Θεωρία αλγορίθμων

3η γνωστική ενότητα : θεωρία αλγορίθμων

1. Υπολογισιμότητα (Computability)
2. Η ιστορία της υπολογισιμότητας
3. Μια απλή γλώσσα προγραμματισμού
4. Οι μηχανές Turing
5. Η Θέση των Church – Turing
6. Το πρόβλημα του τερματισμού
7. Άλλα μη υπολογίσιμα προβλήματα
8. Μερική υπολογισιμότητα

Υπολογισιμότητα (Computability)

- Ύπαρξη αλγορίθμων για μια σειρά από εργασίες όπως
  - έλεγχο των φαναριών
  - παραγωγή παραστατικών μισθοδοσίας
  - εύρεση ενός πρώτου αριθμού
  - ταξινόμηση μιας λίστας

- Ύπαρξη κάποιας δουλειάς που ένας υπολογιστής δεν μπορεί να κάνει?

ΝΑΙ

Ο αριθμός των πραγμάτων που μπορούν να υπολογιστούν είναι απείρως μικρότερος από τον αριθμό των πραγμάτων που θα ήθελε κάποιος να υπολογίσει

Η ιστορία της υπολογισιμότητας

- Τη δεκαετία του ’30 αποδείχτηκε ότι το πρόβλημα της απόφασης δεν είναι υπολογίσιμο
- 1931Kurt Gödel: Θεώρημα της μη πληρότητας (incompleteness problem).
  - Δεν υπάρχει αλγόριθμος του οποίου η είσοδος να είναι οποιαδήποτε δήλωση σχετική με τους ακέραιους αριθμούς και το αποτέλεσμα του να είναι αν η δήλωση είναι αληθής ή όχι
- Περισσότερα προβλήματα χωρίς αλγοριθμική λύση από Alonso Church, Stephen Kleene, Emil Post, Alan Turing
- Σημείωση: τα παραπάνω επιτεύχθηκαν σε εποχή που οι υπολογιστές δεν έχουν ουσιαστικά ακόμα κατασκευαστεί!

Μια απλή γλώσσα προγραμματισμού

- Στόχος: να καθορίσουμε μια γλώσσα προγραμματισμού που θα είναι αρκετά ισχυρή ώστε να καλύπτει ακόμα και τις μελλοντικές ανάγκες του προγραμματισμού
- Μια τέτοια γλώσσα θα πρέπει να είναι τόσο ισχυρή ώστε
  - αν υπάρχει ένας αλγόριθμος που θα επιλύει ένα πρόβλημα να μπορεί να εκφραστεί σε αυτή
  - αν ένα πρόβλημα δεν μπορεί να λυθεί με αυτή τη γλώσσα τότε δεν υπάρχει αλγόριθμος που να μπορεί να λύσει το πρόβλημα

Καθολική γλώσσα προγραμματισμού
Universal Programming Language

Μια απλή γλώσσα προγραμματισμού

- Δηλώσεις περιγραφής δεδομένων
  - Όλες οι μεταβλητές είναι μόνο του τύπου σειράς ψηφιοστοιχείων οποιουδήποτε μήκους (bit patterns of any length)
  - Συντακτικό δηλώσεων: όλα τα ονόματα μεταβλητών καθορίζονται από μόνο από συνδυασμούς γραμμάτων
  - Κάθε δήλωση τερματίζει με ένα “;” για να ξεχωρίζει ο Η/Υ τις δηλώσεις

Μια απλή γλώσσα προγραμματισμού

- Δηλώσεις εκτέλεσης εντολών
  - Εκχώρηση (όπου <name> όνομα μεταβλητής)
    - clear <name>: συσχετισμός με μια σειρά από 0
    - incr <name>: αύξηση κατά 1
    - decr <name>: μείωση κατά 1 (εκτός και αν η αρχική τιμή είναι 0, οπότε και παραμένει)
- **Παράδειγμα**

  clear test → test = 00000

  incr test → test = 00001

  incr test → test = 00010

  incr test → test = 00011

  decr test → test = 00010

Μια απλή γλώσσα προγραμματισμού

- Δομή ελέγχου
  - while <name> not 0 do;
    …
    …
    end;
- Παραδείγματα

```
incr X;
while X not 0 do;
      incr Z;
end;
```

Ατέρμων βρόχος

```
clear Z;
while X not 0 do;
      incr Z;
      decr X;
end;
```

Μετακίνηση τιμών

Μια απλή γλώσσα προγραμματισμού

- Σκοπός της γλώσσας είναι να μας βοηθήσει στην έρευνα του τι είναι δυνατό και όχι τι είναι πρακτικό
- Δυνατότητα εκτέλεσης ενεργειών όπως
  - εκχώρηση μιας τιμής
  - Μετακίνηση μιας τιμής από μια μεταβλητή σε μία άλλη
  - Αντιγραφή μιας τιμής από μια μεταβλητή σε μία άλλη?

