ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μεθόδοι Εξόρυξης Ροής Εργασιών με εφαρμόγη στην εκμάθηση προτιμήσεων

Μαρία Ζαχαρίου

Επιβλέπων Καθηγητής
Γιάννης Δημόπουλος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2010

Ευχαριστίες

Με μεγάλη μου χαρά και ικανοποίηση φτάνω στο τέλος της διεκπεραίωσης της διπλωματικής μου εργασίας και ταυτόχρονα στο τέλος των προπτυχιακών μου σπουδών στο τμήμα Πληροφορικής του Πανεπιστημίου Κύπρου. Με κόπο και προσπάθεια έχω φτάσει μέχρι εδώ, αλλά δεν ήμουν μόνη μου. Αρκετά ήταν να άτομα που με στήριξαν.

Θα ήθελα αρχικά να ευχαριστήσω τον επιβλέποντα καθηγητή μου, τον κ. Γιάννη Δημόπουλο για την άψογη συνεργασία και την πολύτιμη βοήθεια που μου πρόσφερε κατα την διάρκεια της εκπόνησης της διπλωματικής μου εργασίας. Επίσης ευχαριστώ όλους τους φίλους και συμφοιτητές μου με τους οποίους είχα την ευκαιρία να συνεργαστώ κατά τη διάρκεια των σπουδών μου, για την βοήθεια και την στήριξη τους, αφού χωρίς αυτούς όλα θα ήταν πιο δύσκολα.
Τέλος ευχαριστώ την οικογένεια μου που ήταν δίπλα μου κατα τη διάρκεια της προσπάθειας μου και όλης μου της φοίτησης και με στήριζε με το δικό της τρόπο.

Περίληψη

Οι επιχειρησιακές διαδικασίες, καθορίζουν τον τρόπο με τον οποίο πρέπει να χρησιμοποιούνται οι πόροι μιας επιχείρησης. Η απόδοση της επιχείρησης εξαρτάται από την ποιότητα και την ακρίβεια της επιχειρησιακής διαδικασίας. Οι μοντέρνες επιχειρήσεις χρησιμοποιούν το μοντέλο ροής εργασιών για να περιγράψουν τις επιχειρησιακές διαδικασίες. Συγκεκριμένα, ροή εργασιών είναι η συλλογή ενεργειών και αλληλεπιδράσεων που οργανώνονται μαζί για την επίτευξη μιας επιχειρησιακής διαδικασίας. Για την υποστήριξη του σχεδιασμού του μοντέλου ροής εργασιών, προτείνεται η χρήση «Workflow mining», που είναι η εξόρυξη ροών εργασιών μέσα από αρχεία (logs) που καταγράφουν πληροφορίες σχετικά με τις διάφορες διαδικασίες ενός οργανισμού. Η διαδικασία κατασκευής συστημάτων ροής εργασιών ξεκινά με τη συγκέντρωση πληροφοριών για τη ροή εργασιών (διαδικασιών) παρά να ξεκινούμε με τη σχεδίαση του μοντέλου ροής εργασίας.

Στην παρούσα διπλωματική περιγράφονται αλγόριθμους οι οποίοι είναι υπεύθυνοι για την κατασκευή μοντέλων διαδικασιών με βάση αρχεία γεγονότων ή αρχεία εκτελέσεων. Θα δούμε πως αυτοί οι αλγόριθμοι θα αναπαραστήσουν τις σχέσεις οι οποίες υπάρχουν ανάμεσα στις διεργασίες ανάλογα με τα συμπεράσματα που βγάζουν μέσα από τα αρχεία. Υπάρχουν αρκετοί αλγόριθμοι για το σκοπό αυτό και ο καθένας λειτουργεί με διαφορετικό τρόπο. Κάθε αλγόριθμος μπορεί να κατασκευάσει διαφορετικά το μοντέλο διαδικασίας και δεν χρησιμοποιούν όλοι τα ίδια γραφήματα. Επίσης θα δούμε ότι υπάρχουν και κατασκευές τις οποίες δεν μπορούν να κατασκευάσουν σωστά οι αλγόριθμοι.

Στα πλαίσια της έρευνας αυτών των αλγορίθμων, χρησιμοποιήσαμε και το εργαλείο ProM, το οποίο είναι ένα εργαλείο που χρησιμοποιεί πάνω από 20 αλγόριθμους για εξόρυξη διαδικασιών μέσα από αρχεία γεγονότων και κατασκευή μοντέλου διαδικασίας. Επίσης παρέχει τη δυνατότητα ανάλυσης των αποτελεσμάτων αυτών με τη χρήση διάφορων άλλων τεχνικών. Ένα άλλο σημαντικό του χαρακτηριστικό είναι το ότι δίνει τη δυνατότητα μετάφρασης του μοντέλου που έχει κατασκευάσει ένας αλγόριθμος σε άλλο μοντέλο διαφορετικού τύπου.

Τέλος, μελετήθηκαν τα δίκτυα προτιμήσεων CP-nets. Στα πλαίσια της μελέτης αυτών των δικτύων, προτάθηκαν και κάποιες ιδέες για εξόρυξη πληροφοριών μέσα από αρχείο τα οποία περιέχουν προτιμήσεις, έτσι ώστε να κατασκευαστεί το κατάλληλο δίκτυο CP-net. Οι εκτελέσεις στο αρχείο θα κατασκευάζονται μέσα από τις προτιμήσεις και θα αντιστοιχούν στα αρχεία που χρησιοποιούνται για εξόρυξη ροών εργασιών. Μέσα απο αυτό το αρχείο θα κατασκευάζεται μια διαδικασία η οποία αντιστοιχεί στο CP-net.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Επιχειρησιακές Διαδικασίες και ροή εργασιών

Οι επιχειρησιακές διαδικασίες, καθορίζουν τον τρόπο με τον οποίο πρέπει να χρησιμοποιούνται οι πόροι μιας επιχείρησης. Περιγράφουν τη σειρά με την οποία εκτελούνται οι διάφορες διεργασίες. Η απόδοση της επιχείρησης εξαρτάται άμεσα από την ποιότητα και την ακρίβεια της επιχειρησιακής διαδικασίας. Για την περιγραφή της επιχειρισιακής διαδικασίας χρησιμοποιήται από πολλές μοντέρνες επιχειρίσεις το μοντέλο ροής εργασιών. Συγκεκριμένα, ροή εργασιών είναι η συλλογή ενεργειών και αλληλεπιδράσεων που οργανώνονται μαζί για την επίτευξη μιας επιχειρησιακής διαδικασίας.

Τα μοντέλα ροής εργασιών, υποθέτουν ότι μία διαδικασία μπορεί να σπάσει σε μικρότερα κομμάτια που ονομάζονται δραστηριότητες. Η εκτέλεση όλων των δραστηριοτήτων έχει ως αποτέλεσμα την εκτέλεση της διαδικασίας. Κάθε δραστηριότητα μπορεί να εξαρτάται από κάποια άλλη. [1]

Τα συστήματα διαχείρισης ροής εργασιών, είναι τα πληροφοριακά συστήματα υποστήριξης του ανασχεδιασμού επιχειρησιακών διαδικασιών που βοηθούν στον προσδιορισμό ροών εργασιών και διατηρούν τον έλεγχο και συντονισμό κατά την εκτέλεση τους. Εκτός από τα Συστήματα διαχείρισης ροών εργασιών

(WFMS) και άλλα συστήματα λογισμικού υιοθετούν την τεχνολογία της ροής εργασιών.

Παρόλα αυτά, πολλά προβλήματα προκύπτουν με την εισαγωγή αυτής της τεχνολογίας. Τα συστήματα απαιτούν τη σχεδίαση της ροής εργασιών, συγκεκριμένα τη κατασκευή ενός λεπτομερούς μοντέλου που να περιγράφει ακριβώς τη δρομολογημένη εργασία. Επίσης απαιτεί βαθιά γνώση της επιχειρησιακής διαδικασίας. Ο σχεδιασμός είναι χρονοβόρος και επιρρεπής σε λάθη. [2]

Για την υποστήριξη του σχεδιασμού αυτών των συστημάτων, προτείνεται η χρήση «Εξόρυξης ροής εργασιών», που είναι η εξόρυξη ροών εργασιών μέσα από αρχεία (logs) που καταγράφουν πληροφορίες σχετικά με τις διάφορες διαδικασίες ενός οργανισμού. Η διαδικασία κατασκευής συστημάτων ροής εργασιών ξεκινά με τη συγκέντρωση πληροφοριών για τη ροή εργασιών (διαδικασιών) παρά να ξεκινούμε με τη σχεδίαση του μοντέλου ροής εργασίας. Κάθε πληροφοριακό σύστημα που χρησιμοποιεί συστήματα συναλλαγών, όπως τα συστήματα διαχείρισης εργασιών, μπορεί να μας δώσει πληροφορίες σχετικά με γεγονότα τα οποία μπορεί να αναφέρονται i)σε διεργασίες, ii) σε περιπτώσεις (στιγμιότυπα) της ροής εργασίας (διαδικασίας) iii)σε άτομα τα οποία είναι υπεύθυνα για την εκτέλεση κάποιας διεργασίας και μας δίνει την πληροφορία ότι τα γεγονότα δίνονται βρίσκονται σε σειρά.

Η γενικότερη ιδέα εξόρυξης ροής εργασιών ή διαδικασιών (στη συνέχεια θα χρησιμοποιούνται και οι δύο ορισμοί), είναι η ανακάλυψη διαδικασίας , ή η παρακολούθηση και η βελτίωση της πραγματικής διαδικασίας στην περίπτωση που ήδη υπάρχει, με την εξαγωγή γνώσης από τα αρχεία γεγονότων. Υπάρχουν τρεις βασικοί τύποι εξόρυξης διαδικασιών:

- Ανακάλυψη: Σε αυτή την περίπτωση δεν υπάρχει κάποιο προηγούμενο μοντέλο και με βάση το αρχείο των γεγονότων και

κάποιο αλγόριθμο εξόρυξης ροών εργασιών κατασκευάζεται κάποιο μοντέλο.

- Συνέπεια: Υπάρχει ήδη ένα μοντέλο της διαδικασίας. Με τη βοήθεια της εξόρυξης διαδικασιών από το αρχείο ελέγχεται αν το μοντέλο ανταποκρίνεται στην πραγματικότητα.
- Επέκταση: Υπάρχει ήδη ένα μοντέλο μιας διαδικασίας. Με τη βοήθεια του αρχείου γεγονότων, υπάρχει η δυνατότητα επέκτασης και εμπλουτισμού του μοντέλου.

Παραδοσιακά η εξόρυξη διαδικασιών επικεντρώνεται στην “ανακάλυψη”, όπου μέσα από το αρχείο γίνεται προσπάθεια εξαγωγής όσων των δυνατών περισσότερων πληροφοριών και κατασκευής ενός μοντέλου διαδικασίας που να ανταποκρίνεται στην πραγματικότητα. Σε αυτό τον τύπο θα επικεντρωθούμε και εμείς και θα ασχοληθούμε με αλγόριθμους αυτής της κατηγορίας. [6,7]

Την εξόρυξη διαδικασιών μπορούμε να τη δούμε από τρεις διαφορετικές σκοπιές, τη σκοπιά της διαδικασίας, την οργανωτική προοπτική και τη σκοπιά των περιπτώσεων. Η προοπτική της διαδικασίας επικεντρώνεται στη ροή ελέγχου δηλαδή την ταξινόμηση των διεργασιών. Σκοπός αυτής της προοπτικής είναι να βρει ένα καλό χαρακτηρισμό όλων των πιθανών μονοπατιών, αναπαριστώντας τα μέσα από κάποιο μοντέλο, το οποίο μπορεί να είναι για παράδειγμα ένα Petri-net. Η οργανωτική προοπτική επικεντρώνεται στα άτομα τα οποία εμπλέκονται στη διαδικασία και τα οποία είναι υπεύθυνα για την εκτέλεση κάποιων διεργασιών. Σκοπός αυτής της προοπτικής είναι είτε να δομήσει τον οργανισμό κατηγοριοποιώντας τα άτομα ανάλογα με τους ρόλους τους ή να παρουσιάσει τις εξαρτήσεις ανάμεσα σε αυτά τα άτομα, κατασκευάζοντας για παράδειγμα ένα κοινωνικό δίκτυο. Η προοπτική των περιπτώσεων, επικεντρώνεται στις περιπτώσεις (στιγμιότυπα διαδικασίας), οι οποίες μπορούν μα χαρακτηριστούν από το μονοπάτι τους στην διαδικασία ή από τα άτομα (originators) που εργάζονται για την συγκεκριμένη περίπτωση. Όμως η προοπτική η οποία μας ενδιαφέρει περισσότερο και με την οποία θα ασχοληθούμε είναι η προοπτική της διαδικασίας.

Στη συνέχεια θα ασχοληθούμε με αλγόριθμους οι οποίοι είναι υπεύθυνοι για την κατασκευή μοντέλων διαδικασιών με βάση αρχεία γεγονότων ή αρχεία εκτελέσεων. Θα δούμε πως αυτοί οι αλγόριθμοι θα αναπαραστήσουν τις σχέσεις οι οποίες υπάρχουν ανάμεσα στις διεργασίες ανάλογα με τα συμπεράσματα που βγάζουν μέσα από τα αρχεία. Υπάρχουν αρκετοί αλγόριθμοι για το σκοπό αυτό και ο καθένας λειτουργεί με διαφορετικό τρόπο. Κάθε αλγόριθμος μπορεί να κατασκευάσει διαφορετικά το μοντέλο διαδικασίας και δεν χρησιμοποιούν όλοι τα ίδια γραφήματα. Επίσης θα δούμε ότι υπάρχουν και κατασκευές τις οποίες δεν μπορούν να κατασκευάσουν σωστά οι αλγόριθμοι.

Στα πλαίσια της έρευνας αυτών των αλγορίθμων, χρησιμοποιήσαμε και το εργαλείο ProM, το οποίο είναι ένα εργαλείο που χρησιμοποιεί πάνω από 20 αλγόριθμους για εξόρυξη διαδικασιών μέσα από αρχεία γεγονότων και κατασκευή μοντέλου διαδικασίας. Επίσης παρέχει τη δυνατότητα ανάλυσης των αποτελεσμάτων αυτών με τη χρήση διάφορων άλλων τεχνικών. Ένα άλλο σημαντικό του χαρακτηριστικό είναι το ότι δίνει τη δυνατότητα μετάφρασης του μοντέλου που έχει κατασκευάσει ένας αλγόριθμος σε άλλο μοντέλο διαφορετικού τύπου.

1.2 Δρομολόγηση στη Ροή Εργασιών

Οι ροές εργασιών βασίζονται σε περιπτώσεις δηλ. κάθε κομμάτι δουλείας εκτελείται για μια συγκεκριμένη περίπτωση. Στόχος των συστημάτων διαχείρισης ροής εργασιών, είναι η διαχείριση των περιπτώσεων όσο πιο αποτελεσματικά γίνεται. Κάθε διαδικασία ροής εργασιών σχεδιάζεται έτσι ώστε να μπορεί να χειριστεί παρόμοιες περιπτώσεις. Οι περιπτώσεις χειρίζονται εκτελώντας τις δραστηριότητες με συγκεκριμένη σειρά. Αφού οι δραστηριότητες εκτελούνται με συγκεκριμένη σειρά σκοπός είναι να αναγνωριστούν κάποιες υποθέσεις που αντιστοιχούν σε αιτιώδεις εξαρτήσεις ανάμεσα στις δραστηριότητες. Κάθε δραστηριότητα έχει «προϋποθέσεις » οι οποίες πρέπει να ισχύουν πριν την εκτέλεση μιας δραστηριότητας και «μετά-υποθέσεις» οι οποίες πρέπει να ισχύουν μετά την εκτέλεση μιας

δραστηριότητας. Πολλές περιπτώσεις μπορούν να χειριστούν ακολουθώντας τον ίδιο ορισμό διαδικασίας ροής εργασιών, έτσι που μία δραστηριότητα να πρέπει εκτελεστεί για πολλές περιπτώσεις.

Μία δραστηριότητα όταν εκτελείται για συγκεκριμένη περίπτωση ονομάζεται αντικείμενο εργασίας. Τα περισσότερα αντικείμενα εργασίας μπορεί να εκτελούνται από μία πηγή που μπορεί να είναι είτε άνθρωπος είτε μηχανή. Οι περιπτώσεις από τη ματιά της ροής εργασιών δεν επηρεάζονται άμεσα η μία από την άλλη, επηρεάζονται όμως έμμεσα αφού μοιράζονται τις πηγές.

Ένα πολύ σημαντικό θέμα στη διαχείριση ροής εργασιών, είναι η δρομολόγηση (routing) των περιπτώσεων. Ένας ορισμός μίας διαδικασίας ροής εργασιών, ορίζει πως οι περιπτώσεις δρομολογούνται με βάση τις δραστηριότητες οι οποίες χρειάζεται να εκτελεστούν. Υπάρχουν τέσσερα είδη δρομολόγησης:

1. Σειριακή δρομολόγηση (sequential routing): Οι δραστηριότητες εκτελούνται σειριακά, όταν η εκτέλεση μιας δραστηριότητας ακολουθείτε από την εκτέλεση κάποιας δεύτερης.

2. Παράλληλη δρομολόγηση (parallel routing): Όταν δύο δραστηριότητες είναι παράλληλες, αυτό σημαίνει ότι μπορεί να εκτελούνται παράλληλα, ή με οποιαδήποτε σειρά.

3. Υπό συνθήκη δρομολόγηση (conditional routing): Όταν έχουμε δύο δραστηριότητες, από τις οποίες θα πρέπει να επιλέξουμε μία για εκτέλεση.

4. Επαναληπτική (iteration) δρομολόγηση: Όταν ένα μία δραστηριότητα πρέπει να εκτελεστεί πολλές φορές. [4]

1.3 Petri Nets

Ένα είδος γραφήματος που χρησιμοποιείται για την αναπαράσταση ροής εργασιών είναι τα Petri nets (Place/Transition net). Είναι ένα είδος διμερούς γράφου, με δύο είδη κόμβων, τις θέσεις (places) και τις μεταβάσεις (transitions).

Ορισμός Petri net:

-Ένα σύνολο από θέσεις (places) $P= \{p_1,p_2,\ldots,p_m\}$, οι οποίες αναπαριστώνται με κύκλους.

-Ένα σύνολο από μεταβάσεις (transitions) $T=\{t_1,t_2,\ldots,t_n\}$, οι οποίες αναπαριστώνται με κάθετες γραμμές (bars) ή με κουτιά.

-Ένα σύνολο ακμών (arcs) $F \subseteq (PxT)U(TxP)$.

-Μία συνάρτηση βάρους W: F->{1,2,3,…}.

-Την αρχική κατάσταση M_0 (initial marking).

$-P \cap T = \emptyset, \quad P \cup T \neq \emptyset$

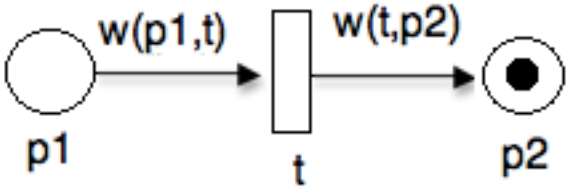
Οι ακμές ξεκινούν είτε από θέσεις σε μεταβάσεις, είτε από μεταβάσεις σε θέσεις. Δεν επιτρέπεται η ένωση ανάμεσα σε δύο κόμβους του ίδιου τύπου. Μία κατάσταση (marking) M αναθέτει σε κάθε θέση ένα θετικό ακέραιο αριθμό. Αυτό σημαίνει ότι δίνει στη θέση ένα αριθμό από « μάρκες » (tokens) . Οι μάρκες χρησιμοποιούνται σε αυτά τα δίκτυα για την προσομοίωση δυναμικών και ταυτόχρονων διαδικασιών των συστημάτων.

Για την περιγραφή της δυναμικής συμπεριφοράς των συστημάτων, η κατάσταση σε ένα Petri net αλλάζει σύμφωνα με τον πιο κάτω κανόνα μετάβασης:

1. Μία μετάβαση t είναι ενεργοποιημένη αν κάθε θέση εισόδου p του t έχει τουλάχιστον w(p,t) μάρκες, όπου w(p,t) είναι το βάρος της ακμής από το p στο t.
2. Μία ενεργοποιημένη μετάβαση μπορεί είτε να πυροδοτήσει (fire) είτε όχι.

3. Η πυροδότηση μιας ενεργοποιημένης μετάβασης t αφαιρεί w(p,t) μάρκες από κάθε θέση εισόδου p του t και προσθέτει w(t,p) μάρκες σε κάθε θέση εξόδου p του t.

