση των λέξεων μήκους $k+1$. Κάθε πρόγραμμα λοιπόν θα εμφανιστεί στη λίστα μας κάποια στιγμή ανάλογα και με το ποιο είναι το μήκος του. Έχουμε λοιπόν μέχρι στιγμής δείξει ότι το πλήθος όλων των υπολογισίμων συναρτήσεων είναι αριθμήσιμο.

Η απόδειξη του θεωρήματος συμπληρώνεται από το ότι το πλήθος *όλων* των συναρτήσεων από το $\mathbf{N}$ στο $\mathbf{N}$ δεν είναι αριθμήσιμο, δε μπορούμε δηλ. να διατάξουμε όλες αυτές τις συναρτήσεις σε μια ακολουθία

$$g_1, g_2, \ldots, g_n, \ldots$$

Αν μπορούσαμε τότε ας κοιτάξουμε τη συνάρτηση

$$f(k) = g_k(k) + 1.$$

Αυτή είναι προφανές μια συνάρτηση από το $\mathbf{N}$ στο $\mathbf{N}$ άρα, αφού υποθέσαμε ότι όλες οι συναρτήσεις αυτές βρίσκονται στη λίστα g_i, υπάρχει κάποιο $i \in \mathbf{N}$ τέτοιο ώστε $f(k) = g_i(k)$ για κάθε $k \in \mathbf{N}$. Αυτό σημαίνει $g_k(k) + 1 = g_i(k)$. Διαλέγοντας $k = i$ παίρνουμε άτοπο. (Το παραπάνω επιχείρημα ονομάζεται 'διαγώνιο επιχείρημα'.)
□

Στην παραπάνω απόδειξη δείξαμε ότι υπάρχουν μη υπολογίσιμες συναρτήσεις δείχνοντας ότι το σύνολο των υπολογισίμων συναρτήσεων είναι αριθμήσιμο ενώ, αντίθετα, το σύνολο όλων των συναρτήσεων δεν είναι, είναι αυτό που λέμε *υπεραριθμήσιμο* σύνολο. Άρα, δεν μπορούν τα δύο σύνολα να είναι ίσα.

Αυτός είναι ένας κλασικός τρόπος απόδειξης στα μαθηματικά, η 'υπαρξιακή απόδειξη', που αν και είναι αρκετά εύκολος και αποδεικνύει αυτό που θέλουμε, δεν είναι τόσο ικανοποιητικός επειδή αποδεικνύεται η ύπαρξη ενός αντικειμένου με ορισμένες ιδιότητες, αλλά, συνήθως, δεν προσδιορίζεται με κάποιο τρόπο κάποιο τέτοιο αντικείμενο. Στην προκειμένη περίπτωση γνωρίζουμε πλέον, μετά την απόδειξη του Θεωρήματος 15, ότι υπάρχουν συναρτήσεις από τους φυσικούς στους φυσικούς που δεν είναι υπολογίσιμες, αλλά δε γνωρίζουμε καμία συγκεκριμένη τέτοια συνάρτηση. Θα δούμε αργότερα συγκεκριμένα παραδείγματα που για το καθένα θα έχουμε και ιδιαίτερο τρόπο απόδειξης του ότι δεν είναι υπολογίσιμα.

Άσκηση 65. Δεν υπάρχει κάποιος ιδιαίτερος λόγος που στο Θεώρημα 15 αναφερόμαστε σε συναρτήσεις $\mathbf{N} \to \mathbf{N}$. Δείξτε, για παράδειγμα, ότι υπάρχουν συναρτήσεις $\mathbf{N} \to \{0,1\}$ και $\Sigma^* \to \Sigma^*$, που δεν είναι υπολογίσιμες (Σ είναι ένα πεπερασμένο αλφάβητο).

6.3 Το Halting Problem. Άλλα μη αποφασίσιμα προβλήματα στα Μαθηματικά.

Τα προγράμματα ως αριθμοί, και το αντίστροφο Για τη συζήτηση από δω και πέρα θα χρειαστούμε να βλέπουμε το τυχόν C πρόγραμμα ως ένα φυσικό αριθμό, και τον τυχόντα φυσικό αριθμό ως ένα C πρόγραμμα.

