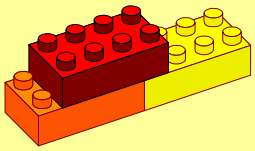
ΗΥ-280

«ΘΕΩΡΙΑ ΥΠΟΛΟΓΙΣΜΟΥ»

θεμελιακές έννοιες της επιστήμης του υπολογισμού



ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Γεώργιος Φρ. Γεωργακόπουλος

Μέρος Γ′

Βασικά στοιχεία περί υπολογισιμότητας:
επιλύσιμα και ανεπίλυτα προβλήματα.

19ο Διάφορα είδη προγραμματισμού, και τα ισχυρότατα από αυτά.

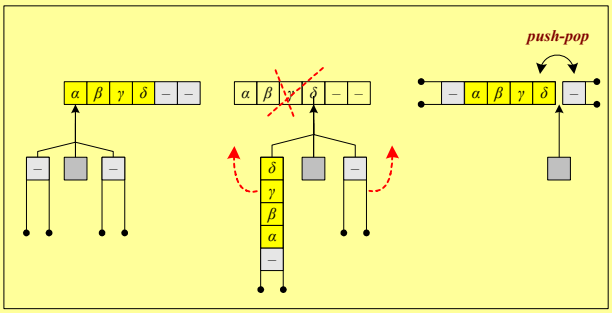
«Πλήρεις» υπολογιστικοί τρόποι, και η εκτίμηση της ισχύος τους.

Ο χώρος στον οποίο αναφέρεται η θεωρία υπολογισμού, όπως το είδαμε πολλές φορές μέχρι τώρα, είναι οι συναρτήσεις Φ από πεπερασμένα δεδομένα σε πεπερασμένα αποτελέσματα· ακριβέστερα, σε πεπερασμένα αποτελέσματα ή καθόλου αποτελέσματα (όταν μια συνάρτηση δεν ορίζεται). Τίποτε περισσότερο δεν μπορούμε να ελπίζουμε ότι θα ήσαν σε θέση να διεκπεραιώσει μια πεπερασμένη μηχανή σε πεπερασμένο χρόνο.

Και επειδή οι εκάστοτε πληροφορίες ακόμα και εάν αυτές έχουν πιο αφηρημένη μορφή (αν είναι, λ.χ. αριθμοί), είναι και πάλι «γραπτές» ως λέξεις κάποιου αλφαβήτου Σ, οι συναρτήσεις προς υπολογισμό έχουν την εξής μορφή:

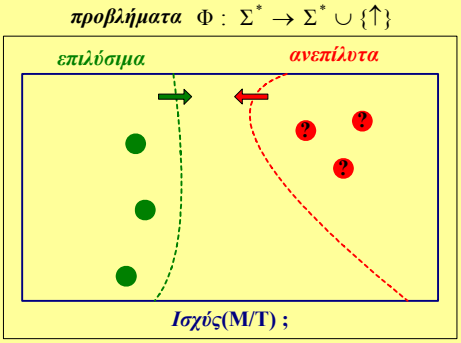
$$\Phi: \Sigma^* \rightarrow \Sigma^* \cup \{\uparrow\}$$

Είδαμε και αναλύσαμε μια σειρά μηχανών που επεξεργάζονται λέξεις εξ ενός αλφαβήτου Σ: μηχανές με μία κατάσταση και 0, 1, 2 στοίβες. Οι τελευταίες δεν είναι παρά οι μηχανές Turing (μT): και πράγματι εάν μια μηχανή φέρει μία κατάσταση και δύο στοίβες, τότε μπορεί να αντιγράψει τα δεδομένα της σε μία από τις στοίβες, και στη συνέχεια, εφόσον είναι σε θέση να αναλαμβάνει δεδομένα από την μία στοίβα και να τα αποθέτει στην άλλη (pop-push), είναι σε θέση ισοδυνάμως να κινεί την κεφαλή σε οποιοδήποτε σημείο των υπό επεξεργασία δεδομένων – όπως και οι μηχανές Turing (βλ. αρχικές ενότητες).



Σχήμα: «1 κατάσταση + 2 στοίβες» = «Μηχανές *Turing*» ...

(Έχει, επίσης, ενδιαφέρον να παρατηρήσουμε ότι οι μηχανές με *τρείς* (ή και περισσότερες στοίβες) δεν έχουν να προσφέρουν περισσότερη υπολογιστική ισχύ, διότι ένα οποιοδήποτε σταθερό πεπερασμένο πλήθος από «στοίβες» είναι δυνατόν να αποθηκεύονται σειριακά σε μία Α κύρια στοίβα, και μια Β βοηθητική στοίβα να χρησιμοποιείται ως προσωρινός χώρος μεταφοράς και φύλαξης δεδομένων, προκειμένου η μηχανή να αποκτήσει πρόσβαση σε μία οποιαδήποτε από τις «στοίβες» που έχουν αποθηκευτεί στην Α.)



Αν Χ είναι ένας τύπος υπολογισμού, ας ονομάσουμε και συμβολισουμεμε ***Ισχύς(Χ)*** $\subseteq \Phi$, το σύνολο των συναρτήσεων που μπορεί να υπολογιστούν από κάποια μηχανή τύπου Χ. Ποιά είναι η Ισχύς(μΤ) των μηχανών Turing (ή «μΤ»); Ποιές συναρτήσεις μπορούν να υπολογιστούν, και ποιές όχι (έαν υπάρχουν τέτοιες);

Και τα δύο ερωτήματα είναι πολύ πιο δύσκολο να αποδειχθούν από ότι στην περίπτωση των μηχανών με 0 ή 1 στοίβα. Αρκετά πιο δύσκολο, ώστε να χρειαστούμε ιδιαίτερες μαθηματικές τεχνικές τεχνικές για να τα απαντήσουμε.

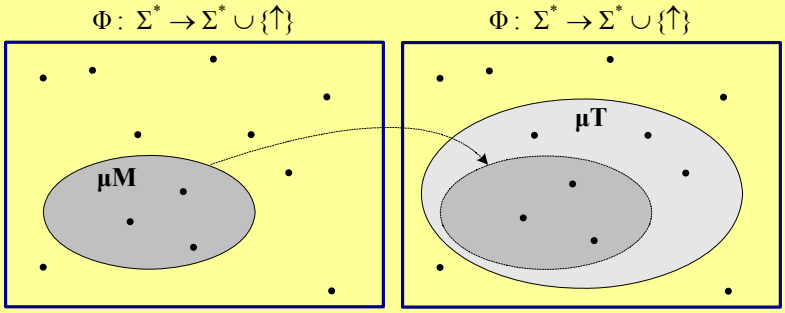
Ως προς το 1ο ερώτημα το σκεπτικό και η τεχνική μας θα είναι η εξής:

> *«δεν είναι εύκολο να περιγράψω τί μπορούν να υπολογίσουν οι μΤ, αλλά δώσε μου έναν άλλο τύπο υπολογισμού Χ και θα δείξω ότι οι μΤ μπορούν να κάνουν ότι και εκείνος, δηλαδή: Ισχύς(Χ) $\subseteq$ Ισχύς(μΤ)».*

Θα δούμε σε αυτή την ενότητα δύο ακόμα τύπους υπολογισμού/προγραμματισμού.

- τις μηχανές με μνήμη[1] (μΜ), και
- τις αναδρομικές περιγραφές (ΑΠ),

και θα δείξουμε ότι και οι δύο δεν έχουν μεγαλύτερη ισχύ από τις μΤ.



Σχήμα: σύγκριση δύο τύπων: Ισχύς(μΜνήμης) $\subseteq$ Ισχύς(μ.*Turing*).

Το δόγμα Church-Turing:[2] Το ίδιο έχει επιτευχθεί και για κάθε άλλο τρόπο υπολογισμού που έχουμε σχεδιάσει μέχρι τώρα: *οι μηχανές Turing είναι ο ισχυρότερος ή πληρέστερος τρόπος υπολογισμού που έχουμε στη διάθεσή μας*. Το να αποδείξουμε ότι αυτό ισχύει δεν είναι δυνατόν διότι αυτός ο ισχυρισμός κείται «εκτός» μαθηματικών: δεν υπάρχει τρόπος να βεβαιώσουμε ότι έχουμε ελέγξει «όλους» τους δυνατούς τρόπους υπολογισμού. Ο σχετικός ισχυρισμός έχει μείνει στην ιστορία ως το *δόγμα Church-Turing* (υπήρξαν ως δάσκαλος και μαθητής αντιστοίχως):

Ως προς το 2ο ερώτημα – υπάρχουν συναρτήσεις που δεν μπορούμε να υπολογίσουμε με καμμία μΤ; – το σκεπτικό και η τεχνική μας θα είναι η εξής:

> *«δώσε μου έναν κατάλογο όσων συναρτήσεων νομίζεις ότι είναι δυνατόν να υπολογίσουν οι μΤ και θα σου δώσω μια συνάρτηση που είναι εκτός αυτού του καταλόγου.»*

Ως προς αυτό το 2ο ερώτημα, είμαστε σε θέση να δώσουμε και πολύ πλουσιότερες απαντήσεις· ατυχώς όμως..., όπως θα φανεί στις τελευταίες ενότητες των σημειώσεων.

[1] Σε αντιπαράθεση με τις μηχανές με «ταινία» όπου η πρόσβαση στα δεδομένα είναι σειριακή και όχι κατ' ευθείαν.
[2] «*Church-Turing* thesis» στα αγγλικά.

I. *μT: μηχανές «Turing».*
Έχουμε ήδη δεί αυτό το είδος «μηχανής», στις αρχικές ενότητες. Θα επανέλθουμε σε αυτό το είδος σε επόμενη ενότητα προκειμένου να την συγκρίνουμε με τα δύο είδη προγραμματισμού που ακολουθούν.

II. *μM: μηχανές με «μνήμες» (ή «καταχωριστές»).*
Οι ***μηχανές με μνήμες*** είναι μηχανές σχεδιασμένες να μιμούνται κατά στοιχειώδη τρόπο τους «μικρο-επεξεργαστές» που είμαστε σε θέση (πλέον) να υλοποιούμε και τεχνολογικά και οι οποίοι στηρίζουν την πληροφορική τεχνολογία της εποχής μας.

Συντακτικοί κανόνες προγραμμάτων για «μηχανές με καταχωριστές».
Τα προγράμματά των μηχανών μM διαθέτουν τρία (μόνο) είδη εντολών:

Εντολές Ε:	Δείκτες:
z_κ	$\kappa \geq 1$
s_κ	$\kappa \geq 1$
$j_{\kappa,\lambda,\rho}$	$\kappa, \lambda \geq 1, \rho \geq 0$

Ένα πρόγραμμα π μιας μηχανής μM, είναι μια πεπερασμένη ακολουθία $\boldsymbol{\pi} = \langle \boldsymbol{\pi}_{[1]}, \boldsymbol{\pi}_{[2]}, ..., \boldsymbol{\pi}_{[\nu]} \rangle$ από τέτοιες εντολές: $\pi_{[\kappa]} \in$ Ε. Ο αριθμός ν θα καλείται το ***μήκος*** του προγράμματος π, και θα τον συμβολίζουμε με ***μήκος(π)*** ή ***μ(π)***. Για να διευκολύνουμε τεχνικά την ανάλυση αυτών των προγραμμάτων θα ζητούμε συμβατικά σε κάθε (*ορθώς συντεταγμένο*) πρόγραμμα π μηχανής μM, η παράμετρος ρ κάθε εντολής «$j_{\kappa,\lambda,\rho}$» του π να κείται στο διάστημα 0...μ(π), δηλαδή ένα πρόγραμμα να μην αναφέρεται σε εντολές που δεν περιέχει – πλήν της υπ. αρ. 0:

«π είναι ορθώς συντεταγμένο» = για κάθε εντολή «$j_{\kappa,\lambda,\rho}$» του π, $0 \leq \rho \leq$ μήκος(π),

Σημασιολογικοί κανόνες προγραμμάτων για «μηχανές με καταχωριστές».
Η «μηχανή» που θα υποθέσουμε ότι είναι σε θέση να εκτελέσει προγράμματα σαν τα παραπάνω αποτελείται από τα εξής:

- *ένα μετρητή προγράμματος (program counter)*, που θα είναι –ακριβολογώντας– μετρητής των εντολών του προγράμματος.
- μια *ακολουθία θέσεων μνήμης* (ή «*καταχωριστών*») $m_{[1]}, m_{[2]}, ... m_{[n]}$, οι οποίες απομνημονεύουν εκάστη από έναν φυσικό αριθμό.

Ο μετρητής αναπαρίσταται από έναν φυσικό αριθμό ***pc***, και η μνήμη της μηχανής αναπαρίσταται από μια συνάρτηση ***m***: $\mathbf{N} \to \mathbf{N}$. Το εκάστοτε *στάδιο υπολογισμού* περιγράφεται από το ζεύγος $\langle pc, \mathbf{m} \rangle$. Το νόημα των εντολών ευρίσκεται στα *βήματα υπολογισμού*, δηλαδή στο πώς ένα στάδιο μετατρέπεται στο επόμενό του, και αυτό είναι που εξηγούμε αμέσως παρακάτω:

- *Αρχικό στάδιο υ_0:*

Αφετηριακό στάδιο της μηχανής μας είναι το $\upsilon_0 = \langle 1, \mathbf{m} \rangle$. Η αφετηριακή μνήμη **m** (πρέπει να) περιέχει στις θέσεις 1, 2, 3, ... τα όποια δεδομένα έχει να δώσει ο «χρήστης» στο υπό εκτέλεση πρόγραμμα. Περιμένουμε το όποιο τελικό αποτέλεσμα κατά παρόμοιο τρόπο.

- *Βήματα υπολογισμού $\upsilon_\kappa \to \upsilon_{\kappa+1}$:*

Έστω ότι το τρέχον στάδιο είναι το $\upsilon_\kappa = \langle pc, \mathbf{m} \rangle$. Εξετάζουμε τον μετρητή pc, και...

- Εάν pc = 0, το πρόγραμμα τερματίζει – βλέπε παρακάτω.
- Εάν $1 \leq pc \leq$ μήκος(π), ως επόμενο στάδιο έχουμε το $\upsilon_{\kappa+1} = \langle pc', \mathbf{m'} \rangle = \langle pc+1, \mathbf{m} \rangle$, όπου η «μνήμη» **m** έχει τροποποιηθεί σε **m'** λόγω της εντολής $\pi_{[pc]}$ σύμφωνα με τον παρακάτω πίνακα.

$\pi_{[pc]} =$	Επίδραση/σημασία	Σχόλιο
z_κ	$m'_{[\kappa]} \leftarrow 0$	Μηδενίζουμε την υπ. αρ. κ θέση μνήμης.
s_κ	$m'_{[\kappa]} \leftarrow m'_{[\kappa]} + 1$	Αυξάνουμε την υπ. αρ. κ θέση μνήμης κατά +1.
$j_{\kappa,\lambda,\rho}$	αν $m_{[\kappa]} = m_{[\lambda]}$ τότε $pc' \leftarrow \rho$	Συνεχίζουμε, υπό συνθήκη, με την εντολή υπ. αρ. ρ.

- Εάν pc > μήκος(π), δεν έχουμε καμμία αλλαγή και το πρόγραμμα *δεν* τερματίζει: $\upsilon_{\kappa+1} = \upsilon_{\kappa}$.

- *Τέλος υπολογισμού υ_{last}:*

Το πρόγραμμα τερματίζει εάν και μόνον εάν ο μετρήτης προγράμματος είναι μηδέν: pc =? 0. Ας προσέξουμε εδώ ότι κατά την εκτέλεση της τελευταίας εντολής ο μετρητής αυξάνεται σε pc ← μήκος(π)+1, και αυτό ίσως να μην αρέσει. Εάν θέλουμε ένα πρόγραμμα να τερματίσει μετά την εκτέλεση της τελευταίας εντολής, θα πρέπει να υπάρχει, ή *να προσθέσουμε*, ως τελευταία εντολή μια εντολή της μορφής $\mathbf{j}_{\kappa,\kappa,0}$ – (μια «τεχνικότητα» ανάμεσα στις μύριες που τηρούν οι προγραμματιστές...).

Παραδειγμα: πρόγραμμα μΜ για την άθροιση: «$\mathbf{m}_{[3]} \leftarrow \mathbf{m}_{[1]} + \mathbf{m}_{[2]}$ »

Δίνουμε στη συνέχεια ένα πρόγραμμα μΜ που υπολογίζει το άθροισμα των αριθμών στη 1η και 2η θέση και το γράφει στη 3η θέση. Χρησιμοποιεί επίσης, βοηθητικά, την 4η θέση.

κ	$\pi_{[\kappa]}$	σχόλιο:
01	z_3	μηδενίζουμε την 3η θέση μνήμης.
02	z_4	μηδενίζουμε την 4η θέση μνήμης.
03	$j_{4,1,7}$	? έχουμε αυξήσει την 3η θέση τόσες φορές όσες λέει 1η θέση (;)
04	s_3	αυξάνουμε την 3η θέση κατά +1, και,
05	s_4	μετράμε πόσες φορές την έχουμε αυξήσει στην 4η θέση.
06	$j_{4,4,3}$	επαναλαμβάνουμε τον κύκλο των εντολών 3-4-5.
07	z_4	μηδενίζουμε την 4η θέση
08	$j_{4,2,0}$	? έχουμε αυξήσει την 3η θέση τόσες φορές όσες λέει 2η θέση (;)
09	s_3	αυξάνουμε την 3 κατά +1, και,
10	s_4	μετράμε πόσες φορές την έχουμε αυξήσει στην 4η θέση.
11	$j_{4,4,8}$	επαναλαμβάνουμε τον κύκλο 8-9-10

Το παραπάνω πρόγραμμα αυξάνει κατά +1 την, αρχικώς μηδενικη, 3η θέση τόσες φορές συνολικά, όσες «λέει» η 1η και στη συνέχεια η 2η θέση, επομένως στο τέλος θα περιέχεται εκεί το άθροισμα $\mathbf{m}_{[1]} + \mathbf{m}_{[2]}$.

III. *ΑΠ: αναδρομικές περιγραφές.*

Οι ***αναδρομικές περιγραφές*** είναι μια τεχνική χειρισμού, και επαγωγικού ορισμού συναρτήσεων. Αντιστοιχούν στο θεμελιακό επίπεδο σε αυτό που στον χώρο του προγραμματισμού κατέληξε στις γλώσσες συναρτησιακού προγραμματισμού. Ας σημειώσουμε εδώ ότι οι αναδρομικές περιγραφές (σε άλλες παραλλαγές) προηγήθηκαν ιστορικά των μηχανών *Turing*.

Συντακτικοί κανόνες ΑΠ:

- Οι *βασικές αναδρομικές περιγραφές*, ορίζονται όπως στον παρακάτω πίνακα. Κάθε αναδρομική περιγραφή χαρακτηρίζεται από ένα φυσικό αριθμό $\nu \geq 0$ που δηλώνει την «παραμετρικότητά» της.

Βασική περιγραφή D	Δείκτες:	Παραμετρικότητα
$\mathbf{Z}_{\nu}$	$\nu \geq 0$	ν-παραμετρική
$\mathbf{I}_{\nu,\kappa}$	$\nu \geq 1, \quad 1 \leq \kappa \leq \nu$	ν-παραμετρική
S	–	μονοπαραμετρική

- Επαγωγικά, ορίζουμε και *σύνθετες αναδρομικές περιγραφές*, κατά τον επόμενο πίνακα:

	Αν δίδονται περιγραφές,	*με παραμετρικότητα,*	*τότε ορίζεται και η D,*	*η οποία έχει παραμετρικότητα:*
«σύνθεση»	**Φ** **H**$_1$**, ..., H**$_\kappa$	κ ν	**C(Φ, H**$_1$**, ..., H**$_\kappa$**)**	ν
«αναδρομή»	**B** **P**	(ν–1) (ν+1)	**R(B, P)**	ν
«αναζήτηση»	**Φ**	(ν+1)	**M(Φ)**	ν

Σημασιολογικοί κανόνες ΑΠ – «υποδηλωτική» εκδοχή:
Οι παραπάνω συμβολισμοί μένουν ακατανόητοι, αν δεν τους δώσουμε «νόημα». Θα νοηματοδοτήσουμε τις αναδρομικές περιγραφές με δύο τρόπους. Ο 1ος είναι ο εξής: κάθε *ν-παραμετρική* αναδρομική περιγραφή **D** (θα) ***υποδηλώνει*** μια –ίσως μερικώς ορισμένη– συνάρτηση $f_D(\cdot)$ από πλειάδες φυσικών αριθμών μήκους ν, προς φυσικούς αριθμούς. Θα επιτρέπουμε επίσης και την «τιμή» *αόριστον*, που θα συμβολίζουμε με το σύμβολο ↑. Ο προσδι-ορισμός της $f_D(\cdot)$ είναι προφανώς επαγωγικός, και δίδεται στον επόμενο πίνακα:

Η περιγραφή D *(ν-παραμετρική)*	*...υποδηλώνει την ν-παραμετρική συνάρτηση:* $f_D(\mathbf{x})$: $\mathbf{N}^\nu \to \mathbf{N} \cup \{\uparrow\}$, όπου $\mathbf{x} = (x_1, ..., x_\nu)$
Z$_\nu$	$f_Z(\mathbf{x}) = 0$
I$_{\nu,\kappa}$	$f_I(x) = x_{[\kappa]}$
S	$f_S(\mathbf{x}) = x+1$
C(Φ, H$_1$**, ..., H**$_\kappa$**)**	$f_D = f_\Phi(f_{H1}(\mathbf{x}), ..., f_{H\kappa}(\mathbf{x}))$
R(B, P)	$f_R(\mathbf{x}, x_{\nu+1}) = \begin{cases} f_B(\mathbf{x}), & \text{εάν } x_{\nu+1} = 0 \quad (\textit{βάση επαγωγής}) \\ f_P(\mathbf{x}, k, f_\Phi(\mathbf{x}, k)), & \text{εάν } x_{\nu+1} = k+1 \quad (\textit{βήμα επαγωγής}) \end{cases}$
M(Φ)	$f_B(\mathbf{x}) = \min\{ \varrho : f_\Phi(\mathbf{x}, \varrho) = 0 \}$

Ακόμα και η παραπάνω νοηματοδότηση ίσως μείνει ακατανόητη αν δεν προσέξει κάποιος σε τί αντιστοιχεί και τί φανερώνει.

Σε αυτό το σημείο θα ήταν καλό ο αναγνώστης να σταθεί και να επεξεργαστεί πρώτα λίγο μόνος του τον παραπάνω πίνακα, διότι, τώρα, το νόημα των παραπάνω είναι *ήδη αποτυπωμένο* στο κείμενό μας. Η δική μας απάντηση ακολουθεί στον επόμενο πίνακα:

	Η περιγραφή D	*υποδηλώνει την δυνατότητά μας...*
«σταθερά 0»	$\mathbf{Z_v}$	να χειριζόμαστε «σταθερές», όπως το 0 (και φυσικά το +1, κττ).
«επιλογή»	$\mathbf{I_{v,\kappa}}$	από ν «δεδομένα» να επιλέγουμε το υπ. αρ. κ, ($1 \le \kappa \le \nu$), από αυτά.
«αύξηση»	**S**	αν όχι να προσθέτουμε, τουλάχιστον να μετράμε, δηλαδή από έναν αριθμό να λαμβάνουμε τον επόμενό του.
«σύνθεση»	$\mathbf{C(\Phi, H_1, ..., H_\kappa)}$	να εκτελούμε εργασίες διαδοχικά, μετατρέποντας τα αποτελέσματα των προηγουμένων εργασιών, σε δεδομένα για τις επόμενες.
«αναδρομή»	**R(B, P)**	να ανατρέχουμε *προς τα οπίσω*, ώστε να ανάγουμε έναν υπολογισμό επί ορίσματος k+1, σε υπολογισμό επί τιμής k – έως ότου k = 0.
«αναζήτηση»	**M(Φ)**	να ανατρέχουμε *προς τα εμπρός*, ώστε να ζητούμε για k = 0, 1, 2, ..., την 1η (άρα *minimum*) τιμή για την οποία ισχύει μια συνθήκη, εδώ του τύπου $f_\Phi(k) = 0$.
	Ας προσέξουμε εδώ ότι μια αναζήτηση M(Φ) ενδέχεται να αποβεί ατελέσφορη (ή ατέρμων). Λ.χ. η '(x+1) = 0' δεν έχει καμμία ρίζα (στους φυσικούς), και ως εκ τούτου είναι αδύνατον να βρεθεί μια πρώτη τέτοια. Επομένως για Φ = S, η συνάρτηση $f_{M(S)}$ μένει *αόριστη*. Αυτός είναι ο λόγος για τον οποίο συμπεριλαμβάνουμε την αόριστη «τιμή» ↑, ως ενδεχόμενη «τιμή» μιας συνάρτησης.	

Σημασιολογικοί κανόνες ΑΠ – «τελεστική» εκδοχή:

Μετά τον ακριβή συντακτικό ορισμό των αναδρομικών περιγραφών, μετά την ***υποδηλωτική*** σημασιοδότησή του, και μετά την ερμηνεία της, παραμένει ακόμα μια απορία: «*Καλώς! Έστω λοιπόν ότι ορίσαμε μία συνάρτηση μέσω αναδρομικής περιγραφής. Πώς θα την υπολογίζουμε;*». Αυτή η ***τελεστική*** σημασιοδότηση είναι η 2η εκδοχή ερμηνείας των αναδρομικών περιγραφών, και την δίδουμε αμέσως στη συνέχεια.

Έστω μια ν-παραμετρική αναδρομική περιγραφή και $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_\nu]$ μια ν-άδα φυσικών αριθμών. Η D περιγράφει μια συνάρτηση f_D την τιμή της οποίας για τα εκάστοτε ***ορίσματα*** $\boldsymbol{\alpha}$ υπολογίζουμε με τα εξής στάδια υπολογισμού:

- *Αρχικό στάδιο* υ_0*:* «Γράφουμε» την αρχική αναδρομική περιγραφή και την πλειάδα των ορισμάτων της $\boldsymbol{\alpha}$ ως μια «λέξη» $D\boldsymbol{\alpha} = D[\alpha_1, \alpha_2, ..., \alpha_\nu]$.

