Εισαγωγή στην Επιστήμη των Υπολογιστών

4ο εξάμηνο Σ.Η.Μ.Μ.Υ. & Σ.Ε.Μ.Φ.Ε.

http://www.corelab.ece.ntua.gr/courses/

4η ενότητα: Λογική, Μοντέλα, Υπολογισιμότητα, Πολυπλοκότητα

Στάθης Ζάχος, Άρης Παγουρτζής

Επιμέλεια: Πάνος Χείλαρης, Βαγγέλης Μπαμπάς, Γεωργία Καούρη

Προτασιακός Λογισμός

Στον προτασιακό λογισμό ονομάζουμε ατομικούς τύπους τις σταθερές TRUE και FALSE καθώς και τις προτασιακές μεταβλητές π.χ. $x_1, x_2, \ldots$

Οι **προτασιακοί τύποι** ορίζονται **επαγωγικά**:

1. Οι ατομικοί τύποι είναι τύποι.
2. Αν Φ είναι τύπος τότε και ο $\neg\Phi$ είναι τύπος.
3. Αν οι Φ και Ψ 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.

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

$$(\Phi \rightarrow \Psi) :\equiv (\neg\Phi \vee \Psi)$$
$$(\Phi \leftrightarrow \Psi) :\equiv (\Phi \rightarrow \Psi) \wedge (\Psi \rightarrow \Phi)$$

Πίνακες Αληθείας

Οι προτασιακοί τύποι είναι συντακτικές συμβολοσειρές που όμως έχουν κάποια σημασία (**σημασιολογία**) δηλαδή είναι αληθείς ή ψευδείς ανάλογα με τις αληθοτιμές που έχουν αποδοθεί στις προτασιακές μεταβλητές. Πιο συγκεκριμένα αληθοτιμές των τυπών $\neg\Phi$, $(\Phi \wedge \Psi)$ και $(\Phi \vee \Psi)$ ορίζονται από τις αληθοτιμές των Φ, Ψ όπως φαίνεται στον παρακάτω πίνακα αληθείας (truth table):

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

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

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

CNF, DNF, Horn

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

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

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

που γράφεται ισοδύναμα:

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

αντίστοιχα.

Όροι

Για να ορίσουμε συνταχτικά τις προτάσεις είναι αναγκαίο πρώτα να ορίσουμε τους **όρους** (οι οποίοι όταν τους αποδοθεί σημασία θα ερμηνεύονται σαν **αντικείμενα** από κάποιο σύνολο). Οι όροι ορίζονται επαγωγικά:

1. Οι μεταβλητές και οι σταθερές είναι όροι.
2. Αν $t_1, t_2, ..., t_n$ είναι όροι και f είναι σύμβολο συνάρτησης n θέσεων τότε $f(t_1, t_2, ..., t_n)$ είναι επίσης όρος.
3. Τίποτε άλλο δεν είναι όρος.

Τύποι

Οι **προτάσεις** του κατηγορηματικού λογισμού ορίζονται επαγωγικά:

1. Αν $t_1, t_2, ..., t_n$ είναι όροι και P είναι σύμβολο κατηγορήματος n θέσεων τότε $P(t_1, t_2, ..., t_n)$ είναι πρόταση (ατομική πρόταση).
2. Αν οι Φ και Ψ είναι προτάσεις και x είναι μεταβλητή τότε και οι $\neg\Phi$, $(\Phi \wedge \Psi)$, $(\Phi \vee \Psi)$, $\forall x\Phi$, $\exists x\Phi$ είναι προτάσεις.
3. Τίποτε άλλο δεν είναι πρόταση.

Σημασιολογία

Οι σταθερές και οι μεταβλητές **ερμηνεύονται** σαν στοιχεία ενός συνόλου A. Τα συναρτησιακά σύμβολα ερμηνεύονται σαν συναρτήσεις: $A^n \to A$. Έτσι κάθε όρος ερμηνεύεται σαν ένα στοιχείο του A.

Τα κατηγορήματα ερμηνεύονται σαν υποσύνολα του A^n. Η πρόταση $P(t_1, t_2, ..., t_n)$ είναι αληθής ανν $(s_1, s_2, ..., s_n) \in R$ όπου $s_1, s_2, ..., s_n$ είναι τα στοιχεία του A με τα οποία ερμηνεύονται οι όροι $t_1, t_2, ..., t_n$ 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 Φ και Ψ όπως και στην προτασιακή λογική. Η πρόταση $\forall x\Phi$ είναι αληθής αν η πρόταση Φ είναι αληθής για **οποιαδήποτε** ερμηνεία της μεταβλητής x, ενώ η πρόταση $\exists x\Phi$ είναι αληθής αν η Φ αληθεύει για **κάποια** ερμηνεία της x.

Prolog

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 σύξευξη από φράσεις Horn.

Σύμβολα του κατηγορηματικού λογισμού

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

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

Ελεύθερη εμφάνιση της μεταβλητής x στον τύπο Φ 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 $\exists$).

