Θέματα στη Μοντελοποίηση Συστημάτων

Στην ενότητα αυτή θα μελετηθούν τα εξής θέματα:

Σειριακά και συντρέχοντα συστήματα

Συστήματα μεταβάσεων

Δικαιοσύνη

Σημασιολογία παρεμβαλλόμενης διάταξης

Σημασιολογία μερικής διάταξης

Μοντελοποίηση

- *Μοντελοποίηση*: η δημιουργία "αντικειμένων" τα οποία συλλαμβάνουν τη συμπεριφορά συστημάτων.
- Ένα μοντέλο επιτρέπει τη διεξαγωγή συμπερασμάτων για τη συμπεριφορά του συστήματος στο οποίο αντιστοιχεί.
- Η μοντελοποίηση συστημάτων συνήθως συνεπάγεται τη διαδικασία της απόσπασης (*abstraction*), δηλαδή, της απλοποίησης της περιγραφής ενός συστήματος διατηρώντας μόνο ένα περιορισμένο αριθμό από τις αρχικές λεπτομέρειες.
- Η απόσπαση (abstraction) μπορεί να καταστεί αναγκαία λόγω
  - της πολυπλοκότητας συστημάτων
  - Αυστηρές μέθοδοι δεν μπορούν να εφαρμοστούν σε φυσικές οντότητες (π.χ. μνήμη του υπολογιστή) έτσι οι σχετικές ιδιότητές τους πρέπει να αποσπαστούν μαθηματικά.

Μοντελοποίηση

- Εφαρμογή τυπικών μεθόδων ξεκινά με τη μοντελοποίηση/διατύπωση του συστήματος στη συγκεκριμένη σύνταξη του εργαλείου που θα χρησιμοποιηθεί.
- Το μοντέλο ενός συστήματος μπορεί να κατασκευαστεί και να μελετηθεί *πριν* ή *παράλληλα* με τον σχεδιασμό και την υλοποίησή του.
- Θα ασχοληθούμε με τη μοντελοποίηση
  - Σειριακών (sequential) συστημάτων
  - Συντρεχόντων (concurrent) συστημάτων
    - Κατανεμημένα συστήματα
    - Αντενεργά (reactive) συστήματα
    - Ενθυλακωμένα (embedded) συστήματα (software + hardware).

Σειριακά (sequential) συστήματα

- Εκτελούν κάποιο υπολογισμό.
- Έχουν κάποια αρχική κατάσταση.

  π.χ. $\forall\ 0 \le i \le n\ A[i] \ge 0$, A[i] integer
- Έχουν κάποιο τελικό στόχο.

  π.χ. $\forall\ 0 \le i \le n-1\ A[i] < A[i+1]$
- Συνήθως, πρέπει να τερματίζουν.

Συντρέχοντα συστήματα

- Εμπεριέχουν πολλούς υπολογιστικούς πράκτορες.
- Ο τερματισμός δυνατόν να υποδηλώνει κάποια ανωμαλία (deadlock, interrupt).
- Μπορεί να χρησιμοποιούν ενέργεια από πολλαπλές πηγές.
- Δυνατόν να υπάρχουν απομακρυσμένες συνιστώσες.
- Αλληλεπίδραση με τους χρήστες (Reactive).
- Μπορεί να εμπεριέχει λογισμικό (Embedded).
- multiprogramming vs multiprocessing
- *Μη-ντετερμινισμός* (nondeterminism): από κάποιο σημείο πιθανόν να υπάρχουν περισσότερες από μία δυνατές εξελίξεις του συστήματος

Συστήματα Μεταβάσεων

- *Κατάσταση*: βασική έννοια στη μοντελοποίηση συστημάτων
  - Συλλαμβάνει πληροφορίες για το σύστημα κάποια δεδομένη στιγμή, π.χ.
    - ανάθεση τιμών στις μεταβλητές ενός προγράμματος, ή/και ανάθεση τιμών σε επιπλέον χρήσιμες μεταβλητές, όπως program counter, ουρές, κλπ
    - ιδιότητες που ικανοποιεί το σύστημα τη δεδομένη στιγμή
- Αρχικές καταστάσεις – τελικές καταστάσεις
- *Συμπεριφορά* – Βήματα που μπορούν να εκτελεστούν σε κάθε κατάσταση.
- *Σύστημα μεταβάσεων*: περιγράφει τις δυνατές καταστάσεις από τις οποίες μπορεί να διέλθει το σύστημα καθώς και τις μεταβάσεις που μπορούν να γίνουν μεταφέροντας το σύστημα από μία κατάσταση σε κάποια άλλη.

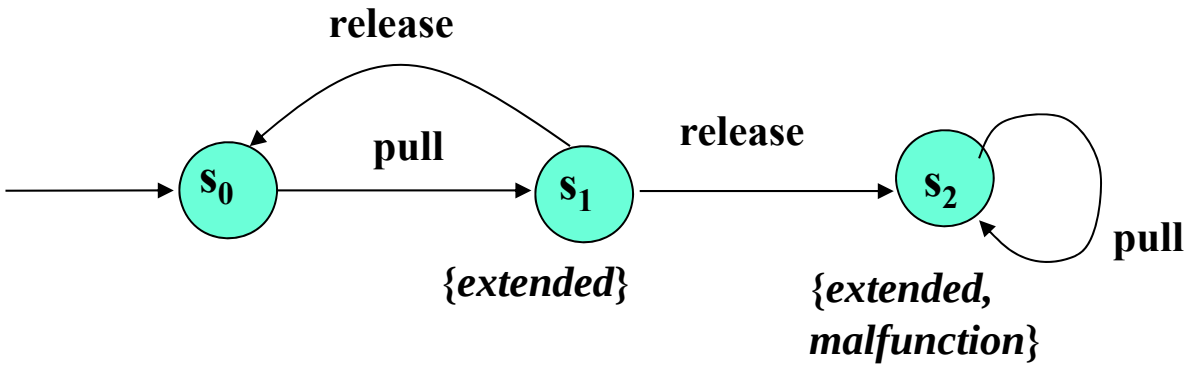
Συστήματα μεταβάσεων

- Ένα σύστημα μεταβάσεων είναι μια πλειάδα

$$T = (S, Act, \rightarrow, S_0, AP, L)$$

  - S είναι ένα αριθμήσιμο σύνολο από καταστάσεις
  - Act, είναι ένα σύνολο ενεργειών
  - $\rightarrow : S \times Act \times S$ είναι το σύνολο των μεταβάσεων, όπου $(s, \alpha, s') \in \rightarrow$ αν από την κατάσταση s μπορούμε να εκτελέσουμε την ενέργεια α και να μεταβούμε στην κατάσταση s'
  - $S_0 \subseteq S$ είναι το σύνολο των αρχικών καταστάσεων
  - AP είναι ένα σύνολο από ατομικές προτάσεις
  - $L : S \rightarrow 2^{AP}$ είναι μια συνάρτηση η οποία συνδέει κάθε κατάσταση με τις ατομικές προτάσεις τις οποίες ικανοποιεί.

Παράδειγμα – Ελατήριο



- Καταστάσεις: S = $\{s_0, s_1, s_2\}$
- Αρχικές Καταστάσεις: $S_0 = \{s_0\}$
- Ενέργειες: Act = {release, pull}
- Σχέση μεταβάσεων: $\rightarrow = \{(s_0, \text{pull}, s_1), (s_1, \text{release}, s_0), (s_1, \text{release}, s_2), (s_2, \text{pull}, s_2)\}$,
- Ατομικές Προτάσεις: AP = {extended, malfunction}
- $L(s_0) = \{\}$, L(s1) = {extended}, $L(s_2)$ = {extended, malfunction}

Μοντελοποίηση προγραμμάτων

- Πως μπορούμε να κατασκευάσουμε συστήματα μεταβάσεων για προγράμματα στα οποία η συμπεριφορά εξαρτάται άμεσα από τις τιμές των μεταβλητών (data-driven computation);

- Έστω ένα πρόγραμμα με μεταβλητές στο σύνολο $V=\{v_0, v_1, v_2, \ldots\}$.

- *Κατάσταση* του προγράμματος: μία ανάθεση τιμών στις μεταβλητές. Πεδίο καταστάσεων του προγράμματος: το σύνολο όλων των δυνατών καταστάσεων στις οποίες μπορεί να βρεθεί.

- Για παράδειγμα, αν $V = \{a,b,c\}$ και οι μεταβλητές είναι τύπου int, τότε το πεδίο καταστάσεων περιέχει τις καταστάσεις:

  <a=0,b=0,c=0> <a=1,b=0,c=0>

  <a=1,b=1,c=0> <a=932,b=5609,c=6658> …

Μοντελοποίηση προγραμμάτων - Μεταβάσεις

- Για να συλλάβουμε τις μεταβάσεις του συστήματος, μπορούμε να χρησιμοποιήσουμε τη γλώσσα φρουρημένων εντολών του Dijkstra (guarded command language)
- Μία *φρουρημένη εντολή* έχει τη μορφή

$$p \quad \rightarrow \quad (v_1,v_2,\ldots,v_n):=(e_1,e_2,\ldots,e_n)$$

όπου p είναι η συνθήκη για να εκτελεστεί η μετάβαση με βήμα την πολλαπλή ανάθεση $(v_1,v_2,\ldots,v_n):=(e_1,e_2,\ldots,e_n)$

- Παράδειγμα:

$$a>b \quad \rightarrow \quad (c,d):=(d,c)$$

Αν a>b η μετάβαση εκτελείται με αποτέλεσμα την ανταλλαγή των τιμών των c και d.

Γράφος Προγράμματος

- Ένας *γράφος προγράμματος* αποτελεί μια πλειάδα (V, S, T, p) όπου
  - V είναι ένα σύνολο μεταβλητών
  - S είναι ένα σύνολο καταστάσεων
  - T είναι ένα πεπερασμένο σύνολο φρουρημένων εντολών.
  - p είναι η αρχική συνθήκη

- Το πρόγραμμα ξεκινά από καταστάσεις που ικανοποιούν την αρχική συνθήκη.

Εκτελέσεις

- Μία *εκτέλεση* είναι μία πεπερασμένη ή μη-πεπερασμένη ακολουθία από καταστάσεις

$$s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow \ldots$$

όπου

  - η αρχική κατάσταση ικανοποιεί την αρχική συνθήκη, δηλαδή, $p(s_0)$.
  - Από την κατάσταση s_i επάγεται η κατάσταση s_{i+1} μέσω εκτέλεσης μιας μετάβασης $e \rightarrow t$ όπου :
    - $e(s_i)$, δηλαδή, η κατάσταση s_i ικανοποιεί τη συνθήκη e, και
    - s_{i+1} λαμβάνεται εφαρμόζοντας το βήμα t στην κατάσταση s_i.

- Από ένα γράφο προγράμματος, μπορούμε να δημιουργήσουμε το σύστημα μεταβάσεων που αντιστοιχεί στο πρόγραμμα.
  - Ξεκινούμε από την αρχική κατάσταση/τις αρχικές καταστάσεις που ικανοποιεί/ούν την αρχική συνθήκη p, και
  - Εφαρμόζουμε όλες τις δυνατές μεταβάσεις σύμφωνα με το T προσθέτοντας καινούριες καταστάσεις και ακμές στο σύστημα μεταβάσεων όπως χρειάζεται.

Παράδειγμα 1

- Θεωρήστε τον πιο κάτω γράφο προγράμματος:

  V = {a, b, c, d, e}

  S: όλες οι αναθέσεις ακεραίων στις μεταβλητές V

  T={ c>0 $\rightarrow$ (c,e):=(c-1,e+1),

  d>0 $\rightarrow$ (d,e):=(d-1,e+1) }

  p: c=a $\wedge$ d=b $\wedge$ e=0

- Ποια η λειτουργία του προγράμματος που μοντελοποιείται από το πιο πάνω σύστημα;

  Αναθέτει στην μεταβλητή e το άθροισμα των c και d, και στις μεταβλητές c και d την τιμή 0 και τερματίζει.

Παράδειγμα 1

- s_0 = <a=2, b=1, c=2, d=1, e=0>
  (ικανοποιεί την αρχική συνθήκη)
- s_1 = <a=2, b=1, c=1, d=1, e=1>
  (Εκτέλεση της πρώτης μετάβασης)
- s_2= <a=2, b=1, c=1, d=0, e=2>
  (Εκτέλεση της δεύτερης μετάβασης)
- s_3 = <a=2, b=1 ,c=0, d=0, e=3>
  (Δεύτερη εκτέλεση της πρώτης μετάβασης)

Παράδειγμα 2

Έστω το πιο κάτω πρόγραμμα που αποτελείται από δύο παράλληλες διεργασίες.

```
while True do
     wait(Turn=0);
     ...
     Turn=1
endwhile
||
while True do
     wait(Turn=1);
     ...
     Turn=0
endwhile
```

- Οι δύο διεργασίες περιμένουν τη σειρά τους για να εκτελέσουν κάποιο υπολογισμό κατά τον οποίο ζητούν αποκλειστική χρήση των πόρων του συστήματος.
- Όταν Turn=0 δίνεται άδεια πρόσβασης στην πρώτη διεργασία και αντίθετα όταν Turn = 1 την άδεια παίρνει η δεύτερη διεργασία.
- Ερωτήματα:
  - Υπάρχει περίπτωση κάποια διεργασία να φθάσει σε αδιέξοδο περιμένοντας επ'άπειρω να της δοθεί άδεια πρόσβασης;
  - Υπάρχει περίπτωση να δοθεί και στις δύο διεργασίες άδεια ταυτόχρονα;

Παράδειγμα 2 (συν.)