Πώς γίνεται αυτό;Πολύ απλά, ένα C πρόγραμμα δεν είναι παρά ένα κομμάτι κειμένου, μια λέξη δηλ. πάνω από ένα συγκεκριμένο αλφάβητο. Το αλφάβητο που χρησιμοποιείται στους υπολογιστές σήμερα είναι το

$$\Sigma = \{b_7 b_6 \cdots b_1 b_0 :\ b_i \in \{0,1\}, i = 0, \ldots, 7\}.$$

Αυτές είναι όλες οι λέξεις (bytes) με οκτώ δυαδικά ψηφία (bits) η κάθε μία και πολλές φορές ταυτίζουμε το συγκεκριμένο αλφάβητο με τους αριθμούς $0, \ldots, 255$. Ένα πρόγραμμα P λοιπόν είναι μια λέξη

$$P = w_1 \cdots w_N, \quad w_i \in \Sigma$$

και δεν έχουμε παρά να αντικαταστήσουμε κάθε w_i με τα δυαδικά του ψηφία ώστε να γράψουμε το P σα μια δυαδική λέξη με $8N$ δυαδικά ψηφία

$$P = b_{8N-1} b_{8N-2} \cdots b_1 b_0$$

την οποία λέξη μπορούμε να ερμηνεύσουμε ως ένα φυσικό αριθμό, που τον συμβολίζουμε στο εξής ως $\alpha(P)$. Αντίστροφα, αν n είναι ένας φυσικός αριθμός τον γράφουμε στο δυαδικό σύστημα και συμπληρώνουμε με μηδενικά (από τα αριστερά) το πλήθος των ψηφίων του αν χρειάζεται ώστε να είναι πολλαπλάσιο του 8, έστω $8k$. Κάθε μια από τις k οκτάδες τώρα τη βλέπουμε σαν ένα στοιχείο του Σ και ο αριθμός μας γίνεται τώρα μια λέξη του Σ^*. Αν αυτή η λέξη είναι ένα συντακτικά σωστό πρόγραμμα σε C τότε ονομάζουμε αυτό το πρόγραμμα $\pi(n)$, αν όχι, θέτουμε $\pi(n)$ να είναι το πρόγραμμα `main() { return 1; }`.

Η απεικόνιση $P \to \alpha(P)$ είναι εύκολο να δεί κανείς ότι είναι ένα προς ένα (αλλά όχι επί), και ότι η απεικόνιση $n \to \pi(n)$ δεν είναι ένα προς ένα. Είναι επίσης σημαντικό το ότι και οι δύο απεικονίσεις είναι υπολογίσιμες. Το ότι η $\alpha(P)$ είναι υπολογίσιμη είναι φανερό, ενώ το μόνο λεπτό σημείο όσον αφορά την υπολογισιμότητα της $\pi(n)$ είναι στο κομμάτι όπου πρέπει να ελέγξουμε αν η λέξη του Σ^* που προκύπτει είναι συντακτικά σωστό C πρόγραμμα ή όχι. Το να αποδείξουμε ότι κάτι τέτοιο είναι υπολογστικά εφικτό σημαίνει ουσιαστικά να γράψουμε ένα πρόγραμμα που να αναγνωρίζει αν ένα κομμάτι κειμένου (μια λέξη του Σ^*) είναι ένα σωστό συντακτικά C πρόγραμμα ή όχι. Τέτοια προγράμματα όμως υπάρχουν και χρησιμοποιούνται ευρέως αφού αποτελούν το πρώτο τμήμα των complilers (μεταφραστών) που παίρνουν ένα πρόγραμμα σε C και παράγουν από αυτό ένα ισοδύναμο πρόγραμμα σε 'γλώσσα μηχανής'.

Προσομοίωση. Θα χρειαστούμε επίσης το εξής: Υπάρχει ένα πρόγραμμα $S(n, x, t)$ με παραμέτρους το οποίο (α) βρίσκει το πρόγραμμα $P = \pi(n)$ και (β) 'Τρέχει' το πρόγραμμα $P(x)$ για τόσα βήματα όσα λέει ο αριθμός t ή επ άπειρον αν ο αριθμός t είναι αρνητικός. Αν και όταν το πρόγραμμα $P(x)$ τελειώσει το $S(n, x, t)$ επιστρέφει ότι επέστρεψε το $P(x)$. Δε θα δείξουμε την ύπαρξη αυτού του προγράμματος που προσομοιώνει άλλα προγράμματα, μια και τέτοια προγράμματα υπάρχουν και χρησιμοποιούνται ευρέως στον προγραμματισμό (π.χ. interpreters, debuggers, κλπ).