- *Βήμα υπολογισμού* $\upsilon_\kappa \to \upsilon_{\kappa+1}$*:* Μετασχηματίζουμε ανά βήμα το τρέχον στάδιο όπως παρακάτω.
  - Εντοπίζουμε την 1η εξ αριστερών εμφάνιση πλειάδας $\boldsymbol{\lambda} = [...]$ η οποία περιέχει μόνον ***ορίσματα***, (δηλαδή καμμία αναδρομική περιγραφή).
  - Εντοπίζουμε την αναδρομική περιγραφή, έστω W, που εμφανίζεται προ της $\boldsymbol{\lambda}$. (Αυτή η περιγραφή είναι αρχικά η διδόμενη D, και όπως θα φανεί στη συνέχεια, προ κάθε πλειάδας ορισμάτων $\boldsymbol{\lambda}$ θα υπάρχει πάντοτε μια σχετική περιγραφή εργασίας W ('working').)
  - Αναλόγως του τύπου της περιγραφής, κάνουμε τα εξής, (όπου ν είναι το μήκος της $\boldsymbol{\lambda}$):

W = ...	Το τμήμα $W\boldsymbol{\lambda}$ του τρέχοντος σταδίου αντικαθίσταται...
Z_ν	από το όρισμα 0.
$I_{\nu,\kappa}$	από το όρισμα $\boldsymbol{\lambda}_{[\kappa]}$ – το υπ. αρ. κ της πλειάδας $\boldsymbol{\lambda}$.
S	από το όρισμα τ+1, όπου τ το ένα και μόνον όρισμα της $\boldsymbol{\lambda}$.
$C(\Phi, H_1, ..., H_\kappa)$	από την προς αποτίμηση «περιγραφή»: $\Phi[H_1\boldsymbol{\lambda}, H_2\boldsymbol{\lambda}, ..., H_\kappa\boldsymbol{\lambda}]$.
R(B, P)	από $B[\boldsymbol{\lambda}_{[1]}, ..., \boldsymbol{\lambda}_{[\nu-1]}]$, εάν $\boldsymbol{\lambda}_{[\nu+1]}$ έχει τιμή 0. από $P[\boldsymbol{\lambda}_{[1]}, ..., \boldsymbol{\lambda}_{[\nu-1]}, k, R(B, P)[\boldsymbol{\lambda}_{[1]}, ..., \boldsymbol{\lambda}_{[\nu-1]}, k]]$, εάν $\boldsymbol{\lambda}_{[\nu+1]}$ έχει τιμή k+1.
M(Φ)	από την 1η τιμή που δίδει $\Phi[\boldsymbol{\lambda}_{[1]}, ..., \boldsymbol{\lambda}_{[\nu]}, k] = 0$. Για να βρούμε αυτή την τιμή αποτιμούμε διαδοχικά (με το τρόπο που περιγράφουμε...) τα $\Phi[\boldsymbol{\lambda}_{[1]}, ..., \boldsymbol{\lambda}_{[\nu]}, \mathbf{k=0}]$, $\Phi[\boldsymbol{\lambda}_{[1]}, ..., \boldsymbol{\lambda}_{[\nu]}, \mathbf{k=1}]$, κοκ.

- *Τέλος υπολογισμού v_{last}:* Ο υπολογισμός τερματίζει όταν δεν εμφανίζεται καμμία προς υπολογισμό περιγραφή W. Εάν ο υπολογισμός δεν τερματίζει (δηλαδή η υπολογιζόμενη συνάρτηση έχει την αόριστη τιμή ↑), τότε, απλά... τη πατήσαμε: δεν θα το μάθουμε ποτέ!

Μετά τα σχόλια, δίνουμε ένα παράδειγμα ορισμού και υπολογισμού αναδρομικής περιγραφής.

Γραφή και γραφή & ένα σχόλιο περί συναρτησιακού προγραμματισμού.

Στα προηγούμενα δεν φαίνεται ιδιαίτερα καθαρά σε τί διαφέρουν οι δύο εκδοχές σημασιοδότησης που περιγράψαμε σε αυτή την ενότητα. Ακόμα και εξ όψεως τα γραφόμενα στις δύο περιπτώσεις φαίνονται τα ίδια με άλλα λόγια! Πόση διαφορά μπορεί να έχουν τα εξής δύο (αντεγραμμένα από τις προηγούμενες σελίδες):

$$f_{\Phi}(f_{H1}(\mathbf{x}), ..., f_{Hk}(\mathbf{x})) \quad \text{και} \quad \Phi[H_1\,\boldsymbol{\lambda}, H_2\,\boldsymbol{\lambda}, ..., H_k\,\boldsymbol{\lambda}] \quad (?)$$

Και πράγματι η διαφορά τους συγκαλύπτεται από το γεγονός ότι όσα «λέμε», δεν τα σκεπτόμαστε απλώς αλλά τα γράφουμε... Πρέπει όμως, (και ιδίως σε ένα μάθημα σαν κι' αυτό...) να προσέχουμε πότε γράφουμε περί *εννοιών*, και πότε γράφουμε περί *γραφής*. Και αυτό συμβαίνει στη περίπτωσή μας:

- στη 1η εκδοχή της σημασιολογίας μας γράφουμε περί των *εννοιών* που εμείς «τώρα» χρησιμοποιούμε, λ.χ. τις συναρτήσεις $f_D(\cdot)$.
- στη 2η εκδοχή της σημασιολογίας μας περιγράφουμε ποιές *γραπτές ενέργειες* [3], που θα μπορούσε να έκανε κάποιο πρόσωπο (ή, ακόμα προτιμότερα, μια *μηχανή*), προκειμένου να *υπολογίσει* στη πράξη τις όποιες υποδηλούμενες συναρτήσεις εννοούμε. Και έχει σημασία να εξασφαλίσουμε πώς ότι μπορεί να γίνει στο ένα επίπεδο μπορεί (?) να γίνει και στο δεύτερο...

Οι δύο σημασιοδοτήσεις των αναδρομικών περιγραφών μας δίνουν την ευκαιρία να δούμε καθαρά δύο πλευρές του προγραμματισμού:

- Από την μια πλευρά έχουμε ένα τρόπο να ***ορίζουμε*** αυτό που μας ενδιαφέρει, και αυτό είναι (θεμελιακά) μια σχέση ή συνάρτηση δεδομένων & αποτελεσμάτων. Είμαστε επιπλέον σε θέση να χειριζόμαστε αυτές τις συναρτήσεις, ώστε από ήδη ορισμένες να *υποδηλώνουμε* νέες. Σε αυτή την διαδικασία δεν θέλουμε, ούτε θα έπρεπε να μας απασχολεί, το πώς θα υπολογιστούν αυτές οι συναρτήσεις – εξ άλλου σε *αυτό ακριβώς είναι εκείνο στο οποίο ζητούμε την συνέργεια των μηχανών*: εμείς ορίζουμε, και ***εκείνες υπολογίζουν***.
- Από την άλλη πλευρά, κάποια στιγμή, θα χρειαστεί να ***εκτελεστούν*** οι προγραμματισμένοι υπολογισμοί [4]. Και εκεί είναι η *τελεστική* σημασιολογία αυτό που μας λέει τι πρέπει και αρκεί να κάνουμε.

Στις αναδρομικές περιγραφές (στοιχειακή μορφή του συναρτησιακού προγραμματισμού) διαφαίνονται με τον πιο κάθετο τρόπο αυτές οι δύο όψεις, και στις παραπάνω δύο σημασιοδοτήσεις έχουμε μια ακτινογραφία του προγραμματισμού. Για την 2η σημασιολογία, την τελεστική, ενδιαφερόμενοι είναι κυρίως οι «συστηματζήδες». Οι «κωδικατζήδες» θα έπρεπε να νοιάζονται κυρίως για την 1η σημασιολογία – την υποδηλωτική: θα πρέπει να σκέφτονται ***τί*** έχουν να υπολογίσουν, και όχι το ***πώς*** αυτό θα συμβεί μέσα σε τούτο ή το άλλο μηχάνημα.

(Για τους «αλγοριθμάδες» διατίθεται και μια 3η οπτική γωνία: η ***αξιωματική*** σημασιολογία, με σαφή λογικό χαρακτήρα, διότι αυτοί έχουν να εξασφαλίσουν το πλέον πολύτιμο και αναντικατάστατο: την *αποδεδειγμένη ορθότητα* των αλγορίθμων.)

[3] «Γραπτές» = δυνάμενες να γραφούν.

[4] Αξίζει να δώσει κάποιος εδώ προσοχή στην ετυμολογική και γραμματική ακρίβεια των όρων: *προ-γραμματ-ισμένη εκ-τέλεση*: σε αυτές τις φράσεις σχεδόν κάθε γράμμα έχει την σημασία του: έχουμε ήδη (*-ισμένη*), εκ των προτέρων (*προ-*), γράψει (*-γραμματ-*) τι θα συμβεί (*-τέλεση*) ως (*εκ-* τούτου) αποτέλεσμα.

Παράδειγμα: αναδρομική περιγραφή της συνάρτησης «άθροισης».
Ας δούμε πώς θα περιγράφαμε αναδρομικά την συνάρτηση της άθροισης ADD(·,·), η οποία παραλαμβάνει δύο παραμέτρους x, y, και παραδίδει το άθροισμά τους x+y:

$$\text{ADD}(x, y) = x+y$$

και πώς σύμφωνα με την περιγραφή-πρόγραμμά μας θα την υπολογίζαμε για x=3 και y=2. Γνωρίζουμε ότι x+(y+1)=(x+y)+1 (μια προσεταιριστική ιδιότητα), ότι δηλαδή:

$$\text{ADD}(x, y+1) = \text{ADD}(x, y) + 1$$

Έχουμε λοιπόν δύο παραμέτρους για την άθροιση ADD, μια αναδρομή στην δεύτερη παράμετρο (από y+1 σε y), και κατά τα παραπάνω η συνάρτηση "βάσης" πρέπει να έχει (2–1) = μία παράμετρο (να δίνει την τιμή x+0) και η συνάρτηση "βήματος" (2+1) = τρείς παραμέτρους:

- την «παράμετρο», $x_1 = x$.
- το «προηγούμενο όρισμα», $x_2 = y$,
- και την «προηγούμενη τιμή», $x_3 = x+y = \text{ADD}(x, y) = \text{ADD}(x_1, x_2)$.

Κατά συνέπεια:

επιδίωξη:	*τιμή =*	*μέσω...*	*περιγραφή:*	*σχόλιο*
$\text{ADD}(x_1, 0) = ??$	x_1	(x_1)	$B = I_{1,1}$	Η ταυτοτική.
$\text{ADD}(x_1, x_2+1) = ??$	$\text{ADD}(x_1, x_2)+1$ ή: x_3+1	(x_1, x_2, x_3)	$P = S(x_3)$ $= S(I_{3,3}(x_1, x_2, x_3))$ $= C(S, I_{3,3})(x_1, x_2, x_3)$	Επιλέγουμε την 3η παράμετρο, (= την προηγούμενη τιμή), και την αυξάνουμε κατά +1.

ή τελικά:

$$D_{\text{ADD}} = \mathbf{R\,(\,I_{1,1}\,,\,C(\,S,\,I_{3,3})\,)}\ .$$

Για τον υπολογισμό της συνάρτησης που περιγράφει η ADD με ορίσματα [3, 2] θα έχουμε τα παρακάτω στάδια και βήματα. Η εκάστοτε προς αποτίμηση περιγραφή W σκιάζεται, και η πλειάδα **λ** των ορισμάτων της γράφεται με παχείς χαρακτήρες.

```
(* ------------------------------------------------------ *)
ADD[3,2] = ...
R(I1,1,C(S,I3,3))[3,  2]
                 [3,1+1]
C(S,I3,3)[3,1,R(I1,1,C(S,I3,3))[3,  1]]   {Προσοχή!}
                               [3,0+1]
          C(S,I3,3)[3,0,R(I1,1,C(S,I3,3))[3,0]]
                               I1,1[3]))
          C(S,I3,3)[3,0,3]
          S[I3,3[3,0,3]]
          S[3]
C(S,I3,3)[3,1,4]
S[I3,3[3,1,4]]
S[4]
[5]
... = 5     (ευτυχώς...)
(* ------------------------------------------------------ *)
```

Παράδειγμα: η αναδρομική περιγραφή της συνάρτησης «υπεροχής».
Ας δώσουμε μια αναδρομική περιγραφή της αφαίρεσης. Επειδή εδώ ασχολούμαστε με *φυσικούς αριθμούς*, θα έχουμε τον εξής ειδικό ορισμό:

$$\text{OVER}(x, y) = \text{εάν } x \geq y \text{ τότε } \text{OVER}(\cdot) = (x–y), \text{ αλλοιώς, (εάν } x < y), \text{ τότε } \text{OVER}(\cdot) = 0.$$

Η αναδρομική περιγραφή της συνάρτησης αυτής ανιχνεύεται με παρόμοιο τρόπο: εάν y = 0 θα είναι OVER(x, 0) = x. Αλλά εάν πρέπει να υπολογίσουμε το OVER(x, y+1) μέσω...

- της «παραμέτρου», $x_1 = x$.
- του «προηγούμενου» ορίσματος, $x_2 = y$,
- και της «προηγούμενης τιμής», $x_3 = x+y = \text{OVER}(x, y) = \text{OVER}(x_1, x_2)$.

τί θα (έπρεπε να) κάναμε; Ο παρακάτω πίνακας μας λέει ότι OVER(x, k+1) = OVER(OVER(x, k), 1):

x	y = k+1	OVER(x, k+1)	y = k	OVER(x, k)	OVER(OVER(x, k), 1)
3	5	0	4	0	0
3	4	0	3	0	0
3	3	0	2	1	0
3	2	1	1	2	1
3	1	2	0	3	2

Έχουμε λοιπόν να λύσουμε ένα απλούστερο πρόβλημα: να βρίσκουμε το x«–1», δηλαδή τον προηγούμενο αριθμό του x, έστω Δ(x), (με προηγούμενο του μηδενός το ίδιο το μηδέν). Αυτό είναι πιο «εύκολο»:

επιδίωξη:	*τιμή =*	*μέσω...*	*περιγραφή:*	*σχόλιο*
$\Delta(0) = ??$	0	(-)	$B = Z_0$	Η μηδενική σταθερά.
$\Delta(x_1+1) = ??$	$\Delta(x_1+1)$ ή: x_1	(x_1, x_2)	$P = I_{2,1}(x_1, x_2)$	Επιλέγουμε την 1η παράμετρο, (= το προηγούμενο όρισμα).

δηλαδή η σχετική αναδρομική περιγραφή D_Δ, (μονοπαραμετρική), είναι,

$$D_\Delta = \mathbf{R(Z_0, I_{2,1})} ,$$

οπότε είμαστε σε θέση να ολοκληρώσουμε την περιγραφή της «υπεροχής»,

επιδίωξη:	*τιμή =*	*μέσω...*	*περιγραφή:*	*σχόλιο*
$\text{OVER}(x_1, 0) = ?$	x_1	(x_1)	$B = I_{1,1}$	Η ταυτοτική.
$\text{OVER}(x_1, x_2+1) = ?$	$\Delta(\text{OVER}(x_1,x_2)) = \Delta(x_3)$	(x_1, x_2, x_3)	$P = D_\Delta(I_{3,3}(x_1, x_2, x_3))$	Επιλέγουμε την 3η παράμετρο, (= την προηγούμενη τιμή), και υπολογίζουμε το Δ(-) επ' αυτής.

το οποίο μας δίνει την συνολική τελική περιγραφή:

$$D_{\text{OVER}} = R(I_{1,1} , C(\mathbf{D_\Delta}, I_{3,3})) = \mathbf{R(I_{1,1} , C(R(Z_0, I_{2,1}), I_{3,3}))}$$

20ο Θετικά νέα: Ισχύς(Αναδρομικές/Περιγραφές) ⊆ Ισχύς(Μηχανές/Μνήμης).

Η ισοδυναμία δύο διαφορετικών τρόπων υπολογισμού.

Έχουμε όπως έχει φανεί διάφορους τρόπους να «προγραμματίζουμε» και ως εκ τούτου να «υπολογίζουμε». Προκύπτει λοιπόν το ζήτημα να συγκρίνουμε αυτούς τους τρόπους. Πώς;

Για τους τρείς (μεταξύ πολλών άλλων) τρόπους προγραμματισμού που έχουμε περιγράψει δώσαμε τους *συντακτικούς* κανόνες για αυτούς. Δώσαμε επίσης και μια *σημασιολογία* των προγραμμάτων που προκύπτουν. Υπάρχουν διάφοροι τρόποι για να αποδίδουμε σημασία στους προγραμματικούς συμβολισμούς. Ένας απλοϊκός τρόπος λ.χ. είναι να θεωρήσουμε ως σημασία ενός προγράμματος τον ίδιο τον εαυτό του «*κατά γράμμα*».

Μια συνοπτική αντίληψη του τι διαφορά δημιουργεί το να αποδίδουμε τούτη ή εκείνη την σημασία, αποκτούμε εάν εξετάσουμε το πότε – ως συνέπεια της σημασιοδότησης – θα έπρεπε δύο προγράμματα να θεωρούνται ισοδύναμα (ή στα Ελληνικά: ***ταυτό-σημα***): προφανώς εάν και μόνον έχουν την ίδια σημασία. Εάν λοιπόν ως σημασία ενός προγράμματος θεωρήσουμε αυτό το ίδιο το κείμενο του προγράμματος τότε δύο προγραμματα (θα) θεωρούνται ταυτόσημα εάν και μόνον συμπίπτουν γράμμα-προς-γράμμα. Πρόκειται λοιπόν για μια ισοπεδωτική σημασιοδότηση διότι λ.χ. αυτό συνεπάγεται ότι ακόμα και ένα απλό διάστημα να βάλουμε στο κείμενο ενός προγράμματος αυτό αποκτά (?) άλλη σημασία!

Από μια άλλη και αφ' υψηλού οπτική γωνία έχουμε ήδη δεί ότι αυτό που, κατά το ελάχιστον, μας ενδιαφέρει σε κάθε πρόγραμμα είναι η σχέση εισόδου-εξόδου, δηλαδή η συνάρτηση που υπολογίζει. Ως προς αυτή τη σημασιοδότηση δύο προγράμματα θεωρούνται ταυτόσημα εάν και μόνον υπολογίζουν την ίδια συνάρτηση. Και αυτή όμως η οπτική είναι κατά κάποια έννοια ισοπεδωτική, αν και προς την αντίθετη κατεύθυνση· διότι μπορεί λ.χ. ένα πρόγραμμα π να υπολογίζει την ίδια συνάρτηση όπως κάποιο άλλο π', αλλά να το επιτυγχάνει μέσω ενός πολύ ταχυτέρου αλγορίθμου από εκείνον του π', και άρα να έχει για μας ουσιώδεις διαφορές και πλεονεκτήματα. Με βάση τα παραπάνω δίνουμε στο ακόλουθο πίνακα 5 διαφορετικά επίπεδα «σημασίας» που μπορούμε να αποδώσουμε στα προγράμματα. (Ο κατάλογος, φυσικά, δεν είναι εξαντλητικός).

«Σημασία» προγράμματος:	***Ταυτόσημα:***
Η συνάρτηση που υπολογίζει.	Υπολογίζουν την ίδια συνάρτηση.
Η συνάρτηση που υπολογίζει και το μήκος του κάθε υπολογισμού.	Υπολογίζουν την ίδια συνάρτηση και με το «ίδιο» πλήθος βημάτων.
Ο υπολογισμός για κάθε δεδομένο.	Ίδιοι υπολογισμοί για τα ίδια δεδομένα
Η συνάρτηση που υπολογίζει, και ο «τρόπος» που χρησιμοποιείται.	Υλοποιούν τον «ίδιο» αλγόριθμο.
Η δομή του προγράμματος	Προκύπτει το ένα από το άλλο με μια μετωνομασία των μεταβλητών
Το κείμενο του προγράμματος	Είναι ίδια «κατά γράμμα».

Εμείς θα σταθούμε αρχικά στο πρώτο και πιο αφηρημένο επίπεδο, θεωρώντας δύο προγράμματα ως ταυτόσημα εάν υπολογίζουν την ίδια συνάρτηση «εισόδου-εξόδου».

Ο αναγνώστης ας δεί τα παρακάτω προγράμματα της δεξιάς στήλης. Ποιά μπορεί να πεί κανείς ότι είναι «ίδια», (λ.χ. υλοποιούν τον «ίδιο αλγόριθμο»), με το πρόγραμμα στη αριστερή στήλη;

```
S←0, Για k = 1 ως N {S←S+T[k]}
```

(i)
```
S←T[1], Για k= 2 ως N {S←(S+T[k])}
```

(ii)
```
Sum←0, Για m = 1 ως N {Sum←(Sum+T[m])}
```

(iii)
```
S←0, Για k = 1 ως N {S←(S+T[k])}
S←0, Για k = 1 ως N {S←(S+T[k])}
```

(iv)
```
S←0, Για k = N κάτω-ως 1 {S←(S+T[k])}
```

(v)
```
S←0, Για k = N κάτω-ως 1 {S←(S+T[N+1-k])}
```

(vi)
```
S←0, Για k = 1 ως N {S←(T[k]+S)}
```

(vii)
```
S←0, Για k = 1 ως N {X←(S+T[k]), S←X}
```

Οι μηχανές μνήμης υπολογίζουν (τουλάχιστον) ότι υπολογίζουν οι αναδρομικές περιγραφές.
Έστω ότι μας δίδεται μια αναδρομική περιγραφή D με ν παραμέτρους. Η περιγραφή αυτή, κατά τα προηγούμενα, αντιστοιχεί σε μια ν-παραμετρική συνάρτηση ϕ_D:

$$\phi_D (x_1, x_2, ..., x_\nu) : \mathbf{N}^\nu \rightarrow \mathbf{N} \cup \{ \uparrow \}$$

Θα δείξουμε ότι μπορούμε να φτιάξουμε ένα πρόγραμμα π_D για μια μηχανή μνήμης, το οποίο να υπολογίζει την ίδια συνάρτηση $\phi_D(\cdot)$, με τις εξής συμβάσεις:

- Εάν το π_D εκκινήσει από μια κατάσταση όπου οι ν πρώτες θέσεις μνήμης περιέχουν τις τιμές των παραμέτρων $(x_1, x_2, ..., x_\nu)$ τότε,
  - εάν $\phi_D(x_1, x_2, ..., x_\nu) \downarrow$, το π_D τερματίζει και στη 1η – κατά σύμβαση – θέση μνήμης περιέχεται η τιμή $\phi_D(x_1, x_2, ..., x_\nu)$.
  - εάν $\phi_D(x_1, x_2, ..., x_\nu) \uparrow$, το π_D επίσης δεν τερματίζει.

Μια περιγραφή D ορίζεται με βάσει τους συντακτικούς κανόνες των αναδρομικών περιγραφών οι οποίοι είναι επαγωγικοί· κατά συνέπεια και η απόδειξή μας και κατασκευή μας θα έχει επαγωγικό χαρακτήρα. Εάν π.χ. έχουμε ορίσει τα προγράμματα π_B, π_P, που υπολογίζουν τις περιγραφές των B και P τότε ορίζεται (?) το πρόγραμμα $\pi_{R(B,P)}$ που υπολογίζει την περιγραφή R(B,P); Επειδή λοιπόν θα χρειαστεί από δεδομένα προγράμματα να ορίζουμε νέα, οι εξής κατασκευές θα μας είναι χρήσιμα:

«Μετατόπιση», π[+δ], ενός προγράμματος π κατά δ.
Ένα πρόγραμμα π για μια συνάρτηση ν-παραμέτρων δέχεται τις τιμές-ορίσματα στις υπ.αρ. 1, 2, ..., ν θέσεις μνήμης, και επιστρέφει το αποτέλεσμα στη 1η θέση μνήμης. Εάν όλες οι αναφορές του π σε θέσεις μνήμης «μετατοπιστούν» κατά δ θέσεις, τότε εάν το π παραλάβει τα ορίσματα στις υπ.αρ. δ+1 έως δ+ν θέσεις μνήμης, θα παραδώσει το αποτέλεσμά του στην δ+1 θέση μνήμης. Συμβολίζουμε λοιπόν με π[+δ] το πρόγραμμα που προκύπτει από το π εάν αντικαταστήσουμε κάθε...

εντολή z_κ	με	$z_{\delta+\kappa}$
εντολή s_κ	με	$s_{\delta+\kappa}$
εντολή $j_{\kappa,\lambda,\varrho}$	με	$j_{\delta+\kappa,\ \delta+\lambda,\ \varrho}$

Είναι σχεδόν προφανές πώς όποια επίδραση θα είχε το π στις θέσεις μνήμης υπ.αρ. 1 ... ν, θα έχει το μετατοπισμένο π[+δ] στις θέσεις υπ.αρ. δ+1 ... δ+ν. Η πρόσθετη (τεχνική) αξία αυτής της μετατόπισης κατά +δ θέσεις είναι η απόλυτη βεβαιότητα ότι το νέο πρόγραμμα δεν αναφέρεται και, κατά συνέπεια, είναι *αδύνατον* να επιδρά στις υπ.αρ. 1...δ θέσεις μνήμης όπου επομένως είναι δυνατόν να διαφυλάξουμε όποια απαραίτητα αρχικά δεδομένα ή/και ενδιάμεσα αποτελέσματα.

«Αλληλουχία», $\pi_1|\pi_2$, δύο προγραμμάτων.-
Όταν έχουμε στη διάθεσή μας δύο προγράμματα π_1, π_2 συχνά επιθυμούμε να τα συνδέσουμε έτσι ώστε το 2ο εξ αυτών να συνεχίσει από εκεί που τέλειωσε το 1ο εξ αυτών. Θα γράφουμε ως $\pi_1|\pi_2$ ένα

πρόγραμμα που προκύπτει από την αλληλουχία των εντολών των δύο προγραμμάτων – κάπως «πειραγμένη». Γράφοντας με π[κ] την υπ.αρ. κ εντολή του προγράμματος π ορίζουμε την αλληλουχία $\pi_1 | \pi_2$ ως εξής:

- *Αλληλουχία εντολών:*
  - για κ=1 έως μ(π_1) $\quad \pi_1 | \pi_2 [\kappa]) = \pi_1[\kappa]$
  - για κ=1 έως μ(π_2) $\quad \pi_1 | \pi_2 [\mu(\pi_1) + \kappa]) = \pi_2[\kappa]$
- *Εντολές τερματισμού του* π_1: επειδή όμως το 1ο πρόγραμμα π_1 τερματίζει με εντολές της μορφής $j_{\kappa,\lambda,0}$, ενώ αντιθέτως στην αλληλουχία $\pi_1 | \pi_2$ θέλουμε να συνεχίσουμε με το 2ο πρόγραμμα π_2, πρέπει να τροποποιήσουμε τις εντολές « j » του π_1 κατά τον εξής τρόπο:
  - μετατρέπουμε κάθε εντολή $j_{\kappa,\lambda,0}$ του π_1 σε $j_{\kappa,\lambda,\mu(\pi 1)+1}$.
- *Εντολές μεταπήδησης του* π_2: επειδή οι εντολές του π_2 έχουν μετατοπιστεί κατά +μ(π_1) θέσεις μέσα στην αλληλουχία $\pi_1 | \pi_2$, πρέπει να τροποποιήσουμε τις « j » εντολές του π_2:
  - μετατρέπουμε κάθε εντολή $j_{\kappa,\lambda,\rho}$ ($\rho \neq 0$) του π_2 σε $j_{\kappa,\lambda,\mu(\pi 1)+\rho}$ [5]

Αυτές οι δύο πράξεις και συμβολισμοί, π[+δ] και $\pi_1 | \pi_2$, μας επιτρέπουν να αποδείξουμε το εξής θεώρημα:

Θεώρημα: οι μηχανές μΜ υπολογίζουν ότι και οι αναδρομικές περιγραφές.
Ισχυρισμός: Για κάθε αναδρομική περιγραφή D, έστω ν παραμέτρων, υπάρχει ένα πρόγραμμα π_D, το οποίο υπολογίζει την ίδια συνάρτηση $\phi_D(x_1, x_2, ..., x_\nu) : \mathbf{N}^\nu \rightarrow \mathbf{N} \cup \{\uparrow\}$.
Σχέδιο απόδειξης: Κάθε αναδρομική περιγραφή D, επιδέχεται ένα «μέγεθος» |D|, ίσο με το πλήθος των συμβόλων τύπου Z, S, I, C, R ή M που χρησιμοποιεί. Θα δείξουμε ότι εάν υπάρχουν προγράμματα μΜ για όλες τις αναδρομικές περιγραφές με μέγεθος έως και k, τότε θα υπάρχουν και προγράμματα για όλες τις περιγραφές μεγέθους έως και k+1.