Αντιγραφή μιας τιμής από μια μεταβλητή σε μία άλλη?

```
copy X to Y
clear tmp;
clear Y;
while X not 0 do;
    incr tmp;
    decr x;
    end;
while tmp not 0 do;
    incr Y;
    incr X;
    decr tmp;
end;
```

Μια απλή γλώσσα προγραμματισμού

- Παράδειγμα (Υπολογισμός X * Y)

```
clear Z;                    /* στο Z αποθηκεύεται το τελικό αποτέλεσμα */
while X not 0 do;
    clear W;
    while Y not 0 do;  /* αντιγραφή του Y στο W και πρόσθεση Y στο Z  */
      incr Z;
      incr W;
      decr Y;
    end;
    while W not 0 do; /* επαναφορά της τιμής του Y  */
      incr Y;
      decr W;
    end;
    decr X;
end;
```

Πώς μπορούμε να αντιστρέψουμε τα περιεχόμενα μιας μεταβλητής?

- Αν χ = 0 τότε χ παίρνει την τιμή 1
- Αν χ ≠ 0 τότε χ παίρνει την τιμή 0

```
invert x
clear aux;
incr aux;
while X not 0
    clear x;
    clear aux;
end;
while aux not 0
    incr x;
    clear aux;
end;
```

Πώς μπορούμε να υλοποιήσουμε μια if-then-else?

- **If X not 0 then S1 else S2**

```
copy x to aux;
while aux not 0 do;
   S1;
   clear aux;
end;
copy x to aux;
invert aux;
while aux not 0 do;
   S2;
   clear aux;
end;
```

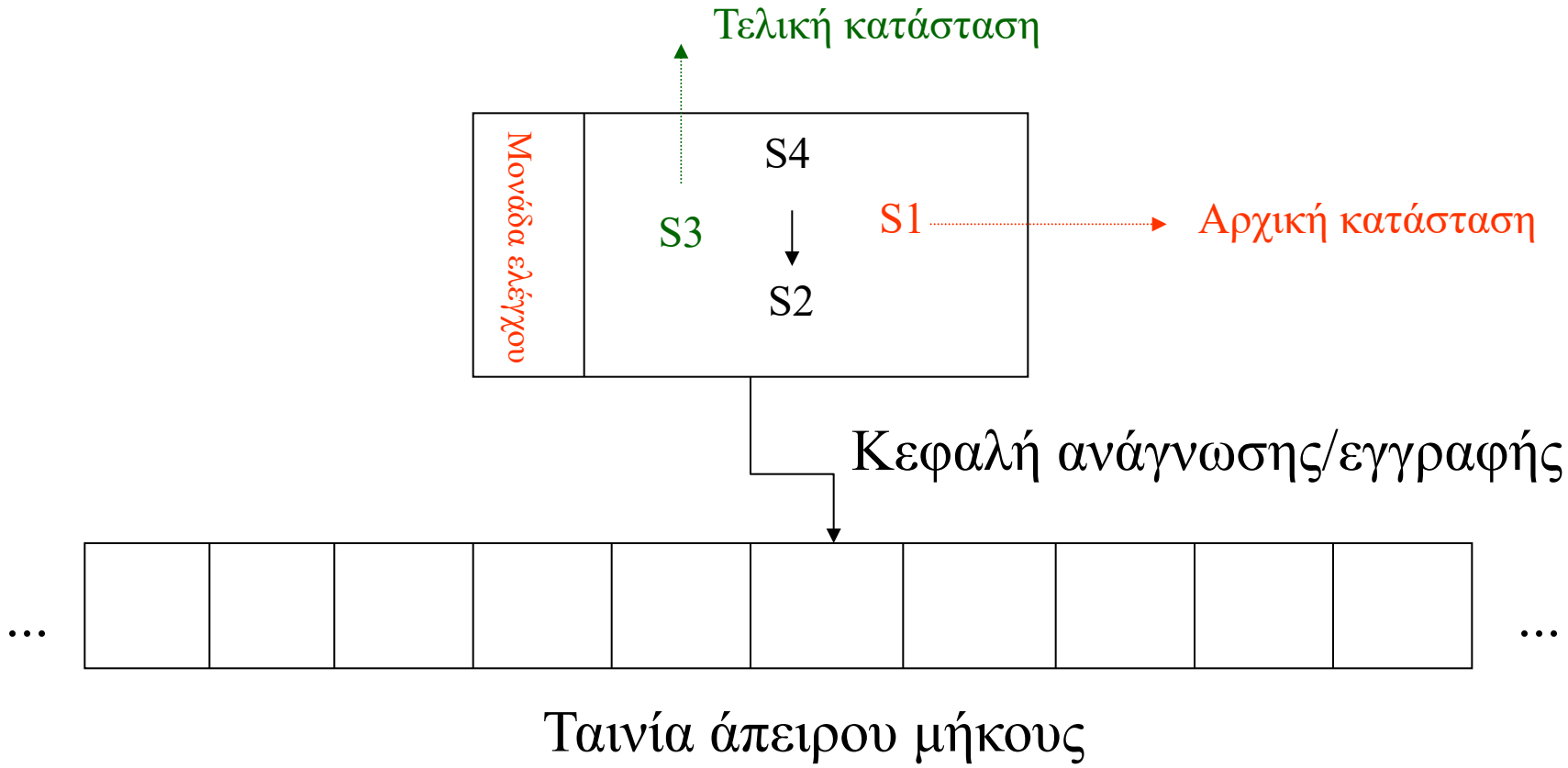
Ισοδυναμία επανάληψης και αναδρομής

```
while x not 0;
    s;
end;

main: begin
                if X not 0 perform unit A;
          end;
UnitA: begin
                S
                if X not 0 perform unit A;
         return;
```

Μηχανές Turing

- Εισήχθησαν από τον A. Turing το 1936 ως εργαλεία μελέτης των αλγοριθμικών διεργασιών και χρησιμοποιούνται ως σήμερα

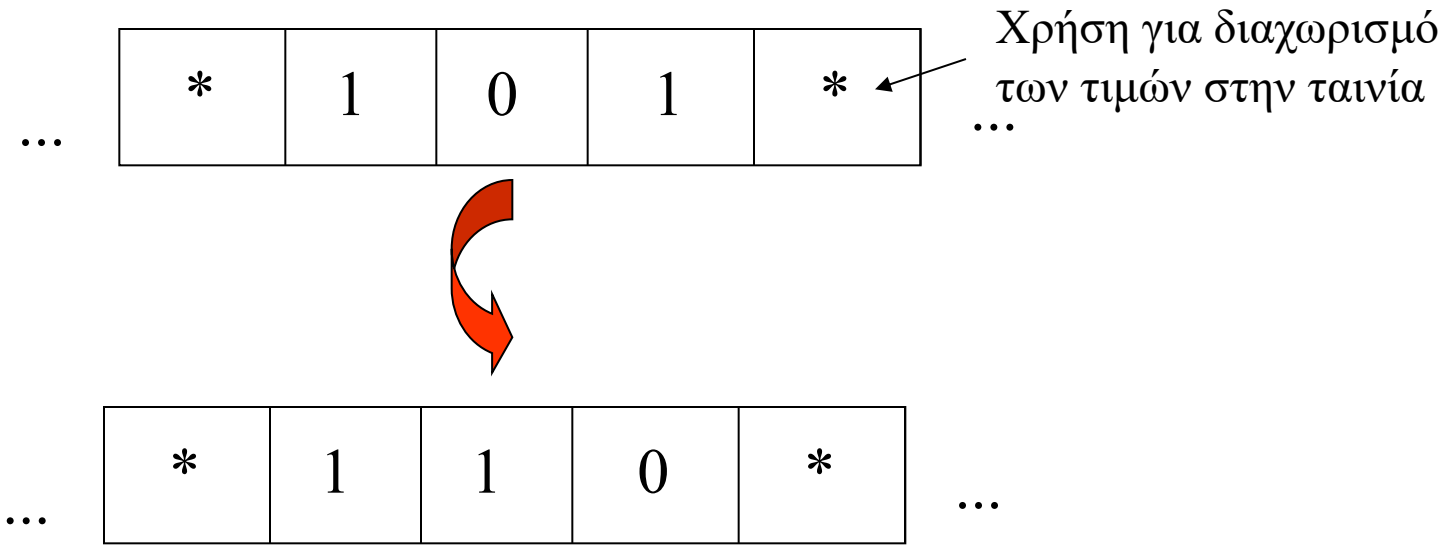


Μηχανές Turing

- Οι σημερινοί Η/Υ είναι ουσιαστικά μηχανές Turing με πεπερασμένη όμως μνήμη & CPU = Μονάδα Ελέγχου
- Η αλγοριθμική ισχύς των μηχανών αυτών είναι τόσο μεγάλη που αν ένα πρόβλημα δεν μπορεί να λυθεί με αυτές τότε δεν επιδέχεται επίλυση από κανένα αλγοριθμικό σύστημα
- Ορίζουν ένα θεωρητικό όριο στις δυνατότητες των μηχανών

Παράδειγμα λειτουργίας μιας μηχανής Turing

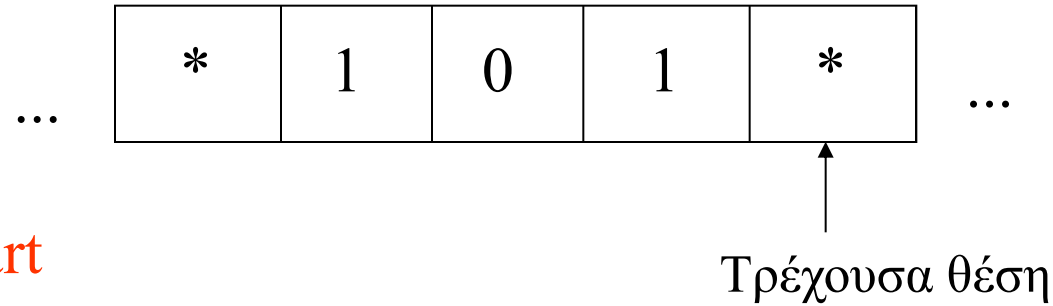
- Η μηχανή υλοποιεί την αύξηση κατά ένα ενός αριθμού στο δυαδικό σύστημα