Είμαστε έτοιμοι τώρα να αναφερθούμε στο Halting problem (πρόβλημα τερματισμού). Αυτό ρωτάει απλά αν υπάρχει ένα πρόγραμμα που να μπορεί να αποφασίζει αν ένα τυχόν άλλο πρόγραμμα που εμείς του προσδιορίζουμε θα τερματίσει ή όχι. (Είναι φανερό ότι υπάρχουν προγράμματα που δεν τερματίζουν ποτέ, π.χ. το `main() { while(1) 1; }`). Ένα τέτοιο πρόγραμμα θα ήταν εξαιρετικά χρήσιμο αν υπήρχε. Φανταστείτε να έχετε ένα C compiler ο οποίος όχι μόνο θα μετάφραζε το πρόγραμμά σας σε γλώσσα μηχανής αλλά θα μπορούσε και να σας πει εκ των προτέρων αν το πρόγραμμά σας θα έπεφτε σε άπειρο βρόγχο (loop) ή όχι. Δυστυχώς όμως τέτοιο πρόγραμμα δεν υπάρχει.

Στο εξής, για τυχόν πρόγραμμα P, θα γράφουμε $P \downarrow$ αν το πρόγραμμα P τερματίζει και $P \uparrow$ αν το P τρέχει επ' άπειρον.

Θεώρημα 16. *Η συνάρτηση*

$$H(x) = \begin{cases} 1 & \alpha\nu \quad \pi(x) \downarrow \\ 0 & \alpha\nu \quad \pi(x) \uparrow \end{cases}$$

δεν είναι υπολογίσιμη.

Η συνάρτηση $H(x)$ επιστρέφει 1 αν το πρόγραμμα $\Pi(x)$ τερματίζει και 0 αλλιώς.

Απόδειξη. Ας υποθέσουμε ότι η συνάρτηση $H(x)$ είναι υπολογίσιμη και ας συμβολίσουμε με P το πρόγραμμα που την υπολογίζει. Φτιάχνουμε τώρα το πρόγραμμα $Q(n, x)$ που χρησιμοποιεί το P ως υποπρόγραμμα:

```
int Q(x)
{
    if (P(α((π(x))(x))) == 0) return 1;
    else {
     while(1) 1;
    }
}
```

Το πρόγραμμα $Q(x)$ που ορίσαμε έχει την ιδιότητα ότι τερματίζει αν και μόνο αν το $(\pi(x))(x)$ δεν τερματίζει. Δηλαδή, δοθέντος του $x \in \mathbf{N}$ υπολογίζουμε το πρόγραμμα $\pi(x)$ που του αντιστοιχεί και του προσδιορίζουμε ως input πάλι τον αριθμό x. Αυτό το νέο πρόγραμμα περνούμε ως παράμετρο στο πρόγραμμα P.

Ποια είναι η συμπεριφορά του προγράμματος $Q(\alpha(Q))$;Τερματίζει ή όχι;Σύμφωνα με τη προηγούμενη παρατήρηση τερματίζει αν και μόνο αν το πρόγραμμα $(\pi(\alpha(Q)))(\alpha(Q))$ δεν τερματίζει. Αλλά $\pi(\alpha(Q)) = Q$, οπότε δείξαμε ότι το πρόγραμμα $Q(\alpha(Q))$ τερματίζει αν και μόνο αν δεν τερματίζει, πράγμα προφανώς αδύνατο. Το συμπέρασμα είναι ότι η συνάρτηση H δεν είναι υπολογίσιμη.
$\square$

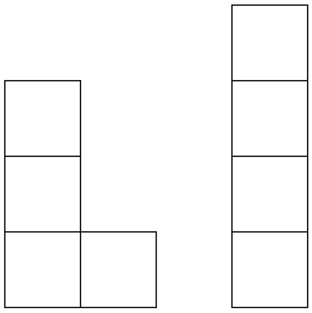
Έχουμε λοιπόν δείξει ότι η, πολύ φυσιολογική, συνάρτηση $H(x)$ που διακρίνει πότε το πρόγραμμα $\pi(x)$ τερματίζει ή όχι δεν είναι υπολογίσιμη. Έχουμε λοιπόν τώρα μια απόδειξη του ότι υπάρχουν μη υπολογίσιμες συναρτήσεις που είναι αρκετά πιο συγκεκριμένη από την προηγούμενη υπαρξιακή απόδειξη του Θεωρήματος 15. Μπορεί βέβαια και πάλι να πει κανείς ότι η $H(x)$ δεν έχει ίσως και τόσο ενδιαφέρον από μαθηματική άποψη μια και είναι μια συνάρτηση 'κομμένη και ραμμένη' στα μέτρα της Θεωρίας Υπολογισμών, και άρα όχι, ίσως, τόσο φυσιολογική.

Υπάρχουν όμως πολλά παραδείγματα αρκετά φυσιολογικών μαθηματικών προβλημάτων που δεν επιδέχονται λύση αλγοριθμική, προβλήματα που προϋπήρχαν της Θεωρίας Υπολογισμών.

1. Δεν υπάρχει αλγόριθμος που να αποφασίζει αν ένα δοσμένο πολυώνυμο με ακεραίους συντελεστές (και οποιοδήποτε πλήθος από μεταβλητές)
$$P(x_1, \ldots, x_n) = \sum_{0 \le j_1, \ldots, j_n \le D} c_{j_1, \ldots, j_n} x_1^{j_1} \cdots x_n^{j_n}, \quad c_{j_1, \ldots, j_n} \in \mathbf{Z},$$
έχει λύση (ρίζα) ανάμεσα στους ακεραίους, αν υπάρχουν δηλ. $x_1, \ldots, x_n \in \mathbf{Z}$ τ.ώ. $P(x_1, \ldots, x_n) = 0$.

2. Είναι ενδιαφέρον ότι αν ρωτήσουμε αν υπάρχει ρίζα πραγματική (και όχι αναγκαστικά ακέραια), αν δηλ. υπάρχουν $x_1, \ldots, x_n \in \mathbf{R}$ τ.ώ. $P(x_1, \ldots, x_n) = 0$, τότε υπάρχει αλγόριθμος που να απαντάει στο ερώτημα.

3. Ένα *πολυόμινο* είναι μια πεπερασμένη ένωση από μη αλληλοεπικαλυπτόμενα, μοναδιαία (πλευράς 1) τετράγωνα στο επίπεδο, τέτοια ώστε οι συντεταγμένες των κορυφών τους είναι ακέραιες. Στο Σχήμα 20 φαίνεται ένα σύνολο από δύο πολυόμινα. Ένα πολύ φυσιολογικό ερώτημα είναι αν ένα δεδομένο πεπε-



Σχήμα 20. Ένα σύνολο από δύο πολυόμινα

ρασμένο σύνολο από πολυόμινα μπορεί να χρησιμοποιηθεί για να 'πλακοστρώσει' ολόκληρο το επίπεδο. Αυτό σημαίνει ότι καλύπτουμε το επίπεδο, χωρίς αλληλο επικαλύψεις, από κομμάτια καθ΄ ένα από τα οποία είναι μια παράλληλη μεταφορά κάποιου από τα δεδομένα πολυόμινα. Για παράδειγμα, είναι σχετικά εύκολο να δει κανείς πως τα δύο πολυόμινα του Σχήματος 20 μπορούν να πλακοστρώσουν (αγγλικά tiling) το επίπεδο, και είναι επίσης εύκολο να φτιαξει κανείς παραδείγματα απο σύνολο πολυομίνων που δε μπορούν, όπως και να μεταφερθούν, να πλακοστρώσουν το επίπεδο. Αποδεικνύεται λοιπόν ότι το να αποφασίσει κανείς αν ένα δεδομένο σύνολο από πολυόμινα μπορεί να πλακοστρώσει το επίπεδο είναι αλγοριθμικά ανεπίλυτο πρόβλημα. Δεν υπάρχει δηλ. πρόγραμμα που να του δίνουμε ένα τυχόν πεπερασμένο σύνολο από πολυόμινα και να μας απαντά αν επιδέχεται πλακόστρωση ή όχι.

6.4 Αναδρομικά απαριθμήσιμα (α.α.) σύνολα.

Ορισμός 39. Ένα μη κενό σύνολο $A \subseteq \mathbf{N}$ ονομάζεται *αναδρομικά απαριθμήσιμο* (α.α.) αν υπάρχει πρόγραμμα $P(x)$ που να απαριθμεί τα στοιχεία του A, δηλ. να έχουμε

$$A = \{P(1), P(2), P(3), \ldots\},$$

με ενδεχόμενη επανάληψη των στοιχείων του A στο δεξί μέλος. Με άλλα λόγια, για κάθε $a \in A$ υπάρχει $n \in \mathbf{N}$, όχι κατ΄ ανάγκη μοναδικό, τέτοιο ώστε $a = P(n)$.

Παρατήρηση: Και πάλι, δεν υπάρχει τίποτα το μαγικό με το σύνολο $\mathbf{N}$ και θα μπορούσαμε στη θέση του να έχουμε, π.χ. το σύνολο Σ^*, όπου Σ ένα πεπερασμένο αλφάβητο.

Είναι σχεδόν άμεσο ότι κάθε αναδρομικό σύνολο είναι και α.α. Πράγματι, αν $\emptyset \neq A \subseteq \mathbf{N}$ είναι αναδρομικό τότε υπάρχει πρόγραμμα $Q(x)$ τέτοιο ώστε $Q(x) = 1 \Leftrightarrow x \in A$. Τότε το ακόλουθο πρόγραμμα απαριθμεί τα στοιχεία του A (m είναι, π.χ., το μικρότερο στοιχείο του A):

```
int P(x)
{
    if(Q(x)) return x;
    else return m;
}
```

Άσκηση 66. Αν $A, B \subseteq \mathbf{N}$ είναι α.α. σύνολα δείξτε ότι το ίδιο ισχύει και για τα σύνολα $A \cap B, A \cup B$.

Από την άλλη μεριά είναι εύκολο να κατασκευάσουμε α.α. σύνολα που δεν είναι αναδρομικά. Για παράδειγμα, το σύνολο

$$A = \{x \in \mathbf{N} :\ \pi(x) \downarrow\}$$

δεν είναι αναδρομικό (αν ήταν τότε το πρόβλημα του τερματισμού θα ήταν υπολογίσιμο, ενώ έχουμε δείξει ότι δεν είναι) αλλά είναι α.α. Για να δούμε αυτό τον τελευταίο ισχυρισμό κάνουμε το εξής. Χρειαζόμαστε ένα πρόγραμμα που να απαριθμεί τα $x \in \mathbf{N}$ για τα οποία το πρόγραμμα $\pi(x)$ τερματίζει. Ισοδύναμα, φτιάχνουμε ένα πρόγραμμα που λειτουργεί επ΄ άπειρον και τυπώνει συνέχεια στην έξοδό του στοιχεία του A, χωρίς να παραλείψει κανένα. Το πρόγραμμα αυτό

- **Βήμα 1:** Πρώτα τρέχει (προσομοιώνει) το πρόγραμμα $\pi(1)$ για 1 βήμα.
- **Βήμα 2:** Έπειτα τρέχει (προσομοιώνει) τα προγράμματα $\pi(1), \pi(2)$ για 2 βήματα το καθένα
- $\ldots$
- **Βήμα k:** Τρέχει τα προγράμματα $\pi(1), \pi(2), \ldots, \pi(k)$ για k βήματα το καθένα,
- Συνεχίζει έτσι επ΄ άπειρον.

Στο τέλος του βήματος k το πρόγραμμά μας ελέγχει ποια από τα προγράμματα $\pi(1), \ldots, \pi(k)$ τελείωσαν μέσα στα πρώτα k τους βήματα, και τυπώνει τα αντίστοιχα x για αυτά που τελείωσαν στην έξοδό του. Είναι έτσι φανερό ότι κάθε x για το οποίο $\pi(x) \downarrow$ θα εμφανιστεί στην έξοδο του προγράμματός μας, και ότι σε αυτή την έξοδο εμφανίζονται μόνο τέτοια x (δηλ. στοιχεία του A). Αν $x \in A$ και το πρόγραμμα $\pi(x)$ τερματίζει μετά από t βήματα τότε ο αριθμός x εμφανίζεται για πρώτη φορά στην έξοδο του προγράμματός μας μετά το βήμα υπ' αριθμόν $\max\{x, t\}$, και εμφανίζεται και σε όλα τα μεταγενέστερα βήματα.

Έχουμε λοιπόν δείξει:

Θεώρημα 17. *Κάθε αναδρομικό σύνολο είναι αναδρομικά απαριθμήσιμο αλλά το αντίστροφο δεν ισχύει.*

Η ιεραρχία των γλωσσών τώρα γίνεται

$$(\text{κανονικές γλώσσες}) \subset (\text{context free γλώσσες}) \subset (\text{αναδρομικές γλώσσες}) \subset (\text{α.α. γλώσσες})$$

και όλοι οι εγκλεισμοί είναι γνήσιοι.

Η σχέση αναδρομικών και α.α. συνόλων φαίνεται πιο καθαρά στα επόμενα δύο θεωρήματα.

Θεώρημα 18. *$A \subseteq \mathbf{N}$ είναι αναδρομικό αν και μόνο αν τα A, A^c είναι και τα δύο α.α.*

Απόδειξη: Αν το A είναι αναδρομικό τότε αναδρομικό είναι και το A^c, άρα και τα δύο είναι α.α. Αντίστροφα τώρα, έστω ότι τα A, A^c είναι και τα δύο α.α., και ότι τα προγράμματα P και Q τα απαριθμούν, έχουμε δηλ.

$$A = \{P(1), P(2), \ldots\},\ A^c = \{Q(1), Q(2), \ldots\}.$$

Το παρακάτω πρόγραμμα $R(x)$, που χρησιμοποιεί τα P και Q ως υποπρογράμματα, επιστρέφει 1 αν $x \in A$ και 0 αλλιώς.

```
int R(x)
{
    int i=1;
    label:
    if(P(i) == x) return 1;
    if(Q(i) == x) return 0;
    i = i+1; goto label;
}
```

Το πρόγραμμα R απλούστατα παράγει όλους τους αριθμούς $P(1), Q(1), P(2), Q(2), \ldots$ και μόλις εμφανιστεί το x επιστρέφει 0 ή 1 ανάλογα με το αν το x εμφανίστηκε στην έξοδο του Q ή του P. Είναι σίγουρο ότι το x κάποτε θα εμφανιστεί (μια και είτε θα ανήκει στο A είτε στο A^c) και άρα για κάθε x το πρόγραμμα $R(x)$ τερματίζει.
$\square$

Θεώρημα 19. *Ένα σύνολο $A \subseteq \mathbf{N}$ είναι α.α. αν και μόνο αν υπάρχει πρόγραμμα $Q(x)$ τέτοιο ώστε*

- *Αν $x \in A$ τότε $Q(x) = 1$, και*
- *Αν $x \notin A$ τότε $Q(x) = 0$ ή $Q(x) \uparrow$.*

Το πρόγραμμα Q στο θεώρημα λέει την αλήθεια για το αν $x \in A$, αν τελειώνει, μπορεί όμως και να τρέχει επ' άπειρον (αλλά αυτό μπορεί να συμβαίνει μόνο αν $x \in A$. Παρατηρείστε ότι υπάρχει ασυμμετρία στις δύο περιπτώσεις $x \in A$ και $x \notin A$, και αυτή η ασυμμετρία εκδηλώνεται και με το ότι υπάρχουν α.α. σύνολα που τα συμπληρώματά τους δεν είναι α.α.

Απόδειξη: Έστω ότι το πρόγραμμα P απαριθμεί τα στοιχεία του A:

$$A = \{P(1), P(2), \ldots\}.$$

Ορίζουμε το πρόγραμμα $Q(x)$ ως εξής:

```
int Q(x)
{
    int i=1;
    label:
    if(P(i) == x) return 1;
    i = i+1; goto label;
}
```

Το πρόγραμμα $Q(x)$ επιστρέφει 1 αν και μόνο αν το x εμφανιστεί στη λίστα των τιμών του P, αλλιώς τρέχει επ' άπειρον.
□