- Για να δημιουργήσουμε τον γράφο προγράμματος του συστήματος προσθέτουμε μεταβλητές PC_0 και PC_1 (program counters) που παίρνουν τιμές οι οποίες δεικνύουν το σημείο εκτέλεσης στην κάθε διεργασία.

```
L0: while True do
    NC0: wait(Turn=0);
    CR0: Turn=1
endwhile ||
L1: while True do
    NC1: wait(Turn=1);
    CR1: Turn=0
endwhile
```

- Οι μεταβάσεις:

T_0: $PC_0=L_0 \rightarrow PC_0:=NC_0$

T_1: $PC_0=NC_0 \wedge Turn=0 \rightarrow PC_0:=CR_0$

T_2: $PC_0=CR_0 \rightarrow (PC_0, Turn):=(L_0, 1)$

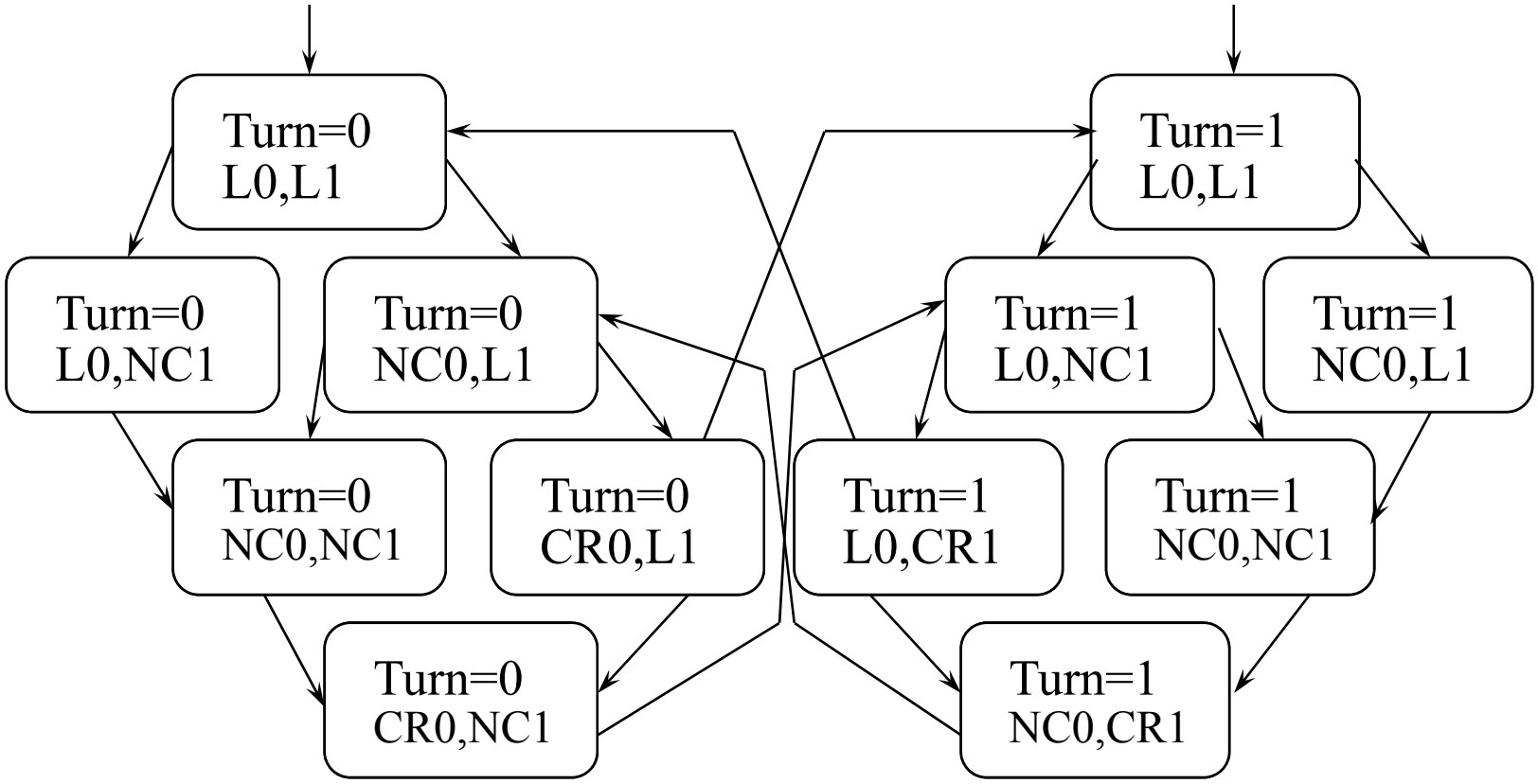
T_3: $PC_1=L_1 \rightarrow PC_1=NC_1$

T_4: $PC_1=NC_1 \wedge Turn=1 \rightarrow PC_1:=CR1$

T_5: $PC_1= CR_1 \rightarrow (PC_1, Turn):=(L_1, 0)$

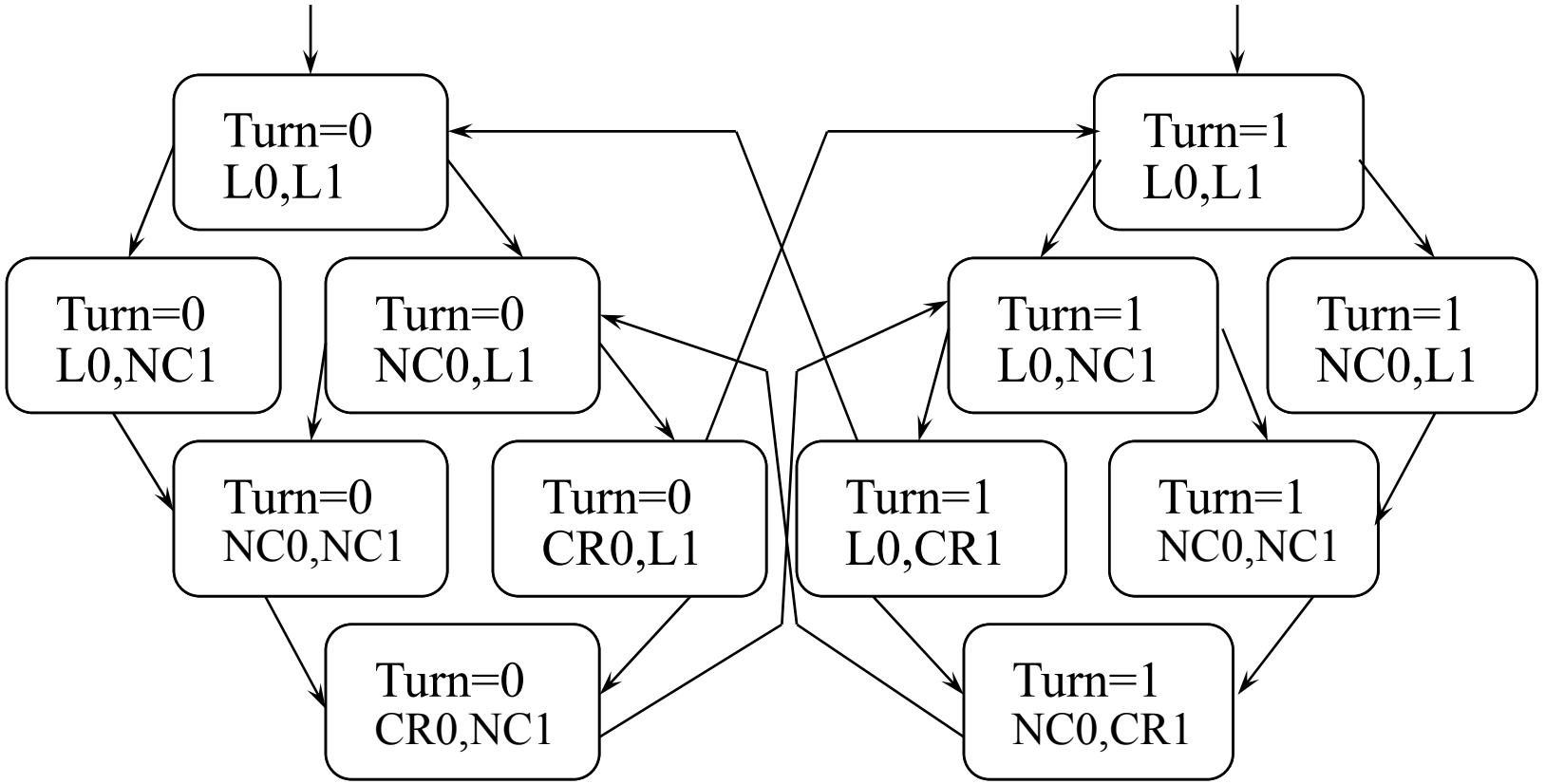
Αρχικά : $PC_0=L_0 \wedge PC_1=L_1$

Παράδειγμα 2 – Το σύστημα μεταβάσεων

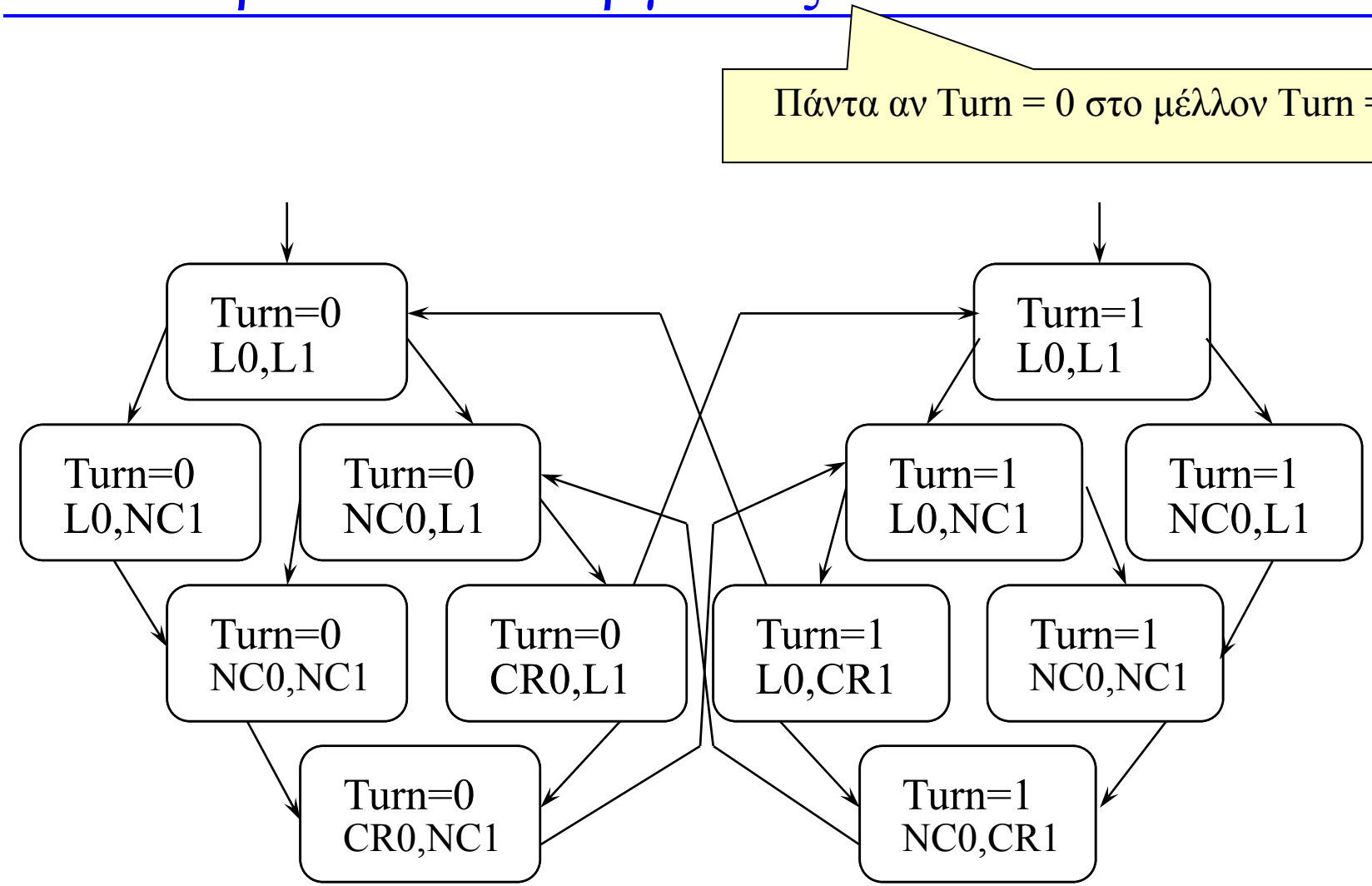