1. *Βάση επαγωγής: προγράμματα μΜ για τις βασικές αναδρομικές περιγραφές.*
Θα δείξουμε ότι υπάρχουν προγράμματα μΜ για όλες τις αναδρομικές περιγραφές D μεγέθους |D| = 1, δηλαδή τις βασικές περιγραφές Z, I, S.

1.α. *Περιγραφές «Z»*
Κάθε βασική αναδρομική περιγραφή $D = Z_\kappa$ υπολογίζεται από το πρόγραμμα: $\boldsymbol{\pi_D = \langle z_1, j_{1,1,0} \rangle}$, το οποίο παραδίδει στη 1η θέση μνήμης την τιμή 0, όπως κάνει και η Z_κ.

1.β. *Περιγραφές «I»*
Κάθε βασική αναδρομική περιγραφή $D = I_{\nu,\kappa}$ υπολογίζεται ως εξής:

- Για κ = 1 δεν έχουμε να κάνουμε τίποτε, οπότε αρκεί το τετριμμένο πρόγραμμα $\boldsymbol{\pi_D = \langle j_{1,1,0} \rangle}$.
- Για κ > 1 η περιγραφή D υπολογίζεται από το πρόγραμμα $\boldsymbol{\pi_D =}$ **COPY(κ, 1)**, το οποίο παραδίδει στη 1η θέση μνήμης το περιεχόμενο της υπ.αρ. κ θέσης μνήμης, όπως κάνει και η $I_{\nu,\kappa}$. Το πρόγραμμα COPY(α, β), α≠β, αντιγράφει το περιεχόμενο της υπ.αρ. α θέσης μνήμης στη υπ.αρ. β θέση μνήμης, και ορίζεται ως εξής:

	COPY(α, β)	*Σχόλιο:*
1	z_β	μηδενίζουμε την θέση β.
2	$j_{\beta,\alpha,0}$	εάν η β έχει εξισωθεί με την α, τότε τέλος.
3	s_β	αυξάνουμε την β κατά +1, και,
4	$j_{1,1,2}$	επαναλαμβάνουμε από την 2η εντολή.

(Προσέξτε πώς αποτυγχάνει το COPY πρόγραμμα όταν α = β ...)

[5] Ας προσέξουμε εδώ ότι αυτές οι δύο πράξεις «ανατοποθετούν» τα προγράμματα μΜ, η μεν 1η στο χώρο της μνήμης, η δε 2η στο χώρο των εντολών – όπως εξ άλλου φαίνεται στην τροποποίηση των εντολών τύπου «j»: στη 1η αλλάζουμε κατά +δ τις αναφορές θέσεων μνήμης, και στη 2η αλλάζουμε κατά +μ(·) τις αναφορές εντολών. Αυτή η δουλειά γίνεται «all the time» στα υπολογιστικά συστήματα από τους λεγόμενους *φορτωτές* (loaders).

1.γ. *Περιγραφές «S»*

Η περιγραφή D = S υπολογίζεται από το πρόγραμμα $\pi_D = \langle s_1, j_{1,1,0} \rangle$, το οποίο παραλαμβάνοντας μια τιμή στην 1η θέση μνήμης, την επιστρέφει στη 1η θέση μνήμης αυξημένη κατά +1, όπως κάνει και η περιγραφή S.

2. *Βήμα επαγωγής: προγράμματα μM για τις σύνθετες αναδρομικές περιγραφές.*

Θα δείξουμε ότι, για κάθε k ≥ 2, εάν υπάρχουν προγράμματα μM για όλες τις αναδρομικές περιγραφές D μεγέθους |D| ≤ k, τότε θα υπάρχουν και για όλες τις περιγραφές μεγέθους |D| ≤ k+1.

2.α. *Περιγραφές «σύνθεσης» D = C(Φ, H1, H2, H3) – παράδειγμα ν = 2 παραμέτρων.*

Θα δώσουμε την «απόδειξη» μέσω ενός παραδείγματος σύνθεσης D = C(Φ, H1, H2, H3), το οποίο συνθέτει μια περιγραφή Φ με ν = 3 παραμέτρους, και τρείς περιγραφές H1, H2 και H3, εκάστη με ν = 2 παραμέτρους. Η αναδρομική περιγραφή Φ έχει *μικρότερο μέγεθος* από την D, δηλαδή |Φ| < |D| = (k+1), και |Φ| ≤ k. Το ίδιο ισχύει και για τις περιγραφές H1, H2, H3. Από την επαγωγική υπόθεση έχουμ5 ότι υπάρχουν για τις Φ, H1, H2, H3 προγράμματα μM π_Φ, π_{H1}, π_{H2}, π_{H3} που τις υπολογίζουν αντιστοίχως. Ας προσέξουμε εδώ ότι C(Φ, H1, H2, H3)[x1, x2] = Φ[H1[x1, x2], H2[x1, x2], H3)[x1, x2], και ότι επομένως αρκεί να υπολογίσουμε τα H1, H2, H3 επί των τιμών [x1,x2], και στη συνέχεια να εφαρμόσουμε επί των αποτελεσμάτων την Φ. Το πρόγραμμα λοιπόν που υπολογίζει την περιγραφή D είναι το εξής:

	D=C(Φ, H1, H2, H3)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
	π_D ***m*[...] =**	x1	x2	...	...	...	...	...	...	...
1	COPY(1, 2+1)	x1	x2	x1	...					...
2..5	COPY(2, 2+2)	x1	x2	x1	x2	...				...
7	π_{H1}[+2]	x1	x2	H1	...					...
...	COPY(1, 3+1)	x1	x2	H1	x1	...				...
	COPY(2, 3+2)	x1	x2	H1	x1	x2	...			...
	π_{H2}[+3]	x1	x2	H1	H2	...				...
	COPY(1, 4+1)	x1	x2	H1	H2	x1	...			...
	COPY(2, 4+2)	x1	x2	H1	H2	x1	x2	...		...
	π_{H3}[+4]	x1	x2	H1	H2	H3	...			...
	π_Φ[+2]	x1	x2	Φ	...					
	COPY(1, 2+1)	D	...	...	...	...	...	...	...	...

Ας προσέξουμε εδώ την τελική εντολή COPY(-, -), η οποία φροντίζει το αποτέλεσμα να παραδοθεί στην 1η θέση μνήμης, κατά την σύμβασή μας.

2.β. *Περιγραφές «αναδρομής» D = R(B, P) – παράδειγμα ν = 3 παραμέτρων.*

Θα δώσουμε την εξήγηση μέσω παραδείγματος μιας περιγραφής D = R(B, P), με ν = 3 παραμέτρους. Οι περιγραφές B (βάση), P (βήμα) θα έχουν, κατά συνέπεια, 2 και 4 παραμέτρους αντιστοίχως. Οι αναδρομικές περιγραφές B, P έχουν *μικρότερο μέγεθος* από την D και μπορούμε επαγωγικά να υποθέσουμε ότι υπάρχουν για αυτές προγράμματα μηχανών-μνήμης π_B, π_P που τις υπολογίζουν. Ας προσέξουμε εδώ ότι π.χ. η τιμή τ της R(B,P)[x, y, 3] υπολογίζεται όπως φαίνεται στην αριστερή στήλη του επόμενου πίνακα, και επομένως είναι δυνατόν να υπολογιστεί και «ευθέως» όπως φαίνεται στη δεξιά στήλη:

R(B,P)[x, y, 3] = P[x, y, 2, R(B,P)[x, y, 2]]	τ ← B[x, y]
R(B,P)[x, y, 2] = P[x, y, 1, R(B,P)[x, y, 1]]	τ ← P[x, y, 0, τ]
R(B,P)[x, y, 1] = P[x, y, 0, R(B,P)[x, y, 0]]	τ ← P[x, y, 1, τ]
R(B,P)[x, y, 0] = B[x, y]	τ ← P[x, y, 2, τ],
	R(B,P)[x, y, 3] = τ

Για να υπολογίσουμε λοιπόν την τιμή της R(B,P)[x1, x2, x3] αρκεί να υπολογίσουμε την τιμή τ = B[x1, x2], και στη συνέχεια τις τιμές τ ← P[x1, x2, κ, τ], για κ = 1, 2, ..., x3. Το πρόγραμμα που υπολογίζει την περιγραφή D = R(B,P), ν = 3 παραμέτρων, είναι λοιπόν το εξής:

	D = R(B, P)	1	2	3	4	5	6	7	8	9
	π_D m[...]	x_1	x_2	x_3	...	...	...	...	...	...
1	z 4	x_1	x_2	x_3	0					...
2	COPY(1, 4+1)	x_1	x_2	x_3	0	x_1				...
...	COPY(2, 4+2)	x_1	x_2	x_3	0	x_1	x_2			...
	π_B[+4]	x_1	x_2	x_3	0	$\tau_{(B)}$				...
	COPY(1+4, 4+4)	x_1	x_2	x_3	0				$\tau_{(B)}$	...
#1	j 4, 3, #2	x_1	x_2	x_3	κ				$\tau_{B/P}$	...
	COPY(1, 4+1)	x_1	x_2	x_3	κ	x_1			$\tau_{B/P}$	...
	COPY(2, 4+2)	x_1	x_2	x_3	κ	x_1	x_2		$\tau_{B/P}$	...
	COPY(1+3, 4+3)	x_1	x_2	x_3	κ	x_1	x_2	κ	$\tau_{B/P}$	...
	π_P[+4]	x_1	x_2	x_3	κ	$\tau_{(P)}$			$\tau_{B/P}$	...
	COPY(1+4, 4+4)	x_1	x_2	x_3	κ				$\tau_{(P)}$	...
	s 4	x_1	x_2	x_3	κ++				$\tau_{(P)}$	...
	j 4, 4, #1	x_1	x_2	x_3	κ				$\tau_{(P)}$	...
#2	COPY(4+1, 1)	D	..	...	...	...	...	...	...	...

όπου,

#1 = 1 + 2μ(COPY) + **μ(π_B)** + 1

#2 = 1 + 2μ(COPY) + **μ(π_B)** + μ(COPY) + 1 + 3μ(COPY) + **μ(π_P)** + μ(COPY) + 1 + 1 + μ(COPY).

2.γ. *Περιγραφές «αναζήτησης»* D = M(Φ) – *παράδειγμα ν = 2 παραμέτρων.*

Θα δώσουμε ένα παράδειγμα για περιγραφή D = M[Φ], για ν = 2 παραμέτρους. Η περιγραφή Φ, (εδώ με ν = 3 παραμέτρους), έχει μικρότερο μέγεθος από την D και μπορούμε επαγωγικά να υποθέσουμε ότι υπάρχει για αυτή πρόγραμμα π_Φ που την υπολογίζει.

Εδώ ας προσέξουμε ότι M(Φ)[x_1, x_2] = min{ κ: Φ(x_1, x_2, κ) = 0 }, και ότι επομένως το ζητούμενο κ είναι η «1η» τιμή x_3 που μηδενίζει την Φ. Αρκεί επομένως να υπολογίζουμε τις τιμές Φ[x_1, x_2, κ] για κ = 1, 2, ..., έως ότου βρούμε (?) την 1η (άρα μικρότερη) που δίδει Φ[x_1, x_2, x_3]. Το πρόγραμμα που υπολογίζει την M[Φ] είναι λοιπόν ως εξής:

	D = M(Φ)	1	2	3	4	5	6	7	8	9
	π_D m[...] =	x_1	x_2	...						...
1	z 3	x_1	x_2	0	...					...
2	z 4	x_1	x_2	0	0	...				...
3..6	COPY(1, 5+1)	x_1	x_2	κ	0	x_1	...			...
...	COPY(2, 5+2)	x_1	x_2	κ	0	x_1	x_2	...		...
	COPY(3, 5+3)	x_1	x_2	κ	0	x_1	x_2	κ	...	...
	π_Φ[+4]	x_1	x_2	κ	0	Φ	...			...
	j 5, 4, #1	x_1	x_2	κ	0	Φ				...
	s 4	x_1	x_2	κ++	0	...				...
	j 5, 5, 3	x_1	x_2	κ	0	...				...
#1	COPY(3, 1)	D	..	...	...	...	...	...	...	...

όπου,

#1 = 1 + 1 + 3μ(COPY) + **μ(π_Φ)** + 1 + 1 + 1 + 1.

Αυτή η τελευταία κατασκευή ολοκληρώνει – αν και απλώς δια μέσου παραδειγμάτων – την επαγωγική απόδειξη. (Προσέξατε ότι κατασκευή μας παράγει ένα *ορθώς συντεταγμένο* πρόγραμμα μM;)

■

Τι πρέπει να «μάθουμε» από το παραπάνω σχέδιο απόδειξης;

Η παραπάνω απόδειξη (ή μάλλον σχέδιο απόδειξης) δίδεται όχι μόνον για να πειστούμε ότι αληθεύει ο ισχυρισμός του θεωρήματος, αλλά γιατί η ίδια μας δίδει πολύτιμες πληροφορίες *για τον τρόπο που αυτό συμβαίνει* – πληροφορίες που θα έπρεπε ίσως να ήσαν μέρος της ίδιας της διατύπωσης αυτού του θεωρήματος:

- Η απόδειξη είναι κατασκευαστική: όχι μόνον αποδεικνύει την ύπαρξη ενός «ισοδύναμου» προγράμματος μM, αλλά και δίδει και τον τρόπο κατασκευής του.
- Η απόδειξη μας δίδει κατά το ήμισυ έναν ***διερμηνέα*** (*interpreter*) για τις αναδρομικές περιγραφές: η μεθοδολογία κατασκευής του κάθε π_D, μας επιτρέπει να σχεδιάζουμε και να «προγραμματίζουμε» στη γλώσσα [6] των αναδρομικών περιγραφών, και μέσω μιας «μετάφρασης» να εκτελούμε το σχετικό πρόγραμμα *σε μια μηχανή μM*, μηχανή που έχει την μορφή των μικροεπεξεργαστών που είμαστε σε θέση τεχνολογικά, (εδώ και καιρό), να κατασκευάζουμε. (Βέβαια η μετάφραση θα πρέπει να γίνεται από εμάς, διότι δεν έχουμε ακόμα ένα πρόγραμμα ικανό προς τούτο – γι' αυτό και λέμε ότι έχουμε μέχρι στιγμής έναν διερμηνέα «κατά το ήμισυ».)
- Το οριζόμενο κατασκευαστικώς πρόγραμμα π_D, όχι μόνον υπολογίζει ότι και η αναδρομική περιγραφή D, αλλά το κάνει και με τον «*ίδιο τρόπο*» που θα το έκανε και εκείνη – με την εξής έννοια: όποια ενδιάμεσα αποτελέσματα θα λαμβάναμε κατά τον υπολογισμό της D επί μιας πλειάδας παραμέτρων $[x_1, x_2, ..., x_v]$, τα «ίδια» ενδιάμεσα αποτελέσματα υπολογίζει (κατά διάφορες διαδοχικές φάσεις) και το ισοδύναμο π_D. Το πρόγραμμα π_D, δηλαδή *εξομοιώνει* την περιγραφή D.
- Τόσο η περιγραφή D όσο και το εξομοιωτικό πρόγραμμα π_D, δεν είναι απλώς αφηρημένες μαθηματικές οντότητες, αλλά επιδέχονται και μια απτή *συντακτική* περιγραφή, ως λέξεις-κείμενα επί ενός κατάλληλου αλφαβήτου. (Εξ άλλου με έναν τέτοιο τρόπο τα *γράφουμε* –συνεχώς– στα γραπτά μας!) Η παραπάνω κατασκευή μας λέει ότι η αντιστοίχιση $D \rightarrow \pi_D$, ως *αντιστοίχιση κειμένων*, είναι (?) μονοσήμαντη, δηλαδή:

$$\text{αν } (D \neq D') \text{ τότε και } (\pi_D \neq \pi_{D'})$$

Αυτή την στιγμή δεν φαίνεται σε τί είναι δυνατόν να μας ενδιαφέρει ή να μα χρησιμεύσει αυτό το γεγονός, και γι' αυτό δεν θα επιμείνουμε στις τεχνικότητες που απαιτεί η βεβαίωσή του. Στις επόμενες ενότητες όμως θα επανέλθουμε για να εξάγουμε από αυτό, ένα από τα 2-3 πιο σπουδαία συμπεράσματα αυτού του μαθήματος!

[6] Ποιό είναι το αλφάβητο αυτής της γλώσσας; Είναι αυτή η γλώσσα ασυμφραστική;

21ο Θετικά νέα: Ισχύς(Μηχανές/Μνήμης) ⊆ Ισχύς(Μηχανές/Turing).

Ότι ισχύει ανάμεσα στα προγράμματα μΜ και τις ΑΠ, ισχύει ανάμεσα στις μηχανές *Turing* και τις μΜ. Υπάρχει εδώ όμως μια (συχνά εμφανιζόμενη) διαφορά: στη 1η περίπτωση το πεδίο δράσης των μηχανών ήταν το ίδιο: οι «φυσικοί αριθμοί». Στη 2η περίπτωση οι μεν μΤ δρούν επί θέσεων ταινίας που περιέχουν σύμβολα, ενώ οι μΜ δρούν επί θέσεων μνήμης που περιέχουν αριθμούς. Για να συγκρίνουμε δύο διαφορετικούς προγραμματικούς συμβολισμούς θα πρέπει να επινοήσουμε και να τηρήσουμε κάποια σύμβαση αντιστοίχισης μεταξύ τους. Σε αυτή την ενότητα θα χρησιμοποιήσουμε μια σύμβασης περιγραφής των οντοτήτων που χειρίζονται οι μΜ, περιγραφή που θα είναι (τί άλλο) μια λέξη γραμμένη στο αλφάβητο μιας μηχανής *Turing*.

Βήμα* 1: *μια στενογραφία για προγράμματα Turing.
Θα χρησιμοποιήσουμε ένα πάγιο αλφάβητο Σ τερματικών συμβόλων, και επ' αυτού θα ορίσουμε ένα σύνολο από θεμελιακά βοηθητικά προγράμματα. Κάθε τέτοιο πρόγραμμα θα φέρει τις τρείς ειδικές καταστάσεις I, Y, N (αφετηριακή, αποδεκτική και απορριπτική τερματική), και τις καταστάσεις K_x και Λ_x για κάθε σύμβολο $x \in \Sigma$. Το σύμβολο/δείκτης x, θα λειτουργεί μνημονικά, για να θυμίζει ότι η σχετική κατάσταση Κ ή Λ ή όποια άλλη «θυμάται» επίσης, για τον οιοδήποτε λόγο, το σύμβολο $x \in \Sigma$.

Για λόγους οικονομίας (και στην γραφή και στην ανάγνωση...), θα χρησιμοποιήσουμε τις εξής στενογραφικές συμβάσεις για τον ορισμό των βοηθητικών προγραμμάτων που θα χρειαστούμε:

(Κ, σ) → (Λ, ρ, +)	Μια εντολή, ως έχει και γράφεται.
(σ): **(..., σ) → (..., ..., ...)**	Εννοεί όλες τις εντολές που προκύπτουν για $\sigma \in \Sigma$.
(ρ): (σ ≠ α): **(K_ρ, σ) → (..., ..., ...)**	Εννοεί όλες τις εντολές που προκύπτουν για όλα τα $\sigma \in \Sigma$ διάφορα του α, και για όλα τα $\rho \in \Sigma$.

Βήμα* 2: *τρείς τρόποι σύνθεσης προγραμμάτων Turing.
Έστω ότι έχουμε ορίσει τα προγράμματα *Turing* π_0, π_1, π_2, ... και ρ_1, ρ_2, ... Τότε οι εξής τρείς τρόποι σχηματισμού νέων προγραμμάτων από τα ήδη ορισμένα, (υπό μορφή «macro» εντολών), θα φανούν εξαιρετικά χρήσιμοι:

μακροεντολές	
$\pi_1 \mid \pi_2$	Εκτελείται το π_1 και, εάν τερματίσει, ▪ *απορριπτικά*, τότε η αλληλουχία $\pi_1 \mid \pi_2$ τερματίζει, επίσης απορριπτικά. ▪ *αποδεκτικά*, εκτελείται στη συνέχεια το π_2, και το όλο πρόγραμμα τερματίζει κατά *τον τρόπο του* π_2.
Repeat(π_0)	Επαναλαμβάνεται η εκτέλεση του π_0 έως ότου αυτό τερματίσει απορριπτικά, οπότε το πρόγραμμα επανάληψης τερματίσει *αποδεκτικά*. (Μια επανάληψη δηλαδή δεν τερματίζει ποτέ απορριπτικά.)
Case{(ρ_1: π_1) (ρ_2: π_2) ... (ρ_ν: π_ν)}	Εκτελούνται τα προγράμματα ρ_1, ρ_2, ... διαδοχικά, έως ότου υπάρξει ένα εξ αυτών το οποίο τερματίζει αποδεκτικά. ▪ Αν αυτό συμβεί για το ρ_κ, τότε εκτελείται το αντίστοιχο π_κ, και το όλο πρόγραμμα τερματίζει κατά τον τρόπο του π_κ. ▪ Αν αυτό δεν συμβεί, το όλο πρόγραμμα τερματίζει *αποδεκτικά*.

Ο ορισμός αυτών των προγραμμάτων είναι πολύ πιο απλός από ότι φαίνεται εκ πρώτης όψεως: αρκεί να «συνδέσουμε» κατάλληλα τις αφετηριακές και τερματικές καταστάσεις των επί μέρους προγραμμάτων. Δίνουμε την εξήγηση για αλληλουχία $\pi_1 \mid \pi_2$, και τα ανάλογα ισχύουν για τις άλλες δύο περιπτώσεις.

Ορισμός μακροεντολής $\pi_1 \mid \pi_2$:
- Λαμβάνουμε από ένα αντίγραφο των π_1 και π_2.

- Μετονομάζουμε τις καταστάσεις των π_1 και π_2, ώστε να είναι αδύνατον οποιαδήποτε οδηγία του ενός να αναφέρεται σε οδηγία του άλλου. Ένας απλός τρόπος είναι ο εξής: αντί για το σύνολο $S(\pi_1)$, $S(\pi_2)$ των καταστάσεων που χρησιμοποιούν τα π_1 και π_2, χρησιμοποιήσουμε τα $S(\pi_1) \times \{1\}$, $S(\pi_2) \times \{2\}$, (κατά την προφανή «μετονομασία» $X \leftrightarrow (X, 1)$ ή $\Upsilon \leftrightarrow (\Upsilon, 2)$).
- Προσθέτουμε τρείς (νέες) καταστάσεις $I(\pi)$, $\Upsilon(\pi)$, $N(\pi)$, ως αφετηριακή και τερματικές καταστάσεις για την υπό κατασκευή αλληλουχία $\pi \leftarrow \pi_1 \mid \pi_2$.
- Συνδέουμε την αφετηριακή κατάσταση $I(\pi)$ με την αφετηριακή κατάσταση $I(\pi_1)$ του π_1· συνδέουμε την αποδεκτική κατάσταση $\Upsilon(\pi_1)$ με την αφετηριακή $I(\pi_2)$, και την απορριπτική $N(\pi_1)$ με την απορριπτική του $N(\pi)$. Εξασφαλίζουμε έτσι ότι το π θα αρχίζει από το π_1 και θα συνεχίσει προς το π_2 αν και μόνο το π_1 τερματίσει αποδεκτικά. Συνδέουμε την αποδεκτική $\Upsilon(\pi_2)$ με την $\Upsilon(\pi)$, και την απορριπτική $N(\pi_2)$ με την $N(\pi)$.

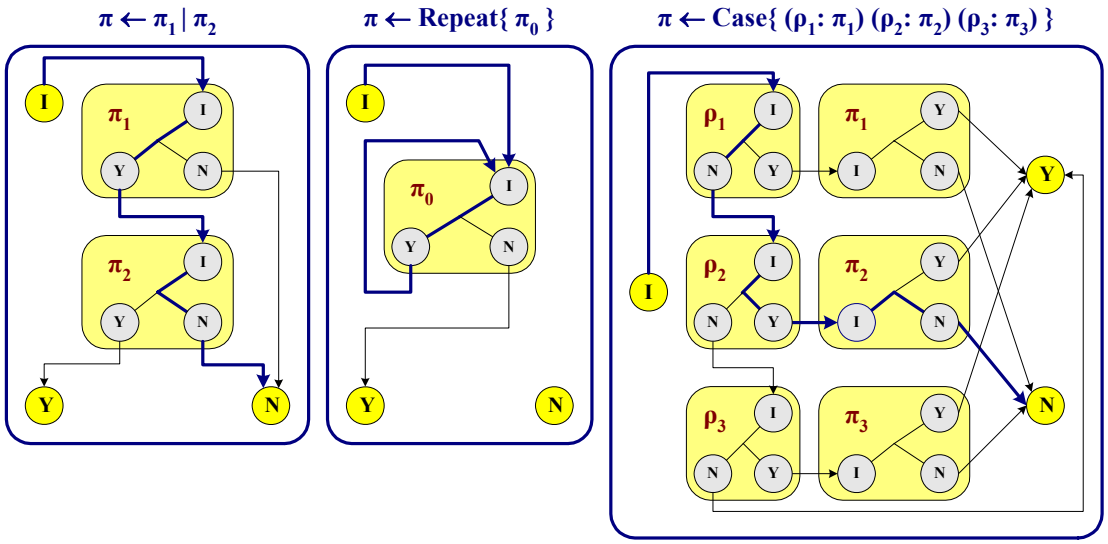
Η «σύνδεση» μιας κατάστασης X ενός 1ου προγράμματος με μια κατάσταση Υ ενός 2ου προγράμματος δεν απαιτεί παρά τις εντολές (στενογραφικά),

$$(\sigma)\text{: } (X, \sigma) \rightarrow (\Upsilon, \sigma, =)$$

οι οποίες εξασφαλίζουν ότι μετά την κατάσταση X, *ανεξάρτητα από το περιεχόμενο της ταινίας* και *χωρίς καμμία άλλη μεταβολή*, θα μεταβούμε στην κατάσταση Υ. Επειδή έχουμε φροντίσει να μετονομάζουμε τις καταστάσεις των επί μέρους προγραμμάτων με διαφορετικά ονόματα κάθε φορά, το σύνολο των παραπάνω εντολών επιτρέπει το 1ο πρόγραμμα να παραδίδει πλήρως τον «έλεγχο της ροής» των εκτελούμενων εντολών στο 2ο.

Ορισμός μακροεντολών Repeat(π_0) και Case{(ρ_1: π_1) (ρ_2: π_2) ... (ρ_v: π_v)}
Κατ' αναλογία με το προηγούμενο: οι σχετικές συνδέσεις για την ***αλληλουχία***, την ***επανάληψη***, και την ***διακλάδωση*** (για $\kappa = 3$), απεικονίζονται στο επόμενο σχήμα.

- Στην 1η περίπτωση (αλληλουχία), τα «μπλέ» βέλη απεικονίζουν την ροή των καταστάσεων εάν το 1ο πρόγραμμα αποδεχθεί και το 2ο απορρίψει.
- Στην 2η περίπτωση (επανάληψη), τα «μπλέ» βέλη απεικονίζουν την ροή των καταστάσεων εάν η επανάληψη καταστεί ατέρμων.
- Στην 3η περίπτωση (διακλάδωση), τα «μπλέ» βέλη απεικονίζουν την ροή των καταστάσεων εάν το ρ_1 απορρίψει, το ρ_2 αποδεχθεί, και το αντιστοίχως επιλεγόμενο π_2, απορρίψει.



Σχήμα: Τρείς θεμελιακές «macro» εντολές για προγράμματα *Turing*.
«κίτρινο χρώμα»: οι αφετηριακές/τερματικές καταστάσεις της σύνθεσης π.
«γκρί χρώμα»: οι αφετηριακές/τερματικές καταστάσεις των π_0, π_1, ..., ρ_1, ρ_1, ...

