Κεφάλαιο 4: Υπολογισιμότητα

4.1 Μηχανές Turing

Μέχρι τώρα είδαμε δύο τύπους αυτομάτων:

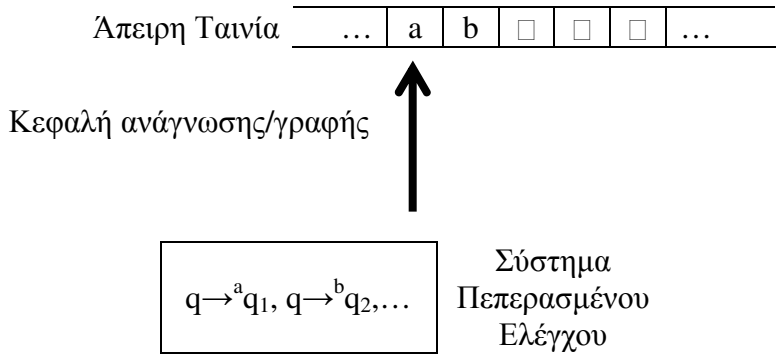
- Πεπερασμένα αυτόματα
- Αυτόματα με στοίβα

Και οι δύο τύποι αδυνατούν να αναγνωρίσουν κάποιες κατηγορίες γλωσσών (Λήμματα Άντλησης).

Οι μηχανές Turing είναι πιο γενικές. Και όπως θα δούμε, μάλλον είναι όσο γενικές γίνεται: δεν μπορούμε να φανταστούμε πιο ισχυρούς υπολογιστικούς μηχανισμούς.

Παρόλα αυτά, οι μηχανές Turing είναι ένα απλό (low level) υπολογιστικό μοντέλο. Συνολικά:

1. Είναι αυτόματα, δηλαδή η κατασκευή και λειτουργία τους διέπεται από το ίδιο γενικό πνεύμα με τις συσκευές (αυτόματα) που μελετήσαμε ήδη.
2. Είναι απλές ως προς την περιγραφή τους.
3. Είναι όσο γίνεται γενικές ως προς τους υπολογισμούς που μπορούν να πραγματοποιήσουν.



Η λειτουργία διαισθητικά είναι η ακόλουθη:

Αρχική συνολική κατάσταση

- Αρχική κατάσταση q_0
- Πεπερασμένη λέξη εισόδου στην ταινία, δεξιά και αριστερά της βρίσκονται μόνο κενά σύμβολα □
- Η κεφαλή δείχνει στο πρώτο σύμβολο της λέξης εισόδου

Κάθε βήμα εξαρτάται από:

- Την κατάσταση q στην οποία βρίσκεται
- Το σύμβολο ο που διαβάζει από την ταινία

Το βήμα κάνει τις εξής ενέργειες:

1. Αλλάζει την κατάσταση
2. Αντικαθιστά το σύμβολο της ταινίας που βρίσκεται κάτω από την κεφαλή με ένα άλλο (πιθανόν το ίδιο)
3. Μετακινεί την κεφαλή δεξιά ή αριστερά, ή την αφήνει αμετάβλητη

Ένας υπολογισμός είναι *επιτυχής* εάν φτάσουμε σε μία τελική κατάσταση.
Έξοδος: η μέγιστη λέξη δεξιά της κεφαλής δίχως κενά σύμβολα (δες παρακάτω).