Παράδειγμα 2 – Αμοιβαίος αποκλεισμός

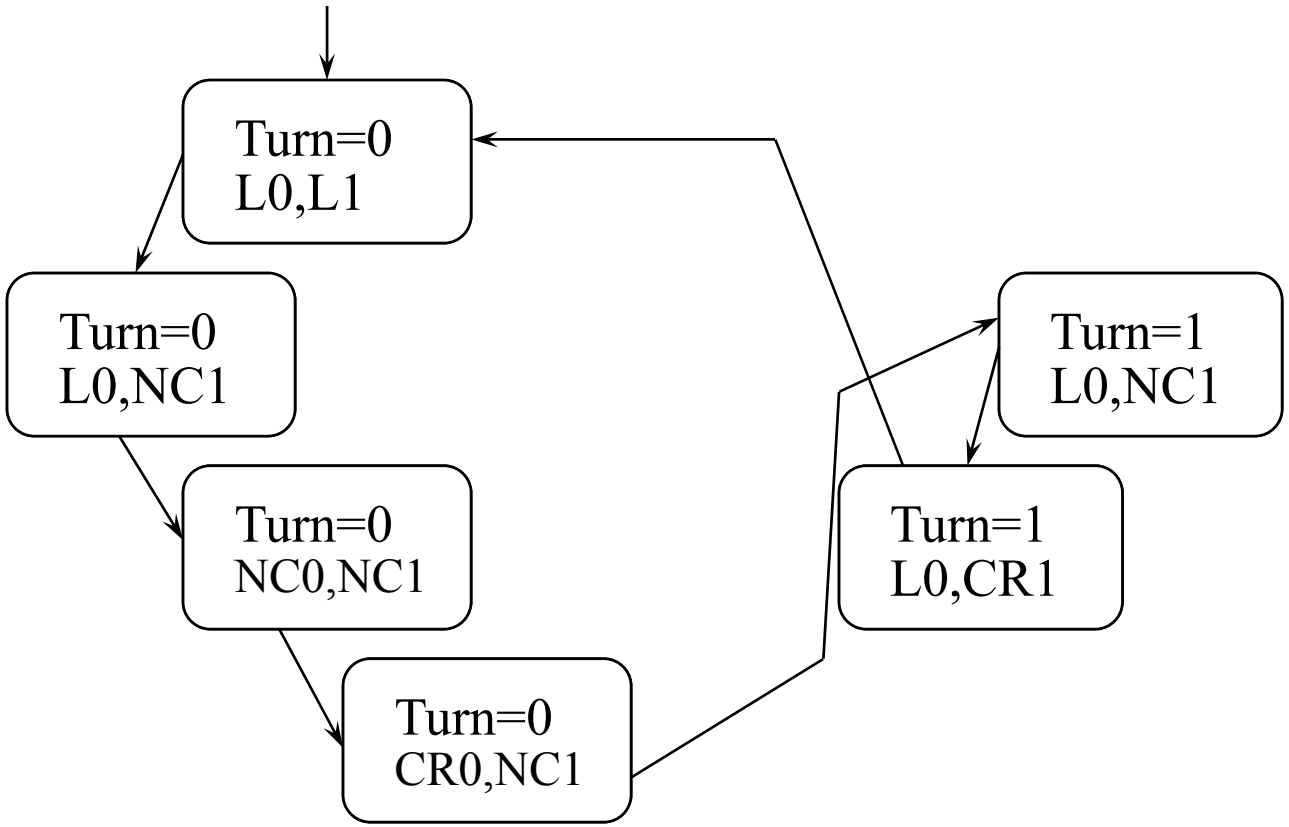
Ποτέ δεν ισχύει ότι και οι δύο διεργασίες έχουν πάρει ταυτόχρονα άδεια



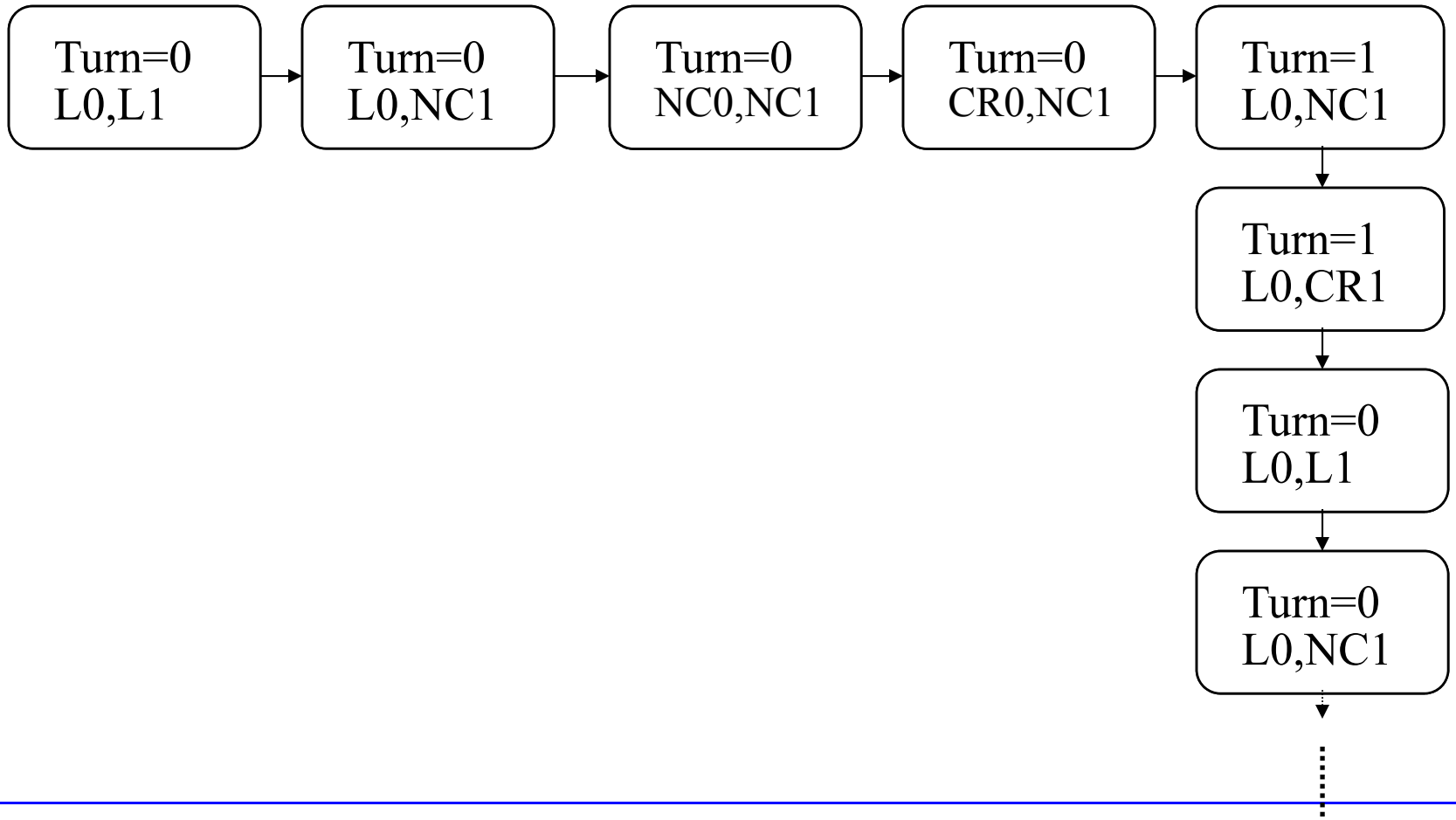
Παρ. 2 – Οι διεργασίες εναλλάσσονται



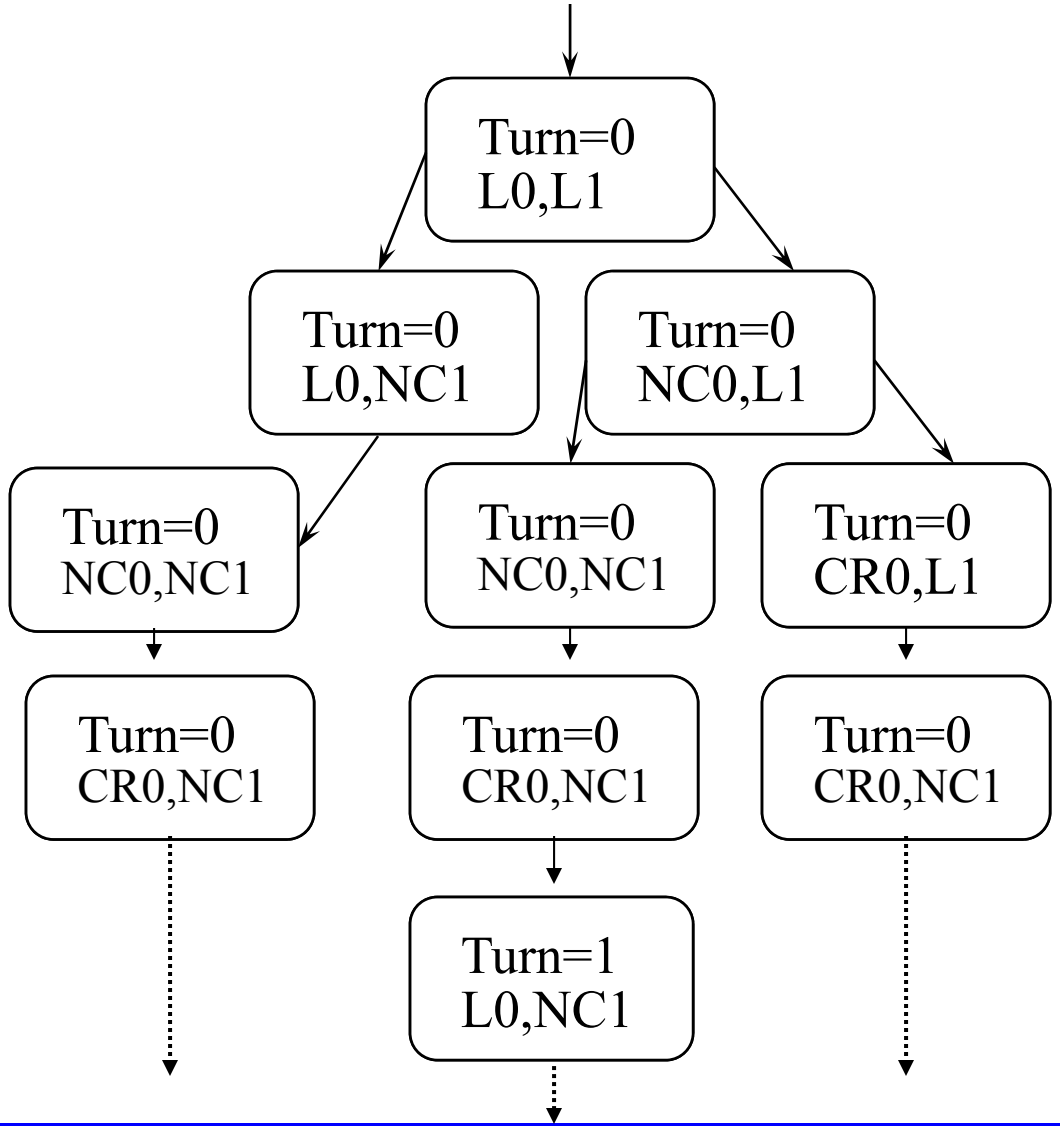
Παράδειγμα *εκτέλεσης* του συστήματος



Παράδειγμα *εκτέλεσης* του συστήματος



(Ενα) *ξεδίπλωμα* του συστήματος μεταβάσεων



Εργασία

Ο πιο κάτω αλγόριθμος επιχειρεί να επιβάλει αμοιβαίο αποκλεισμό ανάμεσα σε δύο διεργασίες P_1 και P_2 κάθε μία από τις οποίες τρέχει τον πιο κάτω κώδικα. Μπορείτε να υποθέσετε ότι αρχικά sema = 0.

```
1.    while true do{
2.        atomic {if sema = 0
3.                        then sema := 1
4.                else
5.                        go to line 2}
6.        critical section;
7.        sema := 0;
8.    }
```

(α) Μοντελοποιήστε το πρωτόκολλο ως ένα γράφο προγράμματος.