Είμαστε τώρα σε θέση να διατυπώσουμε το θεώρημά μας και να δώσουμε ένα σχέδιο απόδειξης:

Θεώρημα: οι μηχανές TURING (μT) υπολογίζουν ότι και οι μηχανές μνήμης (μM).
Ισχυρισμός: Έστω Π ένα οποιοδήποτε πρόγραμμα μηχανών μνήμης, το οποίο υπολογίζει την συνάρτηση $f_{\Pi}(x_1, x_2, ..., x_v) : \mathbf{N}^v \rightarrow \mathbf{N} \cup \{\uparrow\}$. Υπάρχει ένα πρόγραμμα μηχανών *Turing*, T_{Π} το οποίο υπολογίζει την «ίδια» συνάρτηση κατά την εξής έννοια: εάν το T_{Π} παραλάβει επί της ταινίας μια *περιγραφή* των θέσεων μνήμης επί των οποίων θα επιδρούσε το Π, θα παραδώσει την αντίστοιχη *περιγραφή* της μνήμης όπως αυτή θα προέκυπτε μετά την εκτέλεση του Π.

Σχέδιο απόδειξης:
Θα χρησιμοποιήσουμε το εξής αλφάβητο Σ, τόσο για να περιγράψουμε ένα πρόγραμμα Π για μηχανές μνήμης, όσο και για να το εξομοιώσουμε.

$$\Sigma = \{ \text{«}, \text{»}, \mathbf{[}, \mathbf{]}, \mathbf{<}, \mathbf{>}, \bullet, =_{\alpha}, =_{\beta}, =_{\varepsilon}, =_{\mu}, =_{\delta}, \sim, \mathbf{Z}, \mathbf{S}, \mathbf{J}, \mathbf{P}, \mathbf{C}, \mathbf{M} \}$$

Ο σκοπός κάθε συμβόλου του Σ θα είναι – εν συντομία – ο εξής:

Σύμβολα	Χρήση
« »	Ένα ζεύγος σημείων στίξεως που οριοθετούν την ταινία της μηχανής.
[ ] , < >	Δύο ζεύγη σημείων στίξεως που οριοθετούν επί μέρους περιοχές της ταινίας.
•	Η «τελεία»: το σύμβολο της μονάδας για την παράσταση αριθμών.
$=_{\alpha}, =_{\beta}, =_{\delta}, =_{\varepsilon}, =_{\mu}$	Πέντε «κινούμενα» σύμβολα/σημεία, για την σημείωση περιοχών.
~	Ένα «πρόχειρο» βοηθητικό σύμβολο.
Z, S, J	Τρία σύμβολα για την υποδήλωση των εντολών των προγραμμάτων μM.
P, C, M	Τρία «σταθερά» σύμβολα για την σημείωση των περιοχών επί της ταινίας, όπου περιγράφεται το πρόγραμμα, ο μετρητής και η μνήμη, αντιστοίχως.

Με το αλφάβητο αυτό θα περιγράφουμε τόσο ένα πρόγραμμα μM, όσο και την σειρά των θέσεων μνήμης στις οποίες αυτό αναφέρεται. Η περιγραφή έχει ως εξής:

- Περιγράφουμε ένα αριθμό ως μια ***περιοχή***, δηλαδή ένα ζεύγος αγκυλών [ ] που περιέχει τοσες «τελείες» • όσες είναι ο περιγραφόμενος αριθμός. Λ.χ. ο 3 περιγράφεται ως `[•••]`.
- Περιγράφουμε μια εντολή Z_k, S_k από ένα ζεύγος γωνιών ⟨ ⟩ που περιέχουν το σχετικό γράμμα και μια περιοχή που περιγράφει τον αριθμό k. Λ.χ. η S_5 περιγράφεται ως `⟨S[•••••]⟩`.
- Περιγράφουμε μια εντολή $J_{\kappa, \lambda, \rho}$ από ένα ζεύγος γωνιών ⟨ ⟩ που περιέχουν το σχετικό γράμμα J και τρείς περιοχές στη σειρά που περιγράφουν τους αριθμούς κ, λ και ρ. Λ.χ. η $J_{2, 3, 0}$ περιγράφεται ως `⟨J[••][•••][]⟩`.
- Το όλο πρόγραμμα περιγράφεται από την αλληλουχία των περιγραφών των εντολών του. Η σειρά αυτή εγκλείεται σε αγκύλες `[ ]` και σημειώνεται από το διακριτικό σύμβολο P.
- Ο μετρητής προγράμματος περιγράφεται ως μια περιοχή-αριθμός εντός αγκυλών, και σημειωμένη από το διακριτικό C. Λ.χ. αν ο μετρητής έχει τιμή 5, η περιγραφή του είναι `[C[•••••]]`.
- Κάθε θέση μνήμης περιγράφεται από μια περιοχή-αριθμό, και η όλη μνήμη περιγράφεται από την αλληλουχία των περιγραφών αυτών, εντός αγκυλών `[ ]` σημειωμένη από το διακριτικό M.
- Οι περιγραφές του προγράμματος, του μετρητή και της μνήμης παρατίθενται, και περιβάλλονται από τα εισαγωγικά « ».

περιγραφή εντολής J — μια περιοχή

«[P < Z[••]> < J[••][•••][•]> < S[•••]>< J[•][•][]>][C[•]][M[•][••][][•••••••••][][]»

περιγραφή προγράμματος — pc — περιγραφή θέσεων μνήμης

Σχήμα: Η περιγραφή *πρόγραμμα-μετρητής-μνήμη* για $P = \langle Z_2\, J_{2,3,1}\, S_2\, J_{1,1,0} \rangle$, $C = 1$, $M = \langle 1, 2, 0, 9, 0, 0 \rangle$.

Η τεχνική που θα ακολουθήσουμε για την εξομοίωση είναι η τήρηση ορισμένων ***σημείων***, τα οποία οποίοι θα σημειώνουν κρίσιμες περιοχές επί της ταινίας:

Σημεία	Χρήση
$=_{\varepsilon}$	σημειώνει μια *εντολή* του προγράμματος Μ/Μ.
$=_{\delta}$	σημειώνει μια περιγραφή *διεύθυνσης* (ή μνήμης, ή προγράμματος).
$=_{\mu}$	σημειώνει μια θέση μνήμης για το πρόγραμμα Μ.
$=_{\alpha}$	σημειώνει μια περιοχές μνήμης προς σύγκριση με άλλη (την $=_{\beta}$).
$=_{\beta}$	σημειώνει μια περιοχές μνήμης προς σύγκριση με άλλη (την $=_{\alpha}$).

Για να χειριστούμε την περιγραφή επί της ταινίας όπως «ζητά» το περιγραφόμενο πρόγραμμα μΜ, τοποθετούμε κάποια από τα σημεία εντός των εκάστοτε κρίσιμων περιοχών [···], αμέσως μετά την αριστερή αγκύλη. Λ.χ. η περιχοχή $[=_{\delta}\bullet\bullet\bullet]$ θα θεωρείται ως «σημειωμένη» από το σημείο $=_{\delta}$. Τα προγράμματα *Turing* είναι σε θέση να μετακινούν την κεφαλή της μηχανής και να διάβάζουν τα σύμβολα της ταινίας, άρα είναι σε θέση να εντοπίζουν τις θέσεις αυτών των σημείων ώστε να επανέρχονται στις υπό χειρισμό περιοχές, και με αυτό τον τρόπο να πραγματοποιούν τις απαραίτητες ενέργειες επί αυτών των περιοχών.

Π.χ. για να συγκρίνει το εξομοιωτικό πρόγραμμα *Turing* δύο θέσεις μνήμης (όπως ζητά μια εντολή «J»), οι οποίες περιγράφονται, όπως εξηγήσαμε πριν, από δύο περιοχές – π.χ. $[=_{\alpha}\bullet\bullet\bullet\bullet]$ και $[=_{\beta}\bullet\bullet\bullet]$ – σημειωμένες με δύο σημειοσύμβολα $=_{\alpha}$ και $=_{\beta}$, το πρόγραμμα μετακινεί την κεφαλή της μηχανής εναλλάξ από το ένα σημείο στο άλλο, προωθεί και τα δύο κατά μια τελεία • κάθε φορά, – π.χ. $[\bullet=_{\alpha}\bullet\bullet\bullet]$ και $[\bullet=_{\beta}\bullet\bullet]$ – και επαναλαμβάνει το ίδιο έως ότου η κεφαλή φθάσει στην δεξιά αγκύλη της μιας ή της άλλης περιοχής, π.χ $[\bullet\bullet\bullet=_{\alpha}\bullet]$ και $[\bullet\bullet\bullet=_{\beta}]$. Αν και στις δύο περιοχές τα σημεία $=_{\alpha}$ και $=_{\beta}$ έχουν φθάσει προ των αντίστοιχων δεξιών αγκυλών (π.χ. $[\bullet\bullet=_{\alpha}]$ και $[\bullet\bullet=_{\beta}]$), τότε και μόνον τότε οι δύο περιοχές περιέχουν το ίδιο πλήθος από τελείες •, (και άρα οι περιγραφόμενες θέσεις μνήμης φέρουν ίσους αριθμούς).

Χρειάζονται, βέβαια, αρκετά «προγραμματάκια» για να υλοποιηθεί όλη η εξομοίωση, όχι όμως και τόσο πολλά. Τα περισσότερα μάλιστα είναι ιδιαίτερα απλής μορφής. Στις επόμενες τρείς σελίδες καταλογογράφονται,

- Θεμελιακά βοηθητικά προγράμματα μΤ – (με *στενογραφικό τρόπο*).
- Προγράμματα μηχανών *Turing* για την εξομοίωση μηχανών μνήμης – (ως *μακροεντολές*).
- Σύντομα σχόλια για τον τρόπο δράσης των προγραμμάτων που δίδονται.

Το πρόγραμμα *Simulate* (βλ. πίνακα) εάν παραλάβει επί της ταινίας, (με τον τρόπο που έχουμε ήδη εξηγήσει στη αρχή) τα εξής,

- την *περιγραφή* του προγράμματος Π,
- την *περιγραφή* του μετρητή του προγράμματος,
- την *περιγραφή* της μνήμης με τα αρχικά δεδομένα,

τότε θα εξομοιώσει το Π, και θα αφήσει επί της ταινίας την *περιγραφή* της μνήμης όπως θα προέκυπτε από την εκτέλεσή του.

Το πρόγραμμα Simulate και τα σχετικά βοηθητικά προγράμματα δίδονται «top-down» στους εξής πίνακες:

Πρόγραμμα	*Οδηγίες και ερμηνεία*
μεταγραφές	
Insert(α)	*Εισάγεται το σύμβολο α στη θέση της κεφαλής:* (σ): (I, σ) → ($K_σ$, ~, +) (ϱ): (σ ≠ »): ($K_ϱ$, σ) → ($K_σ$, ϱ, +) (ϱ): ($K_ϱ$, ») → ($Λ_»$, ϱ, +) (σ): ($Λ_»$, σ) → ($Λ_α$, », –) (σ ≠ ~): ($Λ_α$, σ) → ($Λ_α$, σ, –) ($Λ_α$, ~) → (Υ, α, =)
Delete	*Διαγράφεται το σύμβολο που βλέπει η κεφαλή:* (σ): (I, σ) → ($K_»$, ~, =) (σ ≠ »): ($K_»$, σ) → ($K_»$, σ, +) ($K_»$, ») → ($Λ_»$, », –) (ϱ): (σ ≠ ~): ($Λ_ϱ$, σ) → ($Λ_σ$, ϱ, –) (ϱ): ($Λ_ϱ$, ~) → (Υ, ϱ, =)
Write(α)	*Γράφει το σύμβολο α στη θέση της κεφαλής:* (σ): (I, σ) → (Υ, α, =)
κινήσεις	
Step	*Η κεφαλή κινείται μία θέση δεξιά – μέχρι το σύμβολο τέλους »:* (σ ≠ »): (I, σ) → (Υ, σ, +) (I, ») → (N, », =)
BackStep	*Η κεφαλή κινείται μία θέση αριστερά – μέχρι το σύμβολο αρχής «:* (σ ≠ «): (I, σ) → (Υ, σ, –) (I, «) → (N, «, =)
διαγνωστικά:	
See(a)	*Απαντά Υ/Ν αν κεφαλή βλέπει ή όχι το σύμβολο α:* (σ ≠ α): (I, σ) → (N, σ, =) (I, α) → (Υ, α, =)
NotSee(a)	*Απαντά Υ/Ν αν κεφαλή βλέπει ή όχι ένα σύμβολο διαφορετικό από το α:* (σ ≠ α): (I, σ) → (Υ, σ, =) (I, α) → (N, α, =)

M/Εντολές		***Εξηγήσεις – Κώδικας***
Find(x)		Μετακινεί την κεφαλή από όπου και εάν ευρίσκεται στην πρώτη εξ αριστερών εμφάνιση επί της ταινίας του συμβόλου x. Repeat{ NotSee(«)|BackStep} | Next(x)
Next(x)		Μετακινεί την κεφαλή προς τα δεξιά έως την πρώτη εμφάνιση επί της ταινίας του συμβόλου x. Repeat{ NotSee(x)|Step }
Pass(x, y)		Η μακροεντολή βρίσκει το σύμβολο/σημάδι x και μετακινεί αυτό το σημάδι αμέσως μετά την 1η προς τα δεξιά εμφάνιση του συμβόλου y. Find(x)|Delete|Next(y)|Step|Insert(x)
UnMark(x)		Βρίσκει το σύμβολο/σημάδι x και το αφαιρεί από την ταινία: Find(x)|Delete
Swap(x)		Το πρόγραμμα πηγαίνει στο σημάδι x και εάν δεξιά από αυτό υπάρχει το σύμβολο •, εναλλάσει τα δύο σύμβολα – ουσιαστικά προωθώντας το σημάδι x κατά μία θέση δεξιά. Προσέχουμε ότι αν δεξιά από το σημάδι x δεν υπάρχει το σύμβολο •, τότε σταματά αμέσως απορριπτικά. Find(x)|Step|See(•)|Write(x)|BackStep|Write(•)
SetZero	$=_\mu$	Διαγράφει από την σημειωμένη με $=_\mu$ περιοχή μνήμης όλα τα σύμβολα • – ουσιαστικά μηδενίζοντας αυτήν. Repeat{ See(•)|Delete }
Equal	$=_\alpha$ $=_\beta$	Συγκρίνει τις περιοχές μνήμης που είναι σημειωμένες με τα $=_\alpha$ και $=_\beta$, και τερματίζει αποδεκτικά εάν και μόνον περιέχουν τον ίδιο αριθμό από σύμβολα •. **«...[M[]…[•••]…[=β•••] … [=α••]… [•]]»** Repeat{ Swap($=_\alpha$)|Swap($=_\beta$) } |Find($=_\alpha$)|See(])|Find($=_\beta$)|See(])
Copy	$=_\delta$ $=_\mu$	Αντιγράφει τη περιοχή που είναι σημειωμένη με το $=_\delta$ στην περιοχή που είναι σημειωμένη με το $=_\mu$. **«…<J…[=δ•••]>…[=μ••]…» → «…<J…[=δ•••]>…[=μ]…» → «…<J…[•••=δ]>…[=μ•••]…»** Find($=_\mu$)|Step|SetZero |Repeat{ Swap($=_\delta$)|Find($=_\mu$)|Insert(•) }
NotZeroPC	**C**	Μεταβαίνει στο (πάγιο) σημείο C της περιοχής που είναι ο μετρητής προγράμματος, **...[C[••]]...**, και διαγιγνώσκει εάν δύο θέσεις δεξιά υπάρχει το σύμβολο •. Find(C)|Step|Step|See(•)
IncreasePC	**C**	Μεταβαίνει στο (πάγιο) σημείο C της περιοχής που είναι ο μετρητής προγράμματος, **...[C[••]]...**, και τον «αυξάνει» κατά +1. Find(C)|Step|Step|Insert(•)

M/Εντολές			***Εξηγήσεις - Κώδικας***
Simulate	**C**	Repeat{ NotZeroPC |MarkCommand |Execute |UnMark($=_\varepsilon$) }	Χρησιμοποιεί το σημάδι C για να κρίνει εάν υπάρχει εντολή προς εκτέλεση. Αν όχι, τερματίζει· αν ναί, σημειώνει με το σημείο $=_\varepsilon$ την εντολή που αριθμεί ο μετρητής προγράμματος, την εκτελεί, σβήνει το σημείο $=_\varepsilon$, και επαναλαμβάνει το ίδιο έως ότου ο μετρητής προγράμματος να δείξει 0.
MarkCommand	**P** **C**	Find(P)|Step|Insert($=_\varepsilon$) |Find(C)|Step|Step|Insert($=_\delta$) |Repeat{ Swap($=_\delta$)|Pass($=_\varepsilon$,<) } |UnMark($=_\delta$)	Βάζει το σημάδι $=_\varepsilon$ εκεί όπου αρχίζουν οι εντολές (σημάδι P), και το σημάδι $=_\delta$ στη περιοχή του μετρητή προγράμματος, και όσο μετά το $=_\delta$ υπάρχει το σημάδι • μετακινεί το σημάδι $=_\varepsilon$ στην επόμενη εντολή: **«[P=$_\varepsilon$< … > < … > … < … >].....[C[=$_\delta$••]]...»**
Execute	$=_\varepsilon$	Find($=_\varepsilon$)|Step|IncreasePC |Case{ (See(Z): Perform_Z) (See(S): Perform_S) (See(J): Perform_J)}	Μεταβαίνοντας στην σημειωμένη (με το $=_\varepsilon$) προς εκτέλεση εντολή, κρίνει τον τύπο της και την εκτελεί αναλόγως. Αυξάνει «προληπτικά» τον μετρητή προγράμματος. **«[P< … > <=$_\varepsilon$Z[•••]> < … > <J[•][•••][••]> …]...»**
Perform_Z		MarkMemory |Find($=_\mu$)|SetZero |UnMark($=_\mu$)	Εντοπίζει την μνήμη όπου αναφέρεται η εντολή «Z» και φροντίζει να μηδενιστεί.
Perform_S		MarkMemory |Find($=_\mu$)|Insert(•) |UnMark($=_\mu$)	Ενεργεί κατ' αναλογία με την «Z» εντολή, μόνο που εδώ αυξάνει το περιεχόμενο της μνήμης κατά +1, εισάγοντας ένα σύμβολο •.
Perform_ J		MarkMemory_A|MarkMemory_B |Case{(Equal: AdjustPC)} |UnMark($=_\alpha$)|UnMark($=_\beta$)	Η εντολή «J» περιέχει δύο «διευθύνσεις» μνήμης. Σημειώνουμε την $1^η$ και εντοπίζουμε την σχετική περιοχή μνήμης που σημειώνουμε με το σημάδι $=_\alpha$. Σημειώνουμε την $2^η$ και την αντίστοιχη μνήμη με το σημάδι $=_\beta$. Συγκρίνουμε τις μνήμες και τροποποιούμε τον μετρητή προγράμματος. Αφαιρούμε τα σημεία $=_\alpha$ και $=_\beta$.
MarkMemory	**M** $=_\varepsilon$	Find(M)|Step|Insert($=_\mu$) |Find($=_\varepsilon$)|Next([)|Step|Insert($=_\delta$) |Repeat{Swap($=_\delta$)|Pass($=_\mu$,[)} |UnMark($=_\delta$)	Σημειώνει την «διεύθυνση» όπου αναφέρεται η εντολή «Z» με το =δ, και (όπως στα προηγούμενα) σημειώνει με =μ την περιοχή μνήμης με την αντίστοιχη διεύθυνση (λ.χ. την $3^η$ για 3 '•'). **«[P< … > <=$_\varepsilon$Z[=$_\delta$•••]> …] ... [M=$_\mu$[…][…][••••••]...]»**
MarkMemory_A	**M** $=_\varepsilon$	Find(M)|Step|Insert($=_\alpha$) |Find($=_\varepsilon$)|Next([) |Step|Insert($=_\delta$) |Repeat{ Swap($=_\delta$)|Pass($=_\alpha$,[) } |UnMark($=_\delta$)	Σημειώνει με $=_\alpha$ την περιοχή μνήμης που αντιστοιχεί στη «διεύθυνση» που είναι σημειωμένη με $=_\delta$ (όπως και στα προηγούμενα). **«[P< … ><=$_\varepsilon$J[=$_\delta$•][•••][••]> …]...[M=$_\alpha$[…]...[…]]»**
MarkMemory_B	**M** $=_\varepsilon$	Find(M)|Step|Insert($=_\beta$) |Find($=_\varepsilon$)|Next([)|Next([) |Step|Insert($=_\delta$) |Repeat{ Swap($=_\delta$)|Pass($=_\beta$,[) } |UnMark($=_\delta$)	Σημειώνει με $=_\beta$ την περιοχή μνήμης που αντιστοιχεί στη «διεύθυνση» που είναι σημειωμένη με $=_\delta$ (όπως και στα προηγούμενα).
AdjustPC	$=_\varepsilon$	Find(C)|Step|Step|Insert($=_\mu$) |Find($=_\varepsilon$)|Next([)|Next([)|Next([) |Step|Insert($=_\delta$) |Copy |UnMark($=_\delta$)|UnMark($=_\mu$)	Σημειώνει την περιοχή της εντολής «J» που περιέχει την νέα τιμή για τον μετρητή προγράμματος, (την $3^η$), με το σημάδι $=_\delta$ και τη περιοχή του ίδιου του μετρητή C με το $=_\mu$, αντιγράφει την $1^η$ στη $2^η$, και αφαιρεί τα σημάδια.

Η ορθότητα της όλης διαδικασίας στηρίζεται –συνοπτικά– στα εξής:

- Στην υπόθεση ότι μας έχει δοθεί ένα *ορθώς συντεταγμένο* πρόγραμμα μΜ.
- Στην ορθότητα των βοηθητικών προγραμμάτων.
- Στο ότι οι μακροεντολές επιδρούν όπως έχουμε ισχυριστεί.
- Στη προσεκτική χρήση των σημείων «=ₓ» : κάθε τμήμα του προγράμματος που δίδουμε (επανα)-τοποθετεί τα σημεία εκεί που «πρέπει», και μετά την άμεση χρήση τους, τα διαγράφει.
- Στην ορθή λογική δομή της εξομοίωσης – ένας ισχυρισμός ελέγξιμος βήμα-προς-βήμα.

Έστω ότι η περιγραφή του προγράμματος Π (όπως την ορίσαμε στην αρχή) είναι η λέξη $\sigma_1 \sigma_2 \ldots \sigma_n$. Προκειμένου να λάβουμε *εκείνο το ένα* πρόγραμμα που εξομοιώνει το Π, συνδέουμε δύο προγράμματα:

- το 1ο, έστω ***Load*-Π**, γράφει στην αρχή της ταινίας τα εξής:
- Την λέξη $\sigma_1 \sigma_2 \ldots \sigma_n$ ως περιγραφή του προγράμματος Π.
- Την λέξη `[C[•]]` ως περιγραφή του μετρητή προγράμματος και πρωτοτιμοδότησή του σε pc = 1.

Το πρόγραμμα είναι το εξής:

$$\texttt{Load-}\Pi \texttt{ = Find(«)|Step|Type([C[•]])|Type(}\sigma_1\sigma_2\ldots\sigma_n\texttt{)|.}$$

$$\texttt{«[M[}\cdots\texttt{] [}\cdots\texttt{] . . . [}\cdots\texttt{] [}\cdots\texttt{]]»} \xrightarrow{\texttt{LOAD-}\Pi} \texttt{«[P<}\cdots\texttt{><}\cdots\texttt{>...<}\cdots\texttt{>][C[•]][M[}\cdots\texttt{] [}\cdots\texttt{] . . . [}\cdots\texttt{] [}\cdots\texttt{]]»}$$

τα δεδομένα — περιγραφή προγράμματος — pc — τα δεδομένα

- το 2ο είναι το εξομοιωτικό πρόγραμμα ***Simulate*** που δίδεται στους πίνακες στις επόμενες σελίδες.

Η αλληλουχία Tπ = Load-Π | Simulate εάν παραλάβει επί της ταινίας την *περιγραφή* των θέσεων μνήμης επί των οποίων θα επιδρούσε το Π, θα το εξομοιώσει και θα παραδώσει την *περιγραφή* της μνήμης όπως θα προέκυπτε από την εκτέλεση του Π. (Είναι ίσως θέμα γούστου να συνδέσετε στο τέλος του Tπ άλλο ένα πρόγραμμα «UnLoad-Π», ώστε για να σβήσει από την ταινία το Π και τον μετρητή προγράμματος, και να αφήσει επ' αυτής μόνον την περιγραφή της μνήμης στην τελική μορφή της.)

■

Και τι έχουμε να «μάθουμε» από αυτό το σχέδιο απόδειξης;

Το παραπάνω σχέδιο απόδειξης (η 2η εξομοίωση αυτής της ενότητας) δίδεται γιατί αποκαλύπτει περισσότερα και σημαντικότερα πράγματα από ότι η προηγούμενη.

- Η απόδειξη είναι επίσης κατασκευαστική, αλλά αυτή τη φορά μας δίδει έναν ολόκληρο διερμηνέα (interpreter) για τα προγράμματα μηχανών μνήμης: αρκεί να περιγράψουμε (με κατάλληλο τρόπο, φυσικά) ένα πρόγραμμα Π για μηχανές μνήμης, και η μηχανή *Turing* «simulate» είναι έτοιμη να το εκτελέσει.
- Το οριζόμενο πρόγραμμα, και πάλι, όχι μόνον υπολογίζει ότι θα έκανε το διδόμενο Π, αλλά το κάνει με τον «*ίδιο τρόπο*» που θα το έκανε και εκείνο: το πρόγραμμα $T_Π$ *εξομοιώνει* το πρόγραμμα Π.
- Ας προσέξουμε ότι το πρόγραμμα $T_Π$ δεν είναι μόνον μια αφηρημένα νοητή οντότητα. Είναι κάτι το οποίο μπορεί και να *γραφεί*. Εξ άλλου αυτό έχουμε ήδη υπονοούμε, όταν γράφουμε μια οδηγία ως λ.χ. (Κ, σ) → (Λ, σ, +). Η αντιστοίχιση Π → $T_Π$, ως αντιστοίχιση *κειμένων* (ή *λέξεων*, ή *περιγραφών*, όπως θέλετε) είναι *μονοσήμαντη*:

  $$\text{αν } (Π \neq Π') \text{ τότε και } T_Π \neq T_{Π'}$$

  Και αυτό, τώρα, δεν είναι μια εκκρεμής υποψία· είναι γραμματικο-συντακτικά πλήρως *εμφανές*, διότι:

  $$T_Π = (\textbf{Load-}Π \,|\, \text{Simulate}) \text{ και } T_{Π'} = (\textbf{Load-}Π' \,|\, \text{Simulate}).$$

και αν π ≠ π′ τότε εμφανώς **Load-Π** ≠ **Load-Π′** και εύλογα $T_Π \neq T_{Π'}$. Είμαστε έτοιμοι πια να εξηγήσουμε, στην επόμενη ενότητα, την βαρύτατη και θεμελιακή σημασία αυτού του γεγονότος.

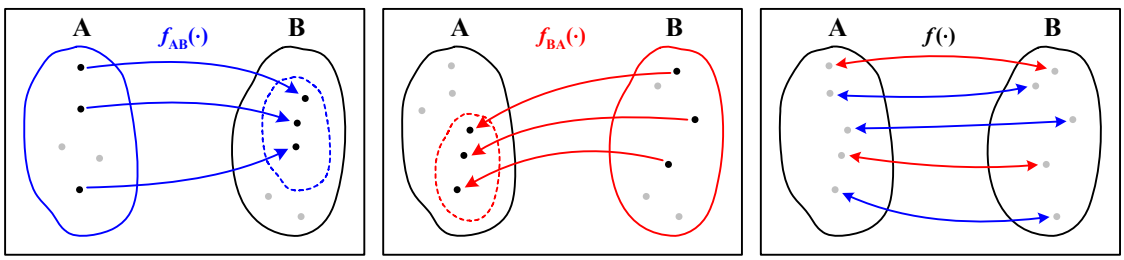
22ο Θετικά νέα: καθολικές μηχανές και η *αλγοριθμική «ισοδυναμία»*.

Οι καθολικές μηχανές και μια καθολική μηχανή Turing.
Στην προηγούμενη ενότητα είδαμε ότι μια υπάρχει μια μηχανή Turing που μπορεί να εξομοιώσει τις μηχανές με μνήμη: αν λάβει επί της ταινίας της την *περιγραφή* π μιας μηχανής M με μνήμη, και την *περιγραφή* δ, των δεδομένων που είναι να λάβει αυτή, τότε επιστρέφει την *περιγραφή* π(δ) των αποτελεσμάτων που θα έδιδε η μηχανή M. Το ίδιο μπορεί να γίνει και για πολλά άλλα είδη υπολογοσιμού/μηχανών/ «προγραμματισμού». Εδώ τίθεται το ερώτημα που θέσαμε και στην τελευταία ενότητα *περί αυτομάτων* (δείτε την πάλι): υπάρχει μια ***καθολική*** μηχανή Turing; Εάν δηλαδή *περιγράψουμε* με μια λέξη π_T μια οποιαδήποτε μηχανή Turing T, (δίνοντας μια σειρά περιγραφών των εντολών της, μία προς μία), και αν παραθέσουμε σε αυτήν μια *περιγραφή* δ της δεδομένων που θα είχε η αρχική ταινία της T, τότε υπάρχει μία καθολική μηχανή T_U η οποία με δεδομένα τα π_T και δ θα παρέδιδε ως αποτέλεσμα ότι θα επέστρεφε η T επί των δεδομένων δ; Η απάντηση είναι «ναί»: υπάρχει κάποια μηχανή Turing T_U, που είναι σε θέση να εξοιμοιώσει οποιαδήποτε άλλη μηχανή Turing. Όπως σχολιάσαμε στην σχετική ενότητα αυτό έχει μείζον τεχνολογικό αντίκτυπο: εάν θέλουμε να έχουμε μια μηχανή T που να υλοποιεί το πρόγραμμα π_T τότε δεν χρειάζεται να κατασκευάσουμε άλλη μία φυσική συσκευή· αρκεί να έχουμε στη διάθεση την «μία» καθολική μηχανή και γράψουμε απλώς την *περιγραφή* της T – δηλαδή το πρόγραμμα π_T και να το φορτώσουμε στην καθολική μηχανή. Αυτό ακριβώς γίνεται «εκεί έξω» στον κόσμο της τεχνολογίας (και του εμπορίου).

> Να σημειώσουμε μόνον εδώ ότι μια καθολική μηχανή θα ήθελε φυσικά να έχει στη διάθεσή της απεριόριστου μήκους ταινία (δηλαδή απεριόριστη ποσότητα μνήμης), αλλά και αυτό είναι φυσικώς υλοποιήσιμο «*κατ' απαίτηση*»: δεν κατασκευάζουμε μια μηχανή με άπειρη μνήμη (κάτι που είναι φυσικώς αδύνατον), απλά φροντίζουμε η μνήμη της να είναι «επεκτάσιμη» ώστε να είμαστε σε θέση να της διαθέσουμε όση μνήμη απαιτηθεί «καθ' οδόν» ενός εκάστου πραγματοποιούμενου υπολογισμού.

Το θεώρημα Cantor-Schröeder-Bernstein.
Οι διάφοροι τρόποι υπολογισμού που έχουμε στη διάθεσή μας είναι ισοδύναμοι ως προς τις συναρτήσεις που υπολογίζουν· είναι όμως ισοδύναμοι και κατά ένα πρόσθετο ιδιαίτερα αξιόλογο τρόπο. «Τόσο» ισοδύναμοι, που θα έλεγε κάποιος ότι δεν έχουν, εν τέλει, καμμία διαφορά μεταξύ τους. Αλλά για να αποκαλύψουμε αυτή την βαθειά ισοδυναμία τους θα χρειαστούμε ένα θεμελιακό θεώρημα που αφορά στους πληθαρίθμους των συνόλων:

Θεώρημα Cantor-Schröeder-Bernstein:
Ισχυρισμός: Έστω A και B δύο σύνολα, και $f_{AB}(\cdot)$ μια ένθεση του A στο B, και $f_{BA}(\cdot)$ μια ένθεση του B στο A. Τότε υπάρχει μια πλήρης αμφιμονοσήμαντη αντιστοίχιση ανάμεσα στα δύο σύνολα f: A → B, η οποία ακολουθεί τις $f_{AB}(\cdot)$ ή $f_{BA}(\cdot)$, δηλαδή εάν (x, y) ∈ $f(\cdot)$, τότε είτε (x, y) = $f_{AB}(\cdot)$ είτε (y, x) = $f_{BA}(\cdot)$.

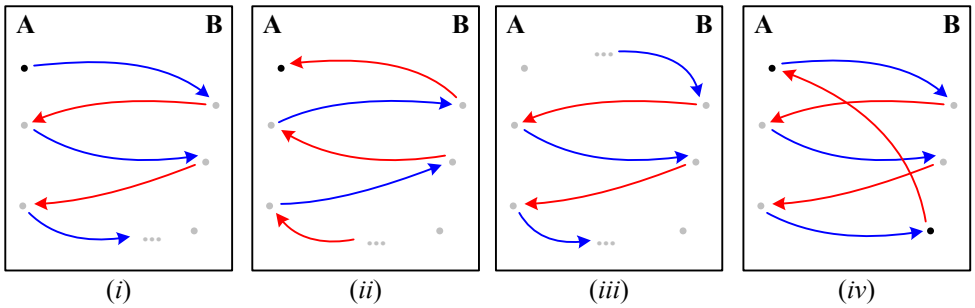


Σχήμα: Ένθεση A στο B (γαλανές ακμές), ένθεση B στο A (ερυθρές ακμές),
και πλήρης αμφιμονοσήμαντη αντιστοίχιση A ↔ B (με ερυθρογάλανες ακμές).

Σχέδιο απόδειξης: Η ιδέα της απόδειξης είναι ιδιαίτερα απλή, σχετικά με το βάθος και του βάρος αυτού του θεωρήματος. Ορίζουμε ως «αλυσίδα» του A∪B, μια ακολουθία στοιχείων ⟨ ..., x_i, y_{i+1}, x_{i+2}, ... ⟩, εναλλάξ από τα A και B , η οποία ακολουθεί τις αντιστοιχίσεις $f_{AB}(\cdot)$ και $f_{BA}(\cdot)$:

$$y_{k+1} = f_{AB}(x_k) \text{ και } x_{k+1} = f_{BA}(y_k)$$

- Οι πιθανές μορφές των αλυσίδων είναι οι εξής:



(*i*) (*ii*) (*iii*) (*iv*)

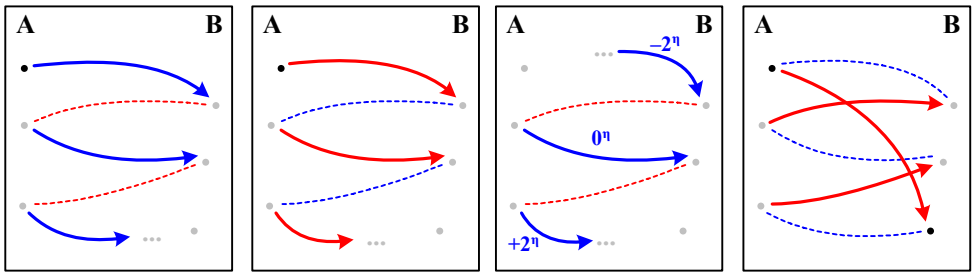
Σχήμα: Τέσσερεις μορφές αλυσίδων.
Οι (*i*), (*ii*) έχουν δύο εκδοχές, μία για το A και μία για το B) αλλά εικονίζεται μόνον η μία.

(i) Η αλυσίδα είναι άπειρη και έχει αρχικό στοιχείο, (είτε στο A, όπως εικονίζεται, είτε στο B).
(ii) Η αλυσίδα είναι άπειρη και έχει τελικό στοιχείο, (είτε στο A, όπως εικονίζεται, είτε στο B).
(iii) Η αλυσίδα είναι άπειρη και προς τις δύο «κατευθύνσεις».
(iv) Η αλυσίδα είναι πεπερασμένη (και ως εκ τούτου σχηματίζει έναν «δακτύλιο»).

- Η καίρια παρατήρηση είναι ότι *δύο διαφορετικές αλυσίδες δεν έχουν κανένα κοινό στοιχείο*. Διότι εάν λ.χ. είχαν κοινό κάποιο στοιχείο $x_k \in A$, θα είχαν κοινό και το επόμενο y_{k+1} , $y_{k+1}= f_{AB}(x_k)$ (διότι η $f_{AB}(\cdot)$ είναι συνάρτηση και το y_{k+1} είναι ένα και μοναδικό), ή/και το προηγούμενο $f_{BA}(y_{k-1}) = x_k$ (διότι η $f_{BA}(\cdot)$ είναι μονοσήμαντη και δεν στέλνει δύο διαφορετικά 'y' στο ίδιο 'x'). Το συμμετρικό θα συνέβαινε εάν είχαν κοινό στοιχείο $y \in B$. Επαγωγικά, αν δύο αλυσίδες έχουν κοινό στοιχείο, τότε έχουν όλα τα στοιχεία τους κοινά.
- Τέλος, κάθε στοιχείο προφανώς ανήκει σε μία τουλάχιστον αλυσίδα, διότι η $f_{AB}(\cdot)$ ορίζεται εφ' όλου του A, και η $f_{BA}(\cdot)$ ορίζεται εφ' όλου του B, επομένως κάποια αλυσίδα αν δεν περνά ή καταλήγει, *τουλάχιστον αρχίζει*, από κάθε $x \in A$ ή κάθε $y \in B$.

Αυτές οι τρείς ιδιότητες των αλυσίδων μας επιτρέπουν να ορίσουμε μια πλήρη και αμφιμονοσήμαντη αντιστοίχιση ανάμεσα στα A και B. Για την αντιστοίχιση $(x, f(x))$ χρησιμοποιούμε (*κατά την φορά* $A \rightarrow B$) τις εξής ακμές (x, y) των αλυσίδων, ανάλογα με τις 4 μορφές που αυτές έχουν:

(i) όλες τις άρτιες ακμές (υπ. αρ. 0, 2, 4, 6, ...) μετρώντας από την αρχική.
(ii) όλες τις άρτιες ακμές (υπ. αρ. 0, 2, 4, 6, ...) μετρώντας από την τελική.
(iii) όλες τις άρτιες, (υπ. αρ. 0, ±2, ±4, ±6, ...), μετρώντας από μία αυθαίρετη ακμή.
(iv) όλες τις άρτιες, μετρώντας από μία αυθαίρετη ακμή.



(*i*) (*ii*) (*iii*) (*iv*)

Σχήμα: οι άρτιας τάξης ακμές ($0^{η}$, $\pm2^{η}$, $\pm4^{η}$, κοκ) ορίζουν, (συλλογικά),
μια ερυθρογάλανη πλήρη αμφιμονοσήμαντη αντιστοίχιση $A \leftrightarrow B$.

Αυτή η αντιστοίχιση έχει όλα τα χαρακτηριστικά που έχουμε προδιαγράψει:

- Οι επιλεγόμενες ακμές (x, y) ανήκουν είτε στην $f_{AB}(\cdot)$, είτε στην $f_{BA}(\cdot)$:(x, y) = $f_{AB}(\cdot)$ είτε (y, x) = $f_{BA}(\cdot)$.
- Ορίζεται εφ' όλου του Α, διότι κάθε στοιχείο του Α ανήκει σε κάποια αλυσίδα, και οι επιλεγόμενες ακμές σε κάθε αλυσίδας (οι άρτιες) περιέχουν *όλα* τα στοιχεία της αλυσίδας.
- Καλύπτει όλο το Β, για τον ίδιο ακριβώς λόγο.
- Είναι αμφιμονοσήμαντη διότι δύο διαφορετικές ακμές (x, y) και (x', y') δεν έχουν ούτε κοινή αφετηρία, ούτε κοινό προορισμό. Και το τελευταίο ισχύει διότι αυτές οι ακμές δεν ανήκουν...
  - ούτε στην ίδια αλυσίδα, (διότι είναι διαδοχικές ακμές και άρα είναι αδύνατον να είναι και οι δύο άρτιας τάξης),
  - ούτε σε διαφορετικές αλυσίδες, (διότι τότε αυτές οι αλυσίδες θα είχαν κοινό στοιχείο, και είδαμε ότι αυτό είναι αδύνατον).

■

Η πλήρης αλγοριθμική και γραμματική ισοδυναμία.

Στις προηγούμενες ενότητες, είδαμε (εν μέρει) ότι κάποιοι τρόποι υπολογισμού, (μΤ, μΜ, ΑΠ), έχουν ισοδύναμη υπολογιστική ισχύ, με την εξής έννοια: αν Α και Β είναι δύο εξ αυτών, τότε για πρόγραμμα π_A ενός εξ αυτών υπάρχει ένα πρόγραμμα π_B, το οποίο εξομοιώνει το πρώτο, και επομένως τα δύο αυτά προγράμματα υπολογίζουν την ίδια συνάρτηση (από πεπερασμένα δεδομένα σε πεπερασμένα αποτελέσματα).

Παρατηρήσαμε όμως, (στο τέλος της προηγούμενης ενότητας), ότι ο τρόπος της εξομοίωσης μας εξασφαλίζει ότι αυτή η αντιστοίχιση είναι *μονοσήμαντη*, δηλαδή πρόκειται για μια *ένθεση* $f_{AB}(\cdot)$ των προγραμμάτων του Α, έστω Π_A, στα προγράμματα Π_B του Β τρόπου προγραμματισμού.

$$f_{AB}(\cdot)\ \Pi_A \rightarrow \Pi_B.$$

Επειδή οι τρόποι προγραμματισμού που αναφέραμε έχουν πλήρη ισχύ, όχι μόνον ο Β εξομοιώνει τον Α, αλλά και αντιστρόφως. (Το ίδιο ισχύει και για όλους τους τρόπος προγραμματισμού με πλήρη ισχύ). Δηλαδή, υπάρχει και μια ***ένθεση*** $f_{BA}(\cdot)$ του Β στο Α:

$$f_{BA}(\cdot)\ \Pi_B \rightarrow \Pi_A.$$

Το χαρακτηριστικό των $f_{BA}(\cdot)$ είναι ότι εάν $\pi_\beta = f_{AB}(\pi_\alpha)$ τότε το π_β εξομοιώνει το π_α, δηλαδή ολόκληρη η σειρά των σταδίων υπολογισμού του π_α έχει αντίστοιχα στάδια εντός του υπολογισμού που πραγματοποιεί το π_β. Δηλαδή, το π_β – κατά κάποια έννοια – ακολουθεί τον «ίδιο αλγόριθμο» με εκείνον που ακολουθεί το π_α. Και συμμετρικά αν $\pi_\alpha = f_{BA}(\pi_\beta)$ τότε το π_α εξομοιώνει το π_β.

Σε αυτό το σημείο επιδρά το θεώρημα των *Cantor-Schröeder-Bernstein*: αν οι συναρτήσεις «εξομοίωσης» $f_{AB}(\cdot)$ και $f_{BA}(\cdot)$ ενθέτουν το Α στο Β και το Β στο Α, τότε υπάρχει μια πλήρης αμφιμονοσήμαντη αντιστοίχιση f: Α → Β, «φτιαγμένη» από ακμές είτε της $f_{AB}(\cdot)$ είτε της $f_{BA}(\cdot)$. Αν δηλαδή π_α και π_β είναι δύο αντίστοιχα κατά την $f(\cdot)$ προγράμματα, τότε είτε $\pi_\beta = f_{AB}(\pi_\alpha)$ είτε $\pi_\alpha = f_{BA}(\pi_\beta)$, οπότε (και στις δύο περιπτώσεις) τα αντίστοιχα προγράμματα π_α και π_β είναι «αλγοριθμικώς» ισοδύναμα κατά την ελάχιστη έννοια ότι το ένα εξ αυτών *εξομοιώνει* το άλλο!

Το συμπέρασμα είναι ότι δύο οποιοιδήποτε πλήρεις τύποι προγραμματισμού[7], είναι ισοδύναμοι σε τρία διαρκώς βαθύτερα επίπεδα:

- *υπολογίζουν τις ίδιες συναρτήσεις του* Φ: $\Sigma^* \rightarrow \Sigma^* \cup \{\uparrow\}$.
- *είναι δυνατόν να εκφράσουν τους «ίδιους αλγορίθμους»*,
- και όλα αυτά «*κατά γράμμα*»: *με 1-προς-1 αντιστοίχιση των προγραμμάτων τους*.

[7] Από τους τρείς που αναφέραμε, (μΤ, μΜ, ΑΠ) – αλλά και από όσους άλλους είναι επίσης «πλήρους (υπολογιστικής) ισχύος».

Για τα μάτια του «θεωρητικού» προγραμματιστή όλες οι (πλήρεις) τύποι υπολογισμού είναι ολόιδιοι: από την πλευρά της υπολογιστικής (ή αλγοριθμικής – αν θέλετε) «εκφραστικότητας» δεν υπάρχει κανένας λόγος προτίμησης καμμίας γλώσσας προγραμματισμού από καμμία άλλη...

23ο Τα άσχημα νέα: ανεπίλυτα προβλήματα.

Καταμέτρηση, άπειρο, και πληθικότητα.

Για να κρίνουμε, στη καθημερινή μας ζωή, από δύο σύνολα στοιχείων ποιό είναι το «μεγαλύτερο», συγκρίνουμε δύο αριθμούς: τα *πλήθη* των στοιχείων αυτών. Αυτό δουλεύει καλά όταν τα σύνολα είναι πεπερασμένα – εξ άλλου τέτοια είναι τα σύνολα που έχουμε συνηθίσει να αντιμετωπίζουμε. Όταν όμως εισέλθουμε στην επικράτεια του απείρου, αυτή η κατάσταση μεταβάλλεται δραστικά: αν ένα σύνολο είναι άπειρο τότε είναι δυνατόν να είναι *ισοπληθές* με κάποιο *γνήσιο* υποσύνολο του – και αυτό είναι μια ιδιότητα που χαρακτηρίζει το «άπειρο».

Π.χ. τα τέλεια τετράγωνα 0, 1, 4, 9, 16, αν και είναι ένα μέρος μόνον των φυσικών αριθμών έρχονται σε μία πλήρη και 1-1 αντιστοίχιση (= *αμφιμονοσήμαντη*) με τους φυσικούς αριθμούς, όπως το ακόλουθο σχήμα εικονίζει κατά προφανή τρόπο:

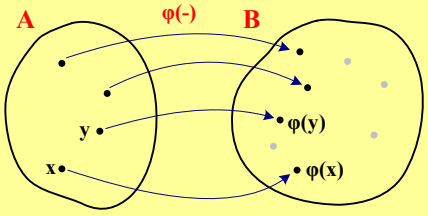
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**	**ν**	**...**
↕	↕	↕	↕	↕	↕	↕	↕		↕	
0	**1**	**4**	**9**	**16**	**25**	**36**	**49**	**...**	**ν²**	**...**

Μια απλή αλλά χαρακτηριστική περίπτωση των παραδόξων του απείρου είναι το λεγόμενο «ξενοδοχείο του *Hilbert*»: έστω ένα άπειρο ξενοδοχείο, με αριθμούς δωματίων 1, 2, 3, ... κοκ, που είναι πλήρες, και στο οποίο εμφανίζεται ένας πρόσθετος πελάτης. Μπορεί να εξασφαλίσει ένα δωμάτιο; Ναί – αρκεί να ζητήσει από κάθε ένοικο να μετακινηθεί στο επόμενο δωμάτιο, και το υπ. αρ. 1 δωμάτιο θα καταστεί διαθέσιμο για τον νέο (αργοπορημένο) πελάτη:

πελάτες:	**ok!**	**1**	**2**	**3**	**4**	**...**	**ν**	**...**
	↕	↕	↕	↕	↕		↕	
δωμάτια:	**1ο**	**2ο**	**3ο**	**4ο**	**5ο**	**...**	**ν+1**	**...**

Το άπειρο, και γεμάτο να είναι, χωρά ακόμα έναν... Γι' αυτό πρέπει να διακρίνουμε δύο παραλλαγές της έννοιας «περισσότερα»:

- Η 1η είναι συνολοθεωρητική: Τα στοιχεία του συνόλου Β είναι συνολοθεωρητικώς περισσότερα από εκείνα του Α, εάν το Α είναι γνήσιο υποσύνολο του Β: $A \subset B$. Σε αυτή την περίπτωση θα ήταν προτιμοτερο να μην λέγαμε ότι το Β έχει «περισσότερα» στοιχεία, αλλά ότι περιέχει ***πρόσθετα*** στοιχεία, (ως προς το Α).
- Η 2η είναι συναρτησιακή: Εάν υπάρχει συνάρτηση ϕ: $A \to B$, η οποία αποτελεί ***ένθεση*** του Α στο Β, δηλαδή $(x \neq y) \Rightarrow (\phi(x) \neq \phi(y))$, τότε επιτρέπεται να λέμε ότι τα στοιχεία του συνόλου Β είναι *πληθικώς τουλάχιστον* όσα του συνόλου Α, (ή συμβολικά $|A| \leq |B|$), αφού το Β περιέχει ένα «αντίτυπο» του Α.



Αυτό που παρατηρήσαμε στη αρχή αυτού του σχολίου είναι ότι υπάρχουν (άπειρα)

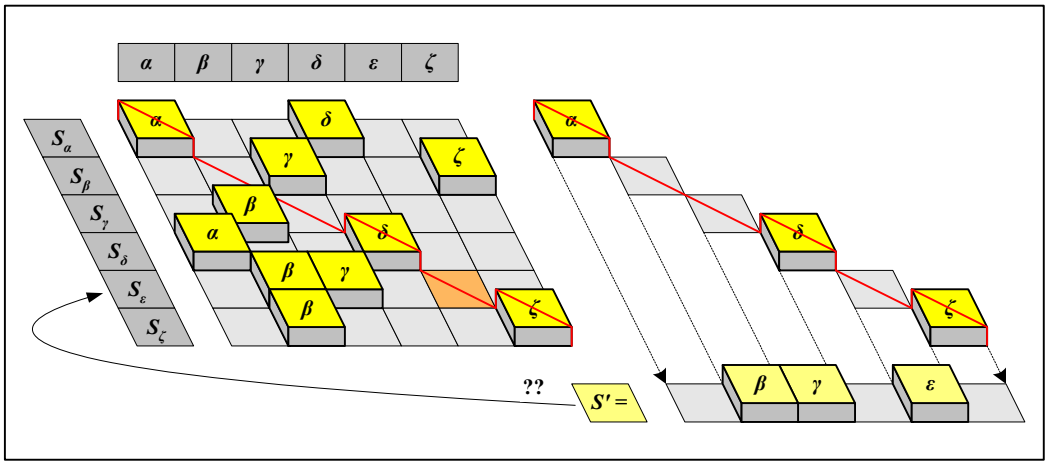
σύνολα Α και Β, που ενώ το Β έχει πρόσθετα στοιχεία, εν τούτοις τα δύο σύνολα είναι πληθικώς ισοδύναμα (ή, ***ισοπληθή***: $|A|\sim|B|$), δηλαδή υπάρχει μια αμφιμονοσήμαντη αντιστοίχιση $A\leftrightarrow B$. Ο εξής ορισμός είναι λοιπόν απαραίτητος: θα λέμε ότι το σύνολο Β έχει ***πληθικώς περισσότερα*** στοιχεία από το Α, (και θα γράφουμε $|A|<|B|$), εάν και μόνον εάν υπάρχει μεν ένθεση του Α στο Β, αλλά *καμμία* ένθεση του Α στο Β δεν καλύπτει όλο το Β.

Ας προσέξουμε εδώ ότι χάρι στο θεώρημα *Cantor-Schröeder-Bernstein* οι παραπάνω έννοιες συμπεριφέρονται όπως θα θέλαμε: $|A|\le|B|$ και $|B|\le|A|$ δίδουν $|A|\sim|B|$.

Στην ενότητα που ακολουθεί θα εξηγήσουμε ένα τέχνασμα (που οφείλεται στον *Cantor*, τέλη 19ου αιώνα), χάρι στο οποίο είναι δυνατόν να αποδείξουμε ότι υπάρχουν διαφόρων ειδών τάξεις απείρου, και συγκεκριμμένα ότι υπάρχουν σύνολα Α και Β τα οποία είναι μεν και τα δύο *άπειρα*, αλλά το Β έχει *πληθικώς περισσότερα* στοιχεία από το Α.

Διαγώνιο τέχνασμα Cantor

Προτού διαβούμε το κατώφλι του απείρου θα ήταν απλούστερο να δούμε το τέχνασμα του *Cantor* εφαρμοζόμενο σε ένα πεπερασμένο σύνολο. Έστω ένα σύνολο Α με έξι στοιχεία: $A=\{\alpha,\beta,\gamma,\delta,\varepsilon,\zeta\}$. Το σύνολο $\wp(A)$ όλων των υποσυνόλων του σίγουρα έχει «περισσότερα ή ίσα» στοιχεία από το Α: κάθε στοιχείο του $x\in A$ ανήκει ως μονοσύνολο $\{x\}$ στο $\wp(A)$, και άρα η αντιστοίχιση $x\to\{x\}$ αποτελεί μια ένθεση του Α στο Β. Αλλά το σημαντικό είναι ότι έχει *πληθικώς* γνησίως περισσότερα στοιχεία: *δεν υπάρχει καμμία ένθεση του Α στο $\wp(A)$ που να καλύπτει όλο το $\wp(A)$*. Και πράγματι έστω ότι υπάρχει μια τέτοια αντιστοίχιση, και ότι στο $x\in A$ αντιστοιχεί το υποσύνολο $S_x\subseteq A$ (ή $S_x\in\wp(A)$).



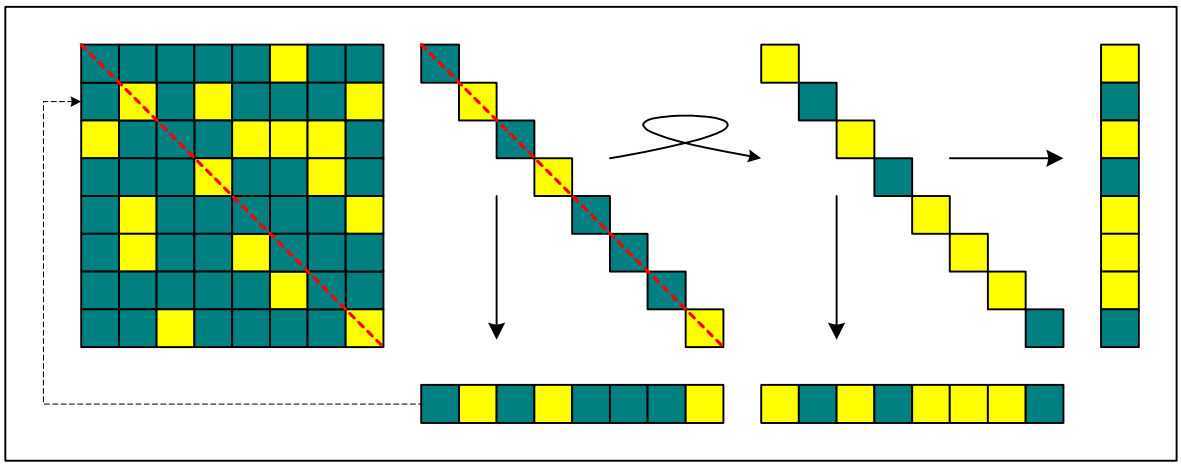
Σχήμα: Τα υποσύνολα ενός συνόλου είναι ***πληθικώς*** περισσότερα.

Αυτή η 1-προς-1 αντιστοίχιση είναι να δυνατόν να εικονιστεί ως ένα τετράγωνο, (βλ. σχήμα), οι στήλες του οποίου επιγράφονται με τα στοιχεία $x\in A$, και οι γραμμές επιγράφονται με τα υποσύνολα S_x, $x\in A$. Από το «τετράγωνο» αυτής της αντιστοίχισης ορίζεται ένα διαγώνιο υποσύνολο $D=\{x: x\in S_x\}$, που καλείται έτσι διότι αποτελείται από τα στοιχεία που εμφανίζονται πάνω στη διαγώνιο (x, S_x), $x\in A$, του τετραγώνου. Το συπληρωματικό αυτού του υποσυνόλου $D'=A-D$, διαφέρει από όλα τα S_x διότι για κάθε $x\in A$,

- αν το $x\in S_x$ τότε $x\in D$ και άρα $x\notin D'=A-D$.
- αν το $x\notin S_x$ τότε $x\notin D$ και άρα $x\in D'=A-D$.

(Στο σχήμα $D=\{\alpha,\delta,\zeta\}$, $D'=\{\beta,\gamma,\varepsilon\}$, και π.χ. $D'\ne D_\varepsilon$ ως προς το διαγώνιο στοιχείο $\varepsilon\in A$.) Κάθε ένθεση λοιπόν του Α στο $\wp(A)$ παραλείπει ένα τουλάχιστον υποσύνολο $D'\in\wp(A)$, και άρα το δυναμοσύνολο έχει πληθικώς περισσότερα στοιχεία από το Α.

Διαγωνιοποίηση Cantor: η υπαρξιακή εκδοχή.
Η αξία της προηγούμενης ανάλυσης είναι ότι ενώ αναφέρεται σε πεπερασμένα σύνολα, δεν χρησιμοποιεί πουθενά το γεγονός ότι είναι πεπερασμένα· ισχύει λοιπόν και για άπειρα σύνολα. Η γενική διατύπωση αυτού του τεχνάσματος έχει ως εξής: Έστω δύο σύνολα S, T (πεπερασμένο, ή άπειρο καθ' οιονδήποτε τρόπο), το οποίο είναι ισοπληθή $|S|\sim|T|$ βάσει μιας αμφιμονοσήμαντης αντιστοίχισης s: $T \to S$. Έστω επίσης ότι κάθε στοιχείο του $\sigma \in S$ παράγει μια ακολουθία «χρωμάτων» $\langle \chi_\sigma(\tau): \tau \in T\rangle$, όπου το κάθε $\chi(\cdot)$ είναι ένα από δύο χρώματα «χ_0» και «χ_1». Το «τετράγωνο» $S \times T$ περιέχει μια ακολουθία από «διαγώνιες» θέσεις $(s(\tau), \tau)$, ο χρωματισμός των οποίων δίδεται από μια διαγώνιο συνάρτηση: $d(\tau) = \chi_{s(\tau)}(\tau)$, $\tau \in T$, η οποία επιδέχεται το *συμπλήρωμά*[8] της: $c(\tau) = \text{OXI}(\delta(\tau))$, όπου $\text{OXI}(\chi_0) = \chi_1$, και $\text{OXI}(\chi_1) = \chi_0$.



Σχήμα: σε ένα διχρωματικό τετράγωνο το «αρνητικό» της διαγωνίου
δεν εμφανίζεται πουθενά, (ούτε ως γραμμή, ούτε ως στήλη).

Τεχνική διαγωνιοποίησης του Cantor: Σε κάθε τετράγωνο διχρωματικό πίνακα $S \times T$, η συμπληρωματική διαγώνιος ακολουθία $c(\cdot)$ δεν εμφανίζεται ως καμμία γραμμή, και επομένως είναι αδύνατον να προέρχεται από κανένα στοιχείο σ του $S \times T$.
(Διότι, όπως είδαμε, για κάθε $\sigma = s(\tau) \in S$, η γραμμή των χρωμάτων $\chi_{s(\tau)}(\cdot)$ διαφέρει από την συμπληρωματική διαγώνιο τουλάχιστον επί του διαγωνίου στοιχείου τ: $\chi_{s(\tau)}(\tau) = d(\tau) \neq c(\tau) = \text{OXI}(\chi_{s(\tau)}(\tau))$.)

Υπάρχουν πολλοί τρόποι για την αξιοποίηση αυτού του γεγονότος, αλλά όλες έχουν στον πυρήνα τους την χρονολογικά πρώτη από αυτές:

Θεώρημα Cantor: $|\mathbf{A}| < |\wp(\mathbf{A})|$.
Ισχυρισμός: Για κάθε σύνολο A, δεν υπάρχει καμμία ένθεση του A στο $\wp(A) = \{S: S \subseteq A\}$ που να καλύπτει όλο το $\wp(A)$, δηλαδή το δυναμοσύνολο $\wp(A)$ είναι *πληθικώς* μεγαλύτερο από το A.

Σχέδιο απόδειξης: Για κάθε υποσύνολο $\Lambda \subseteq A$, ορίζεται μια γραμμή χρωμάτων $\chi_\Lambda(\tau) \in \{\chi_1, \chi_0\}$ αντιστοίχως με το εάν $\tau \in X$, ή $\tau \notin X$, (και αντιστρόφως). Αν υπήρχε μια αμφιμονοσήμαντη αντιστοιχία $s: A \leftrightarrow \wp(A)$, η συμπληρωματική διαγώνιος $c(\cdot)$ θα περιέγραφε με τα χρώματά της ένα σύνολο $C \subseteq A$, διαφορετικό από όλα του δυναμοσυνόλου $\wp(A)$, πράγμα άτοπο. Επομένως η υποτιθέμενη αμφιμονοσήμαντη αντιστοιχία $s: A \leftrightarrow \wp(A)$ δεν υπάρχει, και $|A| < |\wp(A)|$.

■

Ανεπίλυτα προβλήματα: η υπαρξιακή εκδοχή.

Το προηγούμενο θεώρημα και μόνο αρκεί για να διαπιστώσουμε ότι υπάρχουν συναρτήσεις που θα θελαμε να υπολογίζουμε, αλλά κανένα πρόγραμμα μΤ δεν τις υπολογίζει. Ο λόγος είναι οι εξής δύο παρατηρήσεις:

[8] Ή το «*αρνητικό*» της, εάν θέλουμε να χρησιμοποιήσουμε έναν όρο από τον χώρο της (παλαιάς) φωτογραφίας.

- Ανάμεσα στις συναρτήσεις Φ: $\Sigma^* \to \Sigma^* \cup \{ \uparrow \}$ που θα θέλαμε να υπολογίσουμε είναι όλες οι γλώσσες $L = \wp(\Sigma^*)$.
- Τα διαθέσιμα προγράμματα επιδέχονται όλα μια πεπερασμένη περιγραφή (όποιο και εάν είναι το αλφάβητο Σ που χρησιμοποιούμε), και επομένως το πλήθος τους είναι το πολύ $|\Sigma^*|$.

Όμως μόλις είδαμε ότι το $|\Sigma^*| < |\wp(\Sigma^*)|$, δηλαδή τα διαθέσιμα προγράμματα είναι πληθικώς λιγότερα από τις προς υπολογισμό συναρτήσεις...

Δεν μπορούμε όμως να εφησυχάσουμε σε αυτό το σημείο. Διότι ακόμα και εάν υπάρχουν κάποιες ιδιότροπες γλώσσες (όσες και εάν είναι) οι οποίες δεν υπολογίζονται, το καίριο για μας είναι εάν μπορούμε να υπολογίσουμε όλες τις γλώσσες *εφόσον μπορούμε να τις ορίσουμε* – διότι αν μια γλώσσα υπάρχει μεν αλλά διαφεύγει ακόμα και του προσδι-ορισμού της, δεν είναι μία από αυτές που θα χρειαζόταν ποτέ στη πράξη να την υπολογίσουμε· λόγω της αοριστίας της δεν θα φθάναμε ποτέ σε αυτό το σημείο. Και επειδή οι μαθηματικοί ορισμοί δεν είναι – και αυτοί... – παρά πεπερασμένες λέξεις επί πεπερασμένου αλφαβήτου, το πλήθος τους είναι επίσης $|\Sigma^*|$. Αυτές, δηλαδή, είναι όσες και τα διαθέσιμα προγράμματα, και το ερώτημα της υπολογιστικής επάρκειας (λ.χ.των μΤ), επανέρχεται.

Ευτυχώς ή δυστυχώς, δεν χρειαζόμαστε πολύ ισχυρότερα εργαλεία για να το απαντήσουμε: αρκεί μια *κατασκευαστική* εκδοχή του επιχειρήματος του *Cantor*.

Διαγωνιοποίηση Cantor: η κατασκευαστική εκδοχή και το «τετράγωνο» των υπολογισμών.
Για την ανάλυση σε βάθος των δυνατοτήτων ενός τρόπου υπολογισμού/προγραμματισμού θα χρειαστούμε (εύλογα) μια υπολογιστική εκδοχή του διαγώνιου επιχειρήματος. Η πρώτη σημαντική παρατήρησή μας είναι ότι εάν χρησιμοποιούμε ένα πεπερασμένο αλφάβητο Σ, (οποιοδήποτε), τότε έχουμε στη διάθεσή μας ένα πρόγραμμα Λ, το οποίο «καταλογοποιεί» όλες τις δυνατές λέξεις, δηλαδή όλο το σύνολο Σ*. Η τεχνική είναι απλή έως απλοϊκή. Παράγουμε όλες τις λέξεις μία-προς-μία, αρχίζοντας από αυτή με μήκος 0, μετά αυτές με μήκος 1 (το Σ), στη συνέχεια αυτές με μήκος 2 (ισοδυνάμως το Σ×Σ), κοκ. Για Σ = {α, β} ο κατάλογος θα είχε την εξής μορφή:

$$\varnothing,\ \ \alpha,\ \ \beta,\ \ \alpha\alpha,\ \ \alpha\beta,\ \ \beta\alpha,\ \beta\beta,\ \ \alpha\alpha\alpha,\ \ \alpha\alpha\beta,\ \ \alpha\beta\alpha,\ \ \alpha\beta\beta,\ \ \beta\alpha\alpha,\ \ ...,\ \ \text{κοκ.}$$

Εάν εκ παραλλήλου μετρούμε το πόσες λέξεις έχουμε κατασκευάσει, είναι δυνατόν να σταματάμε όταν φθάσουμε στην υπ. αρ. k, και να την παραδώσουμε ως αποτέλεσμα. Το πρώτο μας λήμμα είναι λοιπόν το εξής:

Λήμμα: καταλογοποίηση όλων των λέξεων ενός αλφαβήτου.
Ισχυρισμός: Υπάρχει ένα πρόγραμμα Λ (λ.χ. για μηχανές *Turing*) το οποίο με δεδομένη μια περιγραφή του αριθμού k, παραδίδει ως αποτέλεσμα την δ_k = η υπ. αρ. k λέξη του Σ*.

Σχέδιο απόδειξης: Το εξής σχέδιο προγράμματος θα αρκούσε για την παραγωγή όλων των λέξεων:

Λ =
```
Διαβάζουμε την περιγραφή του αριθμού k
Χρησιμοποιούμε έναν μετρητή ν, αρχικά ν ← 0
Για μ = 0, 1, 2, 3, ...
Περίπτωση
{ μ=0: «Γράφουμε» την κενή λέξη (κατά μία οιαδήποτε σύμβαση).
  μ=1: Γράφουμε ως λέξεις ένα-προς-ένα τα σύμβολα του Σ.
  μ>1: Αντιγράφουμε |Σ| φορές τις λέξεις μήκους μ-1,
       Και προσθέτουμε σε κάθε μία από ένα σύμβολο του Σ. }
Μετρούμε τις λέξεις που σχηματίζουμε ως λν με τον μετρητή ν,
Και τερματίζουμε όταν (ν=k) παραδίδοντας τη λέξη δk ← λν.
```

■

Τέτοιοι κατασκευάσιμοι ή προγραμματικώς υπολογίσιμοι κατάλογοι είναι εξαιρετικά χρήσιμοι για την ανάλυση των υπολογιστικών δυνατοτήτων μας. Και γι' αυτό μας είναι πολύ χρήσιμο το εξής λήμμα, το οποίο εξηγεί γιατί είναι δυνατόν να είμαστε – υπό προϋποθέσεις – ιδιαίτερα παραγωγικοί ως προς αυτό το ζήτημα:

Λήμμα: καταλογοποίηση μέσω μιας διαγνώσιμης ιδιότητας.
Ισχυρισμός: Έστω ότι κάποια ιδιότητα των λέξεων $\lambda \in \Sigma^*$ είναι *διαγνώσιμη* μέσω ενός προγράμματος $\pi_I(\lambda)$, δηλαδή «$\pi_I(\lambda)$ = «1» *εάν η λ έχει την ιδιότητα I*», και «$\pi_I(\lambda)$ = «0» *εάν η λ δεν έχει την ιδιότητα I*». Τότε το σύνολο όλων των λέξεων που έχουν την ιδιότητα I επιδέχεται μια *καταλογοποίηση*, δηλαδή υπάρχει ένα πρόγραμμα Λ_I, το οποίο με δεδομένη την περιγραφή ενός αριθμού k, παραδίδει ως αποτέλεσμα την υπ. αρ. k λέξη με την ιδιότητα I, (χωρίς φυσικά να παραλείπει καμμία).

$$\{ \lambda: \lambda \text{ έχει την ιδιότητα I} \} = \langle \lambda_0, \lambda_1, \lambda_2, \ldots \rangle = \{ \Lambda_I(k): k \in \mathbf{N} \}$$

Σχέδιο απόδειξης: Το εξής σχέδιο προγράμματος θα αρκούσε – κατά (περίπου) προφανή τρόπο:

Λ_I =
```
Διαβάζουμε τον αριθμό «k»
Χρησιμοποιούμε έναν μετρητή ν, αρχικά ν ← 0
Για μ = 0, 1, 2, 3, ...
{ Φτιάχνουμε τις λέξεις λ_i του Σ* μήκους μ μία-προς-μία
  Εκτελούμε το διαγνωστικό π_I(λ)      // μέσω της καθολικής T_U!
  Περίπτωση:
  { π_I(λ_i)=0: Αγνοούμε έτην λ_i.
    π_I(λ_i)=1: Μετρούμε την λ_i με τον μετρητή ν,
                Τερματίζουμε όταν (ν=k) και παραδίδουμε την λ_ν.
} }
```

Το μόνον που δεν είναι προφανές στο παραπάνω πρόγραμμα είναι *η ύπαρξη ενός καθολικού προγράμματος* μT, το συμβολιζόμενο ως T_U (από *Turing* – universal), την ύπαρξη του οποίου επισημάναμε στη προηγούμενη ενότητα. Η ύπαρξη αυτή δεν είναι απλώς χρήσιμο εργαλείο· είναι αναντικατάστατο εργαλείο – ιδίως σε τέτοιου είδους αποδεικτικούς συλλογισμούς σαν αυτούς με τους οποίους θα ασχοληθούμε στη τρέχουσα ενότητα.

■

Λήμμα: καταλογοποίηση όλων των μηχανών Turing.
Ισχυρισμός: Υπάρχει ένα πρόγραμμα Π (εδώ: για μηχανές *Turing*) το οποίο με δεδομένη μια περιγραφή του αριθμού k, παραδίδει ως αποτέλεσμα το υπ. αρ. k πρόγραμμα (εδώ: περιγραφή μηχανής *Turing*).

Σχέδιο απόδειξης: Το εάν μια λέξη $\lambda \in \Sigma^*$ περιγράφει (σύμφωνα με κάποια σύμβαση) μια μηχανή *Turing*, είναι μια διαγνώσιμη ιδιότητα: απλώς αναλύουμε συντακτικά την λ κατά την «γραμματική» που κατά σύμβαση περιγράφει τις μT. Σύμφωνα με τα προηγούμενα λήμματα, εφόσον υπάρχει διαγνωστικό (αυτό της συντακτικής ορθότητας), όλες οι σχετικές λέξεις/περιγραφές επιδέχονται μια προγραμματική καταλογοποίηση.

■

Είμαστε τώρα σε θέση να δώσουμε μια κατασκευαστική εκδοχή του επιχειρήματος του *Cantor*.

Θεώρημα: η «προγραμματιστική» εκδοχή του διαγωνίου επιχειρήματος.
Ισχυρισμός: Έστω $\chi_0 \neq \chi_1$ δύο λέξεις του αλφαβήτου μας, (δύο «χρώματα», το ένα «συμπληρωματικό» του άλλου), και I μια οποιαδήποτε ιδιότητα των προγραμμάτων. Τότε από τους εξής τρείς ισχυρισμούς:

- υπάρχει καταλογοποίηση **Π**, των προγραμμάτων με την ιδιότητα **I** , δηλ. $\mathbf{\Pi(i) = \pi_i}$, και $\mathbf{I = \{ \pi_i: i \in N \}}$.
- υπάρχει πρόγραμμα π_χ χρωματισμού της μορφής: $\pi_\chi(\pi_\kappa, \delta_\lambda) = \phi(\pi_\kappa(\delta_\lambda))$, $\kappa \in \mathbf{N}$, για συνάρτηση $\phi(\cdot)$. (Δηλαδή το χρώμα στηθέση $(\pi_\kappa, \delta_\lambda)$ ***ορίζεται*** από το αποτέλεσμα του π_κ επί των δεδομένων δ_λ.)
- ο διαγώνιος χρωματισμός $\mathbf{\pi_\epsilon(k)}$ = συμπληρωματικό-του-$\pi_\chi(\delta_k, \delta_k)$ περιλαμβάνεται στον κατάλογο **I**.

ένας εξ αυτών είναι ψευδής· δεν είναι δυνατόν να ισχύουν και οι τρείς ταυτοχρόνως.

Σχέδιο απόδειξης: Σε αυτό ακριβώς καταλήγει το διαγώνιο επιχείρημα του *Cantor*. Ορίζουμε ένα πίνακα N×N οι στήλες του οποίου επιγράφονται από όλα τα ενδεχόμενα δεδομένα {δ_κ: κ ∈ N}, και οι γραμμές από όλα τα προγράμματα {π_κ: κ ∈ N} με την ιδιότητα I. Κάθε πρόγραμμα π_i του καταλόγου Π, ορίζει μια «γραμμή» χρωματισμού $\chi_i(j) = \pi_\chi(\pi_i, \delta_j)$, και όλες οι γραμμές ορίζουν έναν χρωματισμό ολόκληρου του πίνακα $\chi(i, j) = \chi_i(j)$. Ο συμπληρωματικός διαγώνιος χρωματισμός c(k) = ΣΥΜΠΛΗΡΩΜΑΤΙΚΟ(χ(k, k)), k∈ N, δεν εμφανίζεται ως καμμία γραμμή $\chi_i(\cdot)$, i ≥ 0, διότι διαφέρει από αυτήν τουλάχιστον ως προς το αντίστοιχο διαγώνιο στοιχείο: $c(i) \neq \chi_i(i)$. Επομένως το πρόγραμα $\pi_c(\cdot)$ που την υπολογίζει (ούτε και κανένα ισοδύναμό του) δεν περιλαμβάνεται στον κατάλογο I, διότι παράγει έναν χρωματισμό γραμμής διαφορετικό από εκείνον οποιουδήποτε προγράμματος με την ιδιότητα I.
Δεν μένουν λοιπόν παρά τρείς δυνατότητες:
(i) είτε «*το Π δεν υπάρχει*» – η καταλογοποίηση του I δεν είναι εφικτή από κανένα πρόγραμμα.
(ii) είτε «*το π_χ δεν υπάρχει*» – ο χρωματισμός χ(-) δεν υπολογίζεται από κανένα πρόγραμμα.
(iii) είτε «$\pi_c \notin I$» – η καταλογοποίηση I δεν περιλαμβάνει πρόγραμμα υπολογισμού της c(·).

■

Ανεπίλυτα προβλήματα: η <u>κατασκευαστική</u> εκδοχή.

Με την προγραμματιστική εκδοχή του επιχειρήματος *Cantor* είμαστε σε θέση να ορίσουμε συναρτήσεις που θέλαμα να υπολογίσουμε, αλλά που αυτό είναι αδύνατον. Δίνουμε στη συνέχεια τρία κεντρικά συμπεράσματα πάνω στο πρόβλημα της διαγνωσιμότητας του τερματισμού των προγραμμάτων, και της δυνατότητας ή όχι αποφυγής αυτού του προβλήματος. Είναι όλα πολύ χαρακτηριστικά της κατάστασης που επικρατεί στο χώρο του «υπολογισμού».

Υπάρχει μια «ιδιότητα» που είναι ορίσιμη μεν, αλλά αλγοριθμικώς αδιάγνωστη.
Ένα από τα πρακτικώς πιο χρήσιμα διαγνωστικά προγραμμάτων είναι (ή μάλλον θα ήταν...) εκείνο που θα διεγίγνωσκε εάν τυχόν πρόγραμμα π είναι ή όχι εκτελεστικά «ασφαλές», εάν δηλαδή τερματίζει για όλα τα δυνατά δεδομένα δ, ή όχι:

$$\Delta_{\text{TOTAL}}(\pi) = \text{«για κάθε δ, ισχύει } \pi(\delta)\downarrow\text{»} \quad \text{ή μήπως «υπάρχει δ για το οποίο } \pi(\delta)\uparrow\text{»};$$

Τα προγράμματα που τερματίζουν για όλα τα δεδομένα θα τα αποκαλούμε τα *ολικά* προγράμματα.

Θεώρημα (i):
Ισχυρισμός: Δεν υπάρχει διαγνωστικό των ολικών προγραμμάτων μT.

Σχέδιο απόδειξης: Εφαρμόζουμε το διαγώνιο επιχείρημα του *Cantor* για να δούμε που θα μας οδηγήσει:

- υπάρχει (?) πρόγραμμα καταλογοποίησης Π όλων των ολικών προγραμμάτων.
- αφού υπάρχει το Π, υπάρχει και το πρόγραμμα π_χ, το οποίο «χρωματίζει» την θέση (κ, λ) ως «χ_1» αν τερματίζει αποδεκτικά, και με «χ_0» αν τερματίζει απορριπτικά. Ο χρωματισμός $\pi_\chi(\kappa, \lambda)$ έχει την εξής μορφή:

$\pi_\chi(\kappa, \lambda)$

```
Διαβάζουμε την περιγραφή των αριθμών κ,λ
Φτιάχνουμε δεδομένα δλ      // μέσω Λ(λ)= υπ.αρ. λ λέξη του Σ*
Φτιάχνουμε πρόγραμμα πκ     // μέσω Π(κ)= υπ.αρ. κ πρόγραμμα μT
Εκτελούμε πκ(δλ)            // μέσω καθολικού TU
Περίπτωση:
  πκ αποδέχεται δλ: γράφουμε «χ1»
  πκ απορρίπτει δλ: γράφουμε «χ0»
```

- αφού υπάρχει το πρόγραμμα π_χ, ο συμπληρωματικός διαγώνιος χρωματισμος c(i): i → ΣΥΜΠΛΗΡΩΜΑΤΙΚΟ(χ(i, i)) υπολογίζεται από (κάποιο) πρόγραμμα του καταλόγου I, δηλαδή από ολικό πρόγραμμα, διότι το π_χ είναι *ολικό* πρόγραμμα και αν το εκτελέσουμε και απλώς αντιστρέψουμε το αποτέλεσμά του, λαμβάνουμε ένα, *επίσης ολικό*, πρόγραμμα π_c για τον συμπληρωματικό διαγώνιο χρωματισμό c(·).

Τί από τα τρία δεν ισχύει; Εάν το 1ο ισχύει τότε παράγονται και τα υπόλοιπα 2ο και 3ο. Μόνον το 1ο είναι δυνατόν λοιπόν να αποτυγχάνει, άρα δεν υπάρχει πρόγραμμα καταλογοποίησης των ολικών προγραμμάτων. Κατά συνέπεια δεν υπάρχει ούτε διαγνωστικό της ιδιότητας «ολικό πρόγραμμα», διότι, σύμφωνα με το σχετικό λήμμα, το δεύτερο εξ αυτών θα εξασφάλιζε και το πρώτο...

■

Ο τερματισμός ή όχι ενός προγράμματος επί τυχόντων δεδομένων είναι αλγοριθμικώς αδιάγνωστος.
Για να είναι ένα πρόγραμμα ολικό θα πρέπει να τερματίζει για όλα τα δεδομένα. Επομένως η επιθυμία διάγνωσης μιας ιδιότητας σαν την παραπάνω ίσως να φαίνεται υπερβολική, διότι ζητά μια διάγνωση που εμπλέκει όλων των (απείρων) ειδών δεδομένα. Θα μπορούσαμε να είχαμε ένα λιγότερο απαιτητικό διαγνωστικό; Λ.χ. υπάρχει διαγνωστικό τερματισμού το οποίο δεχόμενο ένα πρόγραμμα π και κάποια δεδομένα δ, ελέγχει την «λογική» του π και αποφαίνεται αν θα έχουμε τερματισμό επί των δ ή όχι;

$$\Delta_{\text{HALT}}(\pi, \delta) = \text{«}\pi(\delta)\downarrow\text{» ή «}\pi(\delta)\uparrow\text{»;}$$

Θεώρημα (ii):
Ισχυρισμός: Δεν υπάρχει διαγνωστικό τερματισμού τυχόντος προγράμματος επί τυχόντων δεδομένων.

Σχέδιο απόδειξης: Εφαρμόζουμε το διαγώνιο επιχείρημα του *Cantor* για να δούμε που θα μας οδηγήσει:

- υπάρχει πρόγραμμα καταλογοποίησης Π *όλων* των προγραμμάτων.
- αφού υπάρχει το διαγνωστικό τερματισμού Δ, υπάρχει και το πρόγραμμα π_χ το οποίο «χρωματίζει» την θέση (κ, λ) ως «χ_1» αν το $\pi_\kappa(\delta_\lambda)$ τερματίζει, και ως «χ_0» αν το $\pi_\kappa(\delta_\lambda)$ δεν τερματίζει. Ο χρωματισμός $\pi_\chi(\kappa, \lambda)$ έχει την εξής μορφή:

$\pi_\chi(\kappa, \lambda)$

```
Διαβάζουμε την περιγραφή των αριθμών κ,λ
Φτιάχνουμε δεδομένα δλ      // μέσω Λ(λ)= υπ.αρ. λ λέξη του Σ*
Φτιάχνουμε πρόγραμμα πκ     // μέσω Π(κ)= υπ.αρ. κ πρόγραμμα μΤ
Διαγιγνώσκουμε περί πκ(δλ)  // μέσω διαγνωστικου Δ_HALT
Περίπτωση:
  πκ(δλ)↓: γράφουμε «χ1»
  πκ(δλ)↑: γράφουμε «χ0»
```

- ο συμπληρωματικός διαγώνιος χρωματισμός c(i): i → ΣΥΜΠΛΗΡΩΜΑΤΙΚΟ(χ(i, i)) υπολογίζεται από κάποιο πρόγραμμα π_c: απλά εκτελούμε (μέσω του καθολικού Τυ) τον χρωματισμό π_χ επί των π_i / δ_i, και αντιστρέφουμε το αποτέλεσμά του.