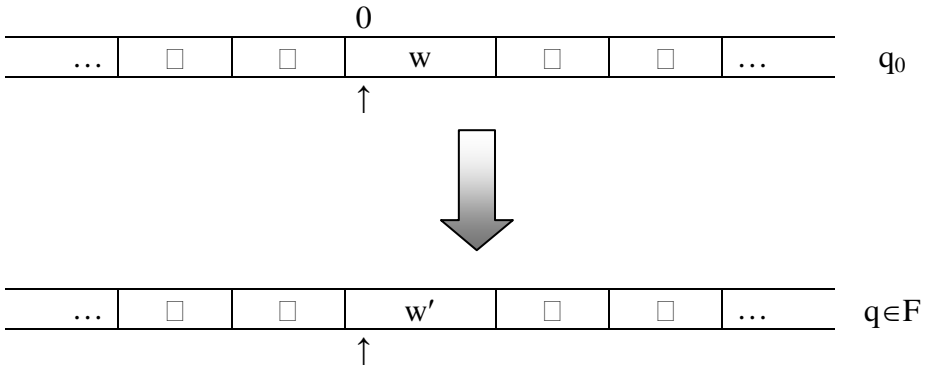
Μία *μηχανή Turing* είναι μία 7-άδα $T = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ όπου:
- Q: πεπερασμένο *σύνολο καταστάσεων*
- Σ: πεπερασμένο *αλφάβητο εισόδου*
- Γ: πεπερασμένο *αλφάβητο ταινίας* με $\Sigma \subseteq \Gamma$
- $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L,R,N\}$, όπου L=left, R=right, N=no movement
- $q_0 \in Q$: *αρχική κατάσταση*
- $\square \in \Gamma - \Sigma$: *κενό σύμβολο*
- $F \subseteq Q$: *τελικές καταστάσεις*

Μία *κατάσταση ταινίας* του T είναι μία τριάδα (q,x,β), όπου $q \in Q$, $x \in \mathbb{Z}$ και $\beta: \mathbb{Z} \to \Gamma$ με $\beta(y) = \square$ για όλα εκτός από πεπερασμένο αριθμό $y \in \mathbb{Z}$.

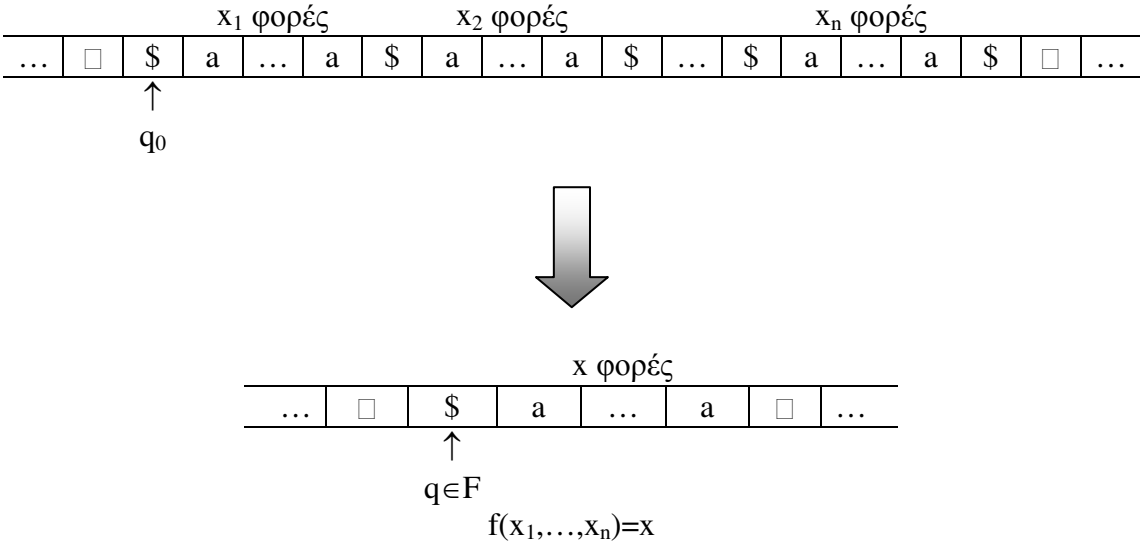
Μετάβαση από μία κατάσταση ταινίας σε μία άλλη: $(q,x,\beta) \to (q',x',\beta')$, όπου:
- $\delta(q,\beta(x)) = (q',\beta'(x),M)$, όπου $\beta(x)$: το σύμβολο ταινίας που διαβάζουμε
- $\beta'(y) = \beta(y)$ για κάθε $y \neq x$
- $x' = x-1$, εάν M=L
  $x' = x+1$, εάν M=R
  $x' = x$, εάν M=N

Ένας *υπολογισμός* του T είναι μία πεπερασμένη ακολουθία $z_0 \vdash z_1 \vdash \ldots \vdash z_n$.
Ο υπολογισμός *τερματίζει* εάν $z_n = (q,x,\beta)$ με $q \in F$.

Για μία πεπερασμένη λέξη $w \in \Sigma^*$ μπορούμε να ορίσουμε την έξοδό της ως εξής:



- Ερώτημα: Μπορούν οι μηχανές Turing να υπολογίζουν *αριθμητικές συναρτήσεις* αντί συναρτήσεις σε αλφαβήτους;
- Ναι!
  - ✓ Οι αριθμοί μπορούν να αναπαρασταθούν σαν λέξεις (δυαδική αναπαράσταση, αριθμός ενός συμβόλου)
  - ✓ Ένα αλφάβητο μπορεί να απαριθμηθεί όπως και οι πεπερασμένες λέξεις σε αυτό



4.2 Αναδρομικές Συναρτήσεις

Τώρα θα εισάγουμε έναν άλλο τρόπο υπολογισμού: *αναδρομικές συναρτήσεις*.

- Βασικές Συναρτήσεις

- $f_{ZERO}^{(n)}(x_1,\ldots,x_n)=0$, για κάθε $n\in\mathbb{N}$
- $f_{SUCC}^{(n)}(x_1,\ldots,x_n)=x_1+1$, για κάθε $n\in\mathbb{N}$
- $f_{PROJ(i)}^{(n)}(x_1,\ldots,x_n)=x_i$ αν $1\le i\le n$
  $f_{PROJ(i)}^{(n)}(x_1,\ldots,x_n)=0$ διαφορετικά

- Σύνθεση

- $f_{COMPOSITION(G,H_1,\ldots,H_m)}^{(n)}(x_1,\ldots,x_n)=f_G^{(m)}(f_{H_1}^{(n)}(x_1,\ldots,x_n),\ldots,f_{H_m}^{(n)}(x_1,\ldots,x_n))$

- Απλή Αναδρομή

- $f_{REC(G,H)}^{(n)}(x_1,\ldots,x_{n-1},0)=f_G^{(n)}(x_1,\ldots,x_{n-1},0)$
- $f_{REC(G,H)}^{(n)}(x_1,\ldots,x_{n-1},y+1)=f_H^{(n+1)}(x_1,\ldots,x_{n-1},f_{REC(G,H)}^{(n)}(x_1,\ldots,x_{n-1},y),y)$

Οι συναρτήσεις σύνθεση και απλή αναδρομή μπορούν να συνδυαστούν και να μας δώσουν πιο πολύπλοκες συναρτήσεις: *απλές αναδρομικές συναρτήσεις*.

Παράδειγμα 1

pred: $\mathbb{N}\to\mathbb{N}$
pred(0)=0
pred(y+1)=y

REC(ZERO,PROJ(2)) | pred(0)=$f_{ZERO}^{(1)}(0)$
pred(y+1)=$f_{PROJ(2)}^{(2)}$(pred(y),y)

Παράδειγμα 2

add: $\mathbb{N}\to\mathbb{N}$
add(x,y)=x+y
x+0=x
x+(y+1)=(x+y)+1

REC(PROJ(1),COMP(SUCC,PROJ(2))) | $add(x,0)=f_{PROJ(1)}^{(2)}(x,0)$
$add(x,y+1)=f_{SUCC}^{(1)}(f_{PROJ(2)}^{(3)}(x,add(x,y),y))$

Παράδειγμα 3

mult: $\mathbb{N}\rightarrow\mathbb{N}$
$mult(x,y)=x\cdot y$
$x\cdot 0=0$
$x\cdot(y+1)=x\cdot y+x$

$mult(x,0)=f_{ZERO}^{(2)}(x,0)$
$mult(x,y+1)=add(f_{PROJ(1)}^{(3)}(x,mult(x,y),y),f_{PROJ(2)}^{(3)}(x,mult(x,y),y))$

Φραγμένη Ελαχιστοποίηση

Έστω $R\subseteq\mathbb{N}^{n+1}$. Ορίζουμε $f:\mathbb{N}^{n+1}\rightarrow\mathbb{N}$ ως εξής:

- $f(x_1,\ldots x_n,b)$=μικρότερο y με $y\leq b$ και $R(x_1,\ldots x_n,y)$ αν υπάρχει
- $f(x_1,\ldots x_n,b)=b$ αλλιώς

Εαν η R είναι απλά αναδρομική, τότε και η f είναι απλά αναδρομική.

- Ερώτημα: Υπάρχουν συναρτήσεις που είναι διαισθητικά υπολογίσιμες αλλά όχι απλά αναδρομικές;
- Ναι, για παράδειγμα η συνάρτηση Ackermann:
  $A(0,y)=y+1$
  $A(x+1,0)=A(x,1)$
  $A(x+1,y+1)=A(x,A(x+1,y))$

Σχέδιο Απόδειξης

Κάθε απλά αναδρομική συνάρτηση είναι total.
Αλλά οι απλά αναδρομικές συναρτήσεις δεν περιλαμβάνουν όλες τις διαισθητικά υπολογίσιμες συναρτήσεις.
$d(n)=f_n(n)+1$
όπου f_n είναι μία αρίθμηση των απλά αναδρομικών συναρτήσεων (πχ λεξικογραφικά).
Η d είναι διαισθητικά υπολογίσιμη.

Μπορεί να είναι απλά αναδρομική; Αν ναι, τότε $d=f_k$ για ένα $k\in\mathbb{N}$.
Τότε $f_k(k)=f_k(k)+1$ (άτοπο).

Μη Φραγμένη Ελαχιστοποίηση

Έστω συνάρτηση g με n+1 θέσεις.
Αν υπάρχει y τέτοιο ώστε:
1. $\forall z\leq y,\ f_g^{(n+1)}(x_1,\ldots,x_n,z)\downarrow$
2. $\forall z<y,\ f_g^{(n+1)}(x_1,\ldots,x_n,z)>0$
3. $f_g^{(n+1)}(x_1,\ldots,x_n,y)=0$

τότε: $f_{MIN(g)}^{(n)}(x_1,\ldots,x_n)=y$
Διαφορετικά: $f_{MIN(g)}^{(n)}(x_1,\ldots,x_n)\uparrow$

Παρατήρηση
1. Διαισθητικά υπολογίσιμη συνάρτηση
2. Μας δίνει συναρτήσεις που δεν έχουν τιμή παντού (όχι total – partial)

μ-αναδρομικές συναρτήσεις: όσες μπορούν να σχηματιστούν από απλές συναρτήσεις χρησιμοποιώντας:
- σύνθεση
- απλή αναδρομή
- μη φραγμένη ελαχιστοποίηση

4.3 Η Θέση του Church
Διάφοροι τρόποι να ορίσουμε υπολογισιμότητα:
- WHILE προγράμματα
- Μηχανές Turing
- Αναδρομικές συναρτήσεις
- Γραμματικές

Τελικά και οι τέσσερις είναι ισοδύναμοι: μπορούν να υπολογίσουν ακριβώς τις ίδιες συναρτήσεις.
Το ίδιο αποδείχθηκε και για όλες τις άλλες προσπάθειες προσδιορισμού του τι υπολογίζεται.

Θέση του Church
1. Όλοι οι μηχανισμοί υπολογισμού που μπορούμε να σχεδιάσουμε δεν είναι ισχυρότεροι από τους παραπάνω.
2. Όλοι αυτοί οι μηχανισμοί περιγράφουν τις διαισθητικά υπολογίσιμες συναρτήσεις.

➢ Δεν είναι θεώρημα (λόγω του 2)!