(β) Αποδώστε τον γράφο προγράμματος του πρωτοκόλλου γραφικά ως ένα σύστημα μεταβάσεων.

(γ) Συζητήστε την καταλληλότητα του πρωτοκόλλου για διασφάλιση αμοιβαίου αποκλεισμού στην παρουσία 2 ή περισσότερων διεργασιών

Παράδειγμα 2 (συν.) – Δικαιοσύνη

```
L0: while True do
    NC0: wait(Turn=0);
    CR0: Turn=1
endwhile ||
L1: while True do
    NC1: wait(Turn=1);
    CR1: Turn=0
endwhile
```

- Οι μεταβάσεις:

T_0: $PC_0=L_0 \rightarrow PC_0:=NC_0$

T_1: $PC_0=NC_0 \wedge Turn=0 \rightarrow PC_0:=CR_0$

T_1': $PC_0=NC_0 \wedge Turn=1 \rightarrow PC_0:=NC_0$

T_2: $PC_0=CR_0 \rightarrow (PC_0,Turn):=(L_0,1)$

T_3: $PC_1=L_1 \rightarrow PC_1=NC_1$

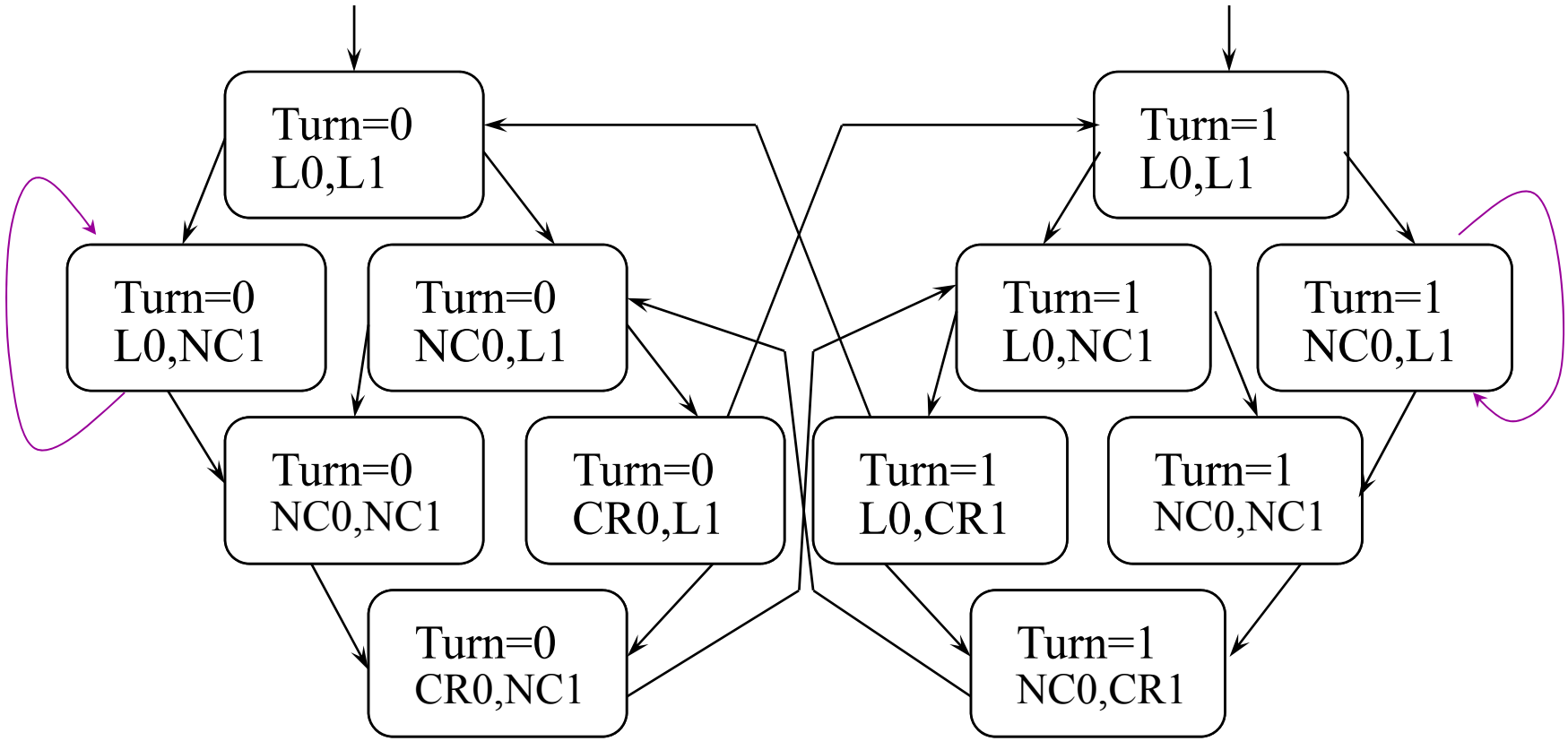
T_4: $PC_1=NC_1 \wedge Turn=1 \rightarrow PC_1:=CR1$

T_4': $PC_1=NC_1 \wedge Turn=0 \rightarrow PC_1:=NC1$

T_5: $PC_1= CR_1 \rightarrow (PC_1,Turn):=(L_1,0)$

Αρχικά : $PC_0=L_0 \wedge PC_1=L_1$

Οι διεργασίες εναλλάσσονται;



Δικαιοσύνη

- Περιορισμός στο σύνολο των εκτελέσεων υπό μελέτη
  - *Ασθενής δικαιοσύνη διεργασιών*: αποκλείει τις εκτελέσεις όπου, από μία κατάσταση και μετά, ενώ μία διεργασία είναι συνεχώς έτοιμη για εκτέλεση δεν εκτελείται ποτέ.
  - *Ασθενής δικαιοσύνη μεταβάσεων*: αποκλείει τις εκτελέσεις όπου, από μία κατάσταση και μετά, ενώ μία μετάβαση είναι συνεχώς έτοιμη για εκτέλεση δεν εκτελείται ποτέ.
  - *Ισχυρή δικαιοσύνη διεργασιών*: αποκλείει τις εκτελέσεις όπου μία διεργασία είναι έτοιμη για εκτέλεση για μη-πεπερασμένο αριθμό φορών (όχι κατ' ανάγκη συνεχόμενα) αλλά εκτελείται μόνο για πεπερασμένο αριθμό φορών.
  - *Ισχυρή δικαιοσύνη μεταβάσεων*: αποκλείει τις εκτελέσεις όπου μία μετάβαση είναι έτοιμη για εκτέλεση για μη-πεπερασμένο αριθμό φορών (όχι κατ' ανάγκη συνεχόμενα) αλλά εκτελείται μόνο για πεπερασμένο αριθμό φορών.

Παράδειγμα 3

```
P1::                    ||      P2::
 L1: x:=1                        L3: while y=0 do
 L2                              L4:      [z:=z+1
                                          OR
                                          if x=1 then y:=1]
                                     end while
                                  L5
```

Γράφος Προγράμματος

P1

A: PC1=L1 → (PC1, x) := (L2, 1)

P2

B: PC2=L3 ∧ y=0 → PC2 := L4

Γ: PC2=L4 → (PC2, z) := (L3, z+1)

Δ: PC2=L4 ∧ x=1 → (PC2, y) := (L3,1)

E: PC2=L4 ∧ x ≠ 1 → PC2:= L3

Z: PC2 = L3 ∧ y≠0 → PC2 := L5

Αρχικά : x=0 /\ y=0 /\ z=0

Παράδειγμα 3

```
P1::x:=1      ||      P2::while y=0 do
                              [z:=z+1
                                OR
                               if x=1 then y:=1]
                            end while
```

Αρχικά : x=0 /\ y=0 /\ z=0

Η P1 τερματίζει πάντα (σε κάθε εκτέλεση του συστήματος); Η P2;

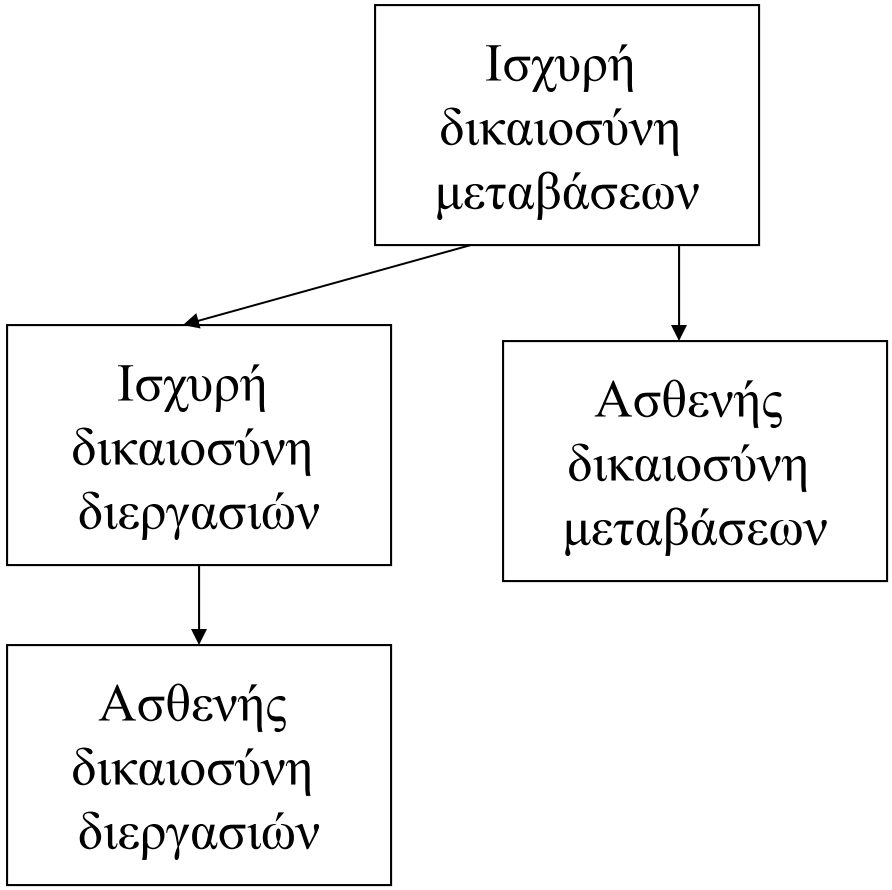
Χωρίς δικαιοσύνη; Δεν υπάρχουν εγγυήσεις

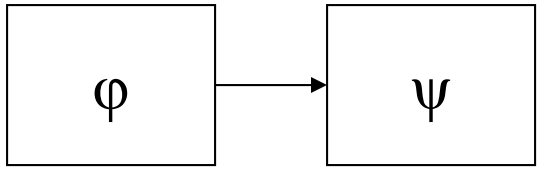
Ασθενής δικαιοσύνη μεταβάσεων (διεργασιών); Τερματίζει η P1

Ισχυρή δικαιοσύνη διεργασιών; Τερματίζει η P1

Ισχυρή δικαιοσύνη μεταβάσεων; Τερματίζουν και η P1 και η P2

Ιεραρχία των ιδιοτήτων δικαιοσύνης





Γράφουμε ότι η φ συνεπάγεται τη ψ αν κάθε εκτέλεση που είναι δίκαια ως προς τη φ είναι δίκαια και ως προς τη ψ.

Σημασιολογία παρεμβαλλόμενης διάταξης

- Σημασιολογία *παρεμβαλλόμενης διάταξης* (interleaving model): μέθοδος αναπαράστασης παραλληλισμού/ταυτοχρονισμού όπου ανά πάσα στιγμή μπορεί να συμβεί μόνο μία μετάβαση.
- Απλό μοντέλο
- Συνοδεύεται από μια πλειάδα μαθηματικών εργαλείων ανάλυσης συστημάτων.
- Και αν δύο μεταβάσεις a και b μπορούν να πραγματοποιηθούν ταυτόχρονα;
  - Στο σύστημα μεταβάσεων παρουσιάζονται δύο εκτελέσεις: μία στην οποία η a εκτελείται πριν από τη b και μία στην οποία η b εκτελείται πριν από την a.
  - Αυτό είναι ικανοποιητικό για την πλειονότητα των συστημάτων που μας ενδιαφέρουν.