Σχήμα 1.1: *t: ενεργοποιημένο, υποθέτουμε ότι το w(p1,t) και w(t,p2) είναι 1.*



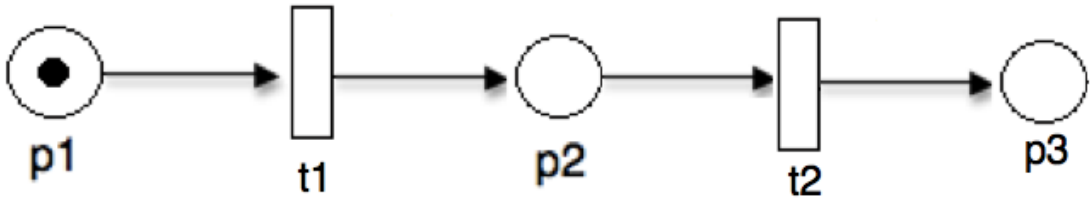
Σχήμα 1.2*: Μετά τον πυροβολισμό του t. Το t δεν είναι ενεργοποιημένο.*

Υπάρχουν μεταβάσεις χωρίς εισόδους και ονομάζονται " source"και αυτού του είδους μεταβάσεις εκτελούνται χωρίς συνθήκες. Επίσης, υπάρχουν μεταβάσεις χωρίς εξόδους οι οποίες ονομάζονται "sink", οι οποίες καταναλώνουν μάρκες χωρίς όμως να παράγουν.

Στην χρήση των Petri-nets για αναπαράσταση ροής εργασιών υπάρχει περιορισμός σχετικά με το βάρος των ακμών. Όλες ακμές έχουν βάρος 1, δεν υπάρχει νόημα να υπάρχουν άλλα βάρη λόγω του ότι οι θέσεις αντιστοιχούν σε υποθέσεις. Κατά τη μοντελοποίηση μίας διαδικασίας ροής εργασιών με Petri-nets, οι δραστηριότητες αναπαριστώνται με μεταβάσεις, οι υποθέσεις με θέσεις και οι περιπτώσεις με μάρκες.

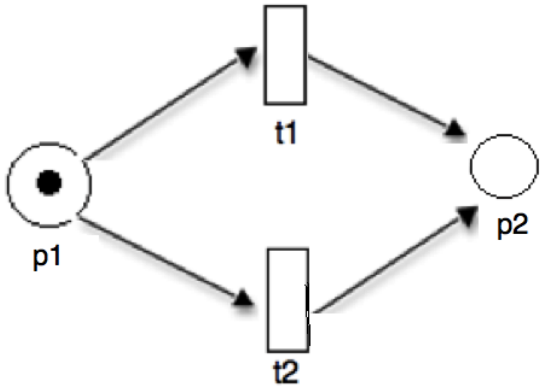
Σειριακή δρομολόγηση: χρησιμοποιείται για τη μεταχείριση των αιτιολογικών σχέσεων ανάμεσα στις δραστηριότητες. Όπως φαίνεται στο πιο κάτω σχήμα η t1 εκτελείται αμέσως μετά το τέλος της εκτέλεσης της t2. Η p2 μοντελοποιεί την

αιτιολογική σχέση ανάμεσα στο t1 και t2, αφού είναι μετά-υπόθεση του t1 και προϋπόθεση του t2. Άρα οι δύο αυτές μεταβάσεις εκτελούνται σειριακά.



Σχήμα 1.3: *Σειριακή Δρομολόγηση*

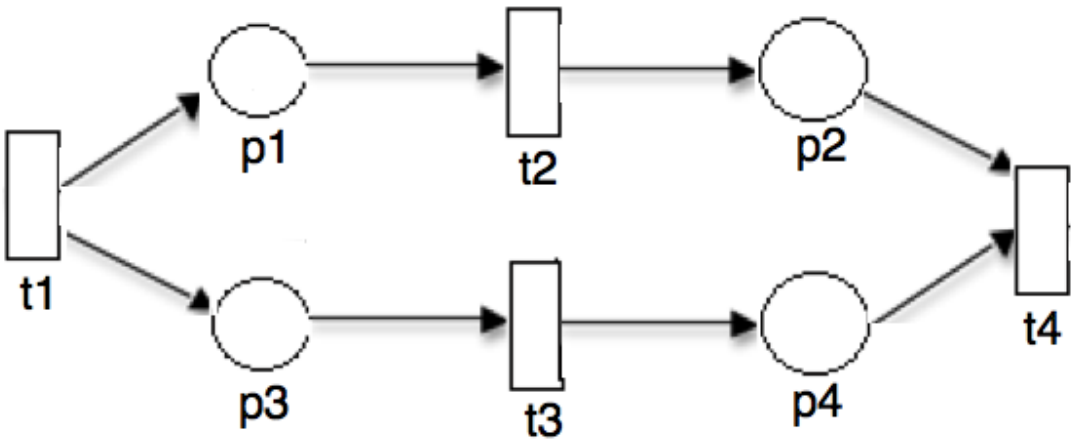
Υπο-συνθήκη (Conditional) δρομολόγηση: έχουμε στις δομές οι οποίες έχουν την μορφή, όπου από μία θέση, όπως η p1 στο *Σχήμα 1.4* φεύγουν δύο ακμές προς δύο διαφορετικές μεταβάσεις και αυτό συμβολίζει το δομικό στοιχείο OR-split. Στην θέση p2, όπου είναι μία θέση με 2 ακμές εισόδου, έχουμε το δομικό στοιχείο OR-join. Αυτή η δομή αναφέρεται και ως επιλογή, ή απόφασή, αφού με κάποιο τρόπο πρέπει να αποφασιστεί ποια μετάβαση θα πυροδοτήσει. Μπορεί να πυροδοτήσει είτε η μία μετάβαση, είτε η άλλη αλλά όχι και οι δύο. Είτε πυροδοτήσει η t1 είτε η t2, θα φύγει η μάρκα (token) από το p1 που είναι η θέση εισόδου και θα πάει στην p2 που είναι η θέση εξόδου.



Σχήμα 1.4: *Υπό-συνθήκη Δρομολόγηση*

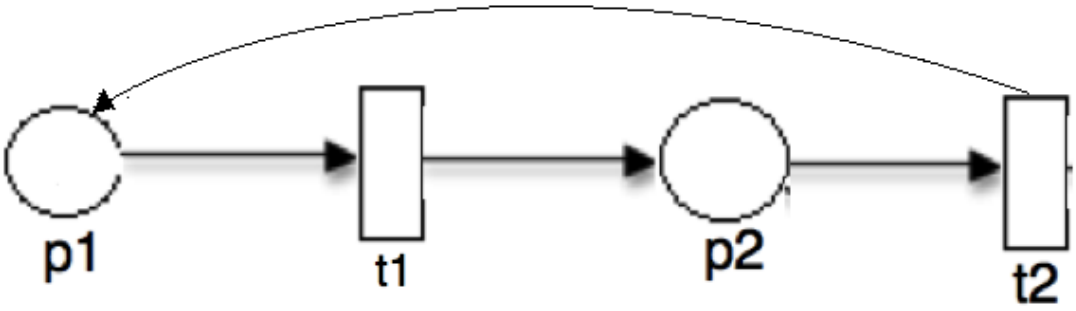
Παράλληλη δρομολόγηση: έχουμε όταν δύο μεταβάσεις μπορεί να πυροβολήσουν με οποιαδήποτε σειρά, ακόμα και ταυτόχρονα. Αυτές οι μεταβάσεις δεν έχουν κάποια αιτιολογική εξάρτηση μεταξύ τους. Για την αναπαράσταση αυτής της

δρομολόγησης βοηθούν δύο δομικά στοιχεία το AND-split και το AND-join. Στο πιο κάτω παράδειγμα το AND-split βρίσκεται στην θέση t1 και το AND-join στη θέση t4. Στο *Σχήμα 1.5* οι t2 και t3 λέμε ότι εκτελούνται παράλληλα γιατί μπορούν να εκτελεσθούν με οποιαδήποτε σειρά.



Σχήμα 1.5: *Παράλληλη Δρομολόγηση*

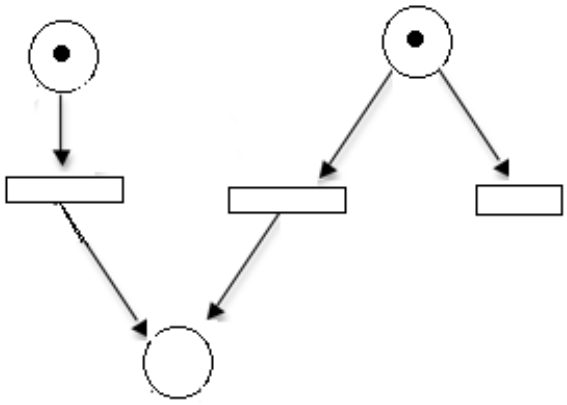
Επαναληπτική δρομολόγηση: Σε αυτή τη δρομολόγηση μία δραστηριότητα μπορεί να εκτελεστεί μία ή περισσότερες φορές. Η t2 μετάβαση είναι μετάβαση ελέγχου και ανάλογα με το αποτέλεσμα της αποφασίζεται αν η t1 πρέπει να εκτελεστεί ξανά ή όχι.



Σχήμα 1.6*: Επαναληπτική Δρομολόγηση*

Μία κατάσταση στην οποία συνδυάζεται απόφαση μαζί με παράλληλες εργασίες ονομάζεται «Σύγχυση» (confusion). Όταν δηλαδή δύο μεταβάσεις είναι

παράλληλες, αλλά ταυτόχρονα πρέπει να υπάρξει επιλογή σε σχέση με κάποια τρίτη μετάβαση. Μια τέτοια περίπτωση φαίνεται στο πιο *Σχήμα 1.7*. [4,5]



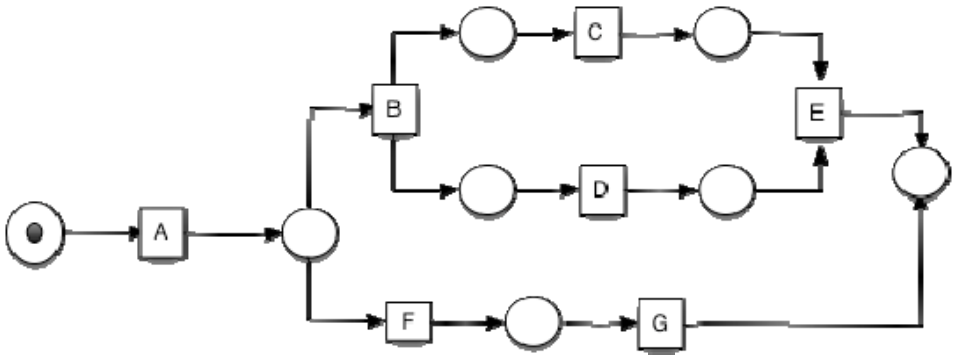
Σχήμα 1.7: *Σύγχυση*

Για την αναπαράσταση των βασικών αρχών εξόρυξης διαδικασιών, ακολουθεί ένα παράδειγμα, όπου μέσα από ένα log (πίνακας 1), κατασκευάζουμε ένα Petri net. Σε αυτό το log, περιέχονται πληροφορίες για τέσσερις περιπτώσεις. Όπως φαίνεται, για όλες τις περιπτώσεις εκτελείται η εργασία A (task A). Για τις περιπτώσεις 1,2,4 εκτελούνται και οι εργασίες B,C, D,E ενώ για την περίπτωση 3 εκτελούνται οι εργασίες F, G. Παρατηρούμαι επίσης ότι όταν εκτελεστεί η εργασία C τότε θα εκτελεστεί και η D , αν και η σειρά με την οποία εκτελούνται δεν είναι πάντα η ίδια, αφού για τις περιπτώσεις 1 και 4 εκτελείται πρώτα η C και μετά D, ενώ για την περίπτωση 2 εκτελείται πρώτα η D και μετά C. Με βάση τον πίνακα 1 και κάνοντας κάποιες υποθέσεις σχετικά με την πληρότητα του αρχείου, κατασκευάζουμε το μοντέλο που φαίνεται στο *Σχήμα 1.8*.

Case identifier	Task identifier
~~Case 1~~	~~Task A~~
~~Case 3~~	~~Task A~~
~~Case 2~~	~~Task A~~

Case 1	Task B
Case 1	Task C
Case 1	Task D
Case 4	Task A
Case 2	Task B
Case 4	Task B
Case 2	Task D
Case 2	Task C
Case 1	Task E
Case 4	Task C
Case 3	Task G
Case 4	Task D
Case 2	Task E
Case 4	Task E

Πίνακας 1.1



Σχήμα 1.8

Το Petri net ξεκινά με την εργασία A, αφού όλες οι περιπτώσεις στο log ξεκινούσαν με αυτή την εργασία. Μετά την εκτέλεση του A, υπάρχει επιλογή στο να εκτελεστεί είτε το B ή το F. Αν εκτελεσθεί το B, τότε στη συνέχεια θα εκτελεσθεί το C και το D με οποιαδήποτε σειρά, αφού είναι παράλληλα και τελευταία θα γίνει η εκτέλεση του E. Αν όμως εκτελεσθεί το F αντί το B, τότε μετά το F θα εκτελεστεί και το G.

Κεφάλαιο 2

Αναπαράσταση ροής εργασιών με κατευθυνόμενο γράφο

2.1 Βασικοί Ορισμοί

Η αρχική ιδέα η οποία περιγράφεται στο [1] για την χρήση εξόρυξης διεργασιών, σε συστήματα διαχείρισης ροών εργασιών βασιζόταν σε αναπαράσταση των ροών εργασιών με κατευθυνόμενο γράφο.

Στον αλγόριθμο που κατασκευάζει αυτό το γράφο δίνεται ένα αρχείο αδόμητων εκτελέσεων μιας διαδικασίας και μέσα από αυτό δημιουργεί ένα μοντέλο της διαδικασίας. Το αποτέλεσμα είναι ένας γράφος που παρουσιάζει τη ροή της επιχειρησιακής διαδικασίας. Ο γράφος αυτός έχει τα ακόλουθα χαρακτηριστικά:

-Πληρότητα: Περιέχει όλες τις εξαρτήσεις που υπάρχουν ανάμεσα στις διεργασίες/δραστηριότητες της διαδικασίας, όπως φαίνεται από τις εκτελέσεις που υπάρχουν στο αδόμητο αρχείο, και επιτρέπει όλες τις εκτελέσεις του αρχείου.

-Απεριτότητα (Irredundancy): Δεν πρέπει να έχει ανεπιθύμητες εξαρτήσεις ανάμεσα στις δραστηριότητες. Δηλαδή εξαρτήσεις οι οποίες δεν υπάρχουν ανάμεσα στις διεργασίες όπως φαίνεται από τις εκτελέσεις του αρχείου, δεν πρέπει να υπάρχουν στο γράφο.

-Ελαχιστότητα (Minimality): Ο γράφος να περιέχει τον ελάχιστο δυνατό αριθμό ακμών.

Οι κόμβοι αποτελούν τις δραστηριότητες και κάθε μία από αυτές μπορεί να θεωρηθεί σαν μια συνάρτηση η οποία τροποποιεί τη κατάσταση της διαδικασίας. Οι ακμές του γράφου αναπαριστούν τη ροή ελέγχου από ένα κόμβο σε άλλο και κάθε μία συσχετίζεται με μια Boolean συνάρτηση η οποία καθορίζει αν η ακμή θα ακολουθηθεί η όχι. [1]

Ορισμός 1:

Μια επιχειρησιακή διαδικασία P ορίζεται σαν ένα σύνολο δραστηριοτήτων $V_p= V_1, V_2,\ldots\ldots,V_n$, ένα κατευθυνόμενο γράφο $G_p=(V_p, E_p)$, μία συνάρτηση εξόδου $O_p: V_p \rightarrow N^k$ και για κάθε (u, v) που ανήκει στο E_p έχουμε μία boolean συνάρτηση $f(u,v): N^k \rightarrow \{0,1\}$.

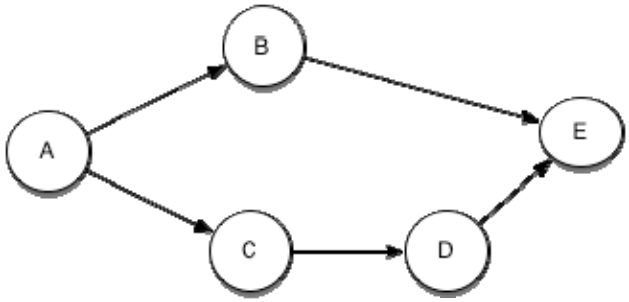
Υποθέτουμε ότι κάθε γράφος G_p έχει ένα κόμβο έναρξης και ένα τερματισμού. Αν δεν υπάρχουν μπορούμε να τους προσθέσουμε έτσι ώστε ο αρχικός θα ενώνεται με ακμές στην δραστηριότητα που εκτελείται πρώτη και ο τελικός θα ενώνεται με ακμές στην δραστηριότητα που εκτελείται τελευταία.

Η εκτέλεση της διαδικασίας, ακολουθεί αυτό τον γράφο δραστηριοτήτων. Για κάθε κόμβο *u* που τερματίζεται, υπολογίζουμε το αποτέλεσμα *o(u)* και με βάση αυτό το αποτέλεσμα ελέγχουμε αν η συνάρτηση ακμής *f(u,v)(o(u))* είναι αληθής, στη συνέχεια ελέγχουμε αν η δραστηριότητα (κόμβος) *v* είναι έτοιμη να εκτελεστεί, και αν είναι έτοιμη, αν δηλαδή έχει εκτελεστεί κάποια ή όλες οι δραστηριότητες που έχουν ακμές προς τη v, τότε τα αποτελέσματα αυτών των ακμών περνούν σαν είσοδοι στο *v*. Γενικά η εκτέλεση όλης της διαδικασίας είναι μία λίστα με τη σειρά εκτέλεσης όλων των δραστηριοτήτων.

Ορισμός 2 :

Αν υπάρχουν δραστηριότητες οι οποίες ξεκινούν ταυτόχρονα τότε αυτές οι δραστηριότητες είναι ανεξάρτητες. Αν υπάρχουν εξαρτήσεις ανάμεσα σε δύο δραστηριότητες τότε εμφανίζονται στην ίδια σειρά σε κάθε εκτέλεση. Στο γράφο οι εξαρτήσεις αναπαρίστανται σαν ακμές από την μια κορυφή στην άλλη, ή σαν μονοπάτια από την μια ακμή στην άλλη.

Παράδειγμα: Όπως φαίνεται στο *Σχήμα 2.1* υπάρχει εξάρτηση ανάμεσα στην Α και Β, αφού για να εκτελεσθεί η Β πρέπει να έχει εκτελεσθεί Α. Το ίδιο ισχύει και για την C, αφού δεν μπορεί να εκτελεσθεί αν δεν εκτελεσθεί η Α. Η D εξαρτάται από την C, άρα και από την Α. Η Ε έχει άμεση εξάρτηση από την Β και D, αλλά εξαρτάται και από την Α και C. Η Β είναι ανεξάρτητη από την C και D, αφού μπορεί να εκτελεστεί παράλληλα είτε με την C είτε με την D και δεν υπάρχει καμία ακμή που να τις ενώνει.



Σχήμα 2.1

Ορισμός 3:

Σε ένα αρχείο εκτελέσεων της ίδιας διαδικασίας η δραστηριότητα Β *ακολουθεί* την Α, αν η Β ξεκινά μετά την Α σε κάθε εκτέλεση που εμφανίζονται και οι δύο, ή αν υπάρχει μία άλλη C η οποία ακολουθεί την Α και η Β ακολουθεί την C.

Ορισμός 4:

Σε ένα αρχείο εκτελέσεων αν μία διαδικασία Β ακολουθεί την Α, αλλά η Α δεν ακολουθεί τη Β, τότε η Β εξαρτάται από τη Α. Αν μία διαδικασία Β ακολουθεί την Α και η Α ακολουθεί τη Β, ή αν η Α δεν ακολουθεί την Β και ούτε η Β ακολουθεί την Α τότε είναι *ανεξάρτητες*.

Με βάση ένα αρχείο εκτελέσεων μπορούμε να ορίσουμε τον γράφο εξαρτήσεων, ο οποίος είναι ένας γράφος που παρουσιάζει όλες τις εξαρτήσεις που βρέθηκαν στο αρχείο.