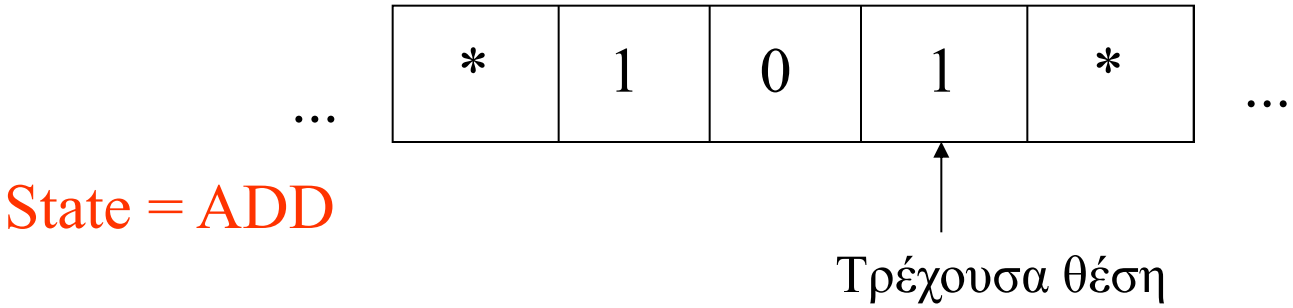


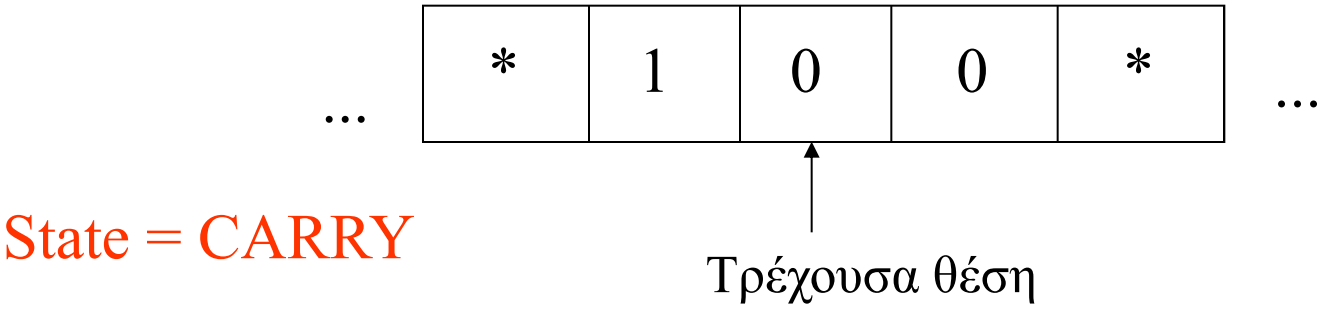
Παράδειγμα λειτουργίας μιας μηχανής Turing

Τρέχουσα κατάσταση	Περιεχόμενο του τρέχοντος κελιού	Τιμή προς εγγραφή	Μετακίνηση κεφαλής	Νέα Κατάσταση
START	*	*	Left	ADD
ADD	0	1	Left	NO CARRY
ADD	1	0	Left	CARRY
ADD	*	*	Right	HALT
CARRY	0	1	Left	NO CARRY
CARRY	1	0	Left	CARRY
CARRY	*	1	Left	OVERFLOW
NO CARRY	0	0	Left	NO CARRY
NO CARRY	1	1	Left	NO CARRY
NO CARRY	*	*	Right	RETURN
OVERFLOW	ignore	*	Right	RETURN
RETURN	0	0	Right	RETURN
RETURN	1	1	Right	RETURN
RETURN	*	*	No move	HALT

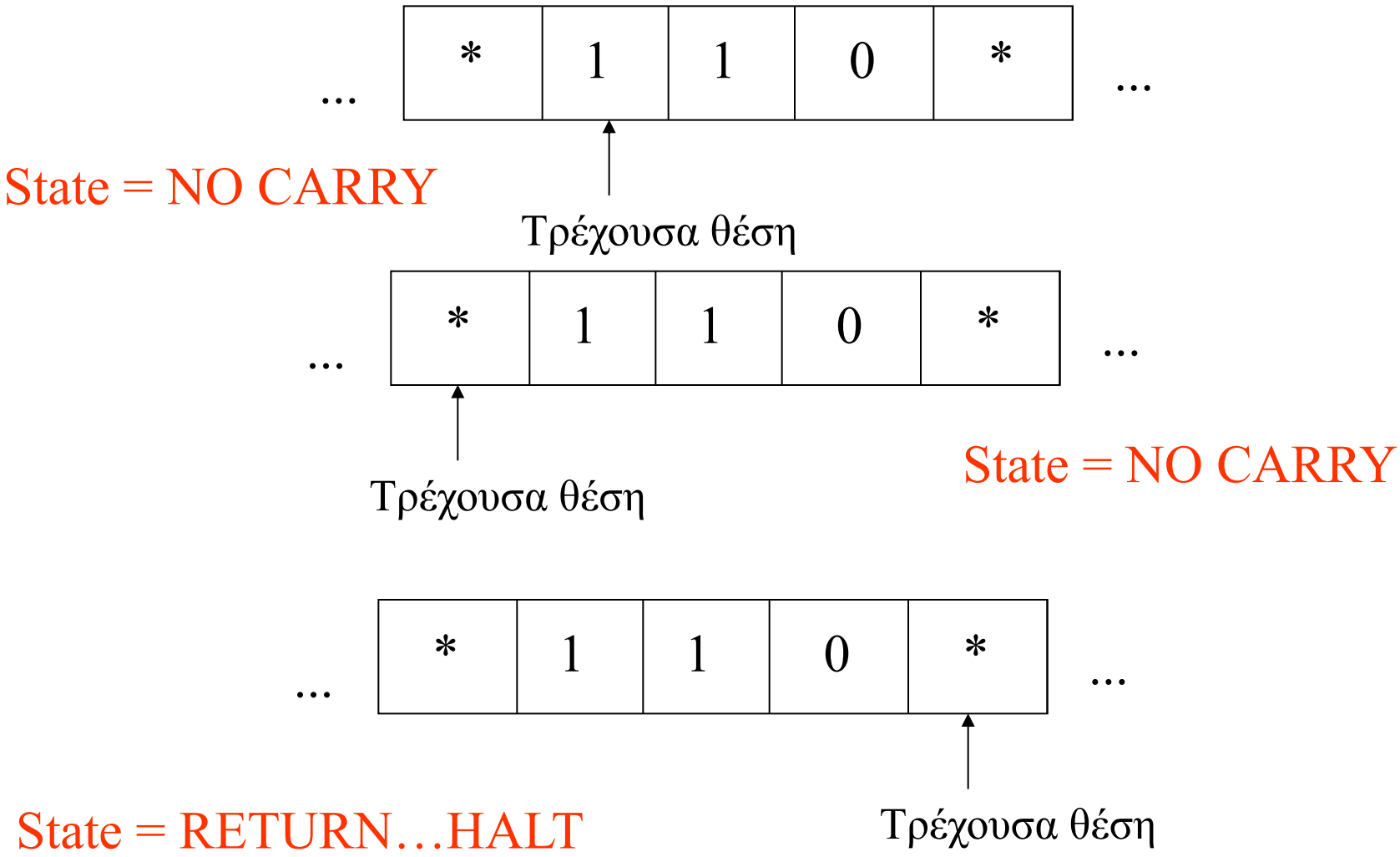
Παράδειγμα λειτουργίας μιας μηχανής Turing