4.4 Μη Υπολογίσιμα Προβλήματα
Θα «αποδείξουμε» (αν και όχι με 100% τυπικό μαθηματικό τρόπο) ότι το *Πρόβλημα Τερματισμού* δεν είναι αλγοριθμικά επιλύσιμο.
➢ Δεν υπάρχει πρόγραμμα το οποίο για είσοδο:
  ✓ Ένα πρόγραμμα P
  ✓ Ένα σύνολο δεδομένων D
  τερματίζει και δίνει μία από τις εξής δύο εξόδους:
  ✓ 1 αν το P με είσοδο D τερματίζει
  ✓ 0 αλλιώς
➢ Προσοχή: Δεν μπορεί να τρέξει απλά το P με είσοδο D, διότι έτσι μπορεί να μην τερματίσει, ενώ πρέπει!

Έστω ότι το Πρόβλημα Τερματισμού μπορεί να επιλυθεί με αλγοριθμικό τρόπο. Τότε υπάρχει πρόγραμμα CheckHalt(P,D) το οποίο τερματίζει πάντα και μας δίνει:
  ✓ 1 αν το P με είσοδο D τερματίζει
  ✓ 0 αν το P με είσοδο D τρέχει επ άπειρον

Ιδέα: Μπορούμε να θεωρήσουμε ένα πρόγραμμα σαν ένα σύνολο δεδομένων.
Η ιδέα ήταν επαναστατική στις πρώτες δεκαετίες του προηγούμενου αιώνα, αλλά είναι πλέον προφανής στην πληροφορική: πχ compilers, parsers κλπ δέχονται προγράμματα σαν δεδομένα.

Ένα πρόγραμμα μπορεί μάλιστα να κωδικοποιηθεί σαν ένας μόνο φυσικός αριθμός (Goedel-οποίηση). Επομένως μπορούμε να τρέξουμε ένα πρόγραμμα P με είσοδο το ίδιο το P. Πιο συγκεκριμένα ορίζουμε ένα πρόγραμμα Test ως εξής:

Test(P):

IF CheckHalt(P,P) τερματίζει με έξοδο 1
THEN x:=1; WHILE x>0, DO x:=x+1 END
END

Πρόκειται για έναν αλγόριθμο που μπορούμε να υλοποιήσουμε (θέση του Church).

Τι συμβαίνει όταν τρέξουμε το Test με είσοδο Test;

- Έστω ότι Test(Test) τερματίζει.
  Τότε εξ ορισμού CheckHalt(Test,Test) τερματίζει με έξοδο 1. Άρα η υπόθεση του IF ισχύει, και άρα το Test με είσοδο Test δεν τερματίζει (άτοπο).
- Έστω ότι Test(Test) δεν τερματίζει.
  Τότε CheckHalt(Test,Test) δίνει 0, άρα το Test δεν ακολουθεί την περίπτωση IF και τερματίζει (άτοπο).

Ο μόνος τρόπος να αποφύγουμε το άτοπο είναι να συμπεράνουμε ότι δεν υπάρχει πρόγραμμα CheckHalt, άρα το πρόβλημα τερματισμού δεν λύνεται αλγοριθμικά.

Το παραπάνω επιχείρημα δεν είναι 100% τυπικό διότι θα έπρεπε να ορίσουμε:

- ✓ Τι σημαίνει ένα πρόγραμμα να τρέχει με είσοδο κάποια δεδομένα
- ✓ Πώς ακριβώς απαριθμούνται τα προγράμματα

Ωστόσο η κεντρική ιδέα αποτυπώνεται με σαφήνεια.

4.5 Στα Σύνορα Υπολογισιμότητας και Μη Υπολογισιμότητας

Το σύνορο μεταξύ υπολογισιμότητας και μη

➢ Το Πρόβλημα Τερματισμού

Μπορούμε να ορίσουμε μία 1-1 συνάρτηση μεταξύ προγραμμάτων P και φυσικών αριθμών $\mathbb{N}$. Έστω P_n το n-στο πρόγραμμα αυτής της απαρίθμησης. Έστω:

- halt(n,m)=1 αν το P_n με είσοδο m τερματίζει
- halt(n,m)=0 αλλιώς

Η συνάρτηση halt <u>δεν</u> είναι υπολογίσιμη.