Τί από τα τρία δεν ισχύει; Το 1ο έχουμε δεί ότι είναι (εύκολα) κατορθωτό, και το 3ο προκύπτει (μέσω 2ου) από την διαθεσιμότητα του διαγνωστικού Δ_{HALT}. Επομένως μόνον το 2ο είναι δυνατόν να αποτυγχάνει, δηλαδή δεν υπάρχει διαγνωστικό Δ_{HALT} για τον τερματισμό ή όχι τυχόντος προγράμματος επί τυχόντων δεδομένων.

■

Υπάρχει ολικώς ορισμένη συνάρτηση που δεν είναι πρωτογενώς αναδρομική.

Είναι οι ατέρμονες υπολογισμοί αναπόφευκτο φαινόμενο;.

Είδαμε στα προηγούμενα τί είδους αδυναμίες προκαλούνται στο υπολογιστικό μας σύστημα (εδώ μΤ και ισοδύναμα), από το γεγονός ότι οι υπολογισμοί ενδέχεται να μην τερματίζουν. Το εξής ερώτημα αποκτά θεμελιακή – αν όχι κεντρική – θέση: είναι οι ατέρμονες «υπολογισμοί» αναπόφευκτο φαινόμενο, ή μήπως είναι απλώς μια θεραπεύσιμη παρενέργεια ενός ατελούς συστήματος προγραμματισμού;

Π.χ. στο σύστημα προγραμματισμού μέσω αναδρομικών περιγραφών είδαμε ότι καμμία από αυτές τις περιγραφές δεν οδηγεί σε ατέρμονες υπολογισμούς – πλην μιας: η περιγραφή Μ[Φ] της (αναζήτησης της) «ελάχιστης ρίζας». Μήπως εάν δεν χρησιμοποιούσαμε αυτήν την δυνατότητα θα εξομαλύναμε την κατάσταση;

Ορίζουμε ως ***πρωτογενώς αναδρομικες περιγραφές*** τις περιγραφές που ορίζονται μόνον μέσω των τριών βασικών αναδρομικών περιγραφών (*μηδενισμός*,

αύξηση +1, *επιλογή*), και των σύνθετων αναδρομικών περιγραφών *σύνθεση* και *αναδρομή*. Είναι εμφανές ότι οι συναρτήσεις που επιδέχονται πρωτογενώς αναδρομικές περιγραφές είναι όλες *ολικές συναρτήσεις*, και επιδέχονται κάποιο *ολικό* πρόγραμμα μηχανής *Turing* που τις υπολογίζει.

Αυτός ο (πρωτογενής) «προγραμματισμός» δεν είναι σε θέση να υπολογίσει συναρτήσεις που είναι μερικώς ορισμένες, αλλά γιατί να ασχοληθούμε με τέτοιες συναρτήσεις; Αν μια συνάρτηση φ(-) δεν ορίζεται για κάποιο όρισμα x, το σχετικό πρόγραμμα θα μπορούσε (;) να τερματίζει και να απαντά «δεν ορίζεται», και όχι να μεταπίπτει σε ατέρμονα υπολογισμό...

Ακόμα και εάν εγκαταλείψουμε τις μερικώς ορισμένες συναρτήσεις στη τύχη τους, η απορία παραμένει, υπό την εξής μορφή: *είναι δυνατόν να υπολογίσουμε κάθε ολική συνάρτηση μέσω μιας πρωτογενώς αναδρομικής περιγραφής*; Ατυχώς, όχι.

Θεώρημα (iii):

Ισχυρισμός: Υπάρχει συνάρτηση φ: $\Sigma^* \to \Sigma^*$, ολικώς ορισμένη, η οποία είναι υπολογίσιμη μεν, αλλά δεν υπολογίζεται από καμμία *πρωτογενώς* αναδρομική περιγραφή.

Σχέδιο απόδειξης: Εφαρμόζουμε το διαγώνιο επιχείρημα του *Cantor* για να δούμε που θα μας οδηγήσει:

- υπάρχει πρόγραμμα καταλογοποίησης Π *όλων* των πρωτογενώς αναδρομικών περιγραφών (ΑΠ).
- αφού υπάρχει το πρόγραμμα εξομοίωσης των αναδρομικών περιγραφών, υπάρχει και πρόγραμμα π_χ το οποίο «χρωματίζει» την θέση (κ, λ) με «χ_0» αν $\pi_\kappa(\delta_\lambda) = 0$, και με «$\chi_1$» αν $\pi_\kappa(\delta_\lambda) \geq 1$. Ο χρωματισμός $\pi_\chi(\kappa, \lambda)$ έχει την εξής μορφή:

$\pi_\chi(\kappa, \lambda)$

```
Διαβάζουμε την περιγραφή των αριθμών κ,λ
Φτιάχνουμε δεδομένα δλ      // μέσω Λ(λ)= υπ.αρ. λ λέξη του Σ*
Φτιάχνουμε πρόγραμμα πκ     // μέσω Π(κ)= υπ.αρ. κ πρόγραμμα μΤ
Διαγιγνώσκουμε περί πκ(δλ)  // μέσω εξομοιωτή ΤΑΠ
Περίπτωση:
  πκ(δλ) δίδει αποτέλεσμα =0: γράψε «χ0»
  πκ(δλ) δίδει αποτέλεσαμ ≥1: γράψε «χ1»
```

- το πρόγραμμα του συμπληρωματικού διαγώνιου χρωματισμού π_c (ακριβέστερα: κάποια αναδρομική περιγραφή του καταλόγου Ι ισοδύναμη με αυτό), συμπεριλαμβάνεται στον κατάλογο του Π, (δηλαδή: επιδέχεται δηλαδή μια πρωτογενώς αναδρομική περιγραφή).

Τί από τα τρία δεν ισχύει; Το 1ο έχουμε δεί ότι είναι κατορθωτό: παράγουμε όλες τις λέξεις και με μια συντακτική ανάλυση[9] φιλτράρουμε μόνον εκείνες που δίδουν μια πρωτογενώς αναδρομική περιγραφή. Το 2ο επίσης ισχύει, διότι προκύπτει από την διαθεσιμότητα του προγράμματος μΤ που εξομοιώνει τις αναδρομικές περιγραφές. Εδώ, μόνον το 3ο είναι δυνατόν να αποτυγχάνει... Επομένως το πρόγραμμα χρωματισμού π_c (και εδώ: *ούτε κανένα ισοδύναμο με αυτό*, λ.χ. μια ΑΠ), δεν συμπεριλαμβάνεται στον κατάλογο Ι: το πρόγραμμα π_c δεν επιδέχεται μια πρωτογενώς αναδρομική περιγραφή. *Είναι* όμως πρόγραμμα μηχανής *Turing*,, και μάλιστα *ολικώς ορισμένο* – αφού όλες οι πρωτογενώς αναδρομικές περιγραφές δίδουν ολικώς ορισμένες συναρτήσεις. Ο συμπληρωματικός διαγώνιος χρωματισμός είναι, εδώ, *ολική* συνάρτηση μεν, αλλά δεν επιδέχεται *καμμία πρωτογενώς* αναδρομική περιγραφή. Κατά συνέπεια οι πρωτογενώς αναδρομικές περιγραφές *δεν* επαρκούν για τον υπολογισμό όλων των ολικών συναρτήσεων, και οι ατέρμονες υπολογισμοί είναι αναπόφευκτο μέρος του υπολογιστικού παιγνιδιού...

■

[9] Ιδού – εν συντομία – μια ασυμφραστική γραμματική που παράγει όλες τις αναδρομικές περιγραφές:

⟨ περιγραφή ⟩ → ⟨ βασική ⟩ | ⟨ σύνθετη ⟩
⟨ δείκτης ⟩ → (η παράσταση ενός φυσικού αριθμού)
⟨ βασική ⟩ → **Z** ⟨ δείκτης ⟩ | **I** ⟨ δείκτης ⟩ **,** ⟨ δείκτης ⟩ | **S**
⟨ σύνθετη ⟩ → **C(** ⟨ σειρά-περιγραφών ⟩ **)** | **R(** ⟨ περιγραφή ⟩ **,** ⟨ περιγραφή ⟩ **)** | **M(** ⟨ περιγραφή ⟩ **)**
⟨ σειρά-περιγραφών ⟩ → ∅ | ⟨ περιγραφή ⟩ **,** ⟨ σειρά-περιγραφών ⟩

Εάν αφαιρέσουμε τον κανόνα ⟨ ... ⟩ → **M(** ⟨ ... ⟩ **)** θα παράγουμε μόνον τις *πρωτογενώς* αναδρομικές περιγραφές.

24ο Η μερική καταστροφή (θ. *Rice*) και η ολική πανωλεθρία (θ. *Gödel*).

Πόση έκταση έχει το φαινόμενο των ανεπίλυτων προβλημάτων;.

Είδαμε στις προηγούμενες ενότητες ότι υπάρχουν (αλγοριθμικώς) ανεπίλυτα προβλήματα, δηλαδή συναρτήσεις από πεπερασμένα δεδομένα προς πεπερασμένα αποτελέσματα, για τις οποίες δεν υπάρχει κανένα πρόγραμμα (μΤ ή και όποιο άλλο) που να τις υπολογίζει.

Εκ πρώτης όψεως αυτό φάνηκε ως εύλογο αποτέλεσμα, αφού και μόνον οι «γλώσσες» επί ενός αλφαβήτου είναι πληθικώς περισσότερες από τα διαθέσιμα προγράμματα. Είδαμε όμως στη συνέχεια οτι ακόμα και εύλογα, και κατά πολύ φυσικό τρόπο ορισμένα προβλήματα, προβλήματα μάλιστα με αξιοσημείωτο πρακτικό αντίκτυπο, (όπως εκείνο του ασφαλούς τερματισμού), είναι επίσης ανεπίλυτα. *Πόση έκταση έχει αυτό το φαινόμενο*; Θα δούμε στην ενότητα που ακολουθεί ότι έχει πολύ μεγαλύτερη έκταση από όση θα μπορούσε κανείς να φανταστεί.

Ας αναλογιστούμε τί είναι μια ιδιότητα Ι των συναρτήσεων Φ: $\Sigma^* \to \Sigma^* \cup \{\uparrow\}$: δεν είναι παρά μια διαμέριση του Φ σε δύο μέρη/υποσύνολα. Σε εκείνο που περιλαμβάνει όσες έχουν την ιδιότητα Ι, και στις υπόλοιπες που δεν έχουν την Ι. Υπάρχουν πολλές κρίσιμες ιδιότητες που θα θέλαμε να διαγιγνώσκουμε περί μιας συναρτήσεως $\phi \in \Phi$. Π.χ. :

- είναι η φ σταθερή, ή λαμβάνει τουλάχιστον δύο τιμές;
- είναι η φ παντού καλώς ορισμένη ή για κάπιο x ισχύει $\phi(x) = \uparrow$;
- υπάρχει πρόγραμμα που να υπολογίζει την φ σε λιγότερο από ν=100 βήματα;
- είναι η φ ίση (παντού) με μια άλλη συνάρτηση ψ;
- λαμβάνει η φ άπειρες τιμές ή το πεδίο τιμών της είναι πεπερασμένο; κοκ.

Μέσω τίνος θα μπορούσαμε να διαγιγνώσκουμε τέτοιες ιδιότητες; Οι συναρτήσεις «υπάρχουν» (σε όποιο φιλοσοφικό κόσμο θέλουν αυτές), αλλά *εμείς* δεν έχουμε πρόσβαση σε αυτές παρά μόνον μέσω των προγραμμάτων που τις ορίζουν (επειδή *και* τις υπολογίζουν). Τα προγράμματα έχουν δύο ειδών ιδιότητες.

Το 1ο είδος αφορά στα ίδια ως λέξεις /κείμενα, έχει δηλαδή ***συντακτικό*** χαρακτήρα. Π.χ.:

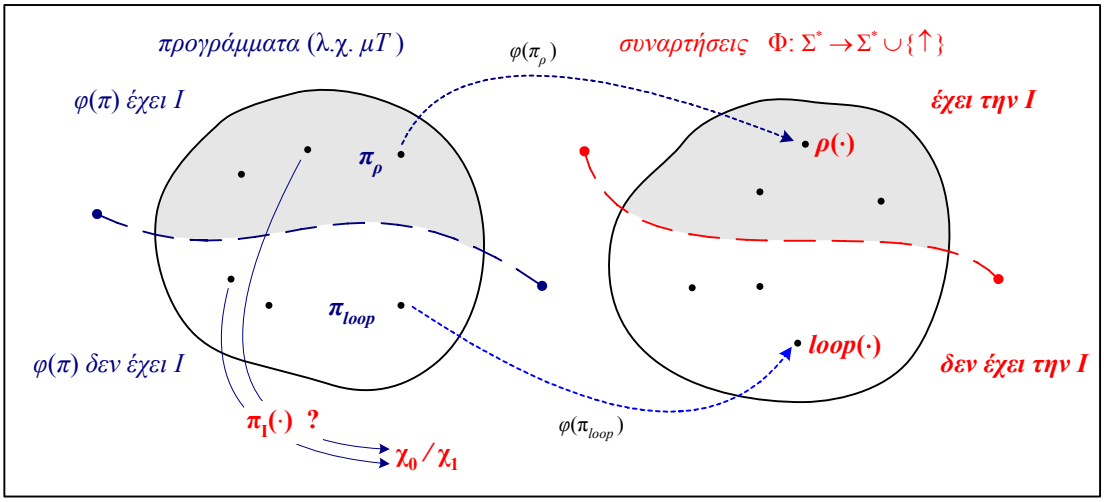
- έχει το πρόγραμμα π μήκος μικρότερο από ν = 1000;
- πόση (σταθερή) μνήμη χρησιμοποιεί το π;
- είναι προσβάσιμη η τάδε μεταβλητή;

Το 2ο είδος αφορά στη *σημασία* που έχουν, (αφορά στις συναρτήσεις που αυτά υπολογίζουν), έχει δηλαδή σημασιολογικό χαρακτήρα. Θα λέμε ότι μια ιδιότητα Ι των προγραμμάτων π είναι ***σημασιολογική***, εάν προκύπτει από την *ερμηνεία* (ή *σημασία*) που δίνουμε στο π· εάν δηλαδή δύο προγράμματα π, π' υπολογίζουν την ίδια συνάρτηση, $\phi(\pi) = \phi(\pi')$, τότε είτε έχουν και τα δύο την Ι, είτε και τα δύο δεν την έχουν: αν $\phi(\pi) = \phi(\pi')$ τότε $(\pi \in I) \Leftrightarrow (\pi' \in I)$.

Προφανώς αυτό που θα μας ενδιέφερε είναι η διάγνωση ***μη-τετριμμένων*** σημασιολογικών ιδιοτήτων, δηλαδή ιδιοτήτων που κάποιες συναρτήσεις τις έχουν, αλλά και κάποιες άλλες δεν τις έχουν. Στην ενότητα αυτή θα αποδείξουμε ότι *κάθε μη-τετριμμένη σημασιολογική ιδιότητα των προγραμμάτων, είναι αλγοριθμικώς αδιάγνωστη*: δεν υπάρχει κανένα πρόγραμμα που να την διαγιγνώσκει...

Θεώρημα *Rice*:
Ισχυρισμός: Κάθε μη-τετριμμένη σημασιολογική ιδιότητα I των προγραμμάτων είναι αδιάγνωστη.

Σχέδιο απόδειξης: Η κατάσταση εικονίζεται ως εξής: το σύνολο των (υπολογίσιμων) συναρτήσεων Φ διαμερίζεται στα δύο: όσες φ ∈ Φ έχουν την ιδιότητα I, και σε όσες δεν έχουν την I. Η ιδιότητα αυτή είναι μη-τετριμμένη, επομένως και τα δύο μέρη είναι διάφορα του κενού. Έστω ότι (χωρίς απώλεια γενικότητας) ότι στο μέρος «όχι-I» περιλαμβάνεται η παντού αόριστη συνάρτηση loop(·) η οποία στέλνει κάθε λέξη λ ∈ Σ* στη αόριστη τιμή ↑, και ότι στο μέρος «ναί-I» περιλαμβάνεται μια συνάρτηση ρ(·). Η μεν 1η υπολογίζεται από κάποιο πρόγραμμα loop το οποίο δεν τερματίζει ποτέ, η δε 2η από ένα κάποιο πρόγραμμα π_ρ. Το ερώτημα που (θα απαντήσουμε αρνητικά) είναι: υπάρχει ένα διαγνωστικό **πι(·)**, το οποίο δεδομένου ενός προγράμματος π θα απαντά ισοδυνάμως με «ναί/όχι» ανάλογα με το εάν η συνάρτηση φ(π) που υπολογίζει το π, έχει την ιδιότητα I ή όχι;



Σχήμα: συναρτήσεις δύο ειδών, και η διάγνωση μέσω προγραμμάτων γι' αυτές.

Αν παρατηρήσουμε ότι η ιδιότητα I παράγει ένα διχρωματισμό των προγραμμάτων (σε «∈I» και «∉I»), μπορούμε να επιδιώξουμε αυτά τα δύο χρώματα να καταστούν ένας κατασκευάσιμος χρωματισμός του τετραγώνου των υπολογισμών – πράγμα άτοπο όπως είδαμε στην προηγούμενη ενότητα. Αυτό (κατά το ήμισυ) επιτυγχάνει το εξής πρόγραμμα, χρωματίζοντας το τετράγωνο με τις συναρτήσεις ρ(·) και *loop*(·):

$\pi_{RICE}(\pi, \delta)$ =

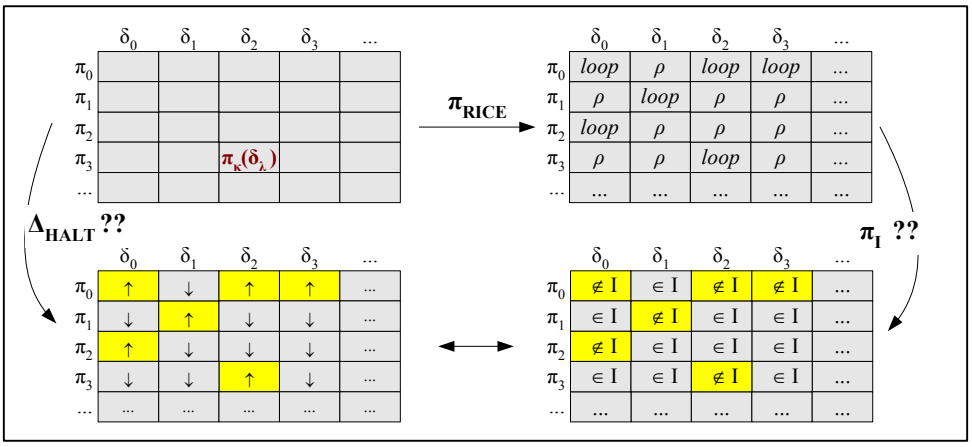
```
Διαβάζουμε τα π και δ
Παραδίδουμε την περιγραφή του εξής προγράμματος r_{π,δ}:

r_{π,δ} =
    Διαβάζουμε τα δεδομένα x
    Εκτελούμε το π(δ)
    Περίπτωση:
      π(δ)↓: Εκτελούμε το π_ρ(x)
      π(δ)↑: ↑ // είμαστε ήδη σε ατέρμονα υπολογισμό!
```

Για κάθε $\pi = \pi_\kappa$ και $\delta = \delta_\lambda$, κ, λ ≥ 0, το παραπάνω πρόγραμμα παράγει ένα διαφορετικό *πρόγραμμα* $r_{\pi,\delta}$. Αλλά τί *συναρτήσεις* υπολογίζουν τα διάφορα $r_{\pi,\delta}$; Μόνον δύο ειδών:

- αν το π(δ) τερματίζει, τότε το πρόγραμμα $r_{\pi,\delta}$ από δεδομένα x θα παραδώσει ότι αποτέλεσμα θα είχε, (αν είχε), το $\pi_\rho(x)$ – άρα υπολογίζει την συνάρτηση ρ(·).
- αν το π(δ) δεν τερματίζει, τότε το $r_{\pi,\delta}$ από δεδομένα x, όποια και εάν είναι, δεν θα τερματίσει ποτέ ούτε αυτό (περίπτωση π(δ)↑) – άρα υπολογίζει τη συνάρτηση *loop*(·).

Μέσω, λοιπόν, του προγράμματος π_{RICE} είναι δυνατόν να χρωματίσουμε κατασκευστικά το τετράγωνο των υπολογισμών με προγράμματα που υπολογίζουν είτε την ρ(·) είτε την loop(·), ανάλογα με το εάν π(δ) τερματίζει ή όχι, (βλ. σχήμα). Εάν διαθέταμε ένα διαγνωστικό πι για την ιδιότητα I, θα μπορούσαμε στην συνέχεια να χρωματίσουμε κατασκευαστικά το τετράγωνο με «ΝΑΙ» ή «ΟΧΙ»,

ανάλογα με το εάν «π(δ) ↓» ή «π(δ) ↑», θα είχαμε δηλαδή κατασκευάσει το διαγνωστικό $\Delta_{\text{HALT}}(\pi, \delta)$ – πράγμα που είδαμε ότι είναι αδύνατον. Επομένως το υποτιθέμενο διαγνωστικό πι για την ιδιότητα I είναι ανύπαρκτο:



Σχήμα: ένα υποτιθέμενο διαγνωστικό πρόγραμμα πι για την ιδιότητα I
θα χρωμάτιζε, (μέσω πRICE), τον πίνακα τερματισμού των προγραμμάτων όπως το $\Delta_{\text{HALT}}(\pi, \delta)$.

■

Θεώρημα Rice – δηλαδή;

Πώς να ερμηνεύσουμε ένα τέτοιο συμπέρασμα; Τί αντίκτυπο έχει στη πρακτική μας; Προφανώς το θεώρημα αυτό δεν λέει ότι τα προγράμματα είναι άχρηστα. Εξ άλλου αυτό αντιβαίνει τόσο στην εμπειρία μας, όπου προγράμματα και υπολογιστές είναι πανταχού παρόντα, όσο και στη θεωρητική μας δραστηριότητα, όπου οι κατασκευαστικές μέθοδοι ήσαν πάντοτε, και παραμένουν, περιζήτητες.

Αυτό που λέει το θ. Rice είναι πώς ακόμα κάθε μη τετριμμένη πληροφορία για μια συνάρτηση φ, ενώ δεν μπορεί παρά να προέλθει από τον *ορισμό* αυτής της συνάρτησης, (λ.χ. ένα πρόγραμμα γι' αυτήν), εν τούτοις δεν μπορεί να προέλθει *κατά αλγοριθμικό τρόπο*.

Δηλαδή αν μας ενδιαφέρει η διάγνωση μιας συναρτησιακής ιδιότητας I, τότε για την μια συνάρτηση θα χρειαστεί να σκεφτούμε με αυτόν τον τρόπο, για μια άλλη συνάρτηση με κάποιο άλλο, και για τρίτη με κάποιο τρίτο τρόπο, χωρίς *καμμία ομοιομορφία να μην αναδύεται πίσω από όλους τους τρόπους που θα χρειαστούμε*, (αν εξετάζαμε ως προς την I «όλες» τις συναρτήσεις).

Και έλλειψη ομοιομορφίας σε μια ανάλυση που θα θέλαμε να πραγματοποιούσαμε μεθοδικά, (αν όχι μηχανικά), σημαίνει ότι αυτή η ανάλυση απαιτεί κάθε φορά μια ίσως «νέα» προσέγγιση, απαιτεί δηλαδή εξω-αλγοριθμικές και μη-μηχανικές ικανότητες, όπως *φαντασία*, *έμπνευση*, *διαίσθηση*, (ή και την *σύμπτωση*, δηλαδή την *τύχη*).

Το εντυπωσιακό στοιχείο σε αυτό το συμπέρασμα είναι ότι δεν αφορά μόνον σε κάποιες ίσως «περίπλοκες» ιδιότητες των συναρτήσεων, αλλά ότι ακόμα και η παραμικρή πληροφορία για μια συνάρτηση φ, οποιαδήποτε, αρκεί να μην ισχύει για όλες τις συναρτήσεις (!), εμφανίζει αυτή την δυσκολία στην εξαγωγή της!

Η θεωρία υπολογισμού είναι ίσως ο μόνος κλάδος που είναι σε θέση να *αποδείξει μαθηματικά* όχι μόνον την χρησιμότητα της συνεισφοράς του, αλλά και ότι τα εκάστοτε πρόσωπα στα οποία αυτή οφείλεται, είναι αναντικατάστατα...

Από την ήττα στην πανωλεθρία: το θεώρημα του Gödel.

Υπάρχει ένας τρόπος υπολογισμού τον οποίο σιωπηρώς παρακάμψαμε (ίσως επίτηδες), παρά το γεγονός ότι διαποτίζει όλες αυτές τις σημειώσεις (όπως και κάθε κλάδο με μαθηματικό περιεχόμενο): ο *λογικός προγραμματισμός*.

Η μαθηματική λογική, όπως πλέον την αντιλαμβανόμαστε, δεν αφορά μόνον έναν τρόπο συλλογισμού και έκφρασης, αλλά και ένα τρόπο *γραφής* και *υπολογισμού*: χρησιμοποιούμε, όπως συνηθίσαμε σε αυτές τις σημειώσεις, ένα *αλφάβητο* για να συντάσσουμε *προτάσεις*, και τηρούμε μια «*λογική*», δηλαδή ένα σύστημα παραγωγής *συμπερασμάτων* από *υποθέσεις*. Τις πλέον αυτονόητες από αυτές τις δεχόμαστε ως αφετηριακές, και τις ονομάζουμε *αξιώματα*, και όσες είναι δυνατόν να παραχθούν *αποδεικτικά* από τα αξιώματά μας τις αποκαλούμε *θεωρήματα*.

Το σύστημα παραγωγής συμπερασμάτων είναι εξαιρετικά απλό· αποτελείται από δύο μόνον κανόνες, και ως «πρόγραμμα υπολογισμού» είναι μάλλον το αρχαιότερο πρόγραμμα που τρέχει επί της οικουμένης, (εδώ και πολλές χιλιάδες χρόνια – αν και όχι υπό την οπτική γωνία και με την ορολογία που χρησιμοποιούμε εδώ):

1ο *κανόνας:* σε ένα «τρέχον» σύνολο προτάσεων επιτρέπεται να προσθέσουμε οποιαδήποτε πρόταση φ δεχόμαστε αξιωματικά.

2ος *κανόνας:* εάν ένα «τρέχον» σύνολο προτάσεων περιλαμβάνει τόσο την υποθετική πρόταση $\phi \to \psi$ («*εάν φ τότε ψ*»), όσο και την *υπόθεση* φ, τότε επιτρέπεται να συμπεριλάβουμε σε αυτό και το συμπέρασμα ψ. (Ο κανόνας αυτός καλείται modus ponens – αν και έχει αναγνωριστεί ως τέτοιος από την αρχαιότητα, πολύ πριν από την εποχή αυτής της ονομασίας.)

Αυτοί οι κανόνες επιτρέπουν μια υπολογιστική δραστηριότητα με βάση την λογική. Τα στοιχεία ενός υπολογισμού υ είναι εδώ τα εξής:

- *Αρχικό στάδιο* υ_0*:*
  Ένα σύνολο προτάσεων υ_0, οι αρχικές *υποθέσεις*, και μια πρόταση χ ως επιδιωκόμενο συμπέρασμα.
- *Βήματα υπολογισμού* $\upsilon_\kappa \to \upsilon_{\kappa+1}$*:*
  Σύμφωνα με τον 1ο κανόνα, μπορούμε να προσθέσουμε στο υ_κ οποιαδήποτε πρόταση φ δεχόμαστε αξιωματικά, παράγοντας το $\upsilon_{\kappa+1} \leftarrow \upsilon_\kappa \cup \{ \phi \}$.
  Και σύμφωνα με τον 2ο κανόνα, εάν το τρέχον στάδιο περιλαμβάνει τις προτάσεις φ και $\phi \to \psi$ τότε μπορούμε να προσθέσουμε στο υ_κ την πρόταση ψ, παράγοντας το $\upsilon_{\kappa+1} \leftarrow \upsilon_\kappa \cup \{ \psi \}$.
- *Τελικό στάδιο* υ_{last1}*:*
  Οποιοδήποτε στάδιο περιλαμβάνει την πρόταση χ ή την άρνησή της $\neg\chi$.

Γνωρίζουμε καλά τον όρο που χρησιμοποιούμε για τέτοιου είδους «υπολογισμούς»: λέγονται *αποδείξεις*, (και ακριβέστερα *τυπικές* αποδείξεις). Ποιά είναι η υπολογιστική τους ισχύς;

Το ελάχιστο φυσικό πλαίσιο για την ανάλυση αυτού του ζητήματος είναι η θεωρία των φυσικών αριθμών, διότι οι υπολογισμοί έχουν ένα αφετηριακό σημείο «0», και για κάθε στάδιο υπολογισμού (ίσως) υπάρχει ένα επόμενο «+1» – όπως συμβαίνει με τους φυσικούς αριθμούς: αρχίζουμε από το μηδέν «0» και τους παράγουμε όλους με βήματα «+1». Για τους φυσικούς αριθμούς έχουμε στη διάθεση μας ένα γλωσσικό εργαλείο και συγκεκριμμένα μια γλώσσα για να γράφουμε *αριθμητικές προτάσεις*, η οποία περιέχει,

- τις βασικές σταθερές όπως 0 και 1.
- σύμβολα για τις βασικές πράξεις (πρόσθεση +, πολλαπλασιασμό ×, κτλ).
- σύμβολα για τις βασικές σχέσεις (ισότητα =, σύγκριση ≥, κτλ).
- λογικά σύμβολα, όπως ∀, ∃, ¬, →, ∨, ∧.

Έχουμε επίσης στη διάθεσή μας ένα λογικό οπλοστάσιο:

- τα αξιώματα της κατηγορηματικής λογικής.
- τα αξιώματα των φυσικών αριθμών(λ.χ. τα αξιώματα Peano).[10]

Με βάση αυτό το υλικό είμαστε σε θέση να κάνουμε *αποδείξεις αριθμητικών προτάσεων*. Ποιά είναι η ισχύς αυτού του αποδεικτικού (ή *υπολογιστικού*, αν θέλετε...) μηχανισμού; Θα δείξουμε στη συνέχεια,

[10] Για το ποιά είναι αυτά τα αξιώματα, βλέπε τη γενική βιβλιογραφία (ή δικτυογραφία) περί λογικής και περί αριθμοθεωρίας.

(ακολουθώντας τις ιδέες του *Gödel* – αν και όχι την ίδια οδό που εκείνος ακολούθησε), ότι η γλώσσα της αριθμητικής είναι ιδιαίτερα εκφραστική: είναι, συγκεκριμμένα, σε θέση να εκφράσει όλα όσα θα θέλαμε να περιγράψουμε περί των ενεργειών μιας μηχανής *Turing*. Και, όπως θα φανεί, αυτή η εκφραστική ισχύς είναι τελικά μια καταδικαστική φλυαρία: η γλώσσα της αριθμητικής μπορεί να εκφράσει και να ισχυριστεί περισσότερα από όσα μπορεί να αποδείξει...

Η αριθμοποίηση ακολουθιών:
Η εκφραστική ισχύς της γλώσσας της αριθμητικής έγκειται (μεταξύ άλλων) στο ότι είναι δυνατόν να περιγράψει την «κωδικοποίηση» μιας πεπερασμένης <u>ακολουθίας</u> φυσικών αριθμών με <u>έναν</u> αριθμό. Ένας τρόπος για να επιτευχθεί αυτό αριθμητικά είναι η κωδικοποίηση κατά *Gödel:*

- Ορίζουμε ως p_i, $i \geq 0$, την ακολουθία των πρώτων αριθμών: $\langle 2, 3, 5, 7, 11, 13, 17, 19, \ldots, \rangle$
- Ως κωδικό της ακολουθίας $\mathbf{m} = \langle m_i : i = 0, 1, 2, \ldots, \mu-1 \rangle$ ορίζουμε το αριθμό $\#\mathbf{m} = \prod_{i=0}^{\mu-1} p_i^{(1+m_i)}$. Ο κωδικός λ.χ. της ακολουθίας $\langle 3, 0, 2, 4 \rangle$ είναι ο $2^{1+3}\,3^{1+0}\,5^{1+2}\,7^{1+4} = 100{,}842{,}000$.

Η ιδέα είναι απλή. Κάθε αριθμός επιδέχεται μια παραγοντοποίηση σε γινόμενο πρώτων αριθμών, *κατά έναν και μοναδικό τρόπο*. Αν λοιπόν ένας αριθμός n γράφεται ως $n = \prod_{i=0}^{\mu-1} p_i^{(1+m_i)}$, είναι δυνατόν από αυτόν να «ανακτήσουμε» την ακολουθία των εκθετών $(1+m_i)$ του p_i για p_i από 2 έως τον μεγαλύτερο και τελευταίο πρώτο-παράγοντα του n, και επομένως να ανακτήσουμε την ακολουθία **m**. Από τεχνικής πλευράς, οι εξής *αριθμητικές* συναρτήσεις αρκούν για να θεμελιώσουν την «***αριθμοποίηση***» όλων όσων θα χρειαστούμε:

Συναρτήσεις	Ερμηνεία (εφόσον ο ν είναι όντως ο κωδικός κάποιας ακολουθίας).
μ = μήκος(n)	Το μήκος μ της ακολουθίας $\langle m_0, m_2, \ldots., m_{\mu-1} \rangle$ που κωδικοποιεί ο αριθμός n.
ν = θέση(n, i)	Το υπ. αρ. *i* στοιχείο της ακολουθίας $\langle m_0, m_1, \ldots., m_{\mu-1} \rangle$ που κωδικοποιεί ο n.

Δεν είναι τόσο σημαντικό να δώσουμε εδώ όλες τις λεπτομέρειες του χειρισμού τέτοιων συναρτήσεων, λίγα δείγματα γραφής είναι όμως αρκετά για να ωθήσουν τη φαντασία του αναγνώστη:

Κατηγόρημα	Γραφή στη γλώσσα της αριθμητικής
Διαιρεί(*d*, *p*) ≡	(ΑΛΗΘΕΣ ⇔ ο d διαιρεί τον p.) $\exists q\,((d \times q) = p)$
Πρώτος(*p*) ≡	(ΑΛΗΘΕΣ ⇔ ο p είναι πρώτος.) $\forall q(\text{Διαιρεί}(q, p) \rightarrow ((q = 1) \vee (q = p)))$
Δύναμη(*p*, *κ*, *ν*) ≡	(ΑΛΗΘΕΣ ⇔ το ν είναι η κ-στή δύναμη του p.) $\Delta(p, \kappa, \nu)$ $\wedge$ $\forall p \forall \kappa \forall \nu\ \Delta(p, \kappa, \nu) \leftrightarrow ((\kappa = 0) \wedge (\nu = 1)) \vee ((\kappa > 0) \wedge \exists \lambda\,(\Delta(p, \kappa-1, \lambda) \wedge (\nu = \lambda \times p)))$
Δείκτης(*p*, *i*) ≡	(ΑΛΗΘΕΣ ⇔ ο p είναι ο υπ. αρ. i πρώτος αριθμός.) $I(p, i)$ $\wedge$ $\forall p \forall i\ I(p, i) \leftrightarrow ((i=0) \wedge (p=2)) \vee ((i>0) \wedge \text{Πρώτος}(p) \wedge \exists q(I(q, i-1) \wedge \forall x((q<x<p) \rightarrow \neg \text{Πρώτος}(x)))$
Μήκος(*n*, *μ*) ≡	(ΑΛΗΘΕΣ ⇔ (μ = μήκος της ακολουθίας που κωδικοποιεί το n).) $\forall \kappa\ ((0 \leq \kappa < \mu) \rightarrow \exists p(\text{Δείκτης}(p, \kappa) \wedge \text{Διαιρεί}(p, n)))$ $\wedge$ $\forall \kappa\ ((\kappa \geq \mu) \rightarrow \neg \exists p(\text{Δείκτης}(p, \kappa) \wedge \text{Διαιρεί}(p, n)))$
Θέση(*n*, *i*, m_i)	(ΑΛΗΘΕΣ ⇔ (m_i = το υπ. αρ. i στοιχείο που κωδικοποιεί το n.) $\exists p \exists \lambda\ (\text{Δείκτης}(p, i) \wedge \text{Δύναμη}(p, 1+m_i, \lambda) \wedge \text{Διαιρεί}(\lambda, n) \wedge \neg \text{Διαιρεί}(\lambda \times p, n))$ $\wedge$ $\exists \mu\ \text{Μήκος}(n, \mu)$

Διαβάζουμε σε «καθημερινή γλώσσα» την 4η σειρά του πίνακα, (*Δείκτης*(*p*, *i*)), ώστε να δώσουμε μια ιδέα αυτού του τρόπου έκφρασης:

«το κατηγόρημα I(p, i) ισχύει εάν και μόνον ο αριθμός p έχει δείκτη i στην ακολουθία των πρώτων, και αυτό διότι το I(p, i) ισοδυναμεί με ότι, είτε (α) ο δείκτης είναι ο «υπ. αρ. 0» και ο p είναι ο αριθμός 2, είτε (β) ο δείκτης είναι τουλάχιστον 1, ο p είναι πρώτος αριθμός, και ανάμεσα σε αυτόν και τον «προηγούμενο» αριθμό q που ικανοποιεί το κατηγόρημα, (δηλαδή: ισχύει I(q, i–1)), δεν μεσολαβεί κανένας άλλος πρώτος αριθμός».

Ως προς την 6η γραμμή του πίνακα, μια ματιά στον ορισμό της κωδικοποίησης $\langle m_i : i = 0...\mu-1 \rangle \rightarrow \#m$ μας πείθει γιατί «ζητείται» ο p_i και ο εκθέτης $(1+m_i)$: $\#m = n = \prod_{i=0}^{\mu-1} p_i^{(1+m_i)}$. Γιατί όμως ζητάμε και το «∃μ Μήκος(n, μ)»; Διότι αυτό μας εξασφαλίζει ότι ο αριθμός είναι όντως κάποιος κωδικός· αλλιώς θα μπορούσε απλώς $n = p_i^{(1+m_i)}$, και η ακρίβεια της περιγραφής θα κινδύνευε: το n θα εμφανιζόταν ως κωδικός ενώ αυτό θα ήταν ψευδές. Έτσι, αν για την απο-κωδικοποίηση χρησιμοποιούμε μόνον τα κατηγορήματα *Μήκος(·,·)* και *Θέση(·,·)*, θα έχουμε εξασφαλίσει την ορθή χρήση των ορθών κωδικών.

Η αριθμοποίηση των μηχανών Turing:
Για να μεταφέρουμε τις μηχανές *Turing* στον κόσμο των φυσικών αριθμών, χρησιμοποιούμε μια απλή κωδικοποίηση των *στοιχείων* αυτών των μηχανών (*σύμβολα*, *καταστάσεις*, *ταινία*, κττ), μέσω φυσικών αριθμών. Ο παρακάτω πίνακας δίδει τις σχετικές «κωδικοποιήσεις»:

Στοιχείο μηχανής	Κωδικοποίηση #
σύμβολο $\alpha \in \Sigma$	#α: Ο αύξων αριθμός (από 1 έως $\lvert\Sigma\rvert$) του συμβόλου $\alpha \in \Sigma$. Για το «κενό»: #_ = 0.
κατάσταση $q \in \Sigma_K$	#q: Ο αύξων αριθμός (από 1 έως $\lvert\Sigma_K\rvert$) της κατάστασης $q \in \Sigma_K$.
ταινία t	#t: Ο κωδικός της ακολουθίας $\langle \#t_{[0]}, \#t_{[1]}, ..., \#t_{[n]} \rangle$, (για n όσο θα χρειαστεί...).
κεφαλή h	#h: Αρκεί το ίδιο το $h \geq 0$.
μετακίνηση δ	#δ: Με $\delta = -1, 0, +1$, αρκεί #δ = (δ+1) (ώστε ο #δ να είναι φυσικός ≥ 0).
εντολή ε	#ε: Αν ε = «(K, α) → (K′, α′, δ)», αρκεί #ε = ο κωδικός της 5άδας ⟨#K, #α, #K′, #α′, #δ⟩.
στάδιο σ	#σ: Για κάθε στάδιο σ = (t, h, q), αρκεί #σ = ο κωδικός της 3άδας ⟨#t, #h, #q⟩.
υπολογισμός υ	#υ: Ο κωδικός της ακολουθίας των σταδίων $\langle \#\upsilon_{[0]}, \#\upsilon_{[1]}, ..., \#\upsilon_{[last]} \rangle$, (αν έχει τέρμα...).

Φυσικά, για την αριθμητική αναπαράσταση της μηχανής θα χρειαστούμε και τις αντίστοιχες «αποκωδικοποιήσεις», που δίδονται στον ακόλουθο πίνακα:

Αποκωδικοποίηση		Ερμηνεία
ταινία(#σ)	= θέση(#σ, 0)	Δίδει τον κωδικό #t της ταινίας κατά το στάδιο #σ.
κεφαλή(#σ)	= θέση(#σ, 1)	Δίδει την θέση h της κεφαλής κατά το στάδιο #σ.
κατάσταση(#σ)	= θέση(#σ, 2)	Δίδει τον κωδικό #q της κατάστασης της μηχανής στο στάδιο #σ.
εντολή(#π, κ)	= θέση(#π, κ)	Δίδει τον κωδικό #ε της υπ. αρ. κ εντολής του προγράμματος #π.
K(#ε) *α(#ε)* *K′(#ε)* *α′(#ε)* *δ(#ε)*	= θέση(#ε, 0) = θέση(#ε, 1) = θέση(#ε, 2) = θέση(#ε, 3) = θέση(#ε, 4)–1	Δίδουν τους κωδικούς των 5 στοιχείων μιας εντολής με κωδικό #ε, η οποία έχει την γενική μορφή: «(K, α) → (K′, α′, δ)».
στάδιο(#υ, κ)	= θέση(#υ, κ)	Δίδει τον κωδικό του υπ. αρ. κ σταδίου του υπολογισμού #υ.

Η λειτουργία της μηχανής μετατρέπεται τώρα σε ***αριθμητικά κατηγορήματα*** ή ***αριθμητικές σχέσεις*** που έχουν (ή δεν έχουν) διάφοροι κωδικοί μεταξύ τους. Η όλη ιδέα να αναζητήσουμε ένα αριθμοκωδικό #υ ο οποίος,

- κωδικοποιεί μια ακολουθία σταδίων,

- κάθε βήμα της οποίας εκτελείται σύμφωνα με μια εντολή του δεδομένου προγράμματος π,
- το αρχικό στάδιο της οποίας περιέχει τον κωδικό των δεδομένων #δ, θέτει την κεφαλή στην αρχή της ταινίας, και την κατάσταση της μηχανής στην αρχική κατάσταση Q_{START},
- και στο τελικό στάδιο της οποίας η κατάσταση της μηχανής είναι (κατά τις συμβάσεις μας) μία από τις δύο τερματικές καταστάσεις κάθε μηχανής: Q_{YES} ή Q_{NO}, με κωδικούς #Q_{YES} ή #Q_{NO}.

Κεντρικό ρόλο παίζει, (εύλογα), το κατηγόρημα «***παράγει(#ε, #σ, #σ′)***» (βλ. πίνακα) το οποίο δεν υλοποιεί παρά την εξαιρετικά απλή σημασιολογία των εντολών των μηχανών *Turing*. Να προσέξουμε εδώ ότι ο κωδικός μιας εντολής (Κ, α) → (Κ′, α′, δ) είναι ο κωδικός της 5άδας ⟨#Κ, #α, #Κ′, #α′, #δ⟩, δηλαδή περιέχει τους κωδικούς των Κ, α, Κ′, α′, δ στις θέσεις 0, 1, 2, 3, 4. Τα υπόλοιπα κατηγορήματα έχουν σχεδόν όλα γενική μορφή, και θα ήσαν περίπου τα ίδια σε οποιοδήποτε σύστημα υπολογισμού...!

Ο επόμενος πίνακας δίδει τα (6 μόνον) κατηγορήματα που περιγράφουν *αριθμητικά*, τί σημαίνει να υπάρχει ένα τερματιζόμενος υπολογισμός που ακολουθεί το πρόγραμμα π επί δεδομένων δ.

Κατηγορήματα	Ερμηνεία και αριθμητική έκφραση
αρχικό(#σ, #δ)	Ο αριθμοκωδικός #σ «είναι» αφετηριακό στάδιο υπολογισμού με ταινία t που φέρει τα δεδομένα #δ (και όσες κενές θέσεις χρειαστούν), με κεφαλή h στην πρώτη θέση της ταινίας (υπ. αρ. 0), και με κατάσταση Κ την αφετηριακή Q_{START}. $\exists\tau\, \exists\mu\, (\ \mu\acute{\eta}\kappa o\varsigma(\tau, \mu)$ $\wedge\ \forall i\, (\ ((i < \mu) \rightarrow (\theta\acute{\epsilon}\sigma\eta(\tau, i) = \theta\acute{\epsilon}\sigma\eta(\delta, i))) \wedge\ ((i \geq \mu) \rightarrow (\theta\acute{\epsilon}\sigma\eta(\tau, \upsilon) = \#\_\ )))\ )$ $\wedge\ (\theta\acute{\epsilon}\sigma\eta(\#\sigma, 0) = \tau) \wedge (\theta\acute{\epsilon}\sigma\eta(\#\sigma, 1) = 0) \wedge (\theta\acute{\epsilon}\sigma\eta(\#\sigma, 2) = \#Q_{START})\ )$
παράγει(#ε,#σ,#σ′)	Η εντολή με κωδικό #ε παράγει από το στάδιο #σ το στάδιο #σ′: $(\ \kappa\alpha\tau\acute{\alpha}\sigma\tau\alpha\sigma\eta(\#\sigma) = \boldsymbol{K(\#\varepsilon)})$ $\wedge\ (\ \theta\acute{\epsilon}\sigma\eta(\tau\alpha\iota\nu\acute{\iota}\alpha(\#\sigma), \kappa\epsilon\phi\alpha\lambda\acute{\eta}(\#\sigma)) = \boldsymbol{\alpha(\#\varepsilon)}\ )$ $\wedge\ (\ \kappa\alpha\tau\acute{\alpha}\sigma\tau\alpha\sigma\eta(\#\sigma') = \boldsymbol{K'(\#\varepsilon)}\ )$ $\wedge\ (\ \theta\acute{\epsilon}\sigma\eta(\tau\alpha\iota\nu\acute{\iota}\alpha(\#\sigma'), \kappa\epsilon\phi\alpha\lambda\acute{\eta}(\#\sigma)) = \boldsymbol{\alpha'(\#\varepsilon)}\ )$ $\wedge\ (\ \kappa\epsilon\phi\alpha\lambda\acute{\eta}(\#\sigma') = (\kappa\epsilon\phi\alpha\lambda\acute{\eta}(\#\sigma) + \boldsymbol{\delta(\#\varepsilon)}\ )$ $\wedge\ \forall\theta\ \ (\ (\theta \neq \kappa\epsilon\phi\alpha\lambda\acute{\eta}(\#\sigma)) \rightarrow (\ \theta\acute{\epsilon}\sigma\eta(\tau\alpha\iota\nu\acute{\iota}\alpha(\#\sigma), \theta) = \theta\acute{\epsilon}\sigma\eta(\tau\alpha\iota\nu\acute{\iota}\alpha(\#\sigma'), \theta)\ )\ )$ // *προσοχή στα κόκκινα... η ζωή είναι δύσκολη!*
βήμα(#σ, #σ′, #π)	Υπάρχει εντολή του προγράμματος #π που παράγει το στάδιο #σ′ από το #σ: $\exists\kappa\ (0 \leq \kappa < \mu\acute{\eta}\kappa o\varsigma(\#\pi)) \wedge \boldsymbol{\pi\alpha\rho\acute{\alpha}\gamma\epsilon\iota}(\varepsilon\nu\tau o\lambda\acute{\eta}(\#\pi, \kappa), \#\sigma, \#\sigma')$
ακολουθεί(#υ, #π)	Όλα τα βήματα του υπολογισμού #υ πραγματοποιούνται σύμφωνα με το #π: $\forall\kappa\ ((0 \leq \kappa < (\mu\acute{\eta}\kappa o\varsigma(\#\upsilon)-1)) \rightarrow \beta\acute{\eta}\mu\alpha(\sigma\tau\acute{\alpha}\delta\iota o(\#\upsilon, \kappa), \sigma\tau\acute{\alpha}\delta\iota o(\#\upsilon, \kappa+1), \#\pi))$.
τελικό(#σ)	Το σ είναι τερματικό στάδιο υπολογισμού: $((\kappa\alpha\tau\acute{\alpha}\sigma\tau\alpha\sigma\eta(\#\sigma) = \#Q_{YES}) \vee (\kappa\alpha\tau\acute{\alpha}\sigma\tau\alpha\sigma\eta((\#\sigma) = \#Q_{NO})$
Τερματίζει(#π, #δ)	Υπάρχει ένας τερματιζόμενος υπολογισμός υ που εκτελείται κατά το π επί των δ, (που θα είναι και ο μοναδικός, αφού περιμένουμε το π να είναι αιτιοκρατικό): $\boldsymbol{\exists(\#\upsilon)}\ (\ \alpha\rho\chi\iota\kappa\acute{o}(\ \sigma\tau\acute{\alpha}\delta\iota o(\#\upsilon, 0), \boldsymbol{\#\delta})$ $\wedge\ \alpha\kappa o\lambda o\upsilon\theta\epsilon\acute{\iota}(\ \#\upsilon, \boldsymbol{\#\pi}\ )$ $\wedge\ \tau\epsilon\lambda\iota\kappa\acute{o}(\ \sigma\tau\acute{\alpha}\delta\iota o(\#\upsilon, \mu\acute{\eta}\kappa o\varsigma(\#\upsilon)-1)\ \ )$

Υπάρχει λοιπόν *στη γλώσσα της αριθμητικής* μια *πρόταση* ***Τερματίζει(·,·)*** – συνοπτικά: ***T(***#π, #δ***)*** – η οποία με παράμετρο τον αριθμοκωδικό #π ενός προγράμματος, και τον αριθμοκωδικό #δ των αρχικών δεδομένων, είναι ΑΛΗΘΗΣ *εάν και μόνον εάν το αντίστοιχο πρόγραμμα π τερματίζει όταν εφαρμόζεται επί των δεδομένων δ*. Ιδού λοιπόν μια ευκαιρία να αποκτήσουμε μέθοδο επίλυσης για το πρόβλημα του τερματισμού: να αναζητούμε μια απόδειξη είτε της αριθμοπρότασης ***T(π, δ)***, είτε της άρνησης αυτής ¬***T(π, δ)***. Το αντίθετο...

Για κάθε πρόγραμμα π και για κάθε δεδομένα δ, το π(δ) είτε θα τερματίσει είτε όχι. Ας υποθέσουμε λοιπόν ότι για κάθε #π και κάθε #δ υπάρχει μια απόδειξη είτε της πρότασης ***T(π, δ)*** είτε της ¬***T(π, δ)***. Τότε θα μπορούσαμε να φτιάξουμε ένα πρόγραμμα μΤ που θα διαγίγνωσκε εάν «π(δ) ↓» ή «π(δ) ↑»: αρκεί να παραγάγουμε όλες τις δυνατές αποδείξεις μία-προς-μία και να τερματίσουμε μόλις εντοπίσουμε μία κατάλληλη – είτε αυτή που υπάρχει για την πρόταση ***T(π, δ)***, είτε αυτή για την πρόταση ¬***T(π, δ)***.

Έχουμε όμως ήδη βεβαιώσει ότι τέτοιο πρόγραμμα δεν υπάρχει. Πού αποτυγχάνει λοιπόν η ιδέα μας; Η μόνη εξήγηση είναι ότι υπάρχει ένα τουλάχιστον πρόγραμμα π και ένα σύνολο δεδομένων δ, για τα οποία *δεν υπάρχει καμμία απόδειξη* στο λογικό σύστημα της αριθμητικής μας, *ούτε μία για την πρόταση* ***T(π, δ)***, *ούτε μία για την άρνησή της* ¬***T(π, δ)***. Μία από αυτές τις δύο όμως *είναι οπωσδήποτε αληθής*! Έχουμε λοιπόν δώσει το περίγραμμα της απόδειξης του εξής:

Θεώρημα Gödel: (το λεγόμενο «της μη-πληρότητας της αριθμητικής»).
Ισχυρισμός: Υπάρχουν προτάσεις της αριθμητικής που είναι ΑΛΗΘΕΙΣ μεν, αλλά δεν είναι δυνατόν να αποδειχθούν ως τέτοιες από το υπάρχον σύστημα αξιωμάτων (αυτά του *Peano*).[11]

Πολλοί έχουν ισχυριστεί ότι το αποτέλεσμα αυτό είναι το πιο βαρυσήμαντο μαθηματικό θεώρημα του 20ου αιώνα – και ίσως όχι μόνον αυτού. Το ποιά επακόλουθα έχει αυτό το θέωρημα, και ποιά είναι η «ορθή» ερμηνεία του, είναι ένα στριφνό και πολύ βαθύ ζήτημα που παραμένει, (και θα συνεχίσει να παραμένει), στο προσκήνιο· το μόνον βέβαιο είναι ότι ξεπερνά κατά πολύ το πλαίσιο αυτών των σημειώσεων...

...και,
κατά ελαφρώς ειρωνικό τρόπο, οι σημειώσεις αυτές περί θεωρίας υπολογισμού, τερματίζουν[12] εκεί που *άρχισε* ιστορικά – περί τα μέσα της 10ετίας του 1920/30 – η όλη περιπέτεια: στην ανακάλυψη από τον *Gödel* των *δυνατοτήτων* μεν, αλλά και των *ορίων*, της λογικής. ■■■

[11] Ούτε και από κανένα άλλο, στο βαθμό που τα αξιώματά του θα επιδέχονταν μια *κατασκευάσιμη καταλογοποίηση* (σαν αυτές που είδαμε στη τρέχουσα ενότητα)· εδώ όμως παραλείπουμε αυτές τις σχετικές (και περισσότερο «τεχνικές») επεκτάσεις.

[12] Αυτό τουλάχιστον ήταν διαγνώσιμο...