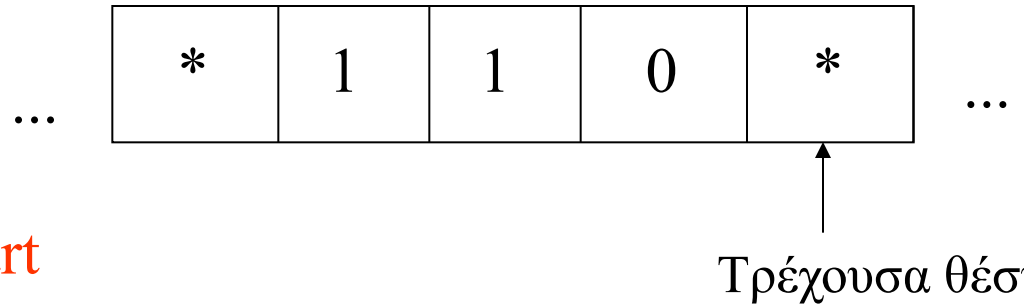




Παράδειγμα λειτουργίας μιας μηχανής Turing



Άσκηση



Εφαρμόστε την παραπάνω είσοδο στη Μηχανή Turing του προηγούμενου παραδείγματος

Η Θέση των Church-Turing

- Τι εννοούμε λέγοντας αλγόριθμος;
- Διαφορετικοί ορισμοί αλγορίθμων
  - Μηχανή Turing
  - Απλή γλώσσα προγραμματισμού
  - Λογισμός λ (Lambda Calculus) – Church
- Όλοι οι φαινομενικά διαφορετικοί και ανεξάρτητα επινοημένοι ορισμοί είναι τελικά υπολογιστικά ισοδύναμοι

Η Θέση των Church-Turing

- Όλοι οι εύλογοι ορισμοί του «αλγόριθμου» που είναι γνωστοί ως τώρα είναι ισοδύναμοι
- Κάθε εύλογος ορισμός του «αλγόριθμου» που κάποιος πιθανό θα δώσει θα αποδειχθεί ισοδύναμος με αυτούς που ξέρουμε

- Κάθε αλγόριθμος που εκτελείται σε ένα υπολογιστή μπορεί να εκτελεστεί και σε οποιοδήποτε άλλο (προσομοίωση –simulation)
- Αν πάρουμε 2 τυχαίους υπολογιστές μπορούμε να γράψουμε ένα πρόγραμμα (διερμηνέας – προσομοιωτής) για τον ένα Η/Υ το οποίο μπορεί να καταλάβει και να εκτελέσει κάθε πρόγραμμα γραμμένο για τον άλλο

Αφού όλοι οι υπολογιστές είναι ισοδύναμοι γιατί επιλέγουμε κάποιους υπολογιστές σε σχέση με κάποιους άλλους?

Το πρόβλημα του τερματισμού

- Δήλωση:Υπάρχουν πολλά προβλήματα που δεν είναι υπολογίσιμα
- Παράδειγμα: Gödel – Θεώρημα μη πληρότητας
- Υπάρχουν προβλήματα της Επιστήμης Υπολογιστών πιο άμεσου ενδιαφέροντος τα οποία να μην είναι υπολογίσιμα;
- Πρόβλημα: Μπορούμε να προσδιορίσουμε αν ένα αυθαίρετο πρόγραμμα τερματίζει την εκτέλεση του ή όχι; (halting problem)
- Λύση: Αλγόριθμος ο οποίος αν του δοθεί ένα αυθαίρετο πρόγραμμα P και τα δεδομένα εισόδου του D, μπορεί να διαπιστώσει αν το P τερματίζει όταν εκτελεστεί με το D

Το πρόβλημα του τερματισμού

- Μπορούμε να χρονομετρούμε ένα πρόγραμμα και να διακόπτουμε πέρα από κάποιο χρονικό όριο
  - Μη πρακτικό αφού κάθε μη τερματίζουσα εκτέλεση θα καταναλώνει όλο το χρόνο πριν αποφανθεί
  - Αν δεν έχουμε υπολογίσει σωστά μπορεί να διακόψουμε την εκτέλεση λίγο πριν τον επιτυχή τερματισμό του προγράμματος

Το πρόβλημα του τερματισμού είναι μη υπολογίσιμο

Αριθμοί Gödel

- Στη θεωρητική επιστήμη των υπολογιστών κάθε πρόγραμμα μπορεί να αντιστοιχηθεί σε ένα μη προσημασμένο ακέραιο
- Π.χ., Σύμβολο Δεκαεξαδικός κωδικός

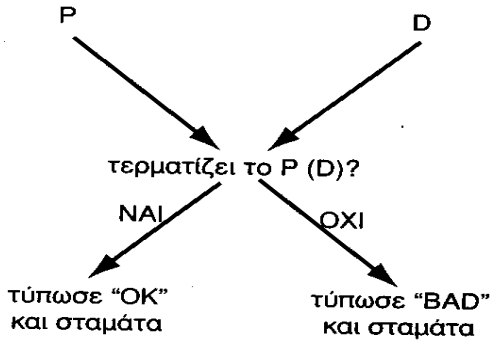
Σύμβολο	Δεκαεξαδικός κωδικός
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
0	A
incr	B
decr	C
while	D
end	E
X	F

Παράδειγμα

- Ποιος είναι ο αριθμός Gödel του προγράμματος incr X;
  - BF ή 191 στο δεκαδικό

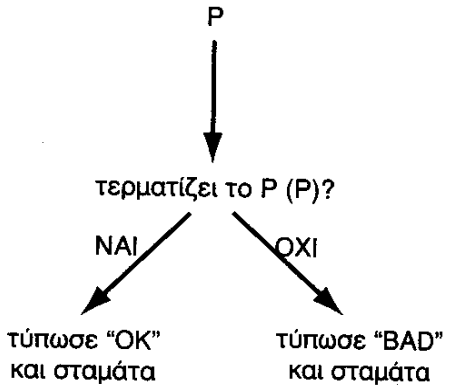
Το πρόβλημα του τερματισμού

- Έστω ότι υπάρχει αλγόριθμος που να επιλύει το πρόγραμμα → ελεγκτής τερματισμού (halttester)



- Νέος αλγόριθμος που δοκιμάζει τον τερματισμό του P όταν τα δεδομένα είναι το ίδιο το P → νέος ελεγκτής τερματισμού (newhalttester)

Module newhalttester(P)
{ελέγχει αν το πρόγραμμα P τερματίζει όταν εκτελεστεί με δεδομένα P
halttester(P,P)



Το πρόβλημα του τερματισμού

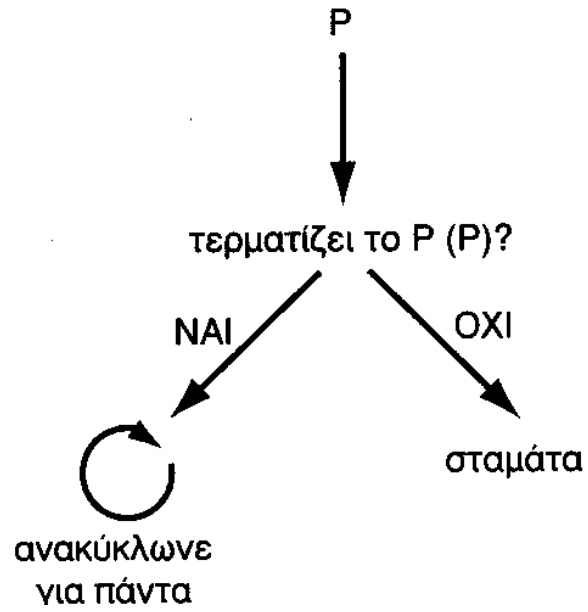
- Αφού υπάρχουν οι halttester & newhalttester τότε μπορούμε να κατασκευάσουμε και τον «παράδοξο» (funny) αλγόριθμο

Module funny(P)

Αν new halttester(P) δίνει απάντηση BAD

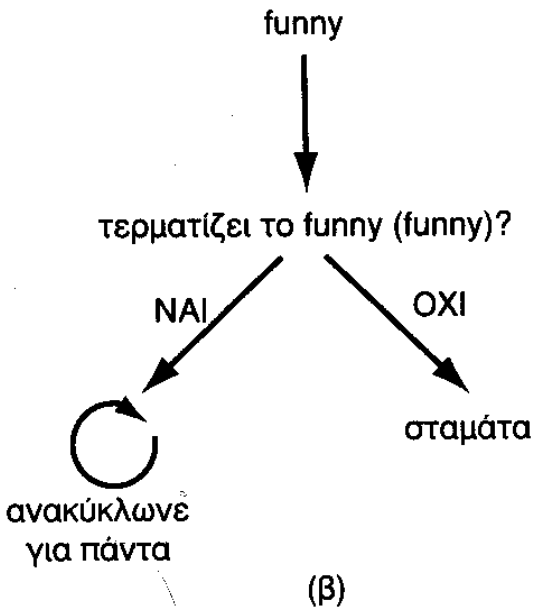
τότε σταμάτα

αλλιώς ανακύκλωνε για πάντα



Το πρόβλημα του τερματισμού

- Τι γίνεται στην περίπτωση που εκτελέσουμε το funny (funny)?



Αντίφαση: Αν το funny(funny) τερματίζει τότε επαναλαμβάνεται για πάντα
Αν το funny(funny) επαναλαμβάνεται επ' άπειρον τότε τερματίζει

Άρα δεν μπορεί να υπάρξει το funny και κατά συνέπεια ο halttester που υπολογίζει το πρόβλημα του τερματισμού

Το πρόβλημα του τερματισμού

- Τα βήματα της απόδειξης
  - Υποθέστε ότι μπορείτε να γράψετε ένα πρόγραμμα halttester
  - Χρησιμοποιείστε το για να φτιάξετε ένα νέο πρόγραμμα funny (μέσω του ενδιάμεσου προγράμματος newhalttester)
  - Δείξτε ότι το funny έχει κάποια αντιφατική ιδιότητα (δεν μπορεί ούτε να τερματίσει, ούτε να επαναλαμβάνεται για πάντα)
  - Συμπέρασμα η υπόθεση του βήματος 1 είναι λάθος

Το πρόβλημα του τερματισμού

- Υπάρχουν προβλήματα που είναι προφανές το αν τερματίζουν ή όχι
- Υπάρχουν και πολλά άλλα που δεν είναι προφανή π.χ., αλγόριθμος ελέγχου του τελευταίου θεωρήματος του Fermat – δεν υπάρχουν θετικοί ακέραιοι a,b,c τ.ω.

$$a^n+b^n = c^n \text{ όπου } n>2$$

```
Module Fermat (n)
    επανέλαβε για a= 1,2,3,…..
        επανέλαβε για b= 1,2,3,…,a
            επανέλαβε για c= 2,3,4,…,a+b
                αν a^n + b^n = c^n
                    τότε δώσε ως έξοδο τα a,b,c και n και τερμάτισε
```

Αν ο αλγόριθμος τερματίσει τότε το τελευταίο θεώρημα του Fermat δεν θα ισχύει!

Το πρόβλημα του τερματισμού

- Είναι εύκολο να αποδείξουμε ότι θα τερματίσει για είσοδο ίση με 1 (a=1,b=1,c=2)
- Όμοια για είσοδο ίση με 2 (a=3,b=4,c=5)
- Για n=3 δεν είναι προφανές, αλλά κάποιος ιδιαίτερα εύστροφος μπορεί να αποδείξει ότι δεν θα τερματίσει
- Τι ισχύει για n=37?

Το θεώρημα του Fermat αποδείχθηκε πριν 18 χρόνια

Άλλα μη υπολογίσιμα προβλήματα (το πρόβλημα της ολότητας)

- Το πρόβλημα του τερματισμού (halt problem): Ένα αυθαίρετο πρόγραμμα θα τερματίσει ή όχι για αυθαίρετα δεδομένα εισόδου
- Το P είναι ολικό (total) αν ένα αυθαίρετο πρόγραμμα P(D) σταματάει για όλα τα δεδομένα εισόδου D.
- Το πρόβλημα της ολότητας είναι μη υπολογίσιμο
- Έστω ότι είναι υπολογίσιμο τότε μπορώ να κατασκευάσω ειδικό πρόγραμμα που να αγνοεί την είσοδο του και να προσομοιώνει το P(D)

module funnypd (I)

{I η είσοδος του αλλά την αγνοεί }

Προσομοίωσε το πρόγραμμα P για τα δεδομένα D

Άλλα μη υπολογίσιμα προβλήματα (το πρόβλημα της ολότητας)

- Η ερώτηση «είναι το funny pd ολικό» είναι ισοδύναμη με την ερώτηση αν το P(D) σταματά
- Ένας αλγόριθμος για το πρόβλημα της ολότητας μας δίνει τον ακόλουθο αλγόριθμο για το πρόβλημα του τερματισμού

```
module halttester (P,D)
{υποθέτουμε ότι υπάρχει ο αλγόριθμος για το πρόβλημα της ολότητας}
κατασκεύασε το κείμενο του προγράμματος funnypd
Αν funnypd είναι ολικό
        τότε έξοδος «OK» και τερμάτισε
        αλλιώς έξοδος «BAD» και τερμάτισε
```

Αφού ξέρουμε ότι δεν υπάρχει αλγόριθμος για το πρόβλημα του τερματισμού σημαίνει ότι δεν υπάρχει αλγόριθμος που να λύνει το πρόβλημα της ολότητας.

Άλλα μη υπολογίσιμα προβλήματα (το πρόβλημα της ισοδυναμίας)

- Πρόβλημα ισοδυναμίας (equivalence problem): η σύγκριση δύο προγραμμάτων για να ελεγχθεί αν κάνουν την ίδια εργασία (δηλ. για την ίδια είσοδο παράγουν την ίδια έξοδο)
- Το πρόβλημα της ισοδυναμίας είναι μη υπολογίσιμο
- Αν ήταν υπολογίσιμο τότε για οποιοδήποτε P μπορούμε να κατασκευάσουμε γι' αυτό το P ένα ειδικό πρόγραμμα που να προσομοιώνει το P και δίνει έξοδο13 αν το P τελικά σταματά.

module funnyp (D)
{Δίνει έξοδο 13 αν το P(D) σταματήσει}
Προσομοίωσε το πρόγραμμα P με είσοδο δεδομένων D
{Το προηγούμενο βήμα μπορεί να επαναλαμβάνεται για πάντα, αν όμως δεν επαναλαμβάνεται τότε}
Έξοδος 13

- Επίσης θεωρείστε ένα απλό πρόγραμμα που δίνει πάντα έξοδο 13

module simple (D)
{Δίνει έξοδο 13 ανεξάρτητα από την είσοδο του}
Έξοδος 13

Άλλα μη υπολογίσιμα προβλήματα (το πρόβλημα της ισοδυναμίας)

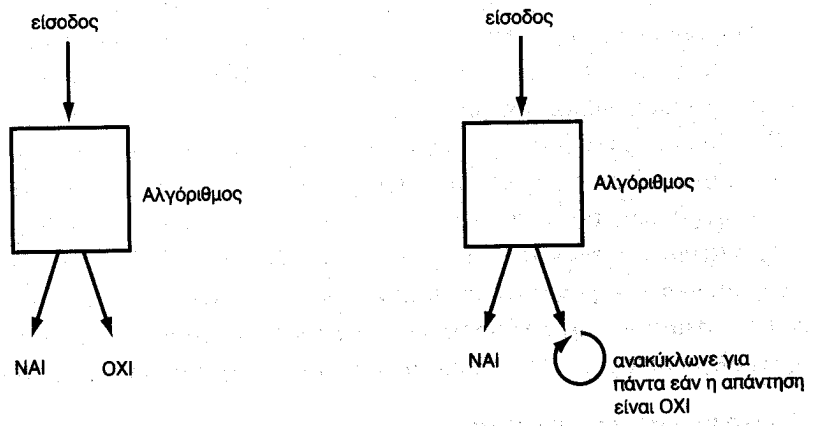
- Ρωτώντας αν είναι το funnyp ισοδύναμο με το simple είναι το ίδιο με το να ρωτήσουμε αν το P είναι ολικό
- Αν το P σταματάει σε κάθε είσοδο τότε το funnyp δίνει έξοδο 13 σε κάθε είσοδο
- Αν υπάρχει κάποια είσοδος στην οποία το P δεν σταματάει τότε το funnyp δεν θα σταματήσει και δεν θα δώσει έξοδο 13
- Αν υπήρχε ο αλγόριθμος της ισοδυναμίας θα μας έδινε ένα αλγόριθμο ολότητας

module total (P)
{υποθέτουμε ότι υπάρχει ο αλγόριθμος για το πρόβλημα της ισοδυναμίας}
κατασκεύασε το κείμενο του προγράμματος funnyp
κατασκεύασε το κείμενο του προγράμματος simple
Αν funnyp και simple είναι ισοδύναμα
τότε έξοδος «το P είναι ολικό»
αλλιώς έξοδος «Το P δεν είναι ολικό»

Αφού ξέρουμε ότι δεν υπάρχει αλγόριθμος για το πρόβλημα της ολότητας σημαίνει ότι δεν υπάρχει αλγόριθμος που να λύνει το πρόβλημα της ισοδυναμίας.

Μερική υπολογισιμότητα

- Κάποια μη υπολογίσιμα προβλήματα είναι λιγότερο μη υπολογίσιμα από κάποια άλλα
- Π.χ. Στο πρόβλημα του τερματισμού αν το P(D) τερματίζει τότε δεν υπάρχει δυσκολία αφού το αφήνουμε να εκτελείται μέχρι να τερματίσει. Αν όμως δεν τερματίζει τότε όσο και να το αφήσουμε δεν μπορούμε να αποφανθούμε για το αν τερματίζει
- Το πρόβλημα του τερματισμού λέγεται μερικώς υπολογίσιμο (partially computable), αφού υπάρχει αλγόριθμος που απαντά ΝΑΙ αν τερματίζει αλλά εισέρχεται σε ατέρμονα βρόχο αν δεν τερματίζει.
- Τα προβλήματα της ολότητας και της ισοδυναμίας δεν είναι μερικώς υπολογίσιμα



Μερική υπολογισιμότητα

- Τα μαθηματικά μιλούν για αποδεικτικά συστήματα (γενικές διαδικασίες ή μεθοδολογίες που χρησιμοποιούνται για να αποδεικνύουμε πράγματα)
- Ουσιαστικά μιλάμε για σύνολο αξιωμάτων και συμπερασματικών κανόνων
- Η απόδειξη είναι μια σειρά γραμμών ξεκινώντας από αξιώματα και προχωρώντας με τους συμπερασματικούς κανόνες έως ότου να φτάσουμε στο αντικείμενο της απόδειξης (θεώρημα)
- Αποδεικνύεται ότι ένα σύστημα είναι μερικώς υπολογίσιμο ανν έχει ένα αποδεικτικό σύστημα
- Αν ένα πρόγραμμα τερματίζει για κάποια δεδομένα μπορούμε να γράψουμε μια πειστική απόδειξη
- Για τα μη μερικώς υπολογίσιμα προβλήματα δεν μπορεί να βρεθεί ομοιόμορφη διαδικασία ρουτίνας για την επαλήθευση τους, και απαιτείται ιδιαίτερη δημιουργικότητα για την αντιμετώπιση τους