

# Θεμελιώδη Θέματα Επιστήμης Υπολογιστών

## 2η ενότητα: Λογική, Μοντέλα Υπολογισμού, Κλάσεις Πολυπλοκότητας

Επιμέλεια: Στάθης Ζάχος – Άρης Παγουρτζής

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών  
Εθνικό Μετσόβιο Πολυτεχνείο

# Προτασιακός Λογισμός

- Boole, Frege.
- **Αλφάβητο**: σύμβολα προτασιακών μεταβλητών και λογικά σημάδια ζεύξης:  $\wedge$  (and),  $\vee$  (or),  $\neg$  (not),  $\rightarrow$  (implies),  $\leftrightarrow$  (equivalent), ...
- Ατομικοί τύποι: σταθερές TRUE και FALSE καθώς και τις προτασιακές μεταβλητές π.χ.  $x_1, x_2, \dots$
- Οι **προτασιακοί τύποι** ορίζονται **επαγωγικά**:
  - 1 Οι ατομικοί τύποι είναι τύποι.
  - 2 Αν  $\Phi$  είναι τύπος τότε και ο  $\neg\Phi$  είναι τύπος.
  - 3 Αν οι  $\Phi$  και  $\Psi$  είναι τύποι τότε και οι  $(\Phi \wedge \Psi)$  και  $(\Phi \vee \Psi)$  είναι τύποι.
  - 4 Ο,τιδήποτε δεν ορίζεται με βάση τα (1)–(3) δεν είναι προτασιακός τύπος.

## Συμβάσεις - Ορολογία

- Μερικές φορές παραλείπουμε παρενθέσεις και υποθέτουμε αριστερό προσηταιρισμό π.χ.  $x_1 \wedge x_2 \wedge \neg x_3$
- Μπορούμε να ορίσουμε νέους τύπους ως **συντομογραφία** άλλων γνωστών π.χ.:

$$(\Phi \rightarrow \Psi) ::= (\neg \Phi \vee \Psi)$$

$$(\Phi \leftrightarrow \Psi) ::= (\Phi \rightarrow \Psi) \wedge (\Psi \rightarrow \Phi)$$

- Μια προτασιακή μεταβλητή ή άρνηση προτασιακής μεταβλητής ονομάζεται **λέκτημα** (literal).
- Μια **φράση** (clause) είναι μια διάζευξη από λεκτήματα (π.χ.  $x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4$ ).

## Παράδειγμα

- $\Delta$ : γράφω διαγώνισμα τη Δευτέρα
- $\Pi$ : παίζει η ομάδα μου την Κυριακή “ντέρμπυ”
- $\Gamma$ : θα πάω γήπεδο την Κυριακή

$$(\neg \Delta \vee (\Delta \wedge \Pi)) \rightarrow \Gamma$$

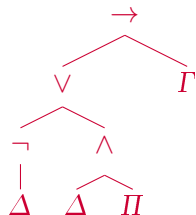
Τι περιγράφει ο παραπάνω τύπος;

## Παράδειγμα (συν.)

$$(\neg\Delta \vee (\Delta \wedge \Pi)) \rightarrow \Gamma$$

Αν δεν γράφω τη Δευτέρα, ή αν γράφω αλλά παίζει η ομάδα μου ντέρμπυ, θα πάω την Κυριακή στο γήπεδο.

Συντακτική δομή:



Απλοποιήσεις:

$$(\neg\Delta \vee (\Delta \wedge \Pi)) \rightarrow \Gamma \equiv (\neg\Delta \vee \Pi) \rightarrow \Gamma \equiv (\Delta \wedge \neg\Pi) \vee \Gamma$$

## Πίνακας Αληθείας (Truth Table)

Οι προτασιακοί τύποι είναι συντακτικές συμβολοσειρές που όμως έχουν κάποια σημασία (**σημασιολογία**) δηλαδή είναι αληθείς ή ψευδείς ανάλογα με τις αληθοτιμές που έχουν απονεμηθεί στις προτασιακές μεταβλητές.

Οι αληθοτιμές των τυπών  $\neg\Phi$ ,  $(\Phi \wedge \Psi)$  και  $(\Phi \vee \Psi)$  ορίζονται από τις αληθοτιμές των  $\Phi$ ,  $\Psi$  όπως φαίνεται στον παρακάτω πίνακα αληθείας (truth table):

$\Phi$	$\Psi$	$\neg\Phi$	$(\Phi \wedge \Psi)$	$(\Phi \vee \Psi)$
TRUE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE		FALSE	TRUE
FALSE	TRUE	TRUE	FALSE	TRUE
FALSE	FALSE		FALSE	FALSE

## Παράδειγμα

Η ελληνική φράση “το φτηνό το κρέας το τρώνε οι σκύλοι” σημαίνει “αν κάτι είναι φτηνό, τότε δεν είναι καλό”. Είναι άραγε αυτό ισοδύναμο με τη φράση “το καλό το πράγμα είναι απαραίτητα ακριβό”;

$\Phi$	$K$	$\Phi \rightarrow \neg K$	$K \rightarrow \neg \Phi$
TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	TRUE	TRUE
FALSE	TRUE	TRUE	TRUE
FALSE	FALSE	TRUE	TRUE

Η απάντηση στο παραπάνω ερώτημα είναι **ΝΑΙ!** Οι δύο φράσεις είναι ισοδύναμες.

Άσκηση: αποδείξτε την ιδιότητα της **αντιθετοαναστροφής** (ή αντιθετοαντιστροφής, contraposition):

$$A \rightarrow B \equiv \neg B \rightarrow \neg A$$

## Ταυτολογίες - Ικανοποιήσιμοι τύποι

Ένας τύπος λέγεται **έγκυρος** (valid) ή **ταυτολογία** αν είναι αληθής για κάθε απονομή αληθοτιμών στις μεταβλητές.

Ένας τύπος λέγεται **ικανοποιήσιμος** (satisfiable) αν υπάρχει απονομή αληθοτιμών που τον καθιστά αληθή.

Άρα  $\Phi$  είναι ικανοποιήσιμος **εάν και μόνο εάν** ο  $\neg\Phi$  δεν είναι ταυτολογία.



## Κανονικές μορφές (CNF, DNF)

Κάθε τύπος της προτασιακής λογικής είναι ισοδύναμος με κάποιον που βρίσκεται σε **συζευκτική κανονική μορφή** (conjunctive normal form) δηλαδή είναι μια σύζευξη από διαζευκτικές φράσεις.

Είναι επίσης ισοδύναμος με τύπο που βρίσκεται σε **διαζευκτική κανονική μορφή** (disjunctive normal form) δηλαδή είναι μια διάζευξη από συζευκτικές φράσεις.

## Φράσεις Horn

Μια φράση λέγεται **φράση Horn** αν έχει το πολύ ένα **θετικό** literal δηλαδή είναι της μορφής:

$$(x_0 \vee \neg x_1 \vee \neg x_2 \vee \dots \vee \neg x_n) \text{ ή } (x_0) \text{ ή } (\neg x_1 \vee \neg x_2 \vee \dots \vee \neg x_n)$$

που γράφεται ισοδύναμα:

$$(x_1 \wedge x_2 \wedge \dots \wedge x_n \rightarrow x_0), (\text{TRUE} \rightarrow x_0), (x_1 \wedge x_2 \wedge \dots \wedge x_n \rightarrow \text{FALSE}),$$

αντίστοιχα.

Αντιστοιχίες στη γλώσσα PROLOG:

$x_0$  : —  $x_1, x_2, \dots, x_n$       (Rule)

$x_0$       (Fact)

# Κατηγορηματικός Λογισμός

Η γλώσσα του κατηγορηματικού λογισμού (ή πρωτοβάθμιας λογικής) αποτελείται από:

- όλα τα σύμβολα που περιέχει ο προτασιακός λογισμός
- επιπλέον σύμβολα για συναρτήσεις, σταθερές, και μεταβλητές π.χ.  $f, g, h, c_1, c_2, \dots$ ,
- σύμβολα για κατηγορήματα π.χ.  $P, Q, =, \dots$
- και τους ποσοδείκτες: καθολικό  $\forall$  και υπαρξιακό  $\exists$ .

# Κατηγορηματικός Λογισμός: όροι και τύποι

Ορίζονται επαγωγικά:

Όροι:

- 1 Οι μεταβλητές και οι σταθερές είναι όροι.
- 2 Αν  $f$  είναι σύμβολο συνάρτησης  $n$  θέσεων και  $t_1, \dots, t_n$  είναι όροι τότε όρος είναι και ο  $f(t_1, \dots, t_n)$ .
- 3 Τίποτα άλλο.

Τύποι:

- 1 Αν  $P$  είναι σύμβολο κατηγορήματος  $n$  θέσεων και  $t_1, \dots, t_n$  είναι όροι τότε  $P(t_1, \dots, t_n)$  και  $t_1 = t_2$  είναι ατομικοί τύποι.
- 2 Αν οι  $\Phi$  και  $\Psi$  είναι τύποι και  $x$  μεταβλητή τότε τύποι είναι και οι:  $\neg\Phi$ ,  $(\Phi \vee \Psi)$ ,  $(\Phi \wedge \Psi)$ ,  $\forall x\Phi$ ,  $\exists x\Phi$ .
- 3 Τίποτα άλλο.

## Ελεύθερες και Δεσμευμένες Εμφάνισεις Μεταβλητών

Η **εμβέλεια** του  $\forall x$  (ή  $\exists x$ ) στον τύπο  $\forall x\Phi$  (ή αντίστοιχα  $\exists x\Phi$ ) είναι ο **υποτύπος**  $\Phi$ .

**Ελεύθερη εμφάνιση** της μεταβλητής  $x$  στον τύπο  $\Phi$  λέγεται μια εμφάνιση της μεταβλητής  $x$  που δεν είναι μέσα στην εμβέλεια ενός ποσοδείκτη  $\forall x$  ή  $\exists x$ .

**Δεσμευμένη εμφάνιση** της  $x$  είναι μέσα στην εμβέλεια ενός ποσοδείκτη ή και ακριβώς δεξιά του συμβόλου  $\forall$  (ή  $\exists$ ).

$$(2 \times x = 6) \wedge \exists x(x = x + 3)$$

Ένας τύπος λέγεται **κλειστός** αν δεν περιέχει ελεύθερες εμφανίσεις μεταβλητών.

# Σημασιολογία

Η **σημασιολογία** τύπων του κατηγορηματικού λογισμού δίνεται με την βοήθεια αλγεβρικών δομών  $A$  που ονομάζουμε **μοντέλα**.

Στην περίπτωση του προτασιακού λογισμού το πεδίο  $A$  είναι  $\{\text{True}, \text{False}\}$ , στον κατηγορηματικό λογισμό μπορεί να είναι οποιοδήποτε μή κενό, πεπερασμένο ή και άπειρο, **σύνολο**.

Όχι **απονομή** αληθοτιμών αλλά **ερμηνεία** (interpretation) των μεν σταθερών και μεταβλητών με στοιχεία του πεδίου  $A$ , των δε συναρτησιακών και κατηγορηματικών συμβόλων με πραγματικές **απεικονίσεις** και **σχέσεις** μεταξύ των στοιχείων του πεδίου  $A$ .

## Σημασιολογία (συν.)

Οι **σταθερές** και οι **μεταβλητές** ερμηνεύονται σαν στοιχεία ενός συνόλου  $A$ .

Τα **συναρτησιακά σύμβολα** ερμηνεύονται σαν συναρτήσεις:  $A^n \rightarrow A$ .

Έτσι κάθε όρος ερμηνεύεται σαν ένα στοιχείο του  $A$ .

Τα **κατηγορήματα** ερμηνεύονται σαν υποσύνολα του  $A^n$ .

Κάθε **όρος** ερμηνεύεται με στοιχείο του  $A$  και κάθε **κλειστός τύπος** αληθεύει (ή όχι) στο μοντέλο  $A$ .

## Σημασιολογία (συν.)

Η πρόταση  $P(t_1, t_2, \dots, t_n)$  είναι αληθής αν  $(s_1, s_2, \dots, s_n) \in R$  όπου  $s_1, s_2, \dots, s_n$  είναι τα στοιχεία του  $A$  με τα οποία ερμηνεύονται οι όροι  $t_1, t_2, \dots, t_n$  και  $R$  το υποσύνολο με το οποίο ερμηνεύεται το  $P$ .

Οι αληθοτιμές των  $\neg\Phi$ ,  $(\Phi \wedge \Psi)$  και  $(\Phi \vee \Psi)$  ορίζονται από τις αληθοτιμές των  $\Phi$  και  $\Psi$  όπως και στην προτασιακή λογική.

Η πρόταση  $\forall x\Phi$  είναι αληθής αν η πρόταση  $\Phi$  είναι αληθής για **οποιαδήποτε** ερμηνεία της μεταβλητής  $x$ , ενώ η πρόταση  $\exists x\Phi$  είναι αληθής αν η  $\Phi$  αληθεύει για **κάποια** ερμηνεία της  $x$ .



## Σημασιολογία: παράδειγμα

- Έστω το μοντέλο  $\langle \mathbb{N}; <, \text{succ}, 0 \rangle$ , όπου  $\mathbb{N}$  το σύνολο των φυσικών.  
Ένας κλειστός (συντακτικός) τύπος ερμηνεύεται στο συγκεκριμένο μοντέλο σαν κάτι που αληθεύει ή όχι.  
Για παράδειγμα, θεωρήστε τον τύπο:

$$\exists x ( L(x, S(Z)) \wedge (\forall y ( L(y, S(Z)) \rightarrow y = x ) ) )$$

Το σύμβολο σταθεράς  $Z$  ερμηνεύεται σαν το στοιχείο  $0 \in \mathbb{N}$ , το συναρτησιακό σύμβολο  $S$  ερμηνεύεται σαν η συνάρτηση επόμενου ( $\text{succ}$ ), και το κατηγορηματικό σύμβολο  $L$  ερμηνεύεται σαν η σχέση ' $<$ '.

Ο παραπάνω τύπος ερμηνεύεται σαν “υπάρχει μοναδικός φυσικός μικρότερος του 1” και αληθεύει.

- Ο ίδιος τύπος ερμηνεύεται σαν “υπάρχει μοναδικός φυσικός μικρότερος ή ίσος του 1” στο μοντέλο  $\langle \mathbb{N}; \leq, \text{succ}, 0 \rangle$  και δεν αληθεύει.

## Φράσεις Horn

Οι φράσεις Horn για τον κατηγορηματικό λογισμό ορίζονται όπως και στην προτασιακή λογική αν αντί για προτασιακές μεταβλητές χρησιμοποιούμε ατομικές προτάσεις.

Ένα πρόγραμμα **Prolog** είναι βασικά μία σύζευξη από φράσεις Horn.

## Θεώρημα Πληρότητας

Συμβολίζουμε  $\Gamma \vdash \Phi$  το γεγονός ότι ο τύπος  $\Phi$  αποδεικνύεται **συντακτικά** από τους τύπους του συνόλου  $\Gamma$ .

Συμβολίζουμε  $\Gamma \models \Phi$  το γεγονός ότι ο τύπος  $\Phi$  αληθεύει σε όλα τα μοντέλα όπου αληθεύουν και οι τύποι του συνόλου  $\Gamma$ .

Το περίφημο **θεώρημα πληρότητας** του Gödel λέει:

$$\Gamma \vdash \Phi \quad \text{ανν} \quad \Gamma \models \Phi$$

# Θεώρημα μη Πληρότητας

Αφ' ετέρου το **θεώρημα μη πληρότητας** του Gödel λέει:

Δεν μπορεί να υπάρξει **συνεπής και πλήρης αξιωματικοποίηση** όλων των αληθών τύπων της Αριθμητικής.

## Μοντέλα Υπολογισμού: μηχανές Turing

Μια μηχανή Turing (TM) είναι ένα απλός ιδεατός υπολογιστής, δηλαδή ένα υπολογιστικό μοντέλο.

Η TM έχει ένα πεπερασμένο αριθμό εσωτερικών καταστάσεων (internal states):

$$Q = \{q_0, q_1, \dots\}$$

Διαθέτει ταινία που προεκτείνεται (δυννητικά) μέχρι το άπειρο και προς τις δύο κατευθύνσεις και υποδιαιρείται σε κύτταρα που το καθένα περιέχει 1 ή 0, δηλαδή το αλφάβητο της μηχανής είναι το  $\Sigma = \{0, 1\}$ .

Σε κάθε χρονική στιγμή η κεφαλή της TM βρίσκεται σε ένα κύτταρο, το οποίο θα λέγεται το τρέχον.

## Μηχανές Turing: βασικές λειτουργίες

- Άλλαξε την εσωτερική κατάσταση
- Διάβασε το περιεχόμενο του τρέχοντος κυττάρου
- Γράψε 1 ή 0 στο τρέχον κύτταρο
- Κάνε τρέχον το αμέσως αριστερότερο ή το αμέσως δεξιότερο κύτταρο

# Πρόγραμμα μηχανής Turing

Ένα πρόγραμμα για μια TM είναι ένα σύνολο από τετράδες της μορφής  $\langle q_i, e, d, q_j \rangle$  όπου:

$$q_i, q_j \in Q, \quad e \in \Sigma, \quad d \in A = \Sigma \cup \{L, R\}$$

με τον εξής συναρτησιακό (ντετερμινιστικό) περιορισμό:

Για κάθε  $\langle q_i, e \rangle$  υπάρχει το πολύ ένα  $\langle d, q_j \rangle$  έτσι ώστε η τετράδα  $\langle q_i, e, d, q_j \rangle$  να ανήκει στο πρόγραμμα, δηλαδή πρόκειται για μια συνάρτηση μετάβασης (transition function)  $\delta : Q \times \Sigma \rightarrow A \times Q$ .

## Πρόγραμμα μηχανής Turing (συν.)

Κατά σύμβαση η μηχανή σταματάει στο ζεύγος κατάστασης-συμβόλου  $\langle q_i, e \rangle$  αν η τιμή  $\delta(q_i, e)$  δεν είναι ορισμένη.

$\Sigma = \{0, 1\}$  (Εναδική αναπαράσταση αριθμού):

Σε κάθε TM μπορούμε να αντιστοιχήσουμε μια μερική συνάρτηση από το  $\mathbb{N}$  στο  $\mathbb{N}$ . Η είσοδος  $n \in \mathbb{N}$  παριστάνεται με  $n + 1$  συνεχόμενα 1 (έτσι ο αριθμός 0 παριστάνεται με 1).

Σαν αρχικό στιγμιότυπο έχουμε την κεφαλή (τρέχον κύτταρο) να δείχνει στο αριστερότερο 1 και να βρίσκεται στην κατάσταση  $q_0$ . Σαν έξοδο λαμβάνουμε το συνολικό αριθμό από 1 που βρίσκεται στην ταινία, όταν και εάν η μηχανή σταματήσει.



# Πρόγραμμα μηχανής Turing: παράδειγμα

Κατασκευή TM που υπολογίζει το  $2 \cdot x$ :

αρχικοποίηση; (\*διαγραφή του πρώτου 1\*)

while είσοδος  $\neq 0$  do

begin

διάγραψε ένα 1 από είσοδο

μετακίνησε κεφαλή δεξιά πέρα από είσοδο και έξοδο

πρόσθεσε δύο 1 στην έξοδο

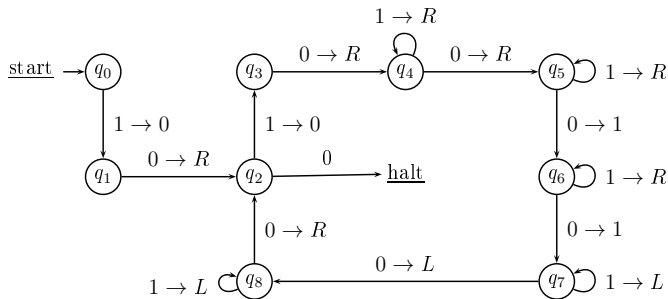
μετακίνησε κεφαλή αριστερά πέρα από έξοδο και είσοδο

end

## Πρόγραμμα μηχανής Turing: παράδειγμα

$\langle q_0$	1	0	$q_1 \rangle$	
$\langle q_1$	0	R	$q_2 \rangle$	
$\langle q_2$	1	0	$q_3 \rangle$	halt για $\langle q_2$ 0 $\rangle$
$\langle q_3$	0	R	$q_4 \rangle$	
$\langle q_4$	1	R	$q_4 \rangle$	
$\langle q_4$	0	R	$q_5 \rangle$	
$\langle q_5$	1	R	$q_5 \rangle$	
$\langle q_5$	0	1	$q_6 \rangle$	
$\langle q_6$	1	R	$q_6 \rangle$	
$\langle q_6$	0	1	$q_7 \rangle$	
$\langle q_7$	1	L	$q_7 \rangle$	
$\langle q_7$	0	L	$q_8 \rangle$	
$\langle q_8$	1	L	$q_8 \rangle$	
$\langle q_8$	0	R	$q_2 \rangle$	

# Πρόγραμμα μηχανής Turing: παράδειγμα



	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$	$q_7$	$q_8$
0		R/ $q_2$	halt	R/ $q_4$	R/ $q_5$	1/ $q_6$	1/ $q_7$	L/ $q_8$	R/ $q_2$
1	0/ $q_1$		0/ $q_3$		R/ $q_4$	R/ $q_5$	R/ $q_6$	L/ $q_7$	L/ $q_8$

## Πρόγραμμα μηχανής Turing: 2ο παράδειγμα

Μηχανή με αλφάβητο  $\Sigma = \{0, 1, \sqcup\}$  (δυναμική αναπαράσταση αριθμού) που υπολογίζει τη συνάρτηση  $x \mapsto x + 1$ .

Περιγραφή **υψηλού επιπέδου**:

Κάνε τρέχον το κύτταρο με το τελευταίο σύμβολο της εισόδου  $x$ ;  
repeat

    Αν τρέχον κύτταρο =  $\sqcup$ , γράψε 1 και σταμάτησε;

    Αν τρέχον κύτταρο =  $1$ , γράψε 0, κάνε τρέχον το αμέσως αριστερότερο κύτταρο, και κρατούμενο := 1;

    Αν τρέχον κύτταρο =  $0$ , γράψε 1, κάνε τρέχον το αμέσως αριστερότερο κύτταρο, και κρατούμενο := 0;

until κρατούμενο = 0;

Κάνε τρέχον το κύτταρο με το πρώτο σύμβολο του  $x + 1$  και σταμάτησε.