Σημασιολογία μερικής διάταξης

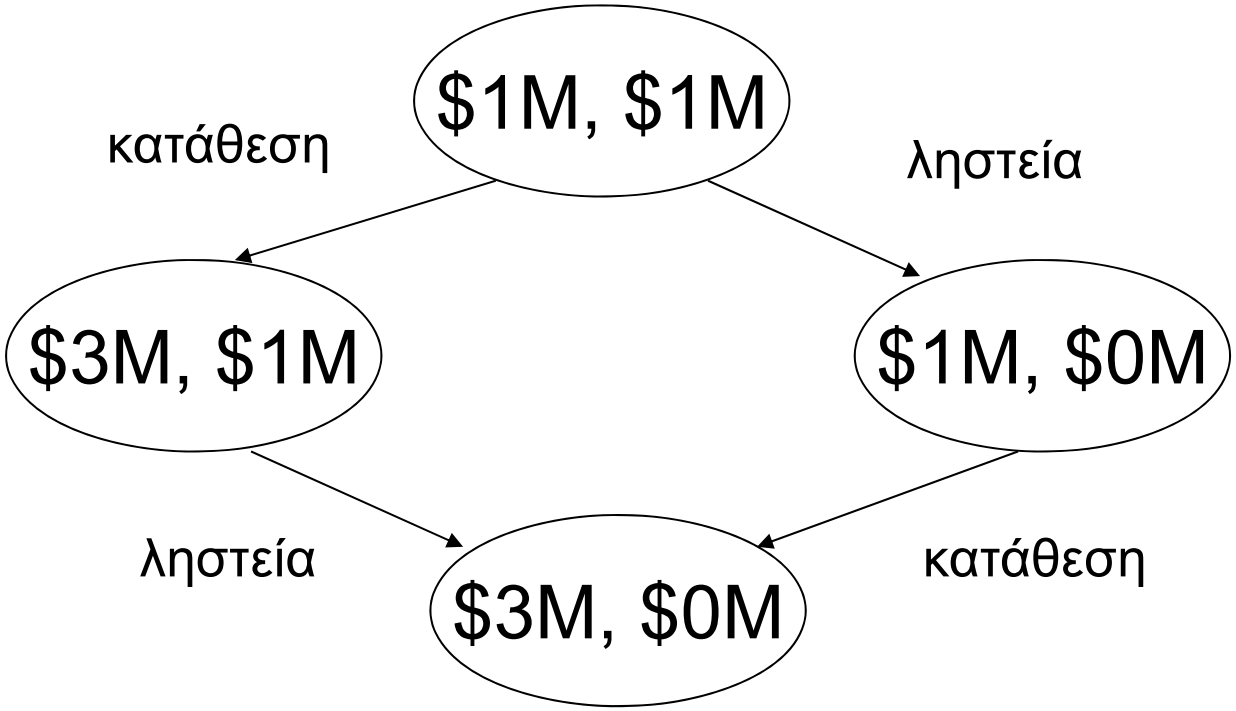
- "true concurrency"
- Δεν υπάρχει ολική διάταξη ανάμεσα στα γεγονότα.
- Πιο διαισθητική. Πιο κοντά στην πραγματική συμπεριφορά του συστήματος
- Δυσκολότερο να αναλυθεί.
- Λιγότερα μαθηματικά αποτελέσματα.
- Μερική διάταξη: (S , <), όπου η σχέση < είναι
  - Αν x<y και y<z τότε x<z. (μεταβατική)
  - Για κανένα ζεύγος x και y δεν ισχύει x<y /\ y<x (αντισυμμετρική).
  - Για κανένα x δεν ισχύει x<x
- Διαισθητικά, x < y αν το x είναι προαπαιτούμενο για το y.

Παράδειγμα 4

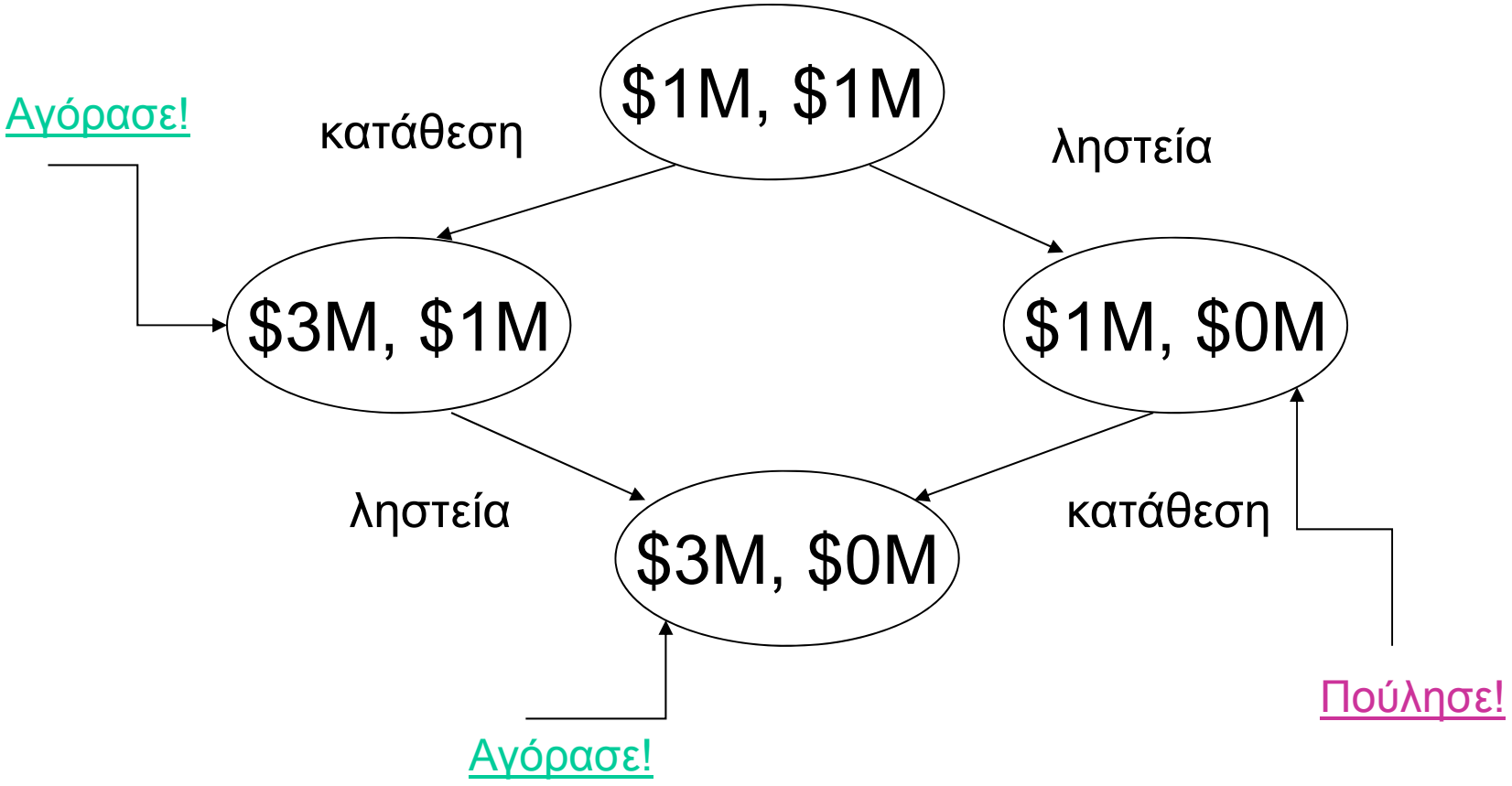
- Μία τράπεζα έχει δύο παραρτήματα, αρχικά $1M στο κάθε ένα.
- Ταυτόχρονα
  - Στο παράρτημα 1: κατάθεση $2M.
  - Στο παράρτημα 2: ΛΗΣΤΕΙΑ.
- Πως μπορεί να μοντελοποιηθεί το σύστημα;
- Θέλουμε να αγοράσουμε μετοχές της τράπεζας αν τα περιουσιακά της στοιχεία στα δύο παραρτήματα ξεπερνούν τα $2M, διαφορετικά, να πουλήσουμε τις μετοχές που έχουμε.

Σημασιολογία Παρεμβαλλόμενης Διάταξης

- Κάθε κατάσταση συλλαμβάνει και τα δύο παραρτήματα.

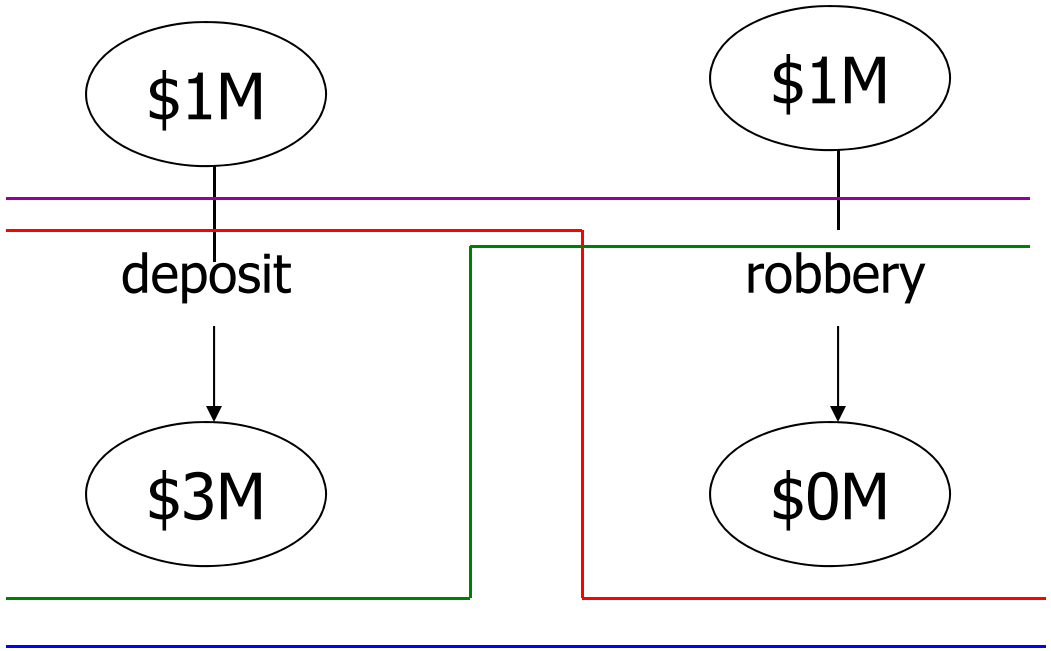


Παράδειγμα 4: Επένδυση



Σημασιολογία Μερικής Διάταξης

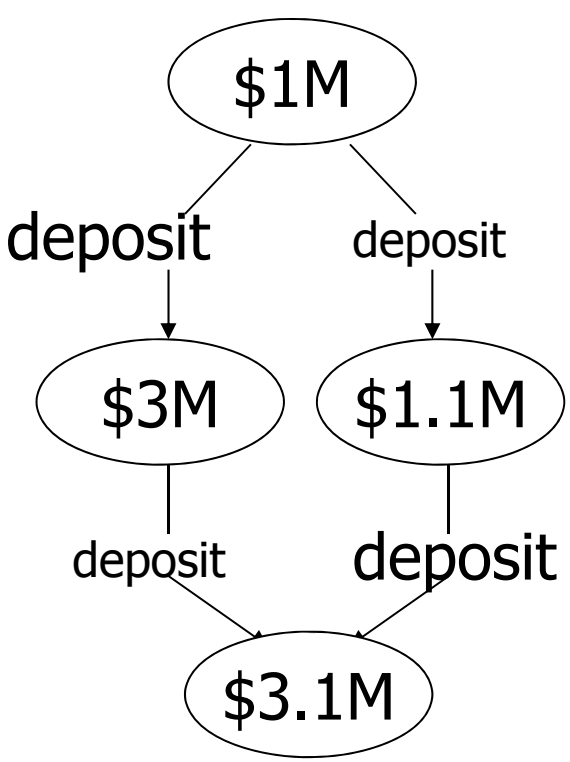
- Δύο συνιστώσες μοντελοποιούν τα δύο ανεξάρτητα παραρτήματα.