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

Σημασιολογία

Η **σημασιολογία** τύπων του κατηγορηματικού λογισμού δίνεται με την βοήθεια των αλγεβρικών δομών A που ονομάζουμε **μοντέλα**. Στην περίπτωση του προτασιακού λογισμού το πεδίο A είναι $\{\text{True}, \text{False}\}$, εδώ όμως μπορεί να είναι οποιοδήποτε μή κενό, πεπερασμένο ή και άπειρο, σύνολο. Εδώ λοιπόν δεν έχουμε **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. Με τέτοια σημασιολογία κάθε όρος ερμηνεύεται με στοιχείο του A και κάθε κλειστός τύπος αληθεύει (ή όχι) στο μοντέλο A.

Θεώρημα Πληρότητας

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

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

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

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

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

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

Μηχανές Turing

Οι βασικές λειτουργίες μιας TM είναι:

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

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

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

Ένα πρόγραμμα για μια TM είναι ένα σύνολο από τετράδες της μορφής q_i, e, d, q_j όπου $q_i, q_j \in Q, e \in \Sigma$, $d \in A = \Sigma \bigcup \{L, R\}$ με τον εξής συναρτησιακό (ντετερμινιστικό) περιορισμό: Για κάθε $< q_i, e >$ υπάρχει το πολύ ένα $< d, q_j >$ έτσι ώστε η τετράδα $< q_i, e, d, q_j >$ να ανήκει στο πρόγραμμα, δηλαδή πρόκειται για μια **συνάρτηση μετάβασης** (*transition function*) $\delta : Q \times \Sigma \to A \times Q$.

Κατά σύμβαση η μηχανή σταματάει στο ζεύγος κατάστασης-συμβόλου $< q_i, e >$ **σε περίπτωση που η** τιμή $\delta(q_i, e)$ δεν είναι ορισμένη.

Σε κάθε TM μπορούμε να αντιστοιχήσουμε μια μερική συνάρτηση από το $\mathbb{N}$ στο $\mathbb{N}$. Η είσοδος $n \in \mathbb{N}$ παριστάνεται με $\underline{n+1}$ 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.

Παράδειγμα 6.1.1. Να κατασκευαστεί μια TM που υπολογίζει το $2 * x$. Η TM θα εργάζεται σύμφωνα με το παρακάτω πρόγραμμα (το οποίο γράφεται πάντα πρώτα σε άτυπη γλώσσα υψηλού επιπέδου):

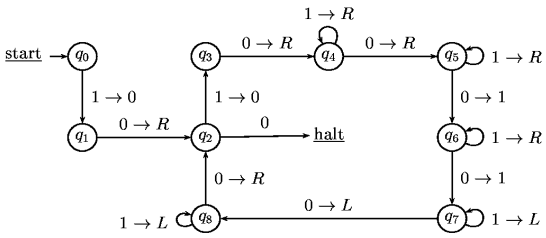
```
αρχικοποίηση; (*διαγραφή του πρώτου 1*)
while είσοδος<>0 do
 begin
   διάγραψε ένα 1 από είσοδο;
   μετακίνησε κεφαλή δεξιά πέρα από είσοδο και έξοδο;
   πρόσθεσε δύο 1 στην έξοδο;
   μετακίνησε κεφαλή αριστερά πέρα από έξοδο και είσοδο
 end
```

$< q_0 \quad 1 \quad 0 \quad q_1 >$
$< q_1 \quad 0 \quad \mathrm{R} \quad q_2 >$
$< q_2 \quad 1 \quad 0 \quad q_3 >$ | halt για $< q_2 \quad 0 >$
$< q_3 \quad 0 \quad \mathrm{R} \quad q_4 >$
$< q_4 \quad 1 \quad \mathrm{R} \quad q_4 >$
$< q_4 \quad 0 \quad \mathrm{R} \quad q_5 >$
$< q_5 \quad 1 \quad \mathrm{R} \quad q_5 >$
$< q_5 \quad 0 \quad 1 \quad q_6 >$
$< q_6 \quad 1 \quad \mathrm{R} \quad q_6 >$
$< q_6 \quad 0 \quad 1 \quad q_7 >$
$< q_7 \quad 1 \quad \mathrm{L} \quad q_7 >$
$< q_7 \quad 0 \quad \mathrm{L} \quad q_8 >$
$< q_8 \quad 1 \quad \mathrm{L} \quad q_8 >$
$< q_8 \quad 0 \quad \mathrm{R} \quad q_2 >$

Πίνακας 5.1: TM πρόγραμμα.

	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

Πίνακας 5.2: TM σε μορφή πίνακα.



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

Πίνακας 5.3: Τυπικό σύνολο εντολών RAM.

Πρόγραμμα RAM		Αντίστοιχη ψευδογλώσσα
	READ 1	read r1
	LOAD 1	
	JGTZ pos	if r1 <= 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	

Πίνακας 6.4: Τυπικό πρόγραμμα RAM.

Πρόγραμμα του Leibniz

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 «Α και όχι Α» ($A \wedge \neg A$) δεν είναι δυνατόν να αποδειχθεί σ' αυτή τη θεωρία).

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 αυτού του 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.

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.

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 «υπολογιστή (*computable*) ή όχι». Αυτό ακριβώς είναι και το αντικείμενο της **Θεωρίας της Υπολογιστότητας** (*computability*). Αν δοθεί ότι μια συνάρτηση f είναι υπολογιστή, 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) .

Διάφοροι επιστήμονες (*Turing, Church, Kleene, Post, Markov*, κ.α.) βάλθηκαν να ξεκαθαρίσουν τις έννοιες: υπολογιστό ή επιλύσιμο (*solvable*) με αλγόριθμο, υπολογιστή 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 "υπολογιστός" 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.

- Υπάρχουν άπειρα μεν, αλλά μόνο αριθμήσιμα (*countable*) διαφορετικά προγράμματα. Εκτός αυτού μπορούμε χρησιμοποιώντας κωδικοποίηση να τα απαριθμήσουμε μηχανιστικά (*effectively enumerate*)
- 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

Θεώρημα 6.1.1. *Το **halting problem (HP)** είναι μη αποκρίσιμο.*

Απόδειξη. Έστω ότι $\pi_0, \pi_1, \pi_2, \ldots$ είναι μια μηχανιστική απαρίθμηση (*effective enumeration*) όλων των προγραμμάτων. Ας υποθέσουμε ότι το **HP** είναι επιλύσιμο. Τότε κατασκευάζουμε ένα πρόγραμμα π, που ελέγχει αν το πρόγραμμα π_n με είσοδο n σταματάει ή όχι και ανάλογα με την απάντηση σε αυτόν τον έλεγχο, το πρόγραμμα π σταματάει αν το $\pi_n(n)$ δεν σταματάει, και αντιστρόφως:

$$\pi\colon \text{read}(n); \textbf{if } \pi_n(n) \text{ terminates } \textbf{then } \text{loop\_forever } \textbf{else } \text{halt}$$

Φυσικά αυτό το πρόγραμμα π κάπου θα εμφανίζεται στην παραπάνω αρίθμηση. Ας πούμε ότι ο δείκτης για το π είναι i, δηλαδή $\pi = \pi_i$. Η ιδέα της διαγωνιοποίησης είναι να δώσουμε το δείκτη i για input στο π_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. □

Ορισμός 6.1.2. Ένα σύνολο S λέγεται αποκρίσιμο ή υπολογιστό 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 $a \notin S$.

Ορισμός 6.1.3. 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.

Ιδιότητες

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

Μοντέλα Υπολογισμού

- 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 μια κυψέλη "cell" της ταινίας κάθε φορά)

Τα χαρακτηριστικά των παραπάνω μοντέλων είναι:

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

Άλλα μοντέλα είναι:

- παραλλαγές από μηχανές Turing
- Thue: κανόνες επανεγγραφής (re-writing rules)
- Post: κανονικά συστήματα (normal systems)
- Church: λογισμός λ (λ-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 6.2.1. *f είναι TM υπολογιστή ανν*

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

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

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

Πολυπλοκότητα

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

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 $C(n)$, τώρα, ενός προβλήματος $\pi(n)$ είναι το $\min(A(n))$ από όλους τους αλγορίθμους A που λύνουν το πρόβλημα π. 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 $\Theta(n \log n)$.

Συνήθως ονομάζουμε αποδοτικό ένα αλγόριθμο αν ο χρόνος του είναι πολυωνυμικός ($n^{O(1)}$) ως προς το μέγεθος της εισόδου. **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 π εάν υπάρχει τουλάχιστον μια από τις δυνατές εκτελέσεις του A που λύνει το π. Προφανώς ισχύει $\mathbf{P} \subseteq \mathbf{NP}$ αλλά εδώ και τριάντα-πέντε χρόνια παραμένει άλυτο το πρόβλημα "$\mathbf{P} \neq \mathbf{NP}$;".

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

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

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.

Ισχύει:

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

Παρατήρηση:
REC = recursive = decidable
R.E. = recursively enumerable = listable

Ταξινόμηση Προβλημάτων

Divide and Conquer

```
    1984
x   6713
--------
    5952
   1984
 13888
11904
--------
13318592
```

$O(n^2)$

```
        A   B       Γ    Δ
       19  84      67   13

ΑΓ = 19x67                 = 1273
ΑΔ+ΒΓ = (A+B)(Γ+Δ)-ΑΓ-ΒΔ =   5875
ΒΔ = 84x13                 =     1092
                             --------
                             13318592
```

$O(n^{1.59})$

Ασυμπτωτική Συμπεριφορά

logn	n	n^2	2^n
3.322	10	100	1024
6.644	100	10000	1267650600228229401496703205376
9.966	1000	1000000	$(1267650600228229401496703205376)^{10}$

Αναδρομικές Σχέσεις

- T(n) = T(n/2) + c — $O(\log n)$ (Binary Search)
- T(n) = 2T(n/2) + c — $O(n)$ (Εύρεση μεγίστου)
- T(n) = 2T(n/2) + cn — $O(n \log n)$ (Mergesort)

Θέση Σειριακών Υπολογισμών (Sequential Computation Thesis)

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

Θέση Παράλληλων Υπολογισμών (Parallel Computation Thesis)

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

ισοζύγιο (tradeoff)