## Πρόγραμμα μηχανής Turing: 2ο παράδειγμα (συν.)

Συνάρτηση μετάβασης:

	0	1	$\sqcup$
$q_0$	$(q_0, 0, R)$	$(q_0, 1, R)$	$(q_1, \sqcup, L)$
$q_1$	$(q_2, 1, L)$	$(q_1, 0, L)$	$(\text{HALT}, 1, S)$
$q_2$	$(q_2, 0, L)$	$(q_2, 1, L)$	$(\text{HALT}, \sqcup, R)$

Σημείωση: Για συντομία επιτρέπουμε ταυτόχρονα εγγραφή συμβόλου και κίνηση κεφαλής (το μοντέλο είναι ισοδύναμο). Για παράδειγμα, η παραπάνω συνάρτηση καθορίζει ότι αν η μηχανή βρίσκεται στην κατάσταση  $q_1$  και το τρέχον σύμβολο είναι '1' τότε η μηχανή παραμένει στην  $q_1$ , το τρέχον σύμβολο γίνεται '0' και η κεφαλή μετακινείται αριστερά (κατά ένα κύτταρο).

## Πρόγραμμα μηχανής Turing: 2ο παράδειγμα (συν.)

Εκτέλεση με είσοδο **1011**:

$$\begin{array}{ccccccc} (q_0, \underline{1}011) & \vdash & (q_0, 1\underline{0}11) & \vdash & (q_0, 10\underline{1}1) & \vdash & (q_0, 101\underline{1}) \vdash \\ (q_0, 1011\underline{\phantom{0}}) & \vdash & (q_1, 101\underline{1}) & \vdash & (q_1, 10\underline{1}0) & \vdash & (q_1, 1\underline{0}00) \vdash \\ (q_2, \underline{1}100) & \vdash & (q_2, \underline{\phantom{1}}1100) & \vdash & (\text{HALT}, \underline{1}100) & & \end{array}$$

Επεξηγήσεις: σε κάθε βήμα δίνεται μια περιγραφή της στιγμιαίας **συνολικής κατάστασης (configuration)** της TM, που αποτελείται από την τρέχουσα κατάσταση της μηχανής ακολουθούμενη από το περιεχόμενο της ταινίας. Το σύμβολο  $\vdash$  λέγεται μπάρα (turnstile) και συμβολίζει τη μετάβαση από μία συνολική κατάσταση στην επόμενη.

Το υπογραμμισμένο σύμβολο δηλώνει το τρέχον κύτταρο, ενώ τα κενά αριστερά και δεξιά του περιεχομένου δεν αναγράφονται εκτός αν το τρέχον κύτταρο περιέχει το κενό.

# Μηχανή τυχαίας προσπέλασης (Random Access Machine – RAM)

Κώδικας εντολής	Διεύθυνση
1. LOAD	operand
2. STORE	operand
3. ADD	operand
4. SUB	operand
5. MULT	operand
6. DIV	operand
7. READ	operand
8. WRITE	operand
9. JUMP	operand
10. JGTZ	label
11. JZERO	label
12. HALT	label

# Τυπικό Πρόγραμμα RAM

Πρόγραμμα RAM	Αντίστοιχη ψευδογλώσσα
READ 1	read r1
LOAD 1	
JGTZ pos	if r1 $\leq$ 0 then write 0
WRITE =0	
JUMP endelse	
pos: LOAD 1	else begin
STORE 2	r2 $\leftarrow$ r1
LOAD 1	
SUB =1	
STORE 3	r3 $\leftarrow$ r1 - 1
while: LOAD 3	
JGTZ continue	while r3 $>$ 0 do
JUMP endwhile	
continue: LOAD 2	begin
MULT 1	
STORE 2	r2 $\leftarrow$ r2 * r1
LOAD 3	
SUB =1	
STORE 3	r3 $\leftarrow$ r3 - 1
JUMP while	
endwhile: WRITE 2	write r2
endelse: HALT	



## Ιστορικό Υπολογισιμότητας και Πολυπλοκότητας

Η Συλλογιστική του Αριστοτέλη αποτέλεσε την πρώτη προσπάθεια θεμελίωσης της λογικής και των μαθηματικών. Ο Leibni(t)z πρότεινε το εξής πρόγραμμα:

- 1 Να δημιουργηθεί μια τυπική γλώσσα (formal language), με την οποία να μπορούμε να περιγράψουμε όλες τις μαθηματικές έννοιες και προτάσεις.
- 2 Να δημιουργηθεί μια μαθηματική θεωρία (δηλαδή ένα σύνολο από αξιώματα και συμπερασματικούς κανόνες συνεπαγωγής), με την οποία να μπορούμε να αποδεικνύουμε όλες τις ορθές μαθηματικές προτάσεις.
- 3 Να αποδειχθεί ότι αυτή η θεωρία είναι συνεπής (consistent), (δηλαδή ότι η πρόταση “A και όχι A” ( $A \wedge \neg A$ ) δεν είναι δυνατόν να αποδειχθεί σ’ αυτή τη θεωρία).

## Ιστορικό Υπολογισιμότητας και Πολυπλοκότητας (συν.)

Η πραγμάτωση αυτού του προγράμματος άρχισε πολύ αργότερα, προς το τέλος του 19ου αιώνα. Πολλοί επιστήμονες ασχολήθηκαν με τον ορισμό της ενιαίας γλώσσας της μαθηματικής (ή συμβολικής) λογικής (Boole, Frege, κ.α). Άλλοι ασχολήθηκαν με τον ορισμό της ενιαίας θεωρίας των συνόλων (Cantor, κ.α.) και άλλοι με την παραγωγή (derivation) όλων των αληθών μαθηματικών προτάσεων με χρήση της Συνολοθεωρίας (Russel, Whitehead, κ.α.).

Στην αρχή του περασμένου αιώνα ο Hilbert βάλθηκε να πραγματοποιήσει το 3ο μέρος του προγράμματος του Leibni(t)z, δηλαδή να βρει έναν αλγόριθμο που να αποκρίνεται (decides) για την ορθότητα κάθε μαθηματικής πρότασης.

## Ιστορικό Υπολογισιμότητας και Πολυπλοκότητας (συν.)

Τελικά, όμως, το 1931 ο Gödel απέδειξε ότι:


- Δεν υπάρχει τέτοιος αλγόριθμος.
- Είναι αδύνατον να αποδειχθεί η συνέπεια της Συνολοθεωρίας.
- Επιπλέον, οποιαδήποτε (δηλαδή όχι μόνο η Συνολοθεωρία) αξιωματική θεωρία των Μαθηματικών, που περιλαμβάνει τουλάχιστον την Αριθμοθεωρία, θα περιλαμβάνει και μη αποκρίσιμες (undecidable) προτάσεις.
- Κωδικοποιώντας προτάσεις με φυσικούς αριθμούς (αυτή η κωδικοποίηση λέγεται σήμερα “Γκεντελοποίηση”: Gödelization) μπόρεσε να παρουσιάσει μια συγκεκριμένη πρόταση που είναι μη αποκρίσιμη.

## Ιστορικό Υπολογισιμότητας και Πολυπλοκότητας (συν.)

Το αποτέλεσμα αυτό του Gödel ήταν η αιτία μιας σημαντικής κρίσης στα κλασσικά μαθηματικά, μα συγχρόνως και η απαρχή των μοντέρνων δυναμικών μαθηματικών. Το κεντρικό ερώτημα δεν είναι πια απλά αν μια πρόταση είναι αληθής η ψευδής, αλλά αν είναι “αποκρίσιμη ή μη αποκρίσιμη”, δηλαδή αν είναι “υπολογίσιμη (computable) ή όχι”. Αυτό ακριβώς είναι και το αντικείμενο της **Θεωρίας της Υπολογισιμότητας (computability)**.

Αν δοθεί ότι μια συνάρτηση  $f$  είναι υπολογίσιμη<sup>1</sup>, ποιο είναι το κόστος ή τα αγαθά (resources) που χρειάζονται για να υπολογίσουμε την  $f$ ; Αυτό είναι το βασικό ερώτημα της **Θεωρίας της Πολυπλοκότητας (complexity)**.

---

<sup>1</sup>Χρησιμοποιείται και ο όρος υπολογιστός αντί για υπολογίσιμος. 

## Ιστορικό Υπολογισιμότητας και Πολυπλοκότητας (συν.)

Διάφοροι επιστήμονες (Turing, Church, Kleene, Post, Markov, κ.α.) βάλθηκαν να ξεκαθαρίσουν τις έννοιες: υπολογίσιμο ή επιλύσιμο (solvable) με αλγόριθμο, υπολογίσιμη συνάρτηση και αποκρίσιμο πρόβλημα. Κατέληξαν, λοιπόν, σε διαφορετικά υπολογιστικά μοντέλα, τα οποία όμως αποδείχθηκαν όλα ισοδύναμα μεταξύ τους.

Η περίφημη **Θέση (thesis) των Church-Turing** λέει λοιπόν απλουστευμένα:

“Όλα τα γνωστά και τα ‘άγνωστα’ μοντέλα της έννοιας ‘υπολογίσιμος’ είναι μηχανιστικά ισοδύναμα (effectively equivalent)”.

Δηλαδή δοθέντος ενός αλγορίθμου σε ένα μοντέλο για μια συγκεκριμένη συνάρτηση  $f$ , μπορούμε μηχανιστικά (με τη βοήθεια μηχανής) να κατασκευάσουμε αλγόριθμο σε ένα άλλο μοντέλο για την ίδια συνάρτηση  $f$ .

## Μη Υπολογισιμότητα

- Υπάρχουν άπειρα μεν, αλλά μόνο αριθμήσιμα (countable) διαφορετικά προγράμματα. Εκτός αυτού μπορούμε χρησιμοποιώντας κωδικοποίηση να τα απαριθμήσουμε μηχανιστικά (effectively enumerate).
- Απόδειξη: Κάθε πρόγραμμα μιας γλώσσας προγραμματισμού είναι στοιχείο του  $\Sigma^*$ , όπου  $\Sigma = \{a_1, a_2, \dots, a_m\}$  το αλφάβητο της γλώσσας. Το  $\Sigma^*$  όμως αποτελεί την ένωση  $\bigcup_{n=0}^{\infty} \Sigma_n$ , όπου  $\Sigma_n$  το σύνολο των συμβολοσειρών του αλφάβητου  $\Sigma$  που έχουν μήκος  $n$ . Κάθε σύνολο  $\Sigma_n$  είναι πεπερασμένο και έτσι αν διατάξουμε τα στοιχεία του αλφαβητικά μπορούμε να θεωρήσουμε την ακόλουθη αρίθμηση για το  $\Sigma^*$ :

$$\Sigma_0 : \{\varepsilon\}$$

$$\Sigma_1 : \{a_1, a_2, \dots, a_m\}$$

$$\Sigma_2 : \{a_1 a_1, a_1 a_2, \dots, a_1 a_m, \dots, a_m a_m\}$$

⋮

## Μη Υπολογισιμότητα (συν.)

- Από την άλλη μεριά όμως, ξέρουμε ότι υπάρχουν μη αριθμήσιμες άπειρες (uncountable) διαφορετικές συναρτήσεις. Αυτό αποδεικνύεται με διαγωνιοποίηση (diagonalization), ανάλογη με αυτή που χρησιμοποιούμε για να δείξουμε ότι το σύνολο  $\mathbb{R}$  είναι μη αριθμήσιμο.
- Απόδειξη: Ας θεωρήσουμε το σύνολο των ολικών συναρτήσεων  $\varphi: \mathbb{N} \rightarrow \mathbb{N}$  και έστω  $\varphi_0, \varphi_1, \varphi_2, \dots$  μια αρίθμηση τους (ολικές ονομάζονται οι συναρτήσεις που ορίζονται για κάθε  $x \in \mathbb{N}$ ). Ορίζουμε μια συνάρτηση  $f$  ως εξής:  $f(x) = \varphi_x(x) + 1, \forall x \in \mathbb{N}$ . Η  $f$  είναι προφανώς ολική συνάρτηση και επομένως θα αντιστοιχίζεται σε κάποιο δείκτη  $y$  στην παραπάνω αρίθμηση μας, δηλαδή  $f = \varphi_y$ . Τότε όμως θα ισχύει ότι  $\varphi_y(y) = f(y) = \varphi_y(y) + 1$  που είναι άτοπο. Επομένως το σύνολο των ολικών συναρτήσεων δεν είναι αριθμήσιμο.

# Μη Υπολογισιμότητα (συν.)

## Θεώρημα

Το **halting problem** (HP) είναι μη αποκρίσιμο.

## Απόδειξη

Έστω ότι  $\pi_0, \pi_1, \pi_2, \dots$  είναι μια μηχανιστική απαρίθμηση (effective enumeration) όλων των προγραμμάτων. Ας υποθέσουμε ότι το **HP** είναι επιλύσιμο. Τότε κατασκευάζουμε το εξής πρόγραμμα  $\pi$ :

```
 $\pi$ : read( $n$ ); if  $\pi_n(n)$  terminates then loop_forever else halt
```

Φυσικά αυτό το πρόγραμμα  $\pi$  κάπου θα εμφανίζεται στην παραπάνω αρίθμηση. Ας πούμε ότι ο δείκτης για το  $\pi$  είναι  $i$ , δηλαδή  $\pi = \pi_i$ . Η ιδέα της διαγωνιοποίησης είναι να δώσουμε το δείκτη  $i$  για input στο  $\pi_i$ . Τότε το  $\pi_i(i)$  σταματάει αν και μόνο αν το  $\pi(i)$  σταματάει και αυτό συμβαίνει αν και μόνο αν το  $\pi_i(i)$  δεν σταματάει. Αντίφαση. □



## Αποκρίσιμα και Καταγράψιμα Σύνολα

- Ένα σύνολο  $S$  λέγεται **αποκρίσιμο ή υπολογίσιμο ή επιλύσιμο** (decidable, computable, solvable) αν και μόνο αν υπάρχει ένας αλγόριθμος που σταματάει ή μια υπολογιστική μηχανή που δίνει έξοδο “ναι” για κάθε είσοδο  $a \in S$  και έξοδο “όχι” για κάθε είσοδο  $a \notin S$ .
- Ένα σύνολο  $S$  λέγεται **καταγράψιμο** (με μηχανιστική γεννήτρια) (listable, effectively generatable) αν και μόνο αν υπάρχει μια γεννήτρια διαδικασία ή μηχανή που καταγράφει όλα τα στοιχεία του  $S$ . Στην, πιθανώς άπειρη, λίστα εξόδου επιτρέπονται οι επαναλήψεις και δεν υπάρχει περιορισμός για την διάταξη των στοιχείων.

# Αποκρίσιμα και Καταγράψιμα Σύνολα: Ιδιότητες

## Theorem

- Αν το  $S$  είναι αποκρίσιμο τότε και το  $\bar{S}$  είναι αποκρίσιμο.
- Αν το  $S$  είναι αποκρίσιμο τότε το  $S$  είναι και καταγράψιμο.
- Αν το  $S$  και το  $\bar{S}$  είναι καταγράψιμα τότε το  $S$  είναι αποκρίσιμο.
- Αν το  $S$  είναι καταγράψιμο με γνησίως αύξουσα διάταξη τότε το  $S$  είναι αποκρίσιμο.

## Υπολογιστικά μοντέλα

- προγράμματα Pascal
- προγράμματα Pascal χωρίς αναδρομή (αφαίρεση αναδρομής με χρήση στοίβας)
- προγράμματα Pascal χωρίς αναδρομή και χωρίς άλλους τύπους δεδομένων εκτός από τους φυσικούς αριθμούς (επιτυγχάνεται με κωδικοποιήσεις)
- προγράμματα WHILE (μόνη δομή ελέγχου το WHILE)
- προγράμματα GOTO και IF
- Assembler-like RAM (random access machine), URM (universal register machine)
- SRM (single register machine) ένας καταχωρητής
- Μηχανή Turing (πρόσβαση μόνο σε ένα κύτταρο "cell" της ταινίας κάθε φορά)

## Υπολογιστικά μοντέλα (συν.)

Τα χαρακτηριστικά των παραπάνω μοντέλων είναι:

- ντετερμινιστική πολυπλοκότητα σε διακριτά βήματα
- πεπερασμένο σύνολο εντολών που εκτελούνται από επεξεργαστή
- απεριόριστη μνήμη

## Υπολογιστικά μοντέλα (συν.)

Άλλα μοντέλα είναι:

- παραλλαγές από μηχανές Turing
- Thue: κανόνες επανεγγραφής (re-writing rules)
- Post: κανονικά συστήματα (normal systems)
- Church: λογισμός  $\lambda$  ( $\lambda$ -calculus)
- Curry: συνδυαστική λογική (combinatory logic)
- Markov: Μ. αλγόριθμοι
- Kleene: γενικά αναδρομικά σχήματα (general recursive schemes)
- Shepherdson-Sturgis, Elgott: URM, SRM, RAM, RASP
- Σχήματα McCarthy (if ... then ... else ...  $\Rightarrow$  LISP)

## Θεώρημα

f είναι TM υπολογίσιμη ανν

- f είναι WHILE-υπολογίσιμη
- f είναι GOTO-υπολογίσιμη
- f είναι PASCAL-υπολογίσιμη
- f είναι μερικά αναδρομική (partial recursive)

## Παραλλαγές μηχανής Turing

Παραλλαγές Μηχανών Turing που έχουν την ίδια υπολογιστική δυνατότητα, όχι όμως και αποδοτικότητα (efficiency) είναι:

- πολλές ταινίες, μνήμη πλέγματος (grid memory), μνήμη περισσότερων διαστάσεων
- μεγαλύτερο  $\Sigma$
- πολλές παράλληλες κεφαλές
- μη ντετερμινιστικές μεταβάσεις
- μίας κατευθύνσεως, απείρου μήκους ταινία
- εγγραφή και κίνηση της κεφαλής σε κάθε βήμα

## Υπολογιστική Πολυπλοκότητα

Μια άλλη (πιο μοντέρνα) ταξινόμηση προβλημάτων (γλωσσών, συνόλων) σε κλάσεις μπορεί να γίνει με κριτήριο την **ποσότητα** των αγαθών (χρόνος, χώρος, επεξεργαστές, κ.ο.κ.) που χρειάζεται ένας βέλτιστος αλγόριθμος για να τα επιλύσει (αναγνωρίσει).



## Υπολογιστική Πολυπλοκότητα (συν.)

Συνήθως μετράμε το κόστος της χειρότερης περίπτωσης για είσοδο μεγέθους  $n$ . Έτσι το κόστος του αλγορίθμου  $A(n)$  είναι το  $\max$  του κόστους του αλγορίθμου  $A$  από όλες τις δυνατές εισόδους μεγέθους  $n$ .

Το κόστος  $C(n)$  ενός προβλήματος  $\pi(n)$  είναι το  $\min(A(n))$  από όλους τους αλγορίθμους  $A$  που λύνουν το πρόβλημα  $\pi$ . Συνεπώς για να προσδιορίσουμε το κόστος  $C(n)$  ενός προβλήματος  $\pi(n)$  χρειαζόμαστε ένα **άνω όριο** (upper bound) δηλαδή έναν αλγόριθμο  $A$  που έχει κόστος  $C(n)$  αλλά και ένα **κάτω όριο** (lower bound), δηλαδή μια απόδειξη ότι το καλύτερο δυνατό κόστος με το τρέχον μοντέλο είναι  $C(n)$ . Έτσι, π.χ., η χρονική πολυπλοκότητα ταξινόμησης με συγκρίσεις (όπου μοντέλο είναι πρόγραμμα Pascal, και μετράμε τον αριθμό συγκρίσεων) είναι  $\Theta(n \log n)$ .

## Κλάσεις Πολυπλοκότητας

**P** λέγεται η κλάση των προβλημάτων που λύνονται με ντετερμινιστικό αλγόριθμο σε χρόνο πολυωνυμικό.

**NP** λέγεται η κλάση των προβλημάτων που λύνονται με μη ντετερμινιστικό αλγόριθμο σε χρόνο πολυωνυμικό.

Λέμε ότι ένας μη ντετερμινιστικός αλγόριθμος  $A$  λύνει ένα πρόβλημα  $\pi$  εάν υπάρχει τουλάχιστον μια από τις δυνατές εκτελέσεις του  $A$  που λύνει το  $\pi$ .

Προφανώς ισχύει  $P \subseteq NP$  αλλά εδώ και 40 σχεδόν χρόνια παραμένει άλυτο το πρόβλημα “ $P \neq NP$ ;”.  $NP$ —πλήρη λέγονται τα δυσκολότερα προβλήματα της κλάσης  $NP$ . Αν πράγματι ισχύει  $P \neq NP$ , τότε για τα  $NP$ —πλήρη προβλήματα δεν υπάρχει αλγόριθμος πολυωνυμικού χρόνου.

## Κλάσεις Πολυπλοκότητας (συν.)

**PSPACE** λέγεται η κλάση των προβλημάτων που λύνονται με (ντετερμινιστικό ή μη ντετερμινιστικό αλγόριθμο) σε πολυωνυμικό χώρο (μνήμη).

**NC** λέγεται η κλάση των προβλημάτων που λύνονται με αλγόριθμο που χρησιμοποιεί πολυλογαριθμικό χρόνο ( $\log^{O(1)} n$ ) και πολυωνυμικό αριθμό επεξεργαστών.

## Κλάσεις Πολυπλοκότητας (συν.)

Μερικά γνωστά προβλήματα σε αυτές τις κλάσεις:

- Στην κλάση NP: το πρόβλημα SAT ικανοποιησιμότητας τύπων της προτασιακής λογικής, το πρόβλημα (TSP) του πλανόδιου πωλητή, κ.τ.λ.
- Στην κλάση PSPACE: το πρόβλημα QBF αποτίμησης τύπων της κατηγορηματικής (boolean) λογικής, το πρόβλημα στρατηγικής σε διάφορα παιχνίδια, κ.τ.λ.
- Στην κλάση NC: το πρόβλημα GAP πρόσβασης (δηλαδή ύπαρξης μονοπατιού) σε ένα γράφο G μεταξύ δυο κόμβων.
- Το πρόβλημα ικανοποιησιμότητας τύπων της κατηγορηματικής λογικής είναι μη επιλύσιμο αλλά καταγράψιμο.

## Θέση Σειριακών Υπολογισμών

Όλα τα λογικά υπολογιστικά μοντέλα είναι πολυωνυμικά συσχετισμένα ως προς την αποδοτικότητά τους

## Θέση Παράλληλων Υπολογισμών

Ο **χρόνος** που απαιτείται σε παράλληλο υπολογισμό είναι πολυωνυμικά συσχετισμένος με τον **χώρο** που απαιτείται σε σειριακό υπολογισμό

# Ιεραρχία Κλάσεων Πολυπλοκότητας

$$\text{NC} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subsetneq \text{REC} \subsetneq \text{R.E.}$$

**REC** = recursive = decidable

**R.E.** = recursively enumerable = listable

# Ιεραρχία Κλάσεων Πολυπλοκότητας