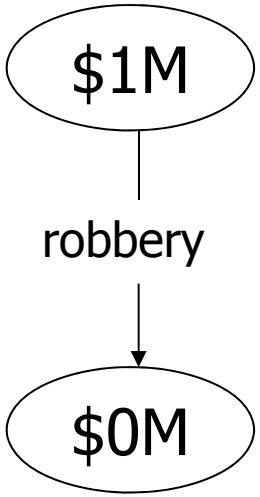


Ταυτοχρονισμός vs μη-ντετερμινισμός

Παράρτημα με ένα υπάλληλο

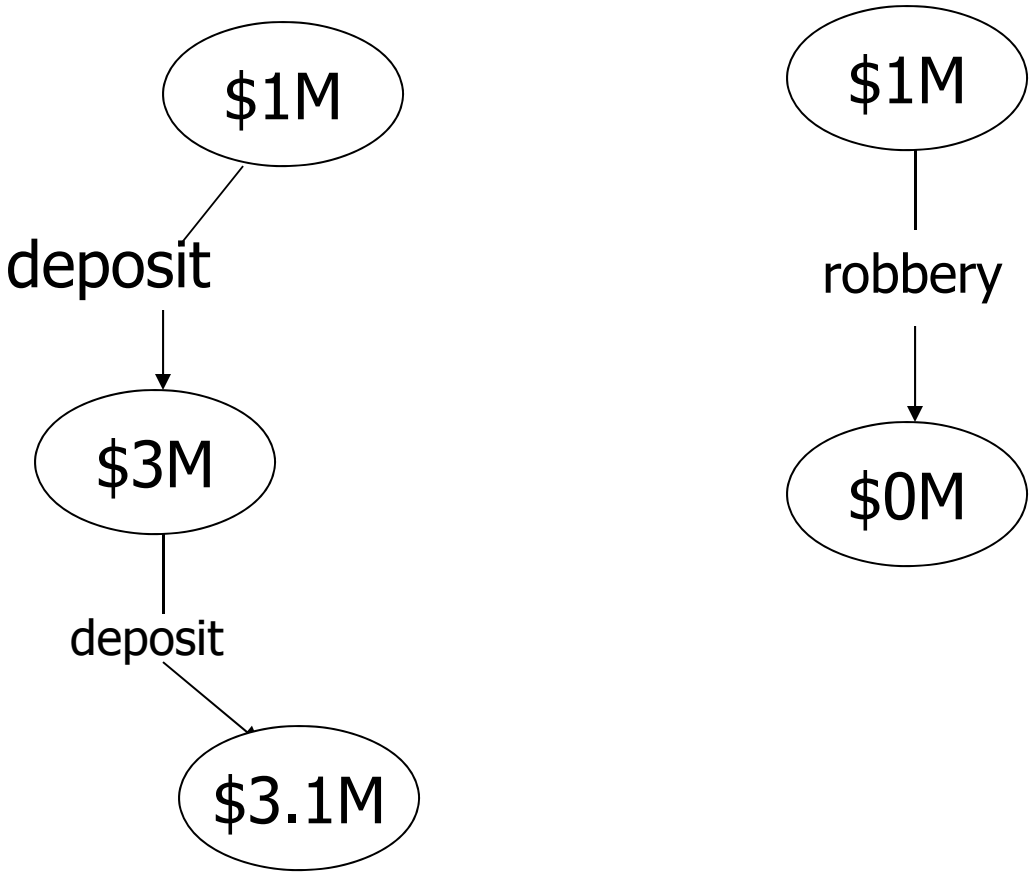
- Έστω ότι στο παράρτημα 1, παρουσιάζονται ταυτόχρονα δύο πελάτες για καταθέσεις $2M και $.1M ο καθένας και υπάρχει μόνο ένας υπάλληλος για να τους εξυπηρετήσει



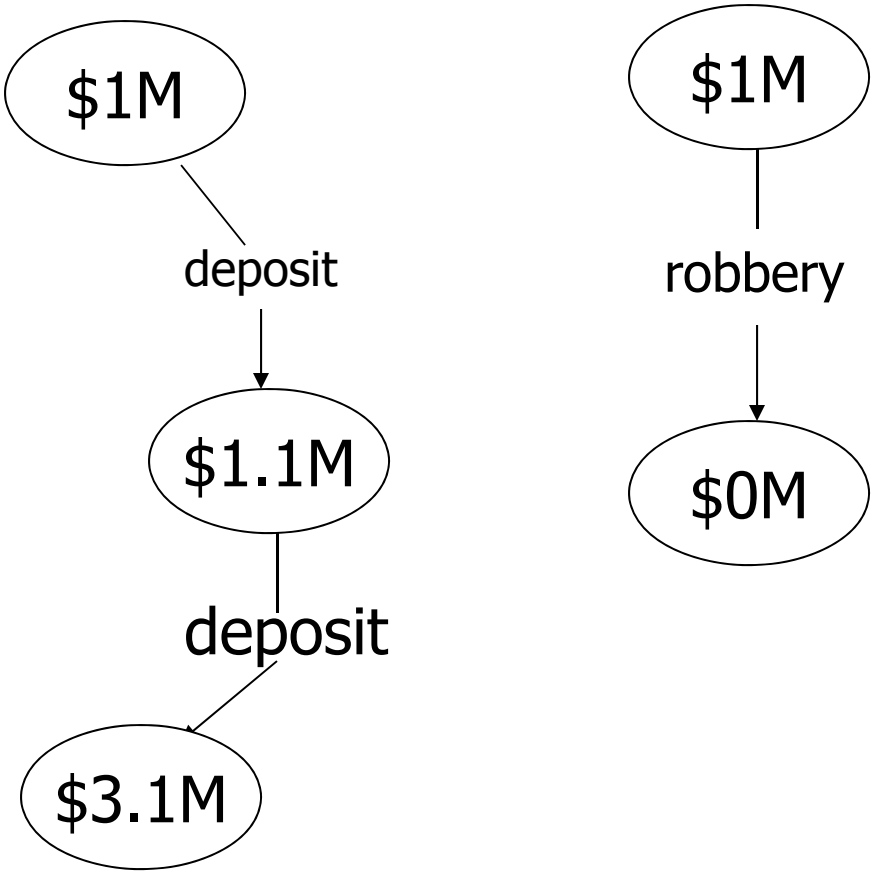


Διαχωρίζεται ο ταυτοχρονισμός από το μη-ντετερμινισμό

Εκτέλεση μερικής διάταξης 1



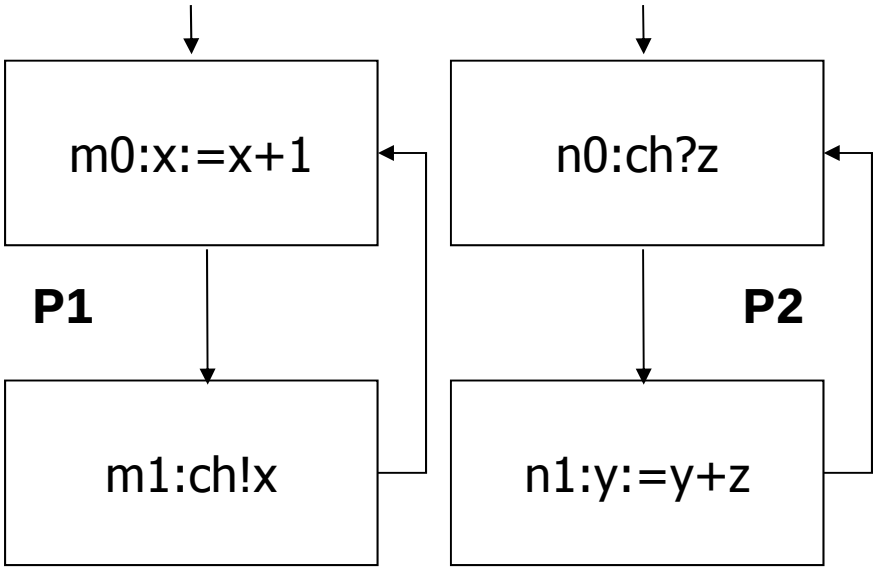
Εκτέλεση μερικής διάταξης 2



Μοντελοποίηση με μερική διάταξη

- Έστω δύο διεργασίες P1, P2 και κανάλι επικοινωνίας ανάμεσα στις δύο, ch.
- Γράφουμε ch!x για την αποστολή του μηνύματος x στο κανάλι ch και ch?y για την παραλαβή μηνύματος στο κανάλι ch το οποίο αποθηκεύεται στη μεταβλητή y.
- Το σύστημα λειτουργεί ως εξής:
  - Η P1 αυξάνει την τιμή του x κατά 1 και τη στέλνει στην P2 μέσω μιας ουράς ch,
  - Η P2 λαμβάνει κάποια τιμή στην ουρά ch και την προσθέτει στην τιμή του y
- Έστω ότι η επικοινωνία στην ουρά ch είναι συγχρονισμένη.

Μοντελοποίηση με μερική διάταξη



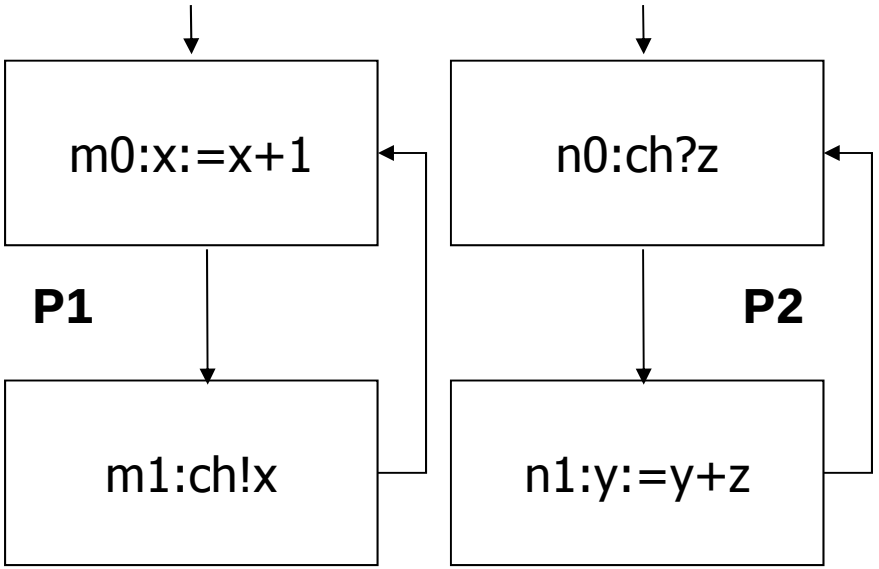
- Υπάρχουν τρεις δυνατές μεταβάσεις:

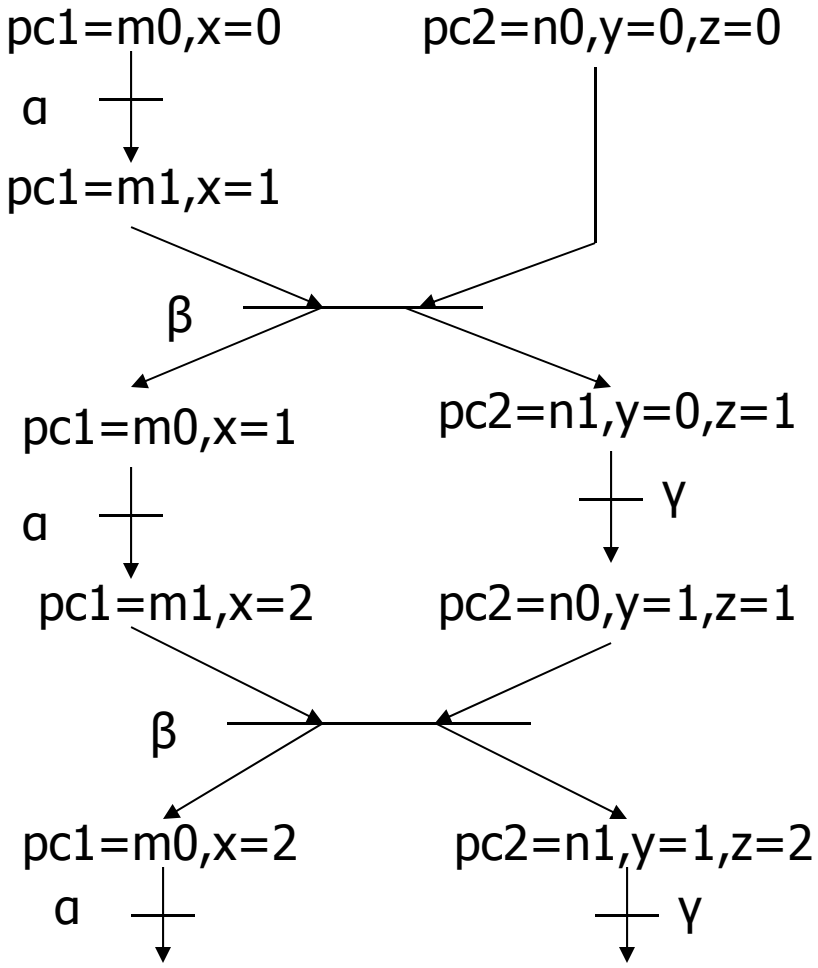
α: pc1=m0 → (pc1,x):= (m1, x+1)

β: pc1=m1 ∧ pc2=n0 → (pc1, pc2, z):=(m0, n1,x)