➢ <u>Παρατήρηση</u>: Στο προηγούμενο μάθημα δεν χρησιμοποιήσαμε την απαρίθμηση των προγραμμάτων. Διαισθητικά τα ίδια επιχειρήματα δείχνουν ότι η halt δεν υπολογίζεται.

<u>Παράδειγμα 1</u>

- $halt_1(n,m)=m$, αν το P_n με είσοδο m τερματίζει
- $halt_1(n,m)=\uparrow$, αλλιώς

➢ Υπολογίσιμη

<u>Ιδέα</u>:

1. Για το n υπολογίζουμε το P_n
2. Τρέχουμε το P_n με είσοδο m
3. Εάν τερματίζει, σταματάμε με έξοδο m
4. Αλλιώς προφανώς δεν σταματάμε ποτέ

Παράδειγμα 2
- $halt_2(n,m)=m$, αν το P_n με είσοδο m τερματίζει
- $halt_2(n,m)=0$, αλλιώς

Έστω ότι η $halt_2$ υπολογίζεται από ένα πρόγραμμα $HALT_2$.
Ανάγουμε τον υπολογισμό της halt σε αυτόν της $halt_2$:
1. Τρέχουμε το $HALT_2$ με είσοδο m
2. Εάν το $HALT_2$ σταματάει με έξοδο 0, δίνουμε έξοδο 0
3. Εάν το $HALT_2$ σταματάει με έξοδο m, δίνουμε έξοδο 1

Αυτό το πρόγραμμα υπολογίζει την halt (άτοπο).
Άρα δεν υπάρχει τέτοιο πρόγραμμα $HALT_2$.

Παράδειγμα 3
- $halt_3(n,m)=\mu(m)$, αν το P_n με είσοδο m τερματίζει
- $halt_3(n,m)=\nu(m)$, αλλιώς

όπου $\mu,\nu:\mathbb{N}\to\mathbb{N}$ υπολογίσιμες συναρτήσεις.

Γενικά θυμίζει το παράδειγμα 2, αλλά μπορεί να είναι υπολογίσιμη. Πχ. υπό την εξής προϋπόθεση:

$\forall m\in\mathbb{N}$: $\nu(m)\downarrow\Rightarrow\mu(m)\downarrow$ και $\mu(m)=\nu(m)$.

Για παράδειγμα:
- $\mu(m)=m^2$, αν $m<200$
- $\mu(m)=\uparrow$, αλλιώς

- $\nu(m)=m^2$, αν $m<100$
- $\nu(m)=\uparrow$, αλλιώς

Υπολογισμός της $halt_3$ σε αυτή την περίπτωση:
1. Υπολόγισε παράλληλα $\nu(m)$ και P_n με είσοδο m
2. Εάν τερματίσει πρώτο το P_n, ισχύει η περίπτωση 1, άρα τερμάτισε τον υπολογισμό του $\nu(m)$ και ξεκίνησε αυτόν του $\mu(m)$
3. Εάν τερματίσει πρώτο το $\nu(m)$, τότε $\mu(m)\downarrow$ άρα $\mu(m)=\nu(m)$. Τότε $halt_3(n,m)=\mu(m)$, το οποίο έχει ήδη υπολογιστεί
4. Εάν δεν τερματίσει ούτε το P_n ούτε το ν, τότε $halt_3(n,m)\uparrow$

Παράδειγμα 4
- $halt_4(n,m,k)=1$, αν το P_n με είσοδο m τερματίζει το πολύ σε k βήματα
- $halt_4(n,m,k)=0$, αλλιώς

➢ Υπολογίσιμη!
Αλγόριθμος
1. Ξεκίνησε να τρέχεις το P_n με είσοδο m
2. Μέτρα τα βήματα
3. Σταμάτα με έξοδο 1 αν ο υπολογισμός τερματίζει με δείκτη $\le k$
4. Μετά από k βήματα σταμάτα με έξοδο 0