Ας πάρουμε για παράδειγμα ένα αρχείο εκτελέσεων {ABCDE, ABDC, ADE}. Η B ακολουθεί την A σε όλες τις εκτελέσεις στις οποίες εμφανίζονται και οι δύο, ενώ η A δεν ακολουθεί την B, άρα η B εξαρτάται από την A σύμφωνα με το αρχείο εκτελέσεων. Το ίδιο ισχύει και για τις υπόλοιπες διεργασίες, εξαρτώνται από την A αφού την ακολουθούν σε όλες τις εκτελέσεις που εμφανίζονται. Η C εκτελείται πριν την D σε μια εκτέλεση, αλλά στην δεύτερη εκτέλεση εκτελείται μετά την D, άρα η C και η D είναι ανεξάρτητες.

Ορισμός 5:
Ένας γράφος G_{VL} είναι *γράφος εξαρτήσεων* αν υπάρχει μονοπάτι από τον κόμβο **u** στο **v**, αν και μόνο αν ο **v** εξαρτάται από το **u**. Γενικά ο γράφος εξαρτήσεων για ένα αρχείο εκτελέσεων δεν είναι μοναδικός. Αλλά αν υπάρχουν δύο γράφοι με το ίδιο μεταβλητή κλειστότητα (transitive closure), τότε μπορεί να αναπαριστούν τις ίδιες εξαρτήσεις.

Ορισμός 6:
Δοθέντος ενός γράφου G = (V,E) μιας διαδικασίας P, και μιας εκτέλεσης R, η R είναι *συνεπής* (consistent) με τον G αν οι δραστηριότητες στην R είναι υποσύνολο V' των δραστηριοτήτων στον G, και ο γράφος που παράγεται
G' = (V', (u,v) $\in$ E| u,v $\in$ V') είναι συνδεδεμένος . Η πρώτη και η τελευταία δραστηριότητα στην R είναι αυτές που ξεκινούν και τελειώνουν τη διαδικασία αντίστοιχα, και κάθε κόμβος στο V΄ μπορεί να διαπεραστεί από την πρώτη δραστηριότητα , και καμιά εξάρτηση στο γράφο δεν παραβιάζεται από τη σειρά των δραστηριοτήτων στην R.

Παράδειγμα: Οι εκτελέσεις ABE και ACDE είναι συνεπείς με τον γράφο στο *Σχήμα 2.1*, ενώ η ACE δεν είναι.

Ορισμός 7: Συμβατός (Conformal) Γράφος

Ένας γράφος G είναι conformal με ένα αρχείο εκτελέσεων L, αν ισχύουν τα εξής:

-Για κάθε εξάρτηση στο L, υπάρχει ένα μονοπάτι στον γράφο G.

-Δεν υπάρχει κανένα μονοπάτι στον G μεταξύ ανεξάρτητων δραστηριοτήτων στο L.

-Κάθε εκτέλεση στο L είναι συνεπής με τον G.

Ακολουθούν δύο αλγόριθμοι για την την εύρεση ενός conformal γράφου χωρίς κύκλους, από ένα αρχείο εκτελέσεων μιας διαδικασίας. [1]

2.2 Αλγόριθμος 1 για κατασκευή κατευθυνόμενου γράφου

Για την εύρεση αυτού του αλγόριθμου γίνονται οι εξής 2 υποθέσεις:

1. Ο γράφος που θα βρούμε δεν θα περιέχει κύκλους, και συνήθως αυτό συμβαίνει σε πολλές περιπτώσεις.
2. Σε κάθε εκτέλεση κάθε δραστηριότητα εμφανίζεται ακριβώς μία φορά. Δηλαδή δεν υπάρχει περίπτωση κάποια δραστηριότητα να λείπει από μία εκτέλεση, ή να εμφανίζεται πάνω από μία φορά.

Αυτή την ειδική περίπτωση την εξετάζουμε, αφού σε αυτή την περίπτωση υπάρχει ένα μοναδικό μοντέλο διαδικασίας , το οποίο είναι και συμβατός γράφος και ελαχιστοποιεί τον αριθμό τω ακμών. Ο αλγόριθμος για αυτή την ειδική περίπτωση έχει μικρότερο χρόνο εκτέλεσης.

Λήμμα 1:

Δοθέντος ενός αρχείου εκτελέσεων της ίδιας διαδικασίας, όπου κάθε δραστηριότητα εμφανίζεται ακριβώς μία φορά σε κάθε εκτέλεση, αν Β εξαρτάται από την Α, τότε η Β ξεκίνα μετά που τελειώνει η δραστηριότητα Α, σε κάθε εκτέλεση στο αρχείο.

Λήμμα 2:

Όταν 2 γράφοι G και G' έχουν το ίδιο μεταβλητή κλειστότητα, τότε και οι δύο γράφοι είναι συνεπείς με το ίδιο σύνολο εκτελέσεων, αν κάθε δραστηριότητα εμφανίζεται ακριβώς μία φορά σε κάθε εκτέλεση.

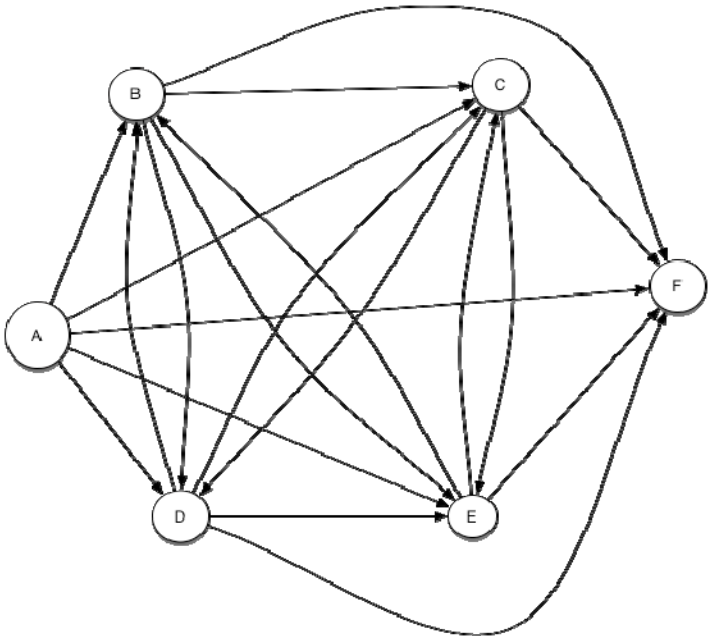
Λήμμα 3:
Δοθέντος ενός αρχείου εκτελέσεων της ίδιας διαδικασίας, όπου κάθε δραστηριότητα εμφανίζεται ακριβώς μία φορά σε κάθε εκτέλεση, ο γράφος εξαρτήσεων G είναι και συμβατός με το αρχείο.

Βήματα Αλγόριθμου

1. Ξεκινούμε με το γράφο G = (V,E), όπου V είναι το σύνολοε δραστηριοτήτων της διαδικασίας και E = Ø .
2. Για κάθε εκτέλεση της διαδικασίας στο L, και κάθε ζευγάρι εκτελέσεων u,v έτσι ώστε η u να τερματίζει πριν ξεκινήσει η v, πρόσθεσε την ακμή (u, v) στο E.
3. Αφαίρεσε από το E τις ακμές που εμφανίζονται και στις δύο κατευθύνσεις
   π.χ A->B, B->A.
4. Υπολόγισε την μεταβλητή μείωση (transitive reduction) του G. (Τον μικρότερο υπογράφο του G, ο οποίος έχει το ίδιο closure με τον G.)
5. Επέστρεψε (V,E). [1]

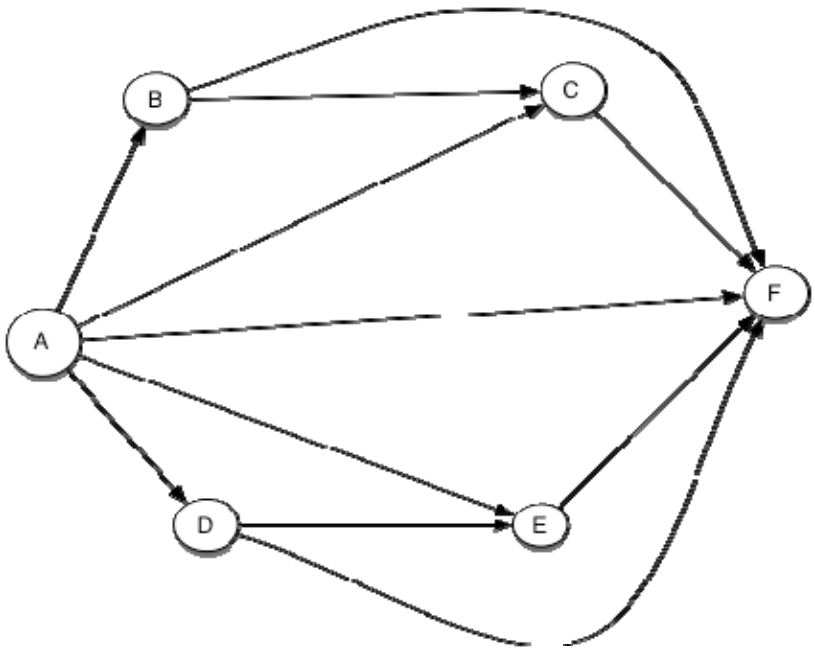
Χρόνος εκτέλεσης $O(n^2m)$

Παράδειγμα. : Έστω ότι έχουμε το αρχείο εκτελέσεων {ABCDEF, ABDECF, ADEBCF, ADBCEF, ADBECF}. Μετά το βήμα 2 έχουμε το γράφο στο *Σχήμα 2.2*.



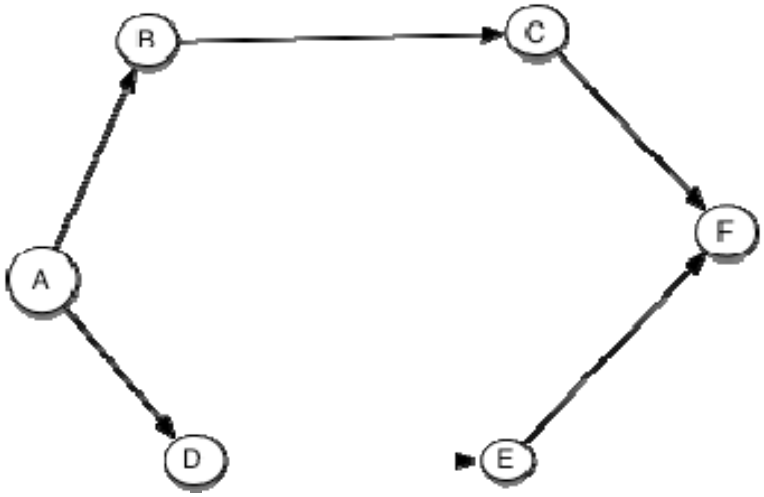
Σχήμα 2.2

Με το βήμα 3 αφαιρούμε τις ακμές που εμφανίζονται και στις δύο κατευθύνσεις, άρα τις ακμές ανάμεσα σε κόμβους που δεν είναι ανεξάρτητοι και παίρνουμε ένα γράφο ο οποίος είναι γράφος εξαρτήσεων, στο *Σχήμα 2.3.*



Σχήμα 2.3

Με το βήμα 4 υπολογίζουμε το transitive reduction του γράφου, δηλαδή βρίσκουμε τον μικρότερο υπογράφο ο οποίος έχει το ίδιο closure. Στο παράδειγμα μας θα αφαιρέσουμε την ακμή AC, αφού για να εκτελεσθεί η εργασία C, δεν αρκεί μόνο η εκτέλεση της A αλλά πρέπει να εκτελεστεί και η B , αφού στο αρχείο εκτελέσεων δεν υπάρχει εκτέλεση κατά την οποία να εκτελείται η C και να μην εκτελείται η B (σε αυτό τον αλγόριθμο υποθέτουμε ότι εκτελούνται όλες οι διεργασίες). Ξέρουμε όμως ότι η A θα έχει σίγουρα εκτελεστεί, αφού η B εξαρτάται από την A, έτσι βλέπουμε ότι η ακμή AC περισσεύει και την αφαιρούμε. Με την ίδια λογική αφαιρούμε και τις ακμές AF, BF, AE, DF. Έτσι παίρνουμε τον τελικό ελάχιστο γράφο εξαρτήσεων ο οποίος με βάση το λήμμα 3 είναι και ελάχιστος conformal γράφος στο *Σχήμα 2.4*.



Σχήμα 2.4

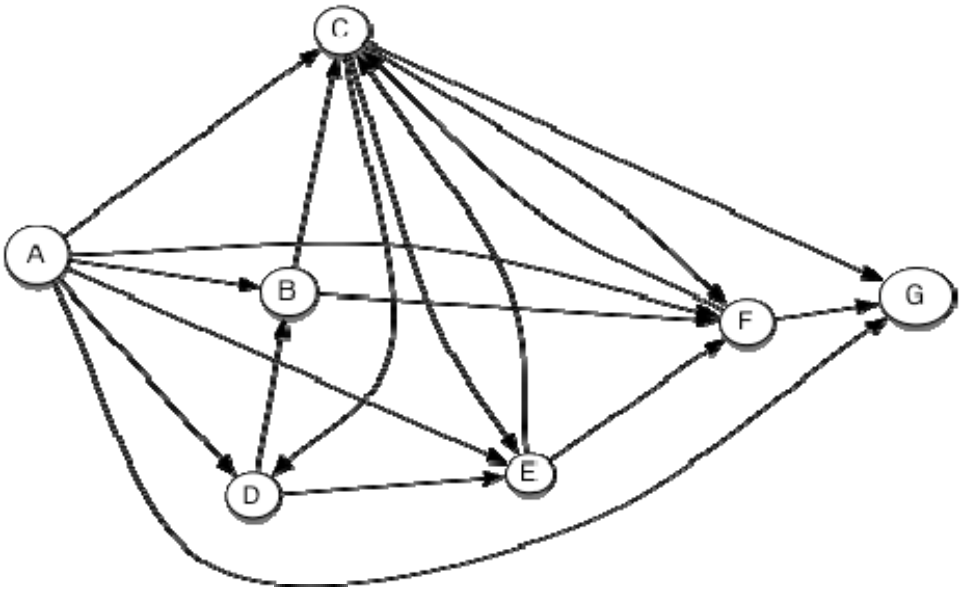
2.3 Αλγόριθμος 2 για κατασκευή κατευθυνόμενου γράφου (επέκταση αλγόριθμου 1)

Χρησιμοποιείται στη περίπτωση όπου και πάλι θέλουμε ακυκλικό γράφο αφού κάθε διαδικασία δεν μπορεί να εμφανιστεί πάνω από μία φορά σε μια εκτέλεση, αλλά τώρα κάθε εκτέλεση μπορεί να μην περιέχει όλες τις διαδικασίες.

1. Ξεκινούμε με το γράφο G = (V,E), όπου V είναι το σετ δραστηριοτήτων της διαδικασίας και E = Ø .

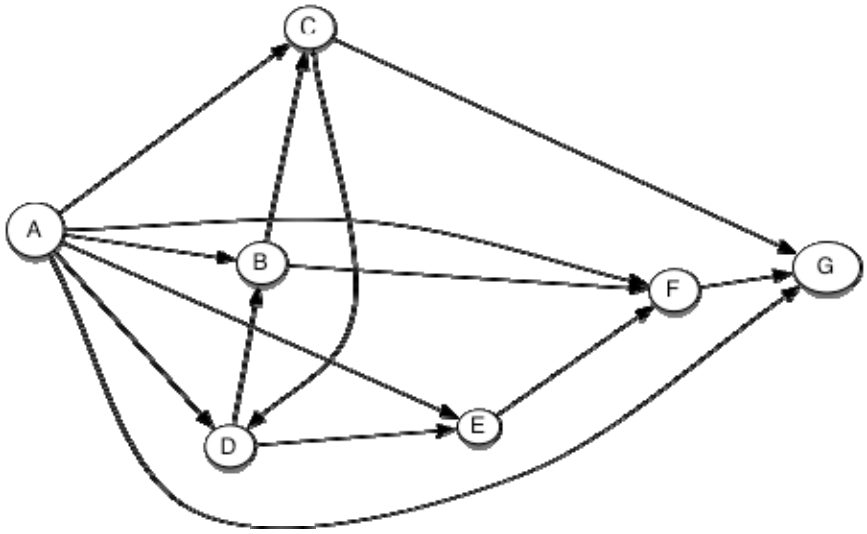
2. Για κάθε εκτέλεση της διαδικασίας στο L, και κάθε ζευγάρι εκτελέσεων u,v έτσι ώστε η u να τερματίζει πριν ξεκινήσει η v, πρόσθεσε την ακμή (u, v) στο E.
3. Αφαίρεσε από το E τις ακμές που εμφανίζονται και στης δύο κατευθύνσεις π.χ A->B, B->A.
4. Για κάθε «ισχυρά συνδεδεμένο μέρος» (strongly connected component) του G, αφαίρεσε από το E όλες τις ακμές ανάμεσα σε κορυφές του ίδιου «ισχυρά συνδεδεμένο μέρος». (για δύο κορυφές στο ίδιο «ισχυρά συνδεδεμένο μέρος» υπάρχει μονοπάτι ακολουθίας από τη πρώτη κορυφή στη δεύτερη και ανάποδα).
5. Για κάθε εκτέλεση της διαδικασίας στο L:
   a) Βρες τον παραγόμενο υπογράφο G΄.
   b) Υπολόγισε το transitive reduction του υπογράφου
   c) Σημείωσε τις ακμές στο E που εμφανίζονται και στο transitive reduction του G΄. [1]
6. Αφαίρεσε τις ακμές που δεν έχεις σημειώσει από το E.
7. Επέστρεψε (V,E).

Παράδειγμα: Έστω ότι έχουμε το αρχείο εκτελέσεων { ABDEFG, ACBFG, ACDEFG, ADEFCG}. Μετά το βήμα 2 παίρνουμε τον πιο κάτω γράφο:



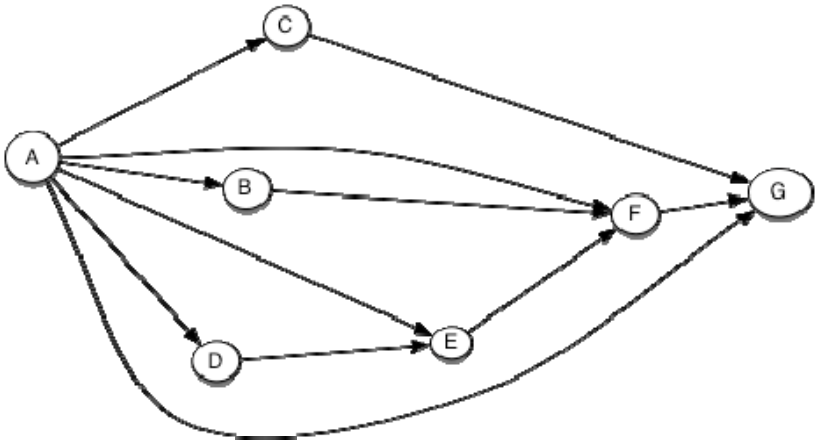
Σχήμα 2.5

Με το βήμα 3 αφαιρούμε τις ακμές που υπάρχουν και στις δύο κατευθύνσεις άρα δεν αποτελούν εξαρτήσεις. Παίρνουμε τον πιο κάτω γράφο:



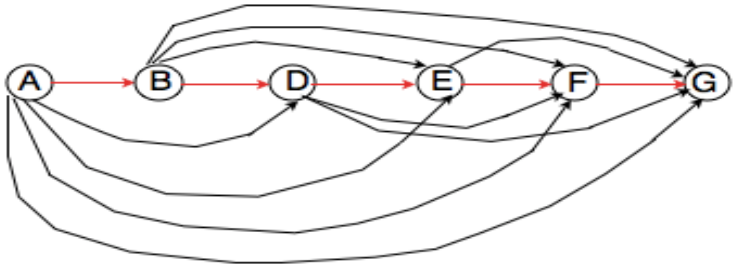
Σχήμα 2.6

Στη συνέχεια με βάση το βήμα 4 αφαιρούμε τις ακμές ανάμεσα στα strongly connected components, όπου στην περίπτωση μας υπάρχει στις ακμές C, B, D.