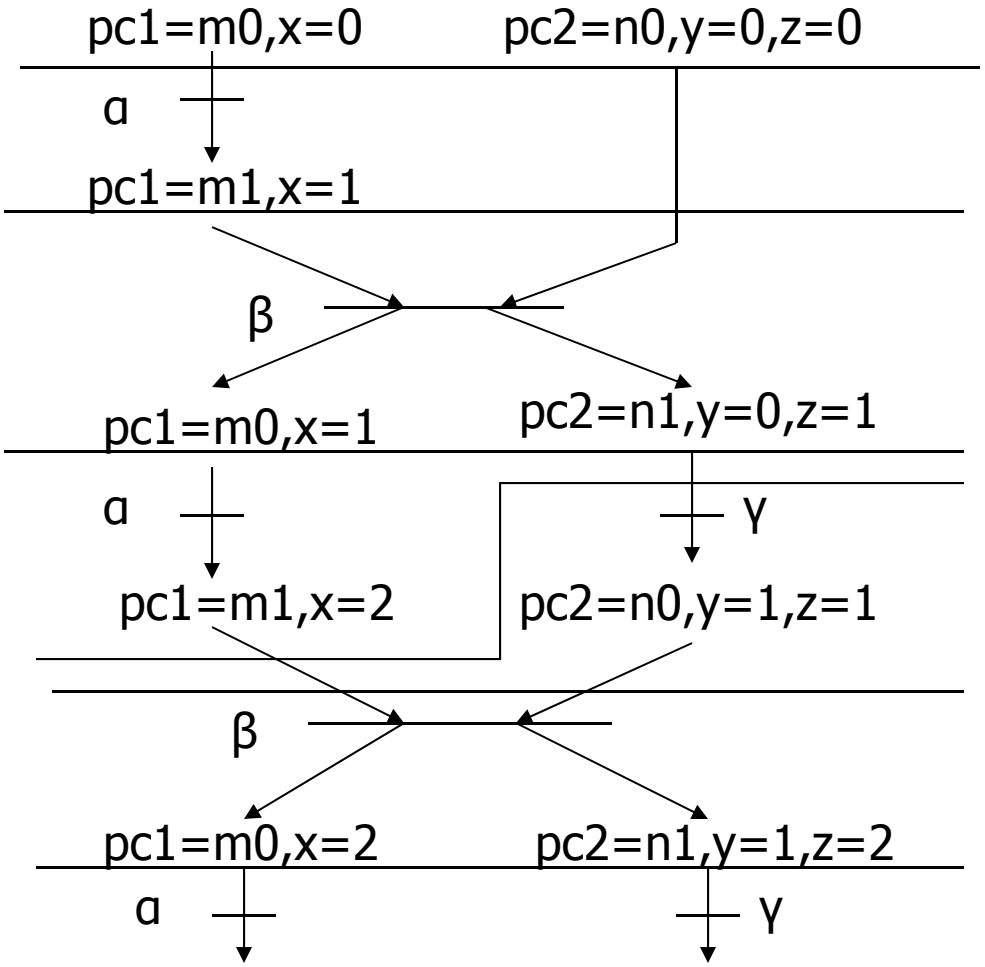
γ: pc2=n1 → (pc2, y):= (n0, y+z)

Μοντελοποίηση με μερική διάταξη





Εκτελέσεις



pc1=m0,x=0,pc2=n0,y=0,z=0

pc1=m1,x=1,pc2=n0,y=0,z=0

pc1=m0,x=1,pc2=n1,y=0,z=1

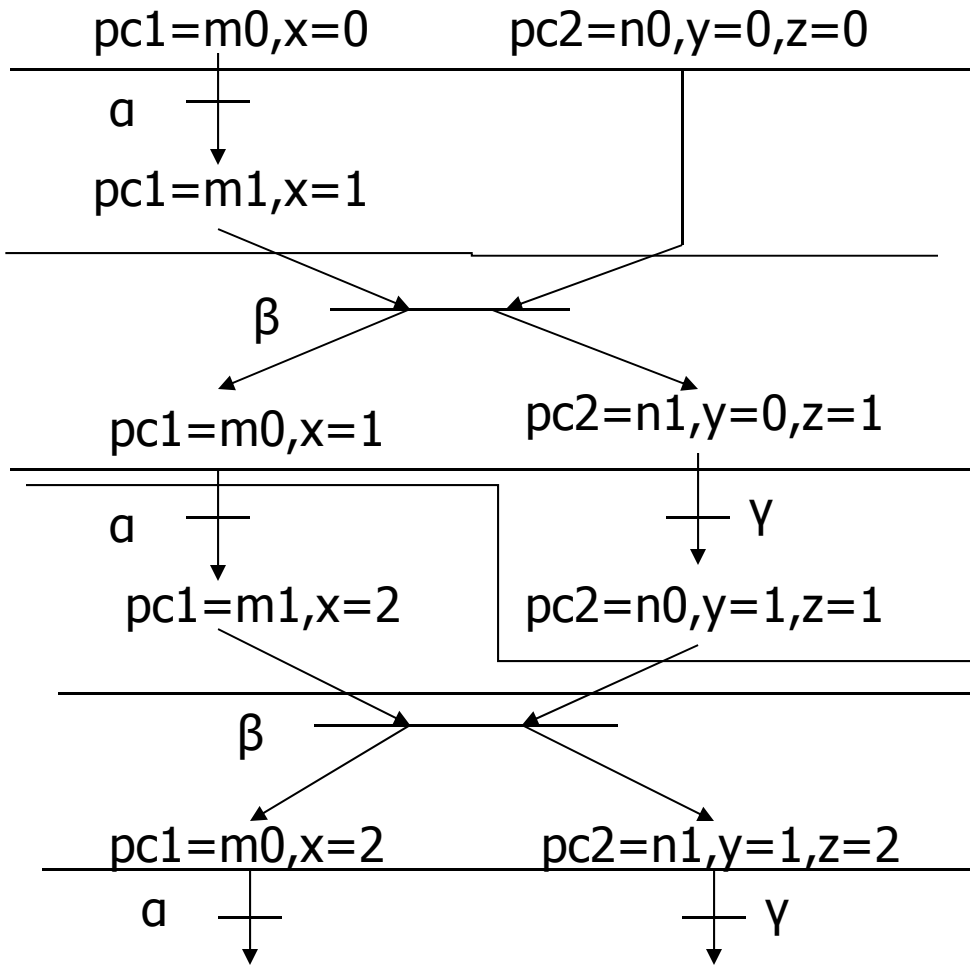
pc1=m1,x=2,pc2=n1,y=0,z=1

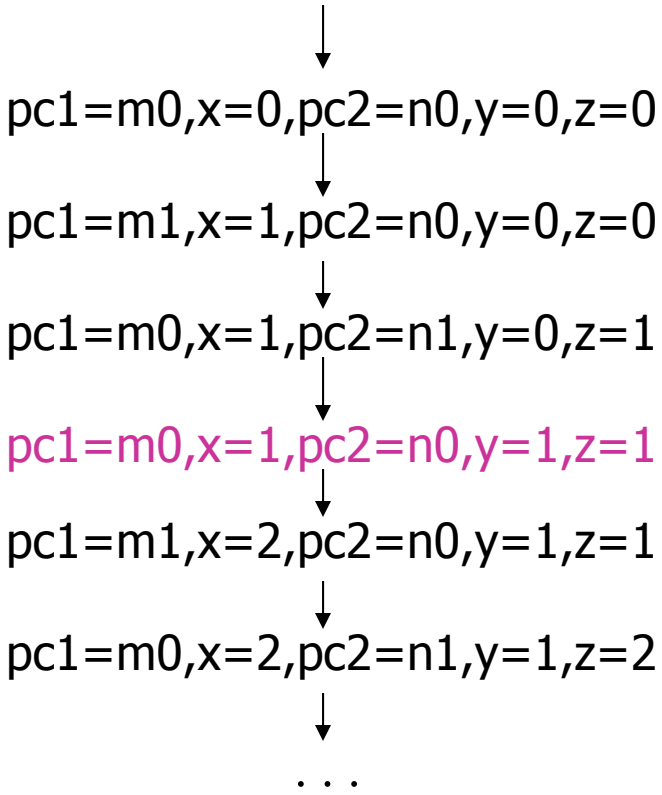
pc1=m1,x=2,pc2=n0,y=1,z=1

pc1=m0,x=2,pc2=n1,y=1,z=2

. . .

Εκτελέσεις


pc1=m0,x=0
pc2=n0,y=0,z=0
α
pc1=m1,x=1
β
pc1=m0,x=1
pc2=n1,y=0,z=1
α
γ
pc1=m1,x=2
pc2=n0,y=1,z=1
β
pc1=m0,x=2
pc2=n1,y=1,z=2
α
γ

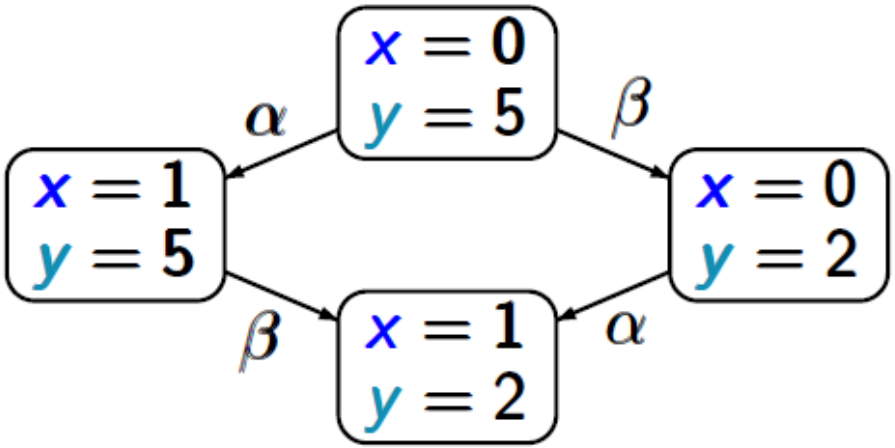

pc1=m0,x=0,pc2=n0,y=0,z=0
pc1=m1,x=1,pc2=n0,y=0,z=0
pc1=m0,x=1,pc2=n1,y=0,z=1
pc1=m0,x=1,pc2=n0,y=1,z=1
pc1=m1,x=2,pc2=n0,y=1,z=1
pc1=m0,x=2,pc2=n1,y=1,z=2
. . .

Μη Ντετερμινισμός

- Στην παρεμβαλλόμενη διάταξη ο μη ντετερμινισμός δυνατόν να επιλύει
  - ταυτοχρονισμό
- Επιπλέον, ο μη ντετερμινισμός δυνατόν να σχετίζεται με τα πιο κάτω, τόσο στη σημασιολογία παρεμβαλλόμενης διάταξης όσο και στη σημασιολογία της μερική διάταξης.
  - Ανταγωνισμός ανάμεσα σε παράλληλες αλληλοεξαρτώμενες ενέργειες.
  - Ελευθερία στην υλοποίηση ή απλοποίηση μοντέλου
  - Ελλιπή γνώση

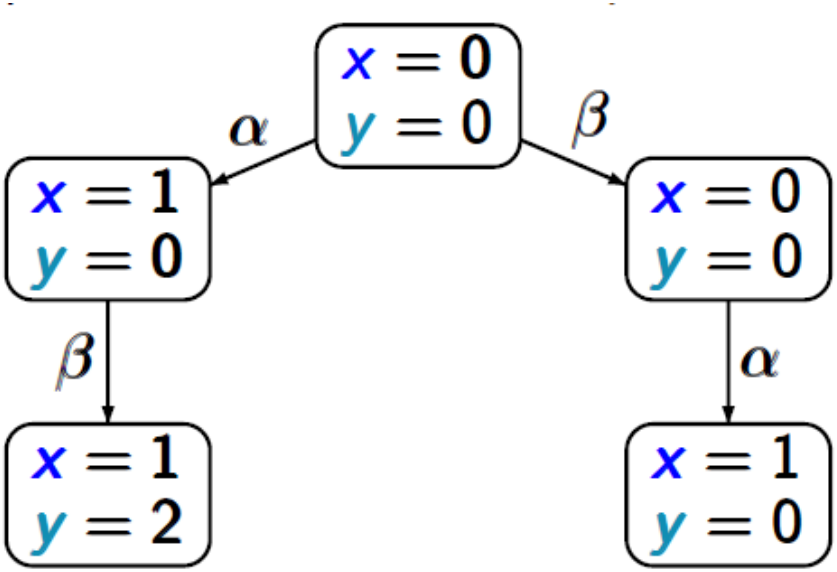
Ταυτοχρονισμός

$\alpha : x := x + 1 \quad \| \quad \beta : y := y - 3$



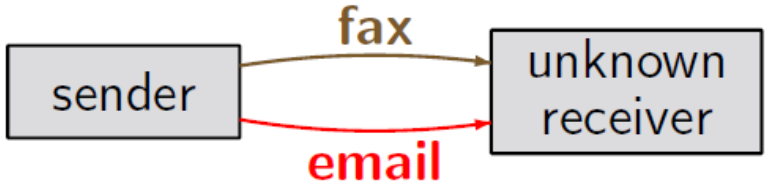
Αλληλοεξαρτώμενες ενέργειες

$\alpha : x := x + 1 \quad \| \quad \beta : y := 2x$

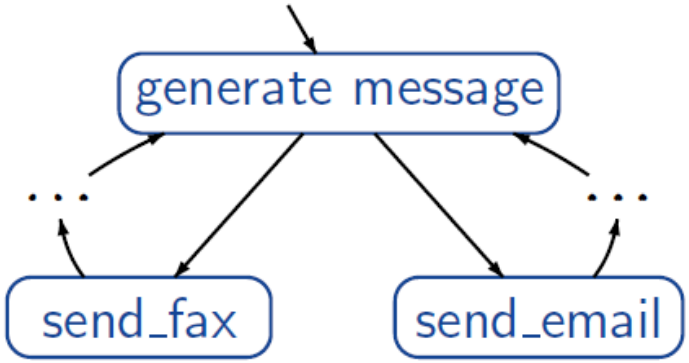


Ελευθερία στην υλοποίηση (1)

- Αβεβαιότητα για τον τύπο της επικοινωνίας που θα πρέπει να υλοποιηθεί:



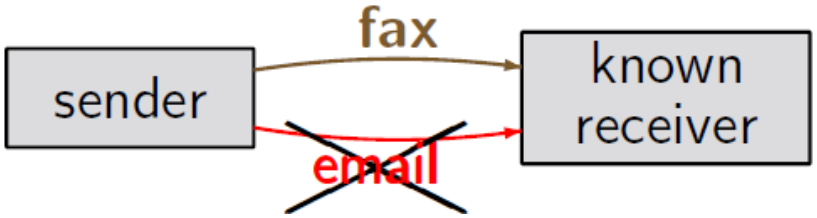
- Σύστημα μεταβάσεων:



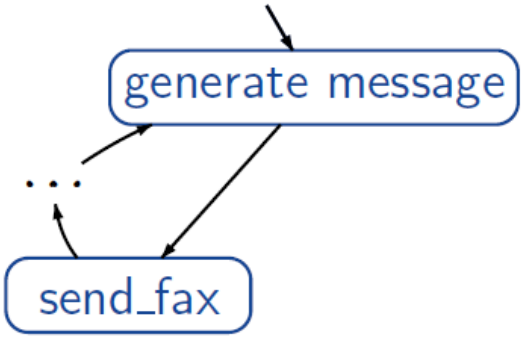
- Σε μεταγενέστερο στάδιο ο μη-ντετερμινισμός μπορεί να αντικατασταθεί από μία εκ των δύο επιλογών.

Ελευθερία στην υλοποίηση (2)

- Το σύστημα δεν θα υποστηρίζει ηλεκτρονικό ταχυδρομείο

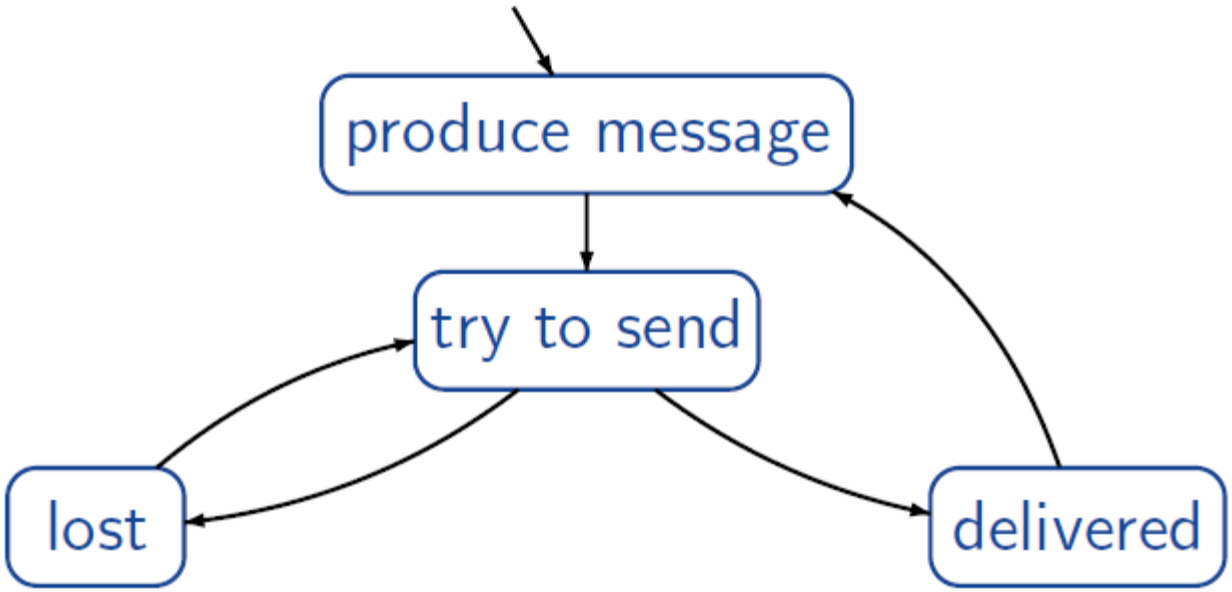


- Εκλεπτυσμένο σύστημα μεταβάσεων:



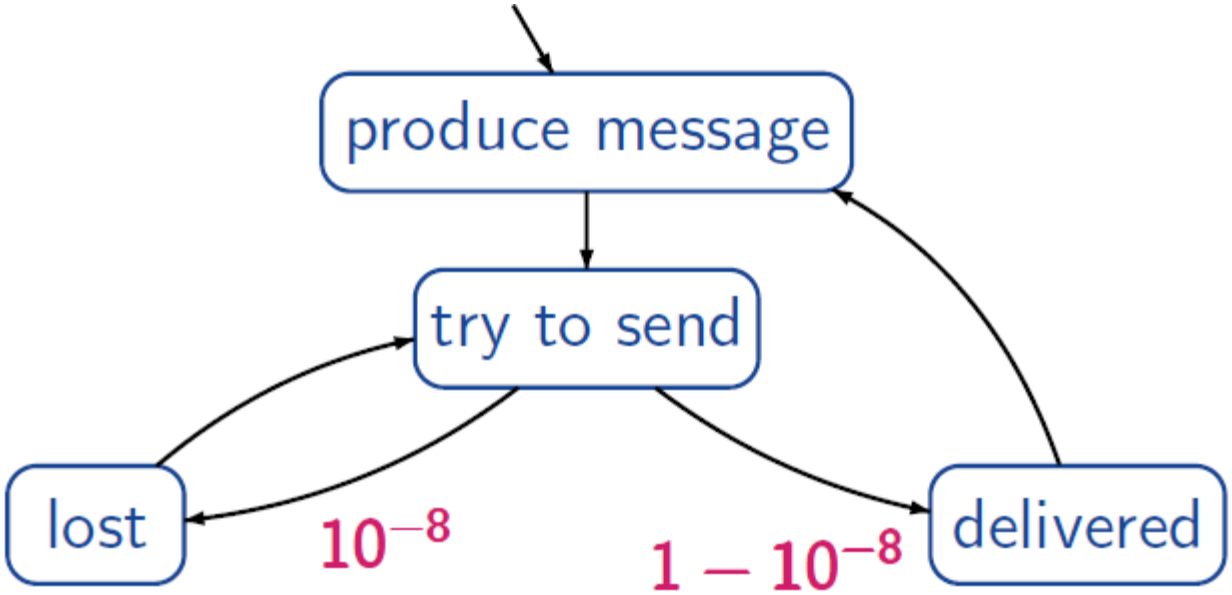
Απλοποίηση μοντέλου (1)

- Η αβεβαιότητα κατά πόσο το μήνυμα θα παραδοθεί ή θα χαθεί.



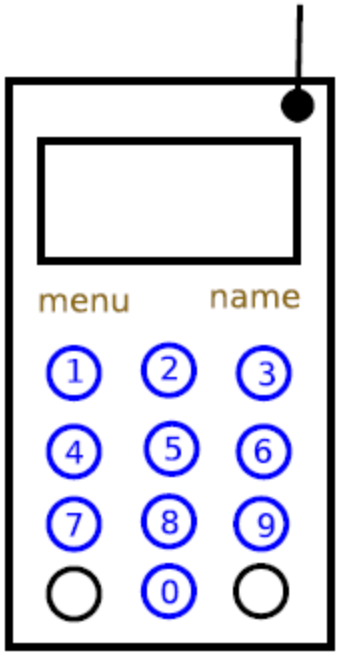
Απλοποίηση μοντέλου (2)

- Πιθανή μελλοντική εκλέπτυνση: ο μη ντετερμινισμός αντικαθίσταται από πιθανοτική επιλογή.

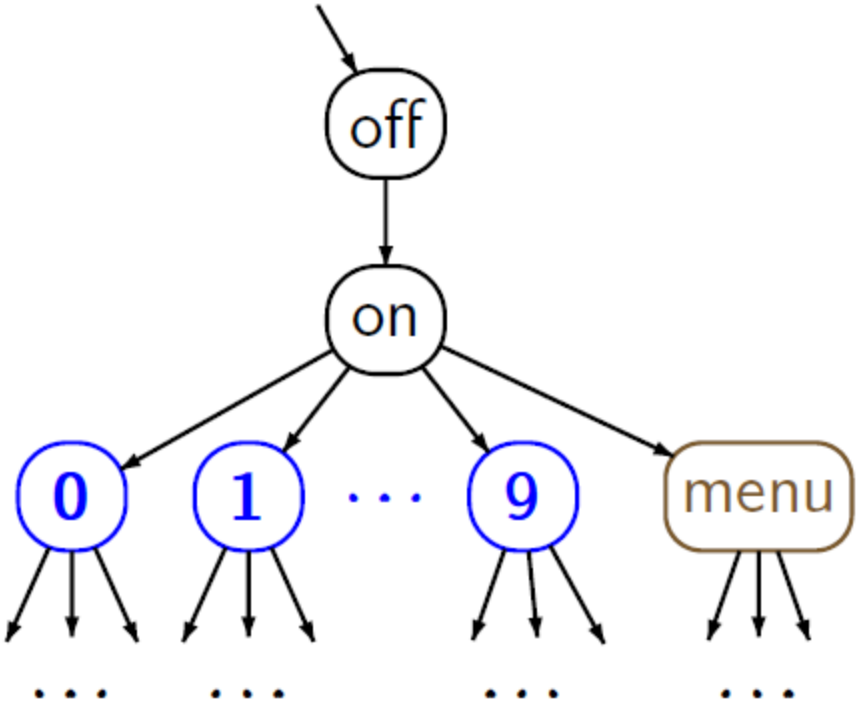


Ελλιπής Γνώση

- Έλλειψη γνώσης που οφείλεται στο περιβάλλον: χρήστης, αισθητήρας, επικοινωνία με άλλα προγράμματα.



mobile phone



Μέγεθος μεταβάσεων

- Κάθε ατομική μετάβαση σε ένα σύστημα μεταβάσεων αντιπροσωπεύει ένα κομμάτι κώδικα το οποίο δεν μπορεί να διασπαστεί.
- Ποιο είναι τα κατάλληλο μέγεθος μιας ατομικής μετάβασης;
  - Μεταβάσεις μικρού μεγέθους συνεπάγονται περιττή πολυπλοκότητα
  - Μεταβάσεις μεγάλου μεγέθους μπορεί να οδηγήσουν στην απώλεια συμπεριφορών του συστήματος υπό μελέτη
- Η εντολή a:=a+1 αντιστοιχεί σε ατομική μετάβαση;

  Μόνο σε συστήματα όπου η εντολή εκτελείται ατομικά ως μία εντολή τύπου inc.

Έλλειψη ατομικότητας

- Εκτέλεσε τα πιο κάτω όταν a=0 σε δύο συντρέχουσες διεργασίες:

```
P1:a=a+1

P2:a=a+1
```

- Αποτέλεσμα: a=2.

- Ισχύει πάντα το αποτέλεσμα;

- Θεωρήστε την πιο κάτω μετάφραση:

```
P1:load R1,a
   inc R1
   store R1,a
P2:load R2,a
   inc R2
   store R2,a
```

- Το a μπορεί να πάρει και την τιμή 1.

Μέθοδοι μοντελοποίησης

- Προφανώς η χρήση συστημάτων μεταβάσεων δεν είναι πρακτική για τη μοντελοποίηση μεγάλων συστημάτων. Ως εκ τούτου διάφοροι φορμαλισμοί έχουν προταθεί. Χωρίζονται σε δύο βασικές κατηγορίες:
  - απλές "γλώσσες προγραμματισμού"
  - παραλλαγές αυτομάτων
- Στόχοι
  - Ύπαρξη απλής και αποδοτικής μετάφρασης ανάμεσα στο φορμαλισμό και κάποια εσωτερική αναπαράσταση
  - Απλότητα και ευκολία χρήσης
  - Αυστηρά διατυπωμένη σημασιολογία
- FDT's (Formal Description Techniques)
  - Estelle, LOTOS, SDL - αναπτύχθηκαν από το ISO (International Standards Organization)
- Άλγεβρες Διεργασιών (CCS, CSP, ACP, …)
- I/O Αυτόματα
- Promela
- Γραφικά πρότυπα (UML, ROOM)