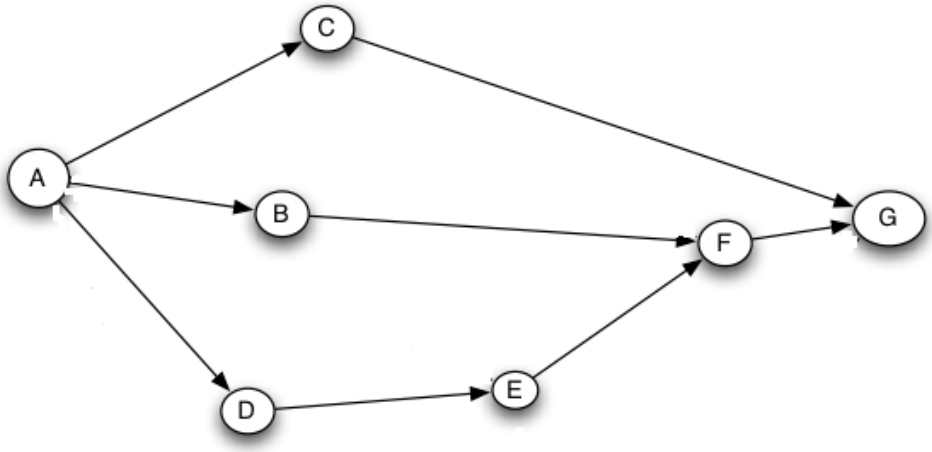
Σχήμα 2.7

Μετά το βήμα 5 και 6, φτιάχνουμε ένα γράφο για κάθε εκτέλεση, βρίσκουμε το transitive reduction όπου είναι ο μικρότερος υπογράφος με το ίδιο κλειστότητα (closure). Συγκρίνουμε τις ακμές που υπάρχουν στον γενικό μας γράφο με αυτές που υπάρχουν στους υπογράφους που πήραμε. Όσες ακμές υπάρχουν στους υπογράφους και υπάρχουν και στο γενικό γράφο τις σημειώνουμε. Στη συνέχεια αφαιρούμε από τον γενικό γράφο αυτές που δεν είναι σημειωμένες και παίρνουμε τον τελικό γράφο.



Σχήμα 2.8

Γράφος εκτέλεσης ABDEFG. Το transitive reduction είναι οι ακμές με κόκκινο. Με τον ίδιο τρόπο φτιάχνουμε γράφους και για τις υπόλοιπες εκτελέσεις. Όσες από τις ακμές του transitive reduction υπήρχαν και στο γενικό γράφο, παρατηρούμε ότι τις δεν τις αφαίρεσε ο αλγόριθμος και υπάρχουν στον τελικό αποτέλεσμα.



Σχήμα 2.9:*Τελικός Γράφος*

2.4 Εύρεση Γενικότερου γράφου:

Στην περίπτωση όπου έχουμε δραστηριότητες οι οποίες εμφανίζονται πάνω από μία φορά σε μια εκτέλεση, δεν θα μπορούμε να εφαρμόσουμε το πιο πάνω αλγόριθμο. Ο λόγος είναι ότι θα εμφανίζονται κύκλοι στο γράφο, τους οποίους ο αλγόριθμος θα διαγράφει, θεωρώντας ότι προέρχονται από ανεξάρτητες δραστηριότητες.

Μια τροποποίηση έτσι ώστε να βρίσκουμε γράφους και τέτοιων εκτελέσεων θα ήταν να φερθούμε σε κάθε εμφάνιση μιας δραστηριότητας στην εκτέλεση, σαν διαφορετική δραστηριότητα. Π.χ. αν έχουμε μια δραστηριότητα Α, για την πρώτη εμφάνιση της σε μια εκτέλεση θα δώσουμε το όνομα A_1, για τη δεύτερη το A_2 κ.ο.κ.

Κεφάλαιο 3

Εξόρυξη ροής εργασιών με α-αλγόριθμο

3.1 Ορισμοί του α-αλγόριθμου

Ο α-αλγόριθμος παίρνει σαν είσοδο ένα αρχείο δεδομένων-γεγονότων και επιστρέφει Place/-Transition (Petri net) δίκτυο. Στη πιο θεωρητική προσέγγιση του αλγόριθμου θεωρούμε ότι δεν υπάρχει θόρυβος στο αρχείο και ερευνούμε πως λειτουργεί ο αλγόριθμος κάτω από ιδανικές καταστάσεις.

Ο αλγόριθμος βασίζεται σε 4 σχέσεις ταξινόμησης που μπορούν να προκύψουν από ένα αρχείο : $>W$, $\rightarrow W$, $\#W$, and $||W$.

Ορισμός 1. (Log-based ordering relations) Έστω ότι *W είναι ένα αρχείο ροών εργασιών πάνω στο* T, δηλ. , $W \in \mathcal{P}(T^*)$. $a, b \in T$:

– $a >_W b$ αν και μόνο αν υπάρχει trace $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ και $i \in \{1, \ldots, n-2\}$

έτσι που $\sigma \in W$ και $t_i = a$ και $t_i+1 = b$. Δηλαδή το α προηγείται του b σε ένα trace σ,

– $a \rightarrow_W b$ αν και μόνο αν $a >_W b$ και $b \not>_W a$, δηλαδή το υπάρχουν εκτελέσεις όπου η εκτελείται α και αμέσως μετά εκτελείται η b. Δεν υπάρχουν όμως εκτελέσεις όπου να εκτελείται η b και αμέσως μετά η α.

– $a \#W\ b$ αν και μόνο αν $a \not>_W b$ και $b \not>_W a$, δηλαδή δεν υπάρχουν εκτελέσεις στο αρχείο όπου η α να εμφανίζεται αμέσως μετά την b, ή η b να εμφανίζεται αμέσως μετά την α. Άρα αυτή η σχέση μπορεί να δημιουργηθεί ανάμεσα στις διεργασίες οι οποίες απλά δεν ακολουθούν άμεσα η μία την άλλη, ή απλά δεν εμφανίζεται ποτέ στην ίδια εκτέλεση (άρα υπάρχει επιλογή ανάμεσα στις δύο διεργασίες.)

– $a||_W b$ αν και μόνο αν $a >_W b$ και $b >_W a$, δηλαδή ισχύει υπάρχει εκτέλεση στο αρχείο στην οποία το α να εμφανίζεται αμέσως μετά το b, αλλά υπάρχει και εκτέλεση στην οποία το b εμφανίζεται αμέσως μετά το α. [2,3]

Η σχέση $a \rightarrow W\ b$ υπονοεί αιτιότητα, αυτό σημαίνει ότι υπάρχει εξάρτηση ανάμεσα σε δύο διεργασίες. Ενώ οι σχέσεις $a \#_W\ b$ και $a||_W b$ χρησιμοποιούνται για διαφοροποίηση ανάμεσα σε παραλληλισμό και επιλογή. Αφού όλες οι πιο πάνω σχέσεις μπορούν να παραχθούν από την $>_W$, υποθέτουμε ότι το αρχείο είναι «πλήρες» σύμφωνα με τη σχέση $>_W$ δηλαδή αν υπάρχει διεργασία α και b όπου η α ακολουθείται άμεσα από την b ο αλγόριθμος υποθέτει ότι αυτό φαίνεται στο αρχείο. Τα δομημένα δίκτυα ροών εργασιών (SWF-nets) είναι υποκατηγορία των δικτύων ροών εργασιών (WF-nets) και των οποίων η δομή δείχνει αποκλειστικά τη συμπεριφορά. Συνεπώς στα SWF i)επιλογή και συγχρονισμός δεν μπορούν να ισχύουν ταυτόχρονα ii) και αν υπάρχει συγχρονισμός όλες οι προηγούμενες μεταβάσεις θα πρέπει να έχουν πυροδοτήσει. [2,3]

Ορισμός 2: ($\in$**,** *first***,** *last***)** Έστω T *ένα σύνολο εργασιών (tasks)*. Έστω $\sigma = a_1 a_2 \ldots a_n \in T^*$ *μία σειρά στο T μήκους n.* $\in$, *first*, και *last ορίζονται ως εξής*:

1. $a \in \sigma$ *αν και μόνο αν* $a \in \{a_1, a_2, \ldots a_n\}$,
2. Αν $n \geq 1$, *τότε* $first(\sigma) = a_1$ *και* $last(\sigma) = a_n$.

Ορισμός 3: *(Αλγόριθμος εξόρυξης «α») Έστω W* αρχείο ροής εργασιών στο T. *Το α(W) ορίζεται ως εξής*:

1. $T_W = \{t \in T \mid \exists \sigma \in W \; t \in \sigma\}$, είναι το σύνολο όλων των μεταβάσεων (διεργασιών) που ανήκουν στο *T* και ανήκουν σε μια εκτέλεση (trace) *σ* που ανήκει στο αρχείο εκτελέσεων *W*.

2. $T_I = \{t \in T \mid \exists \sigma \in W \; t = first(\sigma)\}$, είναι το σύνολο όλων των μεταβάσεων εισόδου που ανήκουν στο *T* δηλ. αποτελούν μετάβαση first για κάποιο trace *σ* που ανήκει στο αρχείο εκτελέσεων *W*.

3. $T_O = \{t \in T \mid \exists \sigma \in W t = last(\sigma)\}$, είναι το σύνολο όλων των μεταβάσεων εξόδου που ανήκουν στο *T* δηλ. αποτελούν μετάβαση last για κάποιο trace *σ* που ανήκει στο αρχείο εκτελέσεων *W*.

4. $X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall a \in A \forall b \in B \; a \rightarrow_W b \wedge$

$\forall a1,a2 \in A \; a1 \#_W a2 \wedge \forall b1,b2 \in B \; b1 \#_W b2\}$, αυτό το σύνολο χρησιμοποιείται για να ορίσουμε τις θέσεις (places) του δικτύου ροής εργασιών. Συγκεκριμένα σε αυτό το βήμα ανακαλύπτουμε ποιες μεταβάσεις έχουν αιτιολογική εξάρτηση ανάμεσα τους. Έτσι για κάθε πλειάδα (A,B) στο Xw, κάθε μετάβαση στο σύνολο A εξαρτάται αιτιολογικά με όλες τις μεταβάσεις του συνόλου B. Όμως καμία μετάβαση του A δεν ακολουθεί άλλη μετάβαση του A σε κάποια σειρά πυροδότησης. Το ίδιο ισχύει και για τις μεταβάσεις του B.

5. $Y_W = \{(A,B) \in X_W \mid \forall (A',B') \in X_W \; A \subseteq A' \wedge B \subseteq B' \Rightarrow (A,B) = (A',B')\}$. Σε αυτό το βήμα ορίζεται ξανά το Xw με βάση μόνο τα μεγαλύτερα στοιχεία του. Δηλ. Αν υπάρχουν κάποια στοιχεία (σύνολα) τα οποία είναι υποσύνολα άλλων μεγαλύτερων στοιχείων, τότε δεν τα συμπεριλαμβάνουμε στο Yw. Σε αυτό το βήμα καθορίζεται ο αριθμός των θέσεων χωρίς να υπολογίζεται όμως σε αυτόν τον αριθμό η θέση εισόδου και η θέση εξόδου.

6. $P_W = \{p(A,B) \mid (A,B) \in Y_W\} \cup \{i_W, o_W\}$, σε αυτό το βήμα κατασκευάζουμε τις θέσεις (places), που θα βρίσκονται ανάμεσα στις μεταβάσεις, καθώς επίσης και τα και τις θέσεις εισόδου και εξόδου.

7. $F_W = \{(a, p(A,B)) \mid (A,B) \in Y_W \wedge a \in A\} \cup \{(p(A,B), b) \mid (A,B) \in Y_W \wedge b \in B\} \cup \{(i_W, t) \mid t \in TI\} \cup \{(t, o_W) \mid t \in T_0\}$. Εδώ οι μεταβάσεις συνδέονται με τις κατάλληλες μεταβάσεις εισόδου και μεταβάσεις εξόδου.

8. $\alpha(W) = (P_W, T_W, F_W)$. [2.3]

Ορισμός 4: (Ability to rediscover): Έστω N=(P,T,F) είναι ένα δίκτυο WF (ροής εργασιών) και «α» ο mining αλγόριθμος ο οποίος από αρχεία ροής εργασιών του N κατασκευάζει ένα WF δίκτυο. Αν από οποιοδήποτε αρχείο W του N ο «α» αλγόριθμος επιστρέφει το N, τότε ο «α» μπορεί να ανακατασκευάσει το N. Έχει αποδειχθεί ότι ο α-αλγόριθμος μπορεί να ανακατασκευάζει SWF δίκτυα αν δεν περιέχουν μικρούς βρόγχους (μήκους 1 ή 2), μπορεί όμως να ανακατασκευάσει και άλλα WF δίκτυα. Υπάρχουν όμως και αρκετά WF δίκτυα που λόγω κάποιων περιορισμών δεν μπορεί να ανακατασκευάσει.

Τα SWF-nets (structured workflow Petri nets), είναι μία κατηγορία δικτύων ροής εργασιών στα οποία η επιλογή και ο συγχρονισμός δεν μπορούν να μπλέκονται και αν υπάρχει συγχρονισμός κάποιων μεταβάσεων θα πρέπει όλες οι προηγούμενες μεταβάσεις να έχουν πυροδοτήσει. [2]

Παράδειγμα 3.1: Έστω L ένα αρχείο εισόδου, το οποίο περιέχει τις περιπτώσεις του *Πίνακα 1.1.* Συνεπώς οι εκτελέσεις που θα περιέχει αυτό το αρχείο είναι :
Case 1: ABCDE, Case 2: ABDCE, Case 3: AFG, Case 4: ABCDE
Οι σχέσεις που δημιουργεί ο α-αλγόριθμος ανάμεσα στις διεργασίες είναι:
$A >_W B$, $B \not>_W A \Rightarrow A \rightarrow_W B$: Άρα η B εξαρτάται από την A.

$A >_W F$, $F \not>_W A$ => $A \rightarrow_W F$: Άρα η F εξαρτάται από την A.

$B >_W C$, $C \not>_W B$ => $B \rightarrow_W C$: Άρα η C εξαρτάται από την B.

$B >_W D$, $D \not>_W B$=> $B \rightarrow_W D$: Άρα η D εξαρτάται από την B.

$C >_W D$, $D >_W C$=> $B \|_W D$: Άρα η C είναι παράλληλη με την B.

$C >_W E$, $E \not>_W C$=> $C \rightarrow_W E$: Άρα η E εξαρτάται από την C.

$D >_W E$, $E \not>_W D$=> $D \rightarrow_W E$: Άρα η E εξαρτάται από την D.

$F >_W G$, $G \not>_W F$ => $F \rightarrow_W G$: Άρα η G εξαρτάται από την F.

Ανάμεσα στις υπόλοιπες διεργασίες κατασκευάζεται η σχέση *a #W b.* Με βάση αυτές τις σχέσεις που κατασκευάζονται με τον α-αλγόριθμο το Petri-net που κατασκευάζεται είναι το ίδιο με αυτό στο *Σχήμα 1.8.*

3.2 Περιορισμοί του α-αλγόριθμου

Ο α-αλγόριθμος αντιπροσωπεύει πολλές πραγματικές ροές εργασιών χρησιμοποιώντας SWF δίκτυα. Υπάρχουν όμως και αρκετές κατασκευές που δεν υποστηρίζουν τα SWF δίκτυα και άλλες που ο α-αλγόριθμος δεν μπορεί να εξορύξει σωστά, όπως τους μικρούς βρόγχους.

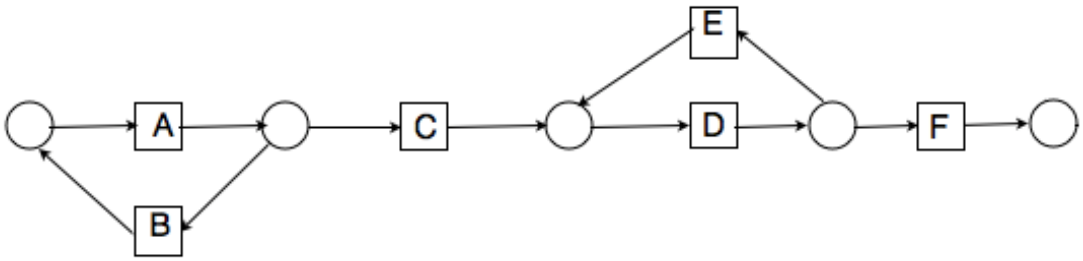
Για να βρούμε τις κατασκευές τις οποίες ο αλγόριθμος δεν μπορεί να εξορύξει σωστά, είναι αναγκαίο να καταλάβουμε πως λειτουργεί:

-Μια διεργασία υπάρχει στο τελικό δίκτυο αν βρίσκεται σε κάποια εκτέλεση του αρχείου.

-Μια διεργασία (μετάβαση) έχει εισερχόμενες ακμές στο τελικό δίκτυο i) αν είναι το η πρώτη διεργασία (first) στις εκτελέσεις του αρχείου ii)αν ακολουθεί αιτιολογικά κάποια άλλη διεργασία.

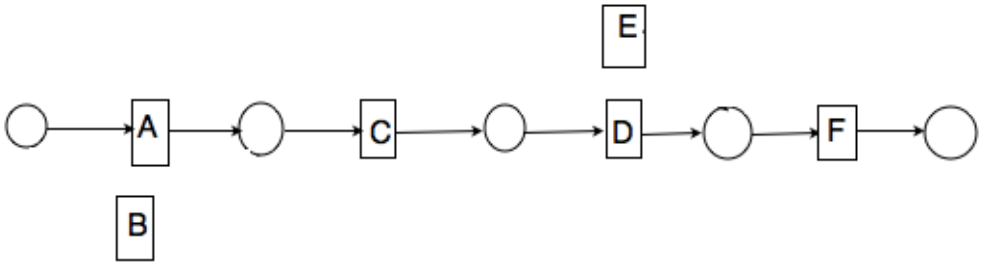
-Μια διεργασία έχει εξερχόμενες ακμές στο τελικό δίκτυο i) αν είναι η τελευταία διεργασία (last) σε εκτελέσεις του αρχείου ii)αν ακολουθείτε αιτιολογικά από κάποια άλλη διεργασία.

Αν μια διεργασία δεν είναι η πρώτη ή η τελευταία σε μια εκτέλεση του αρχείου και δεν συμμετέχει σε οποιαδήποτε αιτιολογική σχέση τότε ο α-αλγόριθμος δεν δημιουργεί εισερχόμενες και εξερχόμενες ακμές για αυτή τη διεργασία [3]. Για παράδειγμα όταν έχουμε το δίκτυο στο *Σχήμα 3.1*, λόγω του ότι για τις μεταβάσεις B και E δεν ισχύουν τα πιο πάνω, δεν συμπεριλαμβάνονται στο δίκτυο που κατασκευάζει ο α-αλγόριθμος *Σχήμα 3.2* .



Σχήμα 3.1

Το δίκτυο μετά την εφαρμογή του αλγορίθμου στα αρχεία εκτελέσεων.

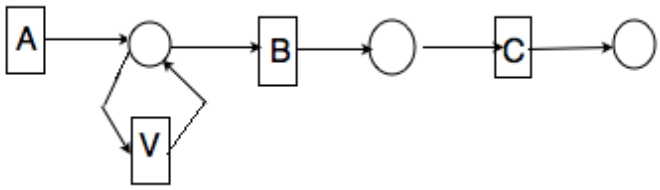


Σχήμα 3.2

Ωστόσο, ακόμα και αν όλες οι μεταβάσεις είναι συνδεδεμένες στο τελικό δίκτυο δεν μας εγγυάται ότι είναι σωστά συνδεδεμένο.

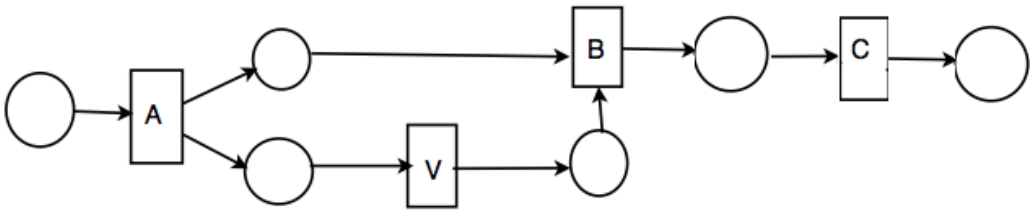
Κατασκευές τις οποίες ο α-αλγόριθμος δεν αναπαριστά σωστά

- *Βρόγχοι μήκους 1* (One-length-loop), όπου η ίδια διεργασία (μετάβαση) μπορεί να εκτελεστεί μία ή περισσότερες φορές. Η θέση εισόδου αυτής της μετάβασης είναι και θέση εξόδου. Ο λόγος που δεν μπορεί να αναπαραστήσει σωστά αυτήν την κατασκευή ο α-αλγόριθμος είναι , ότι για να μπορούσε να την αναπαραστήσει σωστά θα έπρεπε να παράξει μία θέση (place) με την ίδια μετάβαση εισόδου και εξόδου. Όμως ο α-αλγόριθμος απαιτεί την σχέση αιτιότητας $X \rightarrow_W X$, είναι αδύνατον όμως να ισχύουν ταυτόχρονα $X >_W X$ και $X \not>_W X$ [3]. Πιο κάτω ακολουθεί ένα παράδειγμα, με το *Σχήμα 3.2* να δείχνει πως είναι ένα πραγματικό δίκτυο, με ένα βρόγχο μήκους 1, και πως ο α-αλγόριθμος θα το κατασκευάσει με βάση τα αρχεία εκτελέσεων *Σχήμα 3.4* .



Σχήμα 3.3

Δίκτυο που κατασκεύασε ο α-αλγόριθμος



Σχήμα 3.4

- *Βρόγχοι μήκους 2* (Two-length-loop), σε αυτή τη περίπτωση ο α-αλγόριθμος υποθέτει ότι οι εμπλεκόμενες διεργασίες εκτελούνται παράλληλα, άρα ότι μεταξύ τους ισχύει $A||_W B$. Αν ο αλγόριθμος μπορούσε να παράξει για τις δύο διεργασίες τις σχέσεις $A \rightarrow_W B$ και $B \rightarrow_W A$ τότε θα μπορούσε να παράξει σωστά το δίκτυο. Στο

Σχήμα 3.1 και *Σχήμα 3.2* φαίνονται καθαρά δύο περιπτώσεις βρόγχων μήκους δύο [3].

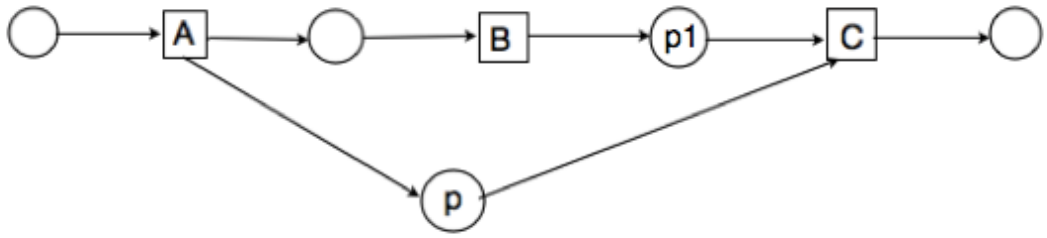
-*Αόρατες διεργασίες* (Invisible tasks): Αυτές οι διεργασίες δεν εμφανίζονται στο αρχείο, άρα δεν ανήκουν στο Tw με αποτέλεσμα να μην μπορούν να εμφανιστούν στο δίκτυο που κατασκευάζει ο α-αλγόριθμος.

-*Διπλές (Duplicated) Διεργασίες* Μερικές φορές υπάρχουν διεργασίες οι οποίες έχουν το ίδιο όνομα, και θα έπρεπε να εμφανίζονται στο δίκτυο ροής εργασιών σαν διαφορετικές μεταβάσεις με το ίδιο όνομα. Π.χ Έχω μια διαδικασία που περιγράφει ένα επαγγελματικό ταξίδι από μια πόλη Α σε κάποια άλλη Β, αν αυτή η διαδικασία περιέχει δύο φορές την διεργασία "Ταξίδι με τρένο" μια για να πάω από την Α στην Β και μια για να επιστρέψω, τότε κανονικά αυτές οι διεργασίες είναι δυο διαφορετικές με το ίδιο όνομα (μπορούσε η μια διαδρομή να γινόταν με τρένο και η άλλη με αυτοκίνητο) [11] . Αυτές οι διεργασίες όμως δεν μπορούν να τα αναγνωριστούν σαν δύο διαφορετικές από τον α-αλγόριθμο, με αποτέλεσμα να μην παράγεται σωστά το τελικό δίκτυο.

-*Υπονοούμενες (Implicit) θέσεις* : Είναι οι θέσεις (places) των οποίων η παρουσία ή η απουσία δεν επηρεάζει τις πιθανές εκτελέσεις του αρχείου. Έτσι είτε υπάρχουν, είτε απουσιάζουν αυτές οι θέσεις, οι σχέσεις αιτιότητας οι οποίες θα παραχθούν θα είναι οι ίδιες. Αυτό έχει ως αποτέλεσμα, ο α-αλγόριθμος να μην μπορεί να συμπεράνει από το αρχείο αυτές τις θέσεις αφού δεν επηρεάζουν τις σχέσεις αιτιότητας και να μην τις συμπεριλαμβάνει στο παραγόμενο δίκτυο. Για τον ίδιο λόγο, ο αλγόριθμος δεν μπορεί να αναπαράξει τις θέσεις που βρίσκονται ανάμεσα σε διεργασίες η οποίες δεν έχουν σχέσεις αιτιότητας ανάμεσα τους [3].

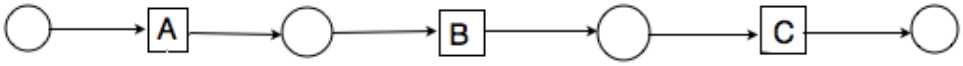
Ένα παράδειγμα δικτύου με υπονοούμενη θέση είναι το πιο κάτω στο *Σχήμα 3.5* και *Σχήμα 3.6*, όπου στο πρώτο σχήμα παρατηρούμε το πραγματικό δίκτυο ενώ το δεύτερο παρουσιάζει το δίκτυο που κατασκευάζει ο α-αλγόριθμος. Όπως παρατηρούμε από το πραγματικό δίκτυο, η εκτέλεση AC δεν μπορεί να παραχθεί, άρα δεν υπάρχει η σχέση $A \rightarrow_W C$. Η μόνη εκτέλεση που μπορεί να παραχθεί είναι

ABC, αφού η θέση *p* είναι υπονοούμενη και για να εκτελεστεί η C δεν αρκεί να πυροδοτήσει μόνο η *p* αλλά και η *p1*. Έτσι με βάση τις εκτελέσεις και τις σχέσεις που μπορούν να παραχθούν ο α-αλγόριθμος κατασκευάζει διαφορετικό δίκτυο από το πραγματικό.



Σχήμα 3.5

Δίκτυο αλγόριθμου

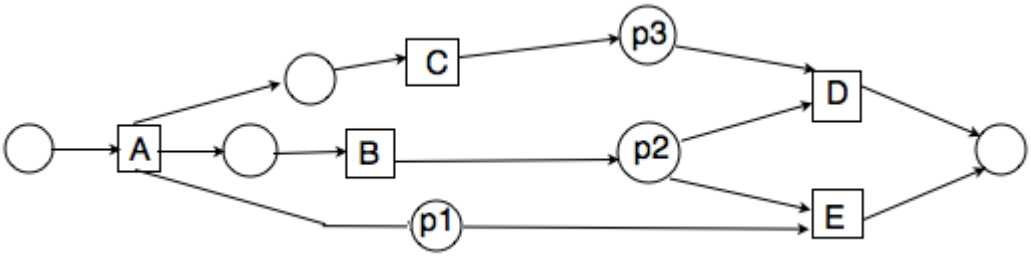


Σχήμα 3.6

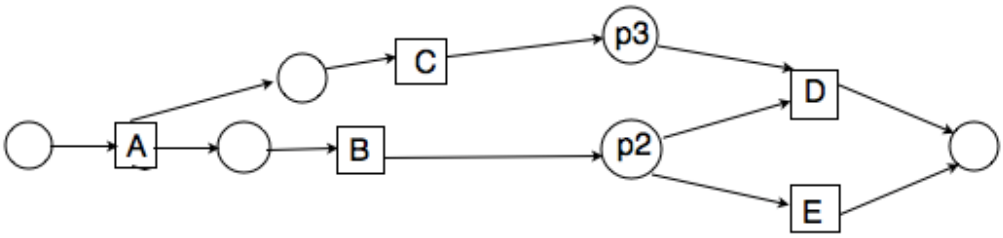
-Μη ελεύθερης επιλογής (Non free choices) κατασκευές: Αυτές οι κατασκευές συνδυάζουν συγχρονισμό και επιλογή, συνδυασμό τον οποίο δεν επιτρέπουν τα δίκτυα SWF. Αυτά τα δίκτυα δεν μπορούν πάντα να παραχθούν σωστά από τον α-αλγόριθμο [3].

Ένα παράδειγμα με « μη ελεύθερη επιλογή » που δεν μπορεί να παράξει σωστά είναι αυτό που φαίνεται στο *Σχήμα 3.7* . Στο συγκεκριμένο παράδειγμα, ο συνδυασμός συγχρονισμού και επιλογής, βρίσκεται στις μεταβάσεις D και E, για τις οποίες η θέση *p2* πρέπει να κάνει επιλογή, αλλά ταυτόχρονα και για τις δύο υπάρχει συγχρονισμός από δύο θέσεις. Για την D υπάρχει συγχρονισμός από την *p2* και *p3*, ενώ για την E υπάρχει συγχρονισμός από την *p1* και *p2*. Γι' αυτό το παράδειγμα συγκεκριμένα το πρόβλημα είναι ότι δεν μπορούμε να έχουμε την εκτέλεση με AE, γιατί για την εκτέλεση του E πρέπει να πυροδοτήσουν δύο tokens, το *p1* και *p2*. Για την πυροδότηση όμως του p2 θα πρέπει να έχει ήδη εκτελεστεί εκτός από την A και η B, αφού . Έτσι χωρίς την ύπαρξη της εκτέλεσης

με substring AE στο αρχείο ,παράγονται οι σχέσεις A->$_W$ B και B->$_W$ E δεν παράγεται η σχέση A->$_W$ E, άρα ο αλγόριθμος δεν θα μπορεί να παράξει την θέση *p1*.



Σχήμα 3.7: *Πραγματικό δίκτυο*



Σχήμα 3.8: *Δίκτυο που θα παράξει ο αλγόριθμος*

Όπως παρατηρούμαι για τη « μη ελεύθερη επιλογή » στην μετάβαση D δεν υπήρξε πρόβλημα, παράγονται οι σχέσεις A→$_W$ C, C//$_W$ B, C→$_W$ D, B→$_W$ D, αφού μπορεί να παραχθούν οι απαραίτητες σχέσεις, A→$_W$ C, C→$_W$ D, τότε αυτό το σημείο μπορεί να αναπαρασταθεί σωστά. Άρα ο α-αλγόριθμος, κάποιες φορές μπορεί να παράξει τέτοιου είδους δίκτυα.

- Συγχρονισμός των OR-join θέσεων: Κατασκευή την οποία επίσης δεν επιτρέπουν τα SWF. Σε αυτού του είδους κατασκευές υπάρχει συγχρονισμός των OR-join. Ο α-αλγόριθμος μπορεί να ανακατασκευάσει δίκτυα με αυτές τις κατασκευές, αυτό όμως δεν ισχύει πάντα [3].

Τρόποι αντιμετώπισης αυτώ των προβλημάτων μέσω προεπεξεργασίας των δεδομένων εισόδου αναπτύσσονται στο [3].

3.3 Σύγκριση α-αλγόριθμου με αλγόριθμο κατασκευής κατευθυνόμενου γράφου

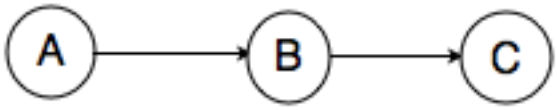
Ο πρώτος αλγόριθμος εξόρυξης ροών εργασιών, που αναφέραμε ο οποίος κατασκευάζει κατευθυνόμενο γράφο ήταν η πρώτη ιδέα που προτάθηκε για εισαγωγή της εξόρυξης διαδικασιών, στο πλαίσιο της διαχείρισης ροών εργασιών. Ο αλγόριθμος αυτός παίρνει σαν είσοδο ένα αρχείο αδόμητων εκτελέσεων και μέσα από αυτό το αρχείο προσπαθεί να κατασκευάσει τον κατευθυνόμενο γράφο, ο οποίος αναπαριστά την ροή ελέγχου της επιχειρησιακής διαδικασίας και ικανοποιεί τις ιδιότητες : πληρότητα, απεριτότητα και ελαχιστότητα που αναφέραμε προηγουμένως.

Ο α-αλγόριθμος παίρνει σαν είσοδο ένα αρχείο από γεγονότα και προσπαθεί να κατασκευάσει ένα Petri-net. Κάθε γεγονός μπορεί να αναφέρεται είτε σε μια διαδικασία, είτε σε περιπτώσεις που αποτελούν στιγμιότυπα της διαδικασίας, είτε σε διεργασίες οι οποίες αποτελούν ολόκληρη τη διαδικασία. Μέσα από αυτό το αρχείο προσπαθεί να βρει τις σχέσεις που υπάρχουν μεταξύ των διεργασιών έτσι ώστε να κατασκευάσει το Petri-net.

Οι δύο αυτοί αλγόριθμοι λειτουργούν με διαφορετικό τρόπο και κατασκευάζουν διαφορετικά γραφήματα. Ο πρώτος αλγόριθμος πρώτα ενώνει τους κόμβους μεταξύ τους ανάλογα με την εμφάνιση της κάθε διεργασίας στις εκτελέσεις του αρχείου και στη συνέχεια διαγράφει τις ακμές που βρίσκονται ανάμεσα σε διεργασίες οι οποίες δεν έχουν αιτιολογική σχέση μεταξύ τους. Ο α-αλγόριθμος πρώτα βρίσκει τις σχέσεις οι οποίες υπάρχουν ανάμεσα στο αρχείο και μετά κατασκευάζει το Petri-net.

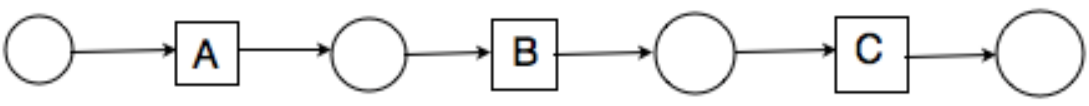
Οι σχέσεις εξάρτησης ανάμεσα στις διεργασίες στους κατευθυνόμενους γράφους αναπαριστώνται με ακμή από τον ένα κόμβο στον άλλο όταν υπάρχει άμεση σχέση, δηλαδή η μια διεργασία εκτελείται αμέσως μετά την εκτέλεση της άλλης. Διαφορετικά αν υπάρχει σχέση εξάρτησης και δεν είναι άμεση, υπάρχει μονοπάτι

που οδηγεί από την μία στην άλλη, με ενδιάμεσους κόμβους. Όπως φαίνεται στο *Σχήμα 3.9* μπορούμε να πούμε ότι η B εξαρτάται από την A, άμεσα γι αυτό και υπάρχει ακμή από την A στην B. Το ίδιο ισχύει και για την C που εξαρτάται άμεσα από την B. Επίσης στο ίδιο σχήμα βλέπουμε ότι η C εξαρτάται από την A, η σχέση όμως δεν είναι άμεση δηλ. η C δεν εκτελείται αμέσως μετά την εκτέλεση της A, υπάρχει όμως μονοπάτι από την A στην C.



Σχήμα 3.9

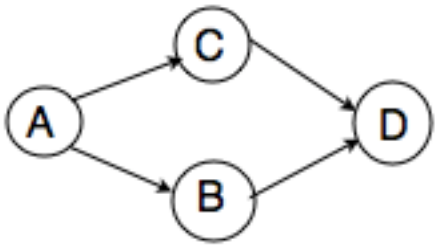
Στα Petri-nets τα πράγματα είναι διαφορετικά. Υπάρχουν δύο είδη κόμβων, οι μεταβάσεις ,οι οποίες αναπαριστούν τις διεργασίες και οι θέσεις. Οι ακμές φεύγουν είτε από μεταβάσεις και φτάνουν σε θέσεις είτε ανάποδα, δεν μπορούν όμως να υπάρχουν ακμές από μετάβαση σε μετάβαση ή από θέση σε θέση. Οι αντίστοιχες εξαρτήσεις που στον κατευθυνόμενο γράφο αναπαριστώνται με ακμή από τον ένα κόμβο στον άλλο, στα Petri-net αναπαρίστανται με ακμή από την μια μετάβαση σε μια θέση, και από την θέση ακμή προς την δεύτερη μετάβαση. Αν υπάρχει σχέση η οποία δεν είναι άμεση, τότε και στα Petri-nets υπάρχει μονοπάτι από την μία μετάβαση στην άλλη, το οποίο εκτός από ενδιάμεσες θέσεις περιέχει και ενδιάμεσες μεταβάσεις. Το παράδειγμα του σχήματος 3.9, στα Petri-net θα είναι όπως φαίνεται στο *Σχήμα 3.10*:



Σχήμα 3.10

Στην περίπτωση που δεν υπάρχει εξάρτηση ανάμεσα σε δύο διεργασίες είτε επειδή δεν εμφανίζονται ποτέ στην ίδια εκτέλεση, είτε επειδή εμφανίζονται μαζί σε

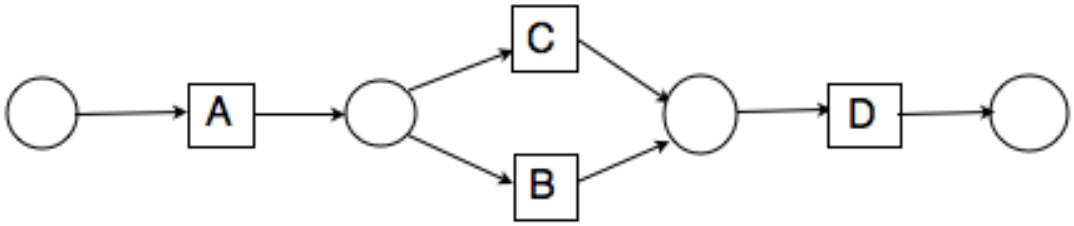
εκτελέσεις αλλά όχι πάντα με την ίδια σειρά, τότε στον κατευθυνόμενο γράφο δεν υπάρχει μονοπάτι από τη μία διεργασία στην άλλη. Αν δύο διεργασίες δεν εμφανίζονται με την ίδια σειρά στις εκτελέσεις, αυτό σημαίνει ότι είναι ανεξάρτητες και μπορούν να εκτελεστούν παράλληλα. Αν δύο διεργασίες της ίδιας διαδικασίας δεν εμφανίζονται πότε στην ίδια εκτέλεση, τότε σημαίνει ότι η εκτέλεση της μιας αναιρεί την εκτέλεση της άλλης και ότι κάθε φορά πρέπει να επιλεγεί να εκτελεστεί μόνο η μία από τις δύο. Στην αναπαράσταση με τον κατευθυνόμενο γράφο δεν υπάρχει ξεκάθαρη διευκρίνιση αν υπάρχει παραλληλισμός ανάμεσα στις δύο διεργασίες ή επιλογή. Ένας γράφος με τέτοιου είδους εξαρτήσεις μπορεί να παράξει εκτελέσεις στις οποίες εμφανίζονται και οι δύο διεργασίες με τυχαία σειρά αλλά και εκτελέσεις όπου μία από τις διεργασίες εμφανίζεται. Ο πιο κάτω γράφος είναι παράδειγμα μιας τέτοιας περίπτωσης, όπου η B και C μπορούν να εκτελεσθούν παράλληλα, ή να εκτελεσθεί η μία από τις δύο. Οι εκτελέσεις που μπορεί να παραχθούν είναι : ABCD, ACD,ABD.



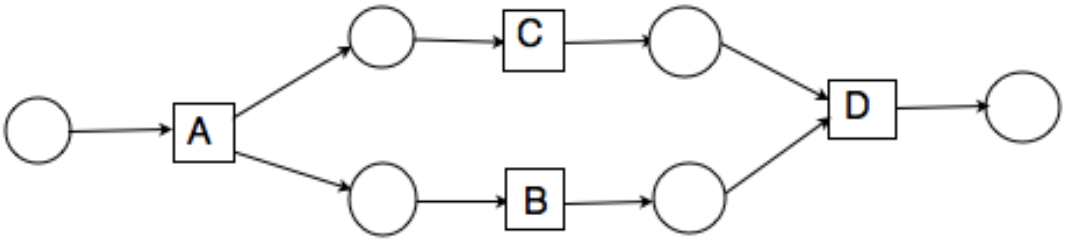
Σχήμα 3.11

Στα Petri-nets που κατασκευάζει ο α-αλγόριθμος υπάρχει ξεκάθαρος διαχωρισμός ανάμεσα στις διεργασίες οι οποίες είναι παράλληλες και σε αυτές που υπάρχει επιλογή ανάμεσα τους. Αυτό συμβαίνει λόγω του ότι ο α-αλγόριθμος έχει διαφορετικές σχέσεις ταξινόμησης για αυτές τις δύο περιπτώσεις. Συγκεκριμένα υπάρχει η σχέση $a \#_W b$ ανάμεσα στις διεργασίες οι οποίες δεν εμφανίζονται μαζί (η μία μετά την άλλη) στις ίδιες εκτελέσεις άρα θα αποτελούν διεργασίες στις οποίες υπάρχει επιλογή ανάμεσα τους. Για τις διεργασίες οι οποίες η μία ακολουθεί την άλλη με διαφορετική σειρά σε κάθε εκτέλεση υπάρχει η σχέση $a||_W b$ η οποία δηλώνει ότι είναι παράλληλες. Στα Petri-nets η αναπαράσταση των παράλληλων διεργασιών γίνεται με τις AND κατασκευές, ενώ των διεργασιών που υπάρχει επιλογή ανάμεσα τους γίνεται με τις OR κατασκευές. Πιο κάτω, στο *Σχήμα 3.12*

βλέπουμε την κατασκευή όπου υπάρχει επιλογή ανάμεσα στις C και B και μόνο μία από τις δύο εκτελείται κάθε φορά ενώ στο *Σχήμα 3.13* εκτελούνται και οι δύο παράλληλα.



Σχήμα 3.12

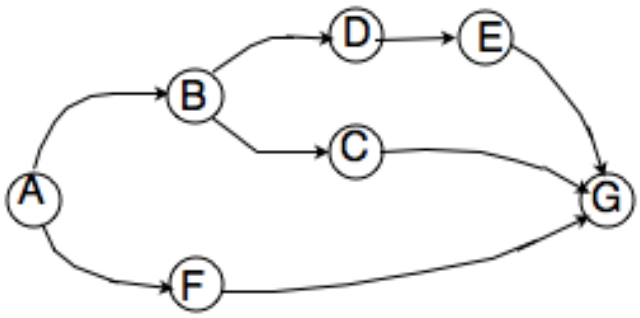


Σχήμα 3.13

Μία άλλη κατασκευή στην οποία υστερεί ο αλγόριθμος που παράγει κατευθυνόμενο γράφο, είναι στην περίπτωση που μία δραστηριότητα εμφανίζεται πάνω από μία φορά. Αυτός ο αλγόριθμος, αναπαριστά τις επαναλήψεις με κύκλους τους οποίους στη συνέχεια διαγράφει θεωρώντας ότι προέρχονται από διεργασίες που δεν έχουν σχέση εξάρτησης μεταξύ τους. Μία λύση που δίνεται σε αυτό το πρόβλημα, είναι να δημιουργείται διαφορετικός κόμβος για κάθε εμφάνιση μιας διεργασίας, και να του δίνεται διαφορετικό όνομα, όπως αναφέρθηκε πιο πάνω. Αυτή η λύση όμως είναι περιοριστική, αφού υπάρχει καθορισμένος αριθμός επαναλήψεων που θα μπορεί να παράξει το μοντέλο που θα κατασκευαστεί. Στον α-αλγόριθμο τα πράγματα είναι διαφορετικά, εδώ υπάρχει πρόβλημα στην αναπαράσταση κατασκευών με βρόγχους μήκους 1 ή 2, αν και έχουν προταθεί τρόποι επίλυσης αυτού του προβλήματος με προεπεξεργασία των δεδομένων εισόδου. Οποιεσδήποτε άλλες επαναλήψεις είναι επιτρεπτές και μπορούν να αναπαρασταθούν σωστά από τον α-αλγόριθμο.

Ένα σημείο στο οποίο υστερεί ο α-αλγόριθμος προκύπτει από το γεγονός ότι α-αλγόριθμος θεωρεί ένα αρχείο πλήρες ως προς τη σχέση >w. Γι αυτό το λόγο δεν μπορεί να βγάλει πάντα τα σωστά συμπεράσματα σχετικά με διεργασίες που υπάρχουν στις ίδιες εκτελέσεις αλλά δεν ακολουθούν άμεσα η μία την άλλη. Για να μπορέσει να βγάλει το επιθυμητό μοντέλο διαδικασίας θα πρέπει να υπάρχουν πολύ περισσότερες εκτελέσεις σε ένα αρχείο από ότι θα χρειάζονταν για την κατασκευή του αντίστοιχου κατευθυνόμενου γράφου. Για την καλύτερη κατανόηση αυτής της περίπτωσης ας δούμε το παράδειγμα που ακολουθεί.

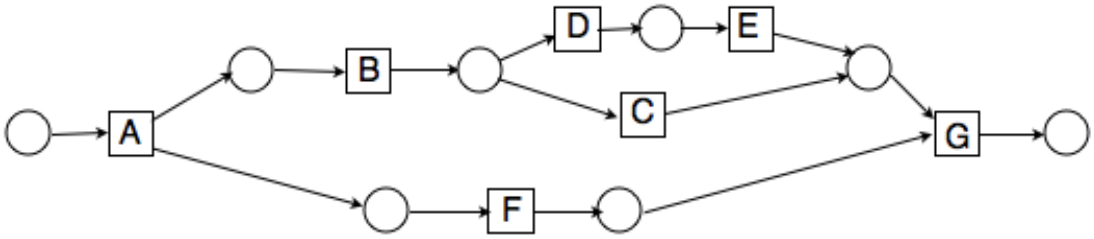
Παράδειγμα 3.2: Έχουμε ένα αρχείο με τις ακόλουθες εκτελέσεις {ABCFG, AFBCG, ABDEFG, AFBDEG}. Αν σε αυτό το παράδειγμα εφαρμόσουμε το δεύτερο αλγόριθμο παραγωγής κατευθυνόμενου γράφου (εφαρμόζουμε τον δεύτερο αφού δεν υπάρχουν όλες οι διεργασίες σε όλες τις εκτελέσεις), θα πάρουμε τον γράφο που ακολουθεί:



Σχήμα 3.14

Με βάση τον γράφο που κατασκευάζει αυτός ο αλγόριθμος και τις εκτελέσεις της διαδικασίας μπορούμε να βγάλουμε κάποια συμπεράσματα σχετικά με τις σχέσεις ανάμεσα στις διεργασίες. Η B και F εξαρτώνται άμεσα από την A, αφού την ακολουθούν στον γράφο και είτε η μία είτε η άλλη την ακολουθούν στις εκτελέσεις. Από την A εξαρτώνται και όλες οι άλλες διεργασίες αφού πάντα εκτελούνται μετά από την εκτέλεση της. Η B και η F με βάση το σχήμα μπορούν να εκτελεστούν είτε παράλληλα και οι δύο, είτε η μία από τις δύο, με βάση όμως τις εκτελέσεις του

αρχείου παρατηρούμε ότι εκτελούνται παράλληλα αφού εμφανίζονται και οι δύο σε όλες τις εκτελέσεις με διαφορετική σειρά κάθε φορά. Μετά την B υπάρχει επιλογή ανάμεσα στην εκτέλεση είτε της C, είτε της D και E αφού στις εκτελέσεις του αρχείου είτε εμφανίζεται η C είτε οι D και E, ποτέ δεν εμφανίζονται και οι τρεις. Η E εξαρτάται από την D, αφού εκτελείται πάντα μετά την E. Παρατηρούμε ότι η F είναι παράλληλη και με τις B, C, D, E. Η G εξαρτάται από όλες και εκτελείται πάντα τελευταία. Άρα λοιπόν αν θέλαμε να μετατρέψουμε τον πιο πάνω γράφο σε Petri-net, με βάση τις πληροφορίες που έχουμε θα κατασκευάζαμε αυτό στο *Σχήμα 3.15*, το οποίο θεωρούμε ότι αναπαριστά την πραγματική διαδικασία και μπορεί να αναπαράξει το αρχείο :



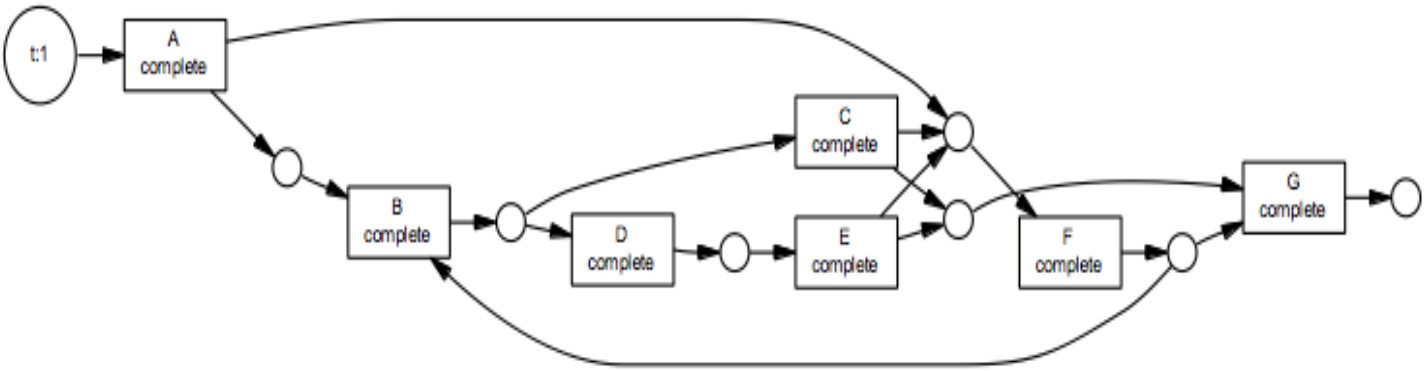
Σχήμα 3.15

Όταν εφαρμόσουμε τον α-αλγόριθμο δεν παίρνουμε το ίδιο μοντέλο. Ο λόγος είναι ότι ο α-αλγόριθμος στο συγκεκριμένο παράδειγμα λόγω της έννοιας της πληρότητας που αναφέραμε προηγουμένως θα παράξει τις σχέσεις:

$A \to_W B$, $A \to_W B$, $B \to_W C$, $B \to_W D$, $D \to_W E$, $F \to_W G$, $C \to_W G$, $E \to_W G$. Μέχρι τώρα τις σχέσεις αυτές τις αναμέναμε γιατί είναι οι σχέσεις ανάμεσα σε διεργασίες που εξαρτώνται άμεσα από άλλες διεργασίες. Αν μια διεργασία εξαρτάται από μια άλλη αλλά όχι άμεσα, όπως π.χ η G από την A, δεν δημιουργείται $A \to_W G$ αφού δεν υπάρχει εκτέλεση που η G να εμφανίζεται αμέσως μετά την A και δεν δημιουργείται η σχέση $A >_W G$. Αυτό δεν επηρεάζει όμως το αποτέλεσμα του αλγόριθμου.

Θα παραχθούν όμως και οι σχέσεις: $C \to_W F$ αφού παρόλο που υπάρχουν δύο εκτελέσεις όπου στη μία η C προηγείται της F και στην δεύτερη η F προηγείται της C, μόνο στην πρώτη η F εκτελείται αμέσως μετά την C γι αυτό και παράγεται

$C >_W F$ και $F \not>_W C$ => $F ->_W B$. Παρομοίως κατασκευάζονται και οι σχέσεις $F ->_W B$ και $E ->_W F$, οι οποίες παρατηρώντας μόνο οπτικά το αρχείο εκτελέσεων καταλαβαίνουμε ότι δεν ισχύουν. Το Petri-net που κατασκευάζει ο α-αλγόριθμος είναι:



Σχήμα 3.16: *Μοντέλο που παίρνουμε με την εφαρμογή του α-αλγόριθμου, χρησιμοποιώντας το εργαλείο ProM.*

Καταλήγουμε λοιπόν στο συμπέρασμα ότι ο α-αλγόριθμος για να μπορεί να παράξει σωστά τις σχέσεις ανάμεσα στις διεργασίες, θα πρέπει για κάθε Χ και Υ διεργασίες οι οποίες η μία μπορεί να ακολουθεί την άλλη να δίνεται εκτέλεση στο αρχείο όπου να ισχύει αυτό.

Κεφάλαιο 4

Εξόρυξη ροής εργασιών: Προσέγγιση με δύο βήματα, Συστήματα μεταβάσεων και Περιοχές .

4.1 Προβλήματα που αντιμετωπίζουν άλλοι αλγόριθμοι

Οι αλγόριθμοι εξόρυξης διαδικασιών που υπάρχουν, όπως ο α-αλγόριθμος έχουν αρκετά προβλήματα και περιορισμούς, όπως μερικούς που αναφέρθηκαν προηγουμένως. Τα κυριότερα προβλήματα που υπάρχουν σε αυτούς τους αλγόριθμους είναι:

1) Πολλοί από τους αλγόριθμους αντιμετωπίζουν προβλήματα με πολύπλοκες κατασκευές ελέγχου ροής. Συγκεκριμένα οι περισσότεροι δεν επιτρέπουν τις

«μη ελεύθερης επιλογής» κατασκευές, όπου συνδυάζουν συγχρονισμό (διεργασίες που πρέπει να εκτελεστούν παράλληλα) με επιλογή όπως έχουμε ήδη αναφέρει και δόθηκε ένα παράδειγμα. Ο α-αλγόριθμος μπορεί να παράξει «μη ελεύθερης επιλογής» κατασκευές αλλά όχι πάντα. Γενικά υπάρχουν και άλλες πολύπλοκες κατασκευές, οι οποίες δεν μπορούν να κατασκευάσουν οι ήδη υπάρχοντες αλγόριθμοι.

2) Ένα άλλο πρόβλημα είναι το γεγονός ότι πολλοί αλγόριθμοι έχουν προβλήματα με διπλές εμφανίσεις διεργασιών.. Συγκεκριμένα μέσα από το αρχείο γεγονότων δεν μπορείς να διακρίνεις ανάμεσα σε δύο διεργασίες οι οποίες έχουν καταγραφτεί με παρόμοιο τρόπο και έχουν αφήσει το ίδιο "αποτύπωμα" στο αρχείο. Έτσι οι περισσότεροι αλγόριθμοι θεωρούν αυτές τις διεργασίες ότι είναι δύο εκτελέσεις της ίδιας διεργασίας και τις αναπαριστούν σαν μία διεργασία στο μοντέλο που κατασκευάζουν. Π.χ μία εκτέλεση έχουμε μία εκτέλεση (στιγμιότυπο) DBAEF το οποίο καταγράφηκε σαν XBAEX, οι περισσότεροι αλγόριθμοι, θα προσπαθήσουν να βάλουν σαν μία διεργασία τα δύο Χ. Σε κάποιες περιπτώσεις αυτό ίσως έχει νόημα, αλλά υπάρχουν και περιπτώσεις όπου πραγματικά οι D και F έχουν διαφορετικούς ρόλους στη διαδικασία και τότε αυτό προκαλεί προβλήματα και λανθασμένα μοντέλα.

3) Το τρίτο πρόβλημα είναι ότι κάποιοι αλγόριθμοι υπερ-γενικεύουν, δηλ. κατασκευάζουν μοντέλα τα οποία επιτρέπουν πολύ περισσότερες εκτελέσεις από αυτές που υπάρχουν στο αρχείο. Από την άλλη μπορεί να υπάρξουν και περιπτώσεις στις οποίες αλγόριθμοι είναι πολύ αυστηροί και επιτρέπουν ακριβώς τις εκτελέσεις που υπάρχουν στο αρχείο.

4) Επίσης, πολλοί αλγόριθμοι έχουν την τάση να παράγουν «ασυνεπή» μοντέλα. Δεν αναφερόμαστε όμως στη σχέση συνέπειας ανάμεσα στο αρχείο γεγονότων και το μοντέλο της διαδικασίας, αλλά την εσωτερική συνέπεια του ίδιου του μοντέλου. Υπάρχουν για παράδειγμα μοντέλα που

Ένας νέος τύπος εξόρυξης διαδικασιών μέσα από αρχείο, ο οποίος ξεπερνά αυτά τα προβλήματα, αποτελείται από δύο βήματα : 1) χρησιμοποιείται ένα σύστημα μεταβάσεων (transition system) σαν ενδιάμεση αναπαράσταση, 2)κατασκευή Petri-net σαν τελική αναπαράσταση, μέσα από περιοχές (regions) . Τα συστήματα συναλλαγών είναι η πιο βασική αναπαράσταση μιας διαδικασίας, και χρησιμοποιώντας τις περιοχές μετατρέπουμε τα συστήματα μεταβάσεων σε Petri-nets. Πρώτα όμως πρέπει να παράξουμε το σύστημα μεταβάσεων μέσα από ένα αρχείο γεγονότων. [10]

4.2 ΄Εννοια πληρότητας

Για την εξόρυξη διεργασιών μέσα από αρχεία γεγονότων, η έννοια της πληρότητας είναι πολύ σημαντική. Όταν έχουμε ένα αρχείο, δεν μπορούμε να υποθέσουμε ότι σε αυτό το αρχείο έχουν καταγραφεί όλες οι πιθανές εκτελέσεις της διαδικασίας. Σε κάποιες περιπτώσεις το μοντέλο που έχει κατασκευαστεί μπορεί να παράξει εκτελέσεις οι οποίες δεν υπάρχουν στο αρχείο. Από την άλλη σε πολλές περιπτώσεις οι αλγόριθμοι κατασκευάζουν μοντέλα που μπορούν να παράξουν μόνο ότι υπάρχει στο αρχείο και τίποτε περισσότερο. Οι αλγόριθμοι στηρίζονται πάντα σε μια έννοια πληρότητας. Για παράδειγμα ο α-αλγόριθμος υποθέτει ότι το αρχείο είναι τοπικά πλήρες δηλ. αν υπάρχουν δύο διεργασίες Χ και Υ και η Χ μπορεί να ακολουθείται άμεσα από την Υ, αυτό πρέπει να φαίνεται στο αρχείο.

Κάθε αλγόριθμος υποθέτει διαφορετική έννοια πληρότητας. Αυτές οι έννοιες αναπαριστούν διαφορετικές προσπάθειες για να επιτευχθεί μία ισορροπία ανάμεσα σε υπερ-ταίριασμα (overfitting) και υποταίριασμα (underfitting). Ένα μοντέλο είναι υπερ-ταίριασμένο όταν μόνο η ακριβής συμπεριφορά που καταγράφεται στο αρχείο μπορεί να κατασκευαστεί το μοντέλο. Αυτό σημαίνει ότι ο αλγόριθμος που το κατασκεύασε υποθέτει μία πολύ αυστηρή έννοια της πληρότητας. Αντίθετα ένα μοντέλο θεωρείται underfitted όταν γενικεύει σε μεγάλο βαθμό τις εκτελέσεις που υπάρχουν στο αρχείο και μπορεί να παράξει πολύ περισσότερα από ότι περιέχει το αρχείο.

Είναι δύσκολο να υπάρξει ισορροπία για ένα μοντέλο ανάμεσα στο να είναι πολύ γενικό ή πολύ συγκεκριμένο. Για αυτό το λόγο δεν αρκεί ένα μοντέλο π.χ Petri-net να μπορεί να αναπαράξει το περιεχόμενο ενός αρχείου εκτελέσεων, αλλά μας ενδιαφέρει να είναι «κατάλληλο». Πρέπει δηλαδή το μοντέλο να περιγράφει την διαδικασία με τον κατάλληλο τρόπο ώστε να ανταποκρίνεται στην πραγματικότητα και να μπορεί να υπάρξει αξιολόγηση ως προς την δομή και τη συμπεριφορά του μοντέλου. Δεν υπάρχει όμως βέλτιστο μοντέλο. Κάθε εφαρμογή απαιτεί διαφορετική διαχείριση και δεν υπάρχει τεχνική που να ανταποκρίνεται σωστά σε όλες τις εφαρμογές. Η τεχνική η οποία αναφέραμε που κατασκευάζει πρώτα ένα σύστημα μεταβάσεων και στη συνέχεια κατασκευάζει το Petri-net, επιτρέπει διαφορετικές ερμηνείες πληρότητας κάθε φορά έτσι ώστε να αποφεύγει το υπερ-ταίριασμα και υποτέριασμα. [10]

4.3 Κατασκευή Συστήματος μεταβάσεων (Transition System)

4.3.1 Εισαγωγικοί Συμβολισμοί και ορισμοί

Αυτοί οι συμβολισμοί είναι για να μπορέσουμε στη συνέχεια να εξηγήσουμε τις διαφορετικές στρατηγικές κατασκευής ενός συστήματος μεταβάσεων.

Έστω A ένα σύνολο. $IB(A) = A \rightarrow IN$ είναι το σύνολο των πολυσυνόλων (multisets) στο A, δηλ. Αν $X \in IB(A)$, είναι ένα πολυσύνολο όπου για κάθε $\alpha \in A$: το $X(\alpha)$

δηλώνει τον αριθμό των εμφανίσεων του α στο πολυσύνολο. Για τα πολυσύνολα ορίζονται και οι σχέσεις αθροίσματος (X+Y), διαφοράς (X-Y) , ύπαρξης ενός στοιχείου στο πολυσύνολο (x ∈ X) και η έννοια του υποσύνολου ($X \leq Y$). Αυτές οι σχέσεις μπορούν να εφαρμοστούν και στα σύνολα, όπου σύνολο υποθέτουμε ότι είναι ένα πολυσύνολο, όπου κάθε στοιχείο εμφανίζεται μόνο , μία φορά.

$|X|=\sum_{\alpha \in A} X(\alpha)$ είναι η πληθυκότητα ενός πολυσυνόλου X στο α, δηλ. το άθροισμα των εμφανίσεων όλων των στοιχείων του A στο X.

Η συνάρτηση set(X) μετατρέπει το πολυσύνολο X σε σύνολο : $set(X)= \{\alpha \in X \mid X(\alpha)>0\}$.

$\mathcal{P}(A)$ είναι το δυναμοσύνολο του α, δηλ. $X \in \mathcal{P}(A)$, αν και μόνο αν $X \subseteq A$.

A* είναι το σύνολο όλων των πεπερασμένων ακολουθιών στο A. Μία πεπερασμένη ακολουθία στο A μήκους n είναι μία απεικόνιση $\sigma \in \{1,\ldots,n\} \rightarrow A$. Αναπαριστάται δηλ. από ένα string $\sigma= < \alpha_1,\alpha_2,...,\alpha_k>$ όπου $\alpha_i=\sigma(i)$ για $1 \leq i \leq n$.

$hd(\sigma,k) = <\alpha_1, \alpha_2,\ldots,\alpha_k>$ είναι η ακολουθία των πρώτων k στοιχείων.
$hd(\sigma,0)=$ είναι η κενή ακολουθία.
$tl(\sigma,k)=<\alpha_{k+1}, \alpha_{k+2},\ldots,\alpha_n>$ είναι η ακολουθία μετά τη αφαίρεση των k πρώτων στοιχείων.
$tl(\sigma,n)$ είναι κενή ακολουθία.
$\sigma\uparrow X$ είναι η προβολή του σ σε κάποιο υποσύνολο $X \subseteq A$ π.χ. $<\alpha,c, b, b, c, \alpha>\uparrow\{\alpha,c\} = <\alpha,c,c,a>$.

par(σ) (διάνυσμα Parikh) αντιστοιχεί κάθε στοιχείο α στο A με τον αριθμό των εμφανίσεων του α στο σ. δηλ. για κάθε $\alpha \in A$: $par(\sigma)(\alpha)= |\sigma\uparrow\{\alpha\}|$. Αυτή η συνάρτηση μετρά τις φορές εμφανίζεται μία μεταβλητή σε μία ακολουθία μεταβλητών.

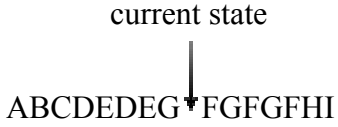
Ορισμός 1 (Ακολουθία, Αρχείο γεγονότων): Έστω A ένα σύνολο διεργασιών . $\sigma \in A^*$ είναι μία ακολουθία (trace) και $L \subseteq \mathcal{P}(A^*)$ είναι ένα αρχείο γεγονότων.

Το σύνολο των διεργασιών μπορεί να βρεθεί ελέγχοντας το αρχείο, ωστόσο το πιο σημαντικό aspect της εξόρυξης διεργασιών είναι ο καθορισμός των καταστάσεων της διαδικασίας. Οι περισσότεροι αλγόριθμοι έχουν μια υπονοούμενη έννοια της κατάστασης δηλ. οι δραστηριότητες είναι κολλημένες μεταξύ τους σε κάποια γλώσσα αναπαράστασης μοντέλων διαδικασιών με βάση την ανάλυση του αρχείου και το τελικό μοντέλο έχει μια συμπεριφορά που μπορεί να αναπαρασταθεί σαν ένα σύστημα μεταβάσεων. Σ΄αυτή την προσέγγιση όμως οι καταστάσεις θα οριστούν ρητά.

Όταν κτίζουμε ένα σύστημα μεταβάσεων, υπάρχουν τέσσερις προσεγγίσεις για να ορίσουμε την κατάσταση στο αρχείο.

-*παρελθόν*, η κατάσταση κατασκευάζεται με βάση την ιστορία μιας περίπτωσης.
-*μέλλον*, η κατάσταση της περίπτωσης βασίζεται στο μέλλον της.
-*παρελθόν* και *μέλλον*, η κατάσταση εξαρτάται από τον συνδυασμό των προηγούμενων δύο.
-*ρητή γνώση* της τρέχουσας κατάστασης.

Αν για παράδειγμα έχουμε μια ακολουθία,



Παρελθόν αυτής της ακολουθίας είναι το ABCDEDEG και *μέλλον* είναι FGFGFHI. *παρελθόν* και *μέλλον* είναι ολόκληρη η ακολουθία.

Ορισμός 2 (παρελθόν και μέλλον περίπτωσης): Έστω A ένα σύνολο διεργασιών και έστω σ=<α1,α2,...,αn> A* να είναι μία ακολουθία που αναπαριστά μία ολοκληρωμένη εκτέλεση μιας περίπτωσης.

Παρελθόν αυτής της περίπτωσης μετά την εκτέλεση k διεργασιών ($0 \leq k \leq n$) είναι hd(σ,k) και το μέλλον tl(σ,k).

Το *παρελθόν* μιας περίπτωσης είναι το πρόθεμα (prefix) ολόκληρης της ακολουθίας και το μέλλον είναι το μετάθεμα (postfix) ολόκληρης της ακολουθίας. Υπάρχουν δύο είδη προθέματος και μεταθέματος:

-πλήρες πρόθεμα (μετάθεμα), η κατάσταση αναπαριστάται από το πλήρες παρελθόν (μέλλον) της περίπτωσης.

-μερικό πρόθεμα (μετάθεμα), λαμβάνεται υπόψιν μόνο ένα υποσύνολο του πλήρες προθέματος (μεταθέματος).

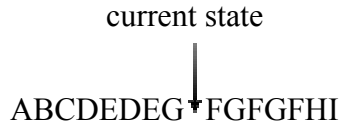
Ένα μερικό πρόθεμα κοιτάζει σε περιορισμένο αριθμό γεγονότων πριν φτάσει στην τρέχουσα κατάσταση. Μπορεί για παράδειγμα να ληφθούν υπόψιν μόνο τα τελευταία k γεγονότα πριν την τρέχουσα κατάσταση. Στο προηγούμενο παράδειγμα το πλήρες πρόθεμα είναι <ABCDEDEG>, αν θέλαμε ένα μερικό πρόθεμα με k=5 τότε αυτό θα ήταν < DEDEG>.

Υπάρχει και η περίπτωση όπου μόνο επιλεγμένο σύνολο διεργασιών λαμβάνεται υπόψιν, το αρχείο φιλτράρεται και μόνο τα γεγονότα που παραμένουν χρησιμοποιούνται σαν είσοδος για την εξόρυξη της διεργασίας. Αν για παράδειγμα υπάρχουν start και complete γεγονότα για τις διεργασίες π.χ "A started" και "A completed" είναι πιθανόν να ληφθούν υπόψιν μόνο τα complete γεγονότα.

Τρίτη περίπτωση είναι η αφαίρεση της σειράς ή της συχνότητας από την πλήρη ακολουθία. Ο λόγος είναι ότι κάποιες φορές δεν μας ενδιαφέρει πόσες φορές συνέβηκε μια διεργασία και πότε (με ποια σειρά), αλλά μας ενδιαφέρει απλά να ξέρουμε ότι συνέβηκε. Υπάρχουν τρεις τρόποι αναπαράστασης γνώσης για το *παρελθόν* και/ή το *μέλλον*:

- σειρά, με την οποία καταγράφονται η διαδικασίες της κατάστασης,
- πολυσύνολο των διεργασιών, ο αριθμός των εκτελέσεων κάθε διεργασίας χωρίς να μας ενδιαφέρει η σειρά,
- σύνολο των διεργασιών, απλά η εμφάνιση των διεργασιών.

Αν επιστρέψουμε στο παράδειγμα μας



πλήρες πρόθεμα: ABCDEDEG
πλήρες μετάθεμα: FGFGFHI
πρόθεμα πολυσύνολο:$ABCD^2E^2G$ (δεν μας ενδιαφέρει η σειρά)
μετάθεμα πολυσύνολο:F^3G^2HI (δεν μας ενδιαφέρει η σειρά)
πρόθεμα σύνολο:ABCDEG (δεν μας ενδιαφέρει η σειρά και η συχνότητα)
μετάθεμα σύνολο:FGHI (δεν μας ενδιαφέρει η σειρά συχνότητα)

Όλοι οι πιο πάνω τρόποι αναπαράστασης μπορούν να συνδυαστούν. Αν υπάρξουν πολλές αφαιρέσεις στην αναπαράσταση των εκτελέσεων, τότε ο αριθμός των καταστάσεων θα είναι μικρότερος και ο κίνδυνος για υπο-ταίριασμα εμφανίζεται. Από την άλλη, αν πολύ αφαιρέσεις υπάρχουν τότε ο αριθμός των καταστάσεων είναι αρκετά μεγάλος με αποτέλεσμα να υπάρχει υπερ-ταίριασμα. [10]

Ορισμός 3 (Φιλτράρισμα): Έστω A ένα σύνολο διεργασιών, $X \subseteq A$ υπόσυνολο διεργασιών και $\sigma \in A^*$ μια ακολουθία. $\sigma\uparrow X$ είναι η ακολουθία που παίρνουμε αφαιρώντας τα γεγονότα που δεν υπάρχουν στο X. Με αυτό τον το τρόπο η ακολουθία φιλτράρεται κρατώντας μόνο της δραστηριότητες στο X.

Ορισμός 4 (horizon): Έστω A σύνολο διεργασιών και $\sigma=<\alpha_1,\alpha_2,...,\alpha_n> \in A^*$. Έστω h ένας φυσικός αριθμός που ορίζει το horizon και έστω k ($0 \leq k \leq n$) δείχνει στην τρέχουσα κατάσταση της ακολουθίας (μετά την εκτέλεση k βημάτων). Το $hd^h=<\alpha(k-h) \max 1,\ldots,\alpha k>$ είναι η ακολουθία h γεγονότων πριν φτάσουμε στην τρέχουσα κατάσταση. Το $tl^h<\alpha k+1,\ldots,\alpha(k+h) \min n>$ είναι η ακολουθία των h γεγονότων που ακολουθούν την τρέχουσα κατάσταση.

Παραδείγματα αναπαράστασης καταστάσεων:

-(hd(σ,k),tl(σ,k)): Αναπαράσταση της τρέχουσας κατάστασης με το πλήρες πρόθεμα και μετάθεμα της ακολουθίας. Για την αποφυγή όμως υπερ-ταιριάσματος μπορούν να χρησιμοποιηθούν διαφορετικές αναπαραστάσεις προθέματος και μεταθέματος.

-par(hd(σ,k)), λαμβάνει υπόψιν μόνο το πλήρες πολυσύνολο, μας ενδιαφέρει το πλήρες πρόθεμα αλλά όχι η σειρά που εκτελούνται οι διεργασίες.

-set(par(hd(σ,k))), μας ενδιαφέρει το πλήρες πρόθεμα αλλά δεν μας ενδιαφέρει η σειρά και η συχνότητα εμφάνισης των διεργασιών.

-set(par(hd^h(σ,k))), μας ενδιαφέρει το μερικό πρόθεμα μήκους h χωρίς να ενδιαφερόμαστε για την σειρά και τις συχνότητες.

Ορισμός 5 (Αναπαράσταση της κατάστασης): Μία αναπαράσταση της κατάστασης είναι μια συνάρτηση η οποία, δεδομένης μιας ακολουθίας σ και ενός αριθμού k που δίχνει τον αριθμό των γεγονότων του σ που έχουν συμβεί, παράγει κάποια αναπαράσταση r.

Ένα παράδειγμα αναπαράστασης μιας κατάστασης είναι state(σ,k)=set(par(tl^h((σ↑X),k))). Σε αυτή την κατάσταση πρώτα η ακολουθία φιλτράρεται και όλες οι διεργασίες που δεν ανήκουν στο X αφαιρούνται, μας ενδιαφέρει το μερικό μετάθεμα μήκους h, και δεν μας ενδιαφέρει η συχνότητα και η σειρά των διεργασιών.

Οι διαφορετικές έννοιες αφαίρεσης που αναφέραμε, μπορούν να χρησιμοποιηθούν έτσι ώστε να αναπαραστήσουμε μια κατάσταση. Υπάρχουν συνολικά 36 στρατηγικές αναπαράστασης καταστάσεων. Αυτές οι στρατηγικές μπορούν να

κατασκευαστούν ως εξής:

1. Φιλτράρισμα του αρχείου αν χρειάζεται, άρα έχουμε δύο πιθανότητες για βάση της ακολουθίας σ ή σ↑X.
2. Αποφαση αν χρειάζεται να χρησιμοποιήσουμε το παρελθόν ή το μέλλον μιας ακολουθίας ή και τα δύο. Υπάρχουν έξι πιθανότητες hd(σ,k), tl(σ,k), (hd(σ,k),tl(σ,k)), hd^h(σ,k), tl^h(σ,k), (hd^h(σ,k), tl^h(σ,k)).
3. Απόφαση αν η σειρά και η συχνότητα μας ενδιαφέρουν ή όχι. Άρα για ένα πρόθεμα ή μετάθεμα έχουμε ακόμα τρεις πιθανότητες, είτε να λάβουμε υπόψιν και τη συχνότητα και τη σειρά, ή να λάβουμε υπόψιν μόνο την συχνότητα αφαιρώντας τη σειρά par(σ), ή να μην λάβουμε υπόψιν ούτε τη σειρά αλλά ούτε τη συχνότητα set(par(σ)).

Αφού αποφασίσουμε ποια θα είναι η κατάσταση, το επόμενο βήμα είναι η κατασκευή του συστήματος μεταβάσεων.

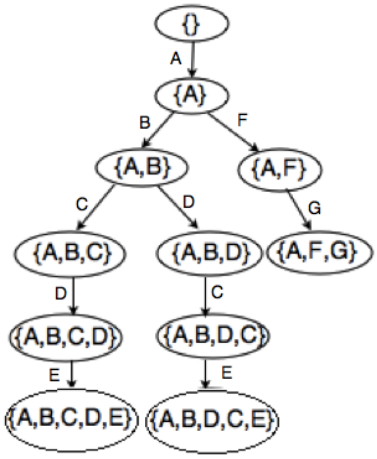
4.3.2 Κατασκευή Συστήματος μεταβάσεων

Ορισμός 6: Έστω A σύνολο διεργασιών και έστω L ⊆ $\mathbb{P}$(A*) είναι ένα αρχείο γεγονότων. Δεδομένης μιας κατάστασης ορίζουμε ένα labeled σύστημα μεταβάσεων TS= (S,E,T) όπου S= {state(σ,k) | σ∈ L$^{\Lambda}$ 0 ≤ k < |σ|} είναι ο χώρος τον καταστάσεων, E=A είναι το σύνολο των γεγονότων (labels) και T ⊆ S x E x S με T= {(state(σ,k), σ(k+1),state(σ, k+1) | σ∈ L$^{\Lambda}$ 0 ≤ k < |σ|} είναι η σχέση μετάβασης.

Όπως φαίνεται από τον ορισμό μπορούμε να κατασκευάσουμε διαφορετικά συστήματα μεταβάσεων ανάλογα με τον τύπο αναπαράστασης της κατάστασης που επιλέχθηκε. Ο αλγόριθμος για κατασκευή του συστήματος μεταβάσεων είναι: Για κάθε ακολουθία σ, για τα k πρώτα βήματα που οδηγούν στην τρέχουσα κατάσταση κατασκευάζουμε νέα κατάσταση state(σ,k) αν δεν υπάρχει ήδη. Μετά οι ακολουθίες

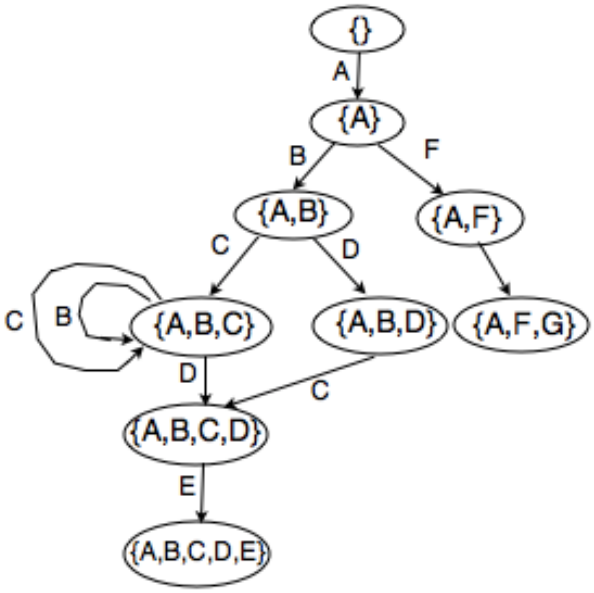
σαρώνονται για να δούμε αν υπάρχουν μεταβάσεις (state(σ, k-1) , σ(k), state(σ, k)) και αυτές προσθέτονται αν δεν υπάρχουν ήδη. [10]

Έχουμε ένα αρχείο L το οποίο έχει τις εκτελέσεις: ABCDE, ABDCE, AFG,ABCBC. Άν χρησιμοποιήσουμε το πλήρες πρόθεμα της κατάστασης, λαμβάνοντας υπόψη τη σειρά και συχνότητα δηλ. state(σ,k)= hd(σ,k) θα πάρουμε το πιο κάτω σύστημα μεταβάσεων:



Σχήμα 4.1

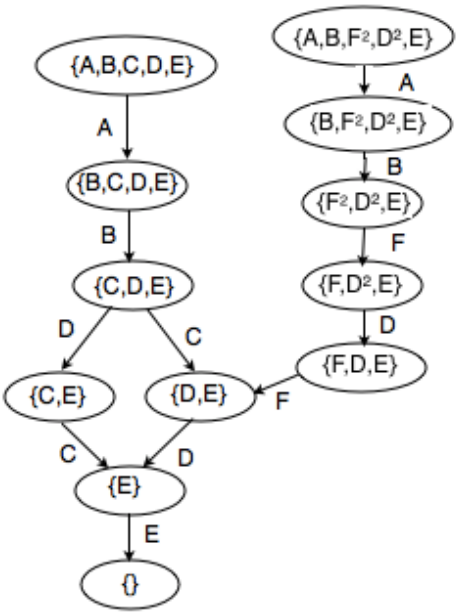
Αν επιλέξουμε σαν κατάσταση μόνο το σύνολο των διεργασιών και δεν λάβουμε υπόψιν την σειρά και συχνότητα εμαφάνισης των διεργασιών δηλ. state(σ,k)= set(par(hd(σ,k))), θα πάρουμε το σύστημα μεταβάσεων που ακολουθεί:



Σχήμα 4.2

Παρατηρούμε ότι οι καταστάσεις σε αυτό το σύστημα μεταβάσεων *Σχήμα 4.2*, λόγω του ότι δεν μας ενδιαφέρει η σειρά των διεργασιών, για τις καταστάσεις που είχαν τις ίδιες ακριβώς διεργασιές αλλά με διαφορετική σειρά, παραμένει μία αντιπροσωπευτική κατάσταση. Παρατηρούμε επίσης ότι αυτό το σύστημα μεταβάσεων περιέχει δύο "self-loop" ({A,B,C},C,{A,B,C}) και ({A,B, C},E,{A,B,C}).

Ας πάρουμε ένα άλλο αρχείο L1 το οποίο περιέχει τις εκτελέσεις ABCDE, ABDCE, ABFDFDE. Αν για αυτό το αρχείο επιλέξουμε την αναπαράσταση των καταστάσεων με βάση το πλήρες μετάθεμα χωρίς όμως να μας ενδιαφέρει η σειρά δηλ. par(tl(σ,k)), τότε θα πάρουμε το σύστημα μεταβάσεων:



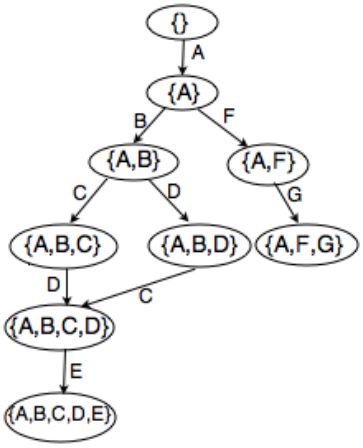
Σχήμα 4.3

4.3.3 Τροποποιήσεις

Υπάρχουν τρία είδη τροποποιήσεων τα οπόια μπορούμε να εφαρμόσουμε.

Στρατηγική "Kill loops": Όταν χρησιμοποιείται αναπαράσταση συνόλων για την κατασκευή συστήματος μεταβάσεων συνήθως υπάρχουν self-loops. Η στρατηγική αυτή χρησιμοποιείται για να αγνοούνται τα loops, αφαιρώντας τις ακμές που τα

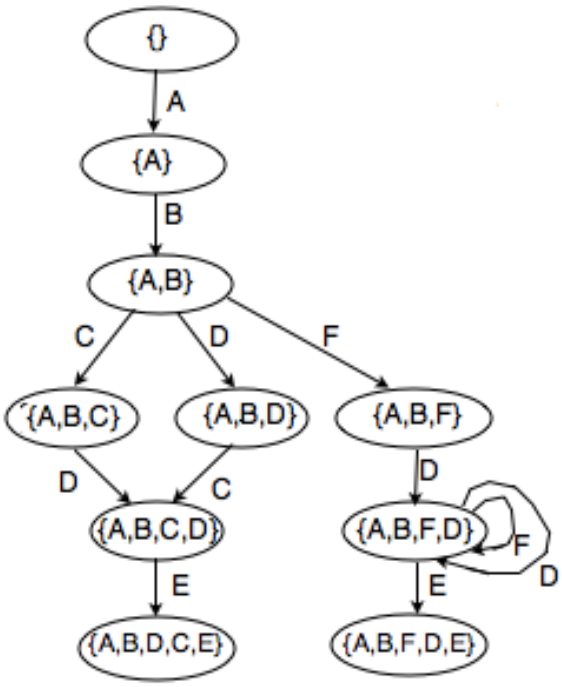
δημιουργούν. Αυτό γίνεται αφού απο τη στιγμή που είδη έχουμε επιλέξει την αναπαράσταση των καταστάσεων μόνο με τα σύνολα, αυτό σημαίνει ότι μας ενδιαφέρει απλά να ξέρουμε ότι υπάρχουν αυτές οι καταστάσεις. [10] Εφαρμόζοντας αυτή τη στρατηγική το σύστημα μεταβάσεων που κατασκευάστηκε με βάση το αρχείο L1 είναι:



Σχήμα 4.4

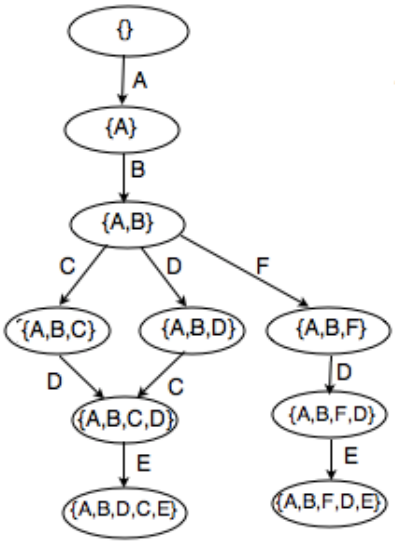
Στρατηγική Επέκτασης (Extend): Είναι ιδιαίτερα χρήσιμη για τα αρχεία που έχουν αναπαράσταση συνόλου (για την κατασκευή του συστήματος μεταβάσεων δεν λαμβάνεται υπόψη η σειρά και η συχνότητα). Η στρατηγική αυτή δημιουργεί μεταβάσεις ανάμεσα σε δύο καταστάσεις οι οποίες κατασκευάστηκαν από διαφορετικές εκτελέσεις του αρχείου αλλά θα μπορούσαν να ακολουθούν η μία την άλλη αφού υπάρχει μόνο μία διεργασία η οποία μπορεί να εκτελεστεί και να οδηγήσει από τη μία κατάσταση στην άλλη. Το κίνητρο για την εφαρμογή αυτής της στρατηγικής είναι ότι, σε πολλές περιπτώσεις δεν θεωρείται ρεαλιστικό όλες οι πιθανές εκτελέσεις της διαδικασίας εμφανίζονται στο αρχείο. Έτσι η στρατηγική επέκτασης επεκτείνει το σύστημα μεταβάσεων προσθέτωντας μεταβάσεις που δεν υπήρχαν στο σύστημα με βάση το αρχείο, αλλά μπορεί να υπήρχαν με βάση τη δομή του συστήματος. [10]

Αν πάρουμε το αρχείο εκτελέσεων L1, και χρησιμοποιήσουμε $\mathrm{set}(\mathrm{par}(\mathrm{hd}(\sigma,k)))$ για να κατασκευάζουμε το σύστημα μεταβάσεων, παίρνουμε το :



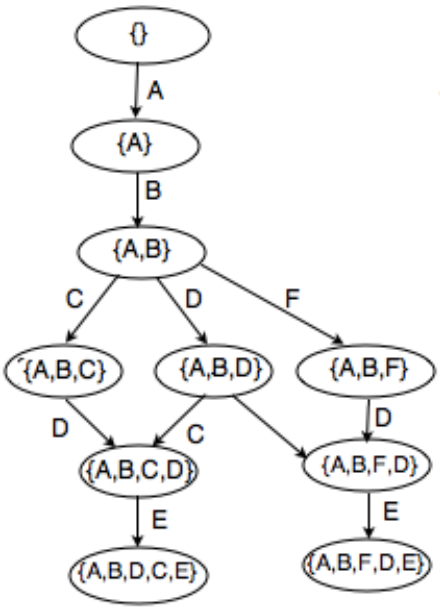
Σχήμα 4.5

Εφαρμόζοντας τη στρατηγική kill-loops θα πάρουμε:



Σχήμα 4.6

Αν στο *Σχήμα 4.6* εφαρμόσουμε και τη στρατηγική επέκτασης:



Σχήμα 4.7

4.4 Σύνθεση χρησιμοποιώντας Περιοχές

Το δεύτερο βήμα αυτής της προσέγγισης είναι η κατασκευή ενός Petri-net με τη χρήση περιοχών. Για να γίνει αυτό χρησιμοποιείται η θεωρία των Περιοχών.

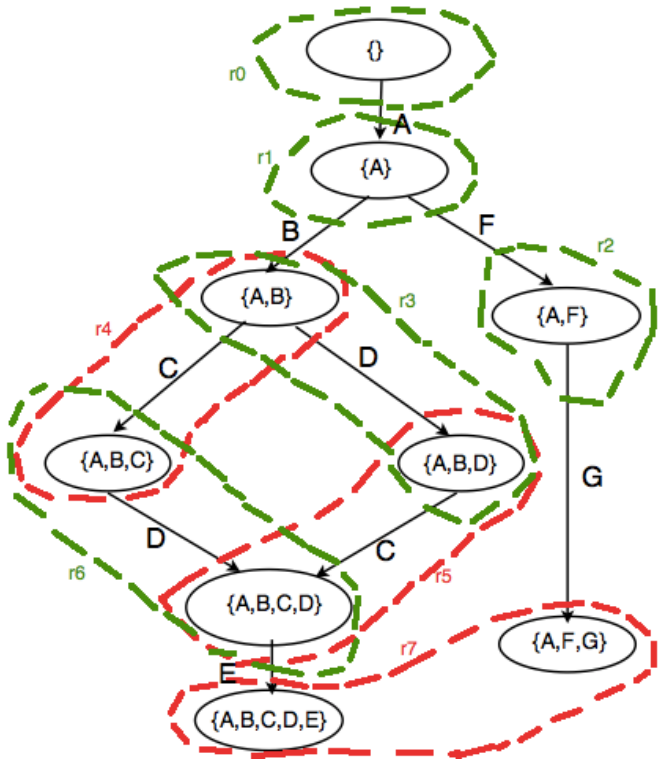
Ορισμός 7 (region): Έστω TS= (S,E,T) ένα σύστημα μεταβάσεων και S′⊆S ένα υποσύνολο καταστάσεων. S′ είναι μια περιφέρεια (region), αν για κάθε γεγονός (δραστηριότητα) e ∈ E ισχύει μια από τις πιο κάτω υποθέσεις:

1. Όλες οι μεταβάσεις $s1 \xrightarrow{e} s2$ (s1,e,s2) μπαίνουν στο S′, δηλ. s1 ∉ S’, s2 ∈ S’.
2. Όλες οι μεταβάσεις $s1 \xrightarrow{e} s2$ βγαίνουν από το S′, δηλ. s1 ∈ στο S’, s2 ∉ S’.
3. Όλες οι μεταβάσεις $s1 \xrightarrow{e} s2$ δεν το διασχίζουν, δηλ. s1,s2 ∈ S’, ή s1,s2 ∉ S’.

Αφού χωρίσω το σύστημα μεταβάσεων μου σε περιοχές, στη συνέχεια θα κατασκευάσω το Petri-net. Δεν θα ασχοληθούμε με περισσότερες λεπτομέρειες σχετικά με τις περιοχές και με τον τρόπο με τον οποίο επιλέγονται, αρκεί να ξέρουμε ότι για να ισχύει ότι ένα S’ είναι region πρέπει να ισχύουν τα πιο πάνω. Το region αντιστοιχεί σε θέση του Petri-net, άρα για κάθε region θα έχω μία θέση, και

οι ακμές του συστήματος μεταβάσεων που χαρακτηρίζονται από την εκτέλεση μιας διεργασίας, θα αντιστοιχούν στις μεταβάσεις του Petri-net. [10]

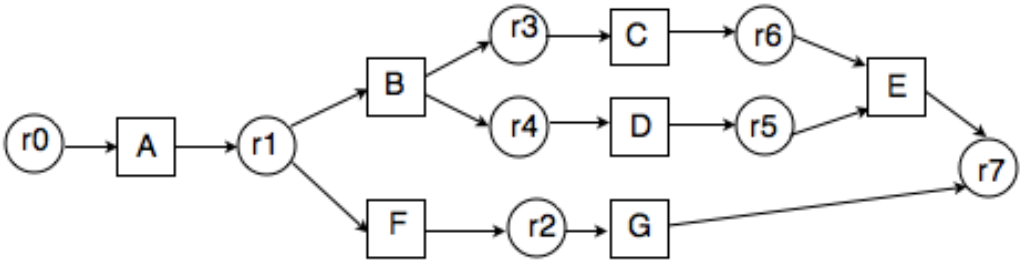
Από το σύστημα μεταβάσεων που κατασκευάστηκε για το αρχείο L, αφού αφαιρέσαμε και τους βρόγχους το χωρίζουμε σε περιοχές. Παίρνουμε το *Σχήμα 4.8*. Όπως φαίνεται το r0={{}}, είναι region αφού οι μεταβάσεις A φεύγουν από αυτό και όλες οι υπόλοιπες δεν το διαπερνούν. Το r1={{A}} έχει σαν είσοδο τη μετάβαση A, και εξόδους τις μεταβάσεις B και F, ενώ οι υπόλοιπες δεν το διαπερνούν κ.ο.κ.



Σχήμα 4.8

Για κάθε region του πιο πάνω συστήματος θα κατασκευάσουμε μία θέση. Ξεκινούμε από το r0, κατασκευάζουμε την θέση r0, και σχεδιάζουμε ακμή από το r0 στην A, όπου η A θα είναι η πρώτη μετάβαση στο Petri-net. Η A όπως φαίνεται από το σχήμα μπαίνει στο r1, άρα θα έχουμε ακμή από την A στην θέση r1. Έξοδοι του r1 είναι η B και F, έτσι θα έχουμε δύο ακμές στο Petri-net από το r1 να φεύγουν και να οδηγούν στις αντίστοιχες μεταβάσεις B και F. Η B οδηγεί στο r3 και r4, έτσι κατασκευάζουμε τις θέσεις r3 και r4 και σχεδιάζουμε ακμές από το B στην r3 και

από το B στην r4. Η F οδηγεί στην r2 και έτσι έχουμε ακμή από την F στη θέση r2. Συνεχίζουμε με τον ίδιο τρόπο και κατασκευάζουμε το πιο κάτω Petri-net.



Σχήμα 4.9

Κεφάλαιο 5

Εργαλείο ProM

Για την υποστήριξη της εξόρυξης ροών εργασιών, από αρχεία συναλλαγών (ή αρχεία εκτελέσεων) σε διάφορα πληροφοριακά συστήματα, έχουν κατασκευαστεί πολλά εργαλεία όπως το EMiT, Little Thump, Process Miner και πολλά άλλα. Κάθε εργαλείο διαβάζει και αποθηκεύει τα αρχεία με διαφορετικό τρόπο και επίσης παρουσιάζουν τα αποτελέσματα τους με διαφορετικό τρόπο. Γι αυτό το λόγο είναι δύσκολο να χρησιμοποιήσεις διαφορετικά εργαλεία στα ίδια δεδομένα και να συγκρίνεις τα αποτελέσματα. Επιπλέον, παρόλο που μερικά εργαλεία αναπαριστούν έννοιες οι οποίες θα μπορούσαν να είναι δομή για άλλα εργαλεία, δυστυχώς είναι δύσκολο να συγκρίνεις τα διάφορα εργαλεία. Οι ερευνητές αν θέλουν να εργαστούν σε μία νέα τεχνική για εξόρυξη ροής εργασιών θα πρέπει να φτιάξουν μια νέα υποδομή από το μηδέν, ή να συγκρίνουν τις τεχνικές τους απομονωμένα χωρίς τη χρήση κάποιας πρακτικής εφαρμογής. Για να ξεπεραστούν αυτά τα προβλήματα, έχει κατασκευαστεί το εργαλείο ProM ένα “pluggable” περιβάλλον για εξόρυξη ροών εργασιών. Είναι ένα πολύ ευέλικτο εργαλείο όσο αφορά την μορφή των εισόδων και εξόδων και γενικά όσο αφορά τις νέες ιδέες σε εξόρυξη ροών εργασιών.

5.1 Αρχιτεκτονική ProM

Η βάση για όλες τις τεχνικές για εξόρυξη διαδικασιών, είναι τα αρχεία εκτελέσεων διαδικασιών. Ένα τέτοιο αρχείο παράγεται από κάποια πληροφοριακά συστήματα με πληροφορίες σχετικά με την εκτέλεση κάποιας διαδικασίας. Κάθε πληροφοριακό σύστημα έχει τη δική του μορφή με την οποία αποθηκεύει τα αρχεία. Για το ProM, έχει κατασκευαστεί μία γενική μορφή για αποθήκευση στα αρχεία, η XML. Αυτή η μορφή βασίζεται κυρίως στις ανάγκες των ήδη υπαρχόντων εργαλείων για εξόρυξη διαδικασιών και τις πληροφορίες που υπάρχουν συνήθως σε αρχεία που κατασκευάζονται από πολύπλοκα πληροφοριακά συστήματα. [6]

5.1.1 XML

Η δομή του XML αρχείου είναι ως εξής :

WorkflowLog
- -Data
  - Attributes
- -Source
  - Data
- -Process
  - Data
  - ProcessInstance
    - Data
    - AuditTrailEntry
      - Data
      - WorkflowModelElement
      - EventType
      - Timestamp
      - Originator

Το “WorkflowLog” που αποτελεί τη ρίζα του αρχείου μας, περιέχει ένα πεδίο “Data” στο οποίο μπορούμε να αποθηκεύσουμε αυθαίρετα δεδομένα κειμένου , το “Data” περιέχει μία λίστα από χαρακτηριστικά (Attributes). Το “WorkflowLog” περιέχει επίσης το πεδίο “Source”, το οποίο μπορεί να περιέχει στοιχεία σχετικά με το πληροφοριακό σύστημα που κατασκεύασε το αρχείο. Τέλος στο “WorkflowLog” υπάρχει και το πεδίο “Process” που αναφέρεται σε συγκεκριμένη διαδικασία του πληροφοριακού συστήματος. Αφού ένα πληροφοριακό σύστημα αποτελείται από

πολλές διαδικασίες, το αρχείο μπορεί να περιέχει πάνω από μία διαδικασία. Στο πεδίο “Process”, υπάγεται το πεδίο “Data” το οποίο δεν είναι υποχρεωτικό και μπορεί να περιέχει αυθαίρετα δεδομένα σχετικά με τη διαδικασία και το πεδίο “Process Instance” το οποίο αποτελεί ένα στιγμιότυπο της διαδικασίας. Το “ProcessInstance” περιέχει το “AuditTrailEntry” που μπορεί να αναφέρεται σε μια διεργασία (WorkflowModelElement), σε τύπο γεγονότος (EventType), σε χρονοδιάγραμμα (Timestamp), ή σε κάποιο άτομο (Originator). Το “AuditTrailEntry” μπορεί επίσης να περιέχει πεδίο “Data”. [6]

5.1.2 Plug-ins

Ένα σημαντικό χαρακτηριστικό του εργαλείου, είναι το ότι επιτρέπει την αλληλεπίδραση ανάμεσα σε ένα μεγάλο αριθμό plug-ins. Το plug-in, είναι η αναπαράσταση ενός αλγόριθμου που έχει κάποια σχέση με τον τομέα της εξόρυξης διαδικασιών. Είναι απλή η εισαγωγή νέων plug-ins στο ProM αφού δεν χρειάζεται οποιαδήποτε τροποποίηση του εργαλείου, όπως για παράδειγμα recompiling, για να μπορεί κάποιος να προσθέσει κάποιο σχετικό αλγόριθμο.

Το εργαλείο αποτελείται από το Log Filter, μέσα από το οποίο το πλαίσιο του ProM μπορεί να διαβάσει αρχεία σε XML μορφή. Το Log Filter μπορεί να χειριστεί μεγάλα σύνολα δεδομένων και να ταξινομήσει τα γεγονότα σε μια περίπτωση, στα χρονοδιαγράμματα τους (timestamps) πριν ξεκινήσει η εξόρυξη.

Mining Plug-ins: Κάνουν την εξόρυξη της διαδικασίας και το αποτέλεσμα φυλάγεται στη μνήμη και αναπαριστάται σε ένα παράθυρο στην οθόνη του ProM. Αναπαριστούν διάφορους αλγόριθμους εξόρυξης ροών εργασιών, όπως για παράδειγμα τον α-αλγόριθμο.

Import Plug-ins: Ένα μεγάλο σύνολο μοντέλων μπορεί να φορτωθεί μέσα από Import plug-ins, όπως για παράδειγμα το Petri-net.

Export Plug-ins: Έχουν να κάνουν με την αποθήκευση κάποιων αντικειμένων, όμως

είναι τα Petri-nets, EPCs (Event Driven Process Chains) κ.λ.π.

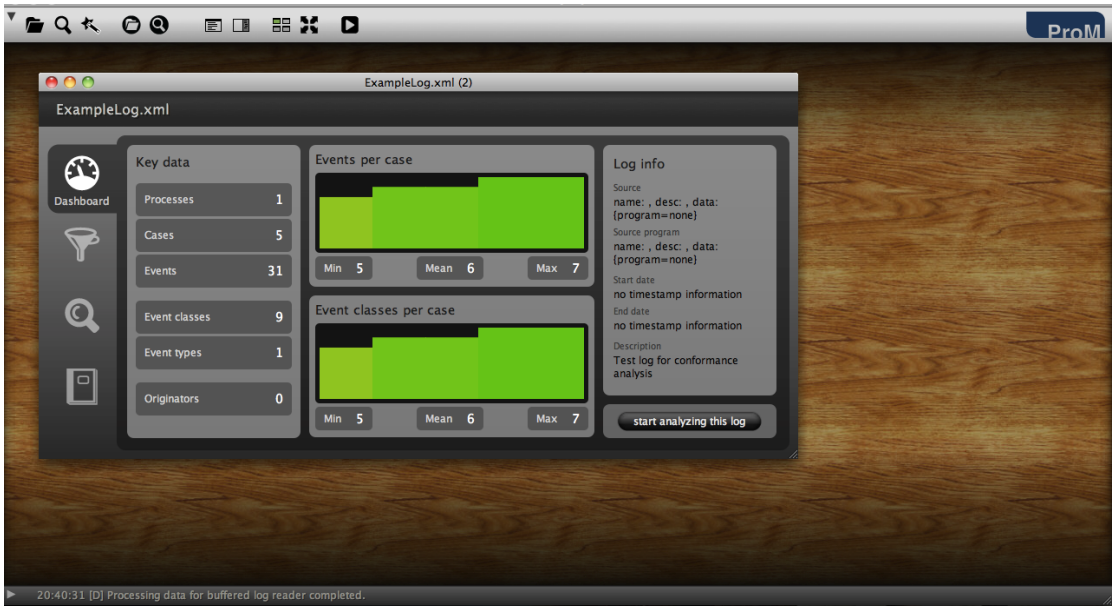
Analysis plug-ins: παίρνουν το αποτέλεσμα της εξόρυξης και το αναλύουν. Για παράδειγμα στα Petri-nets υπάρχει ένα plug-in που κατασκευάζει place invariants, transition invariants και coverability graph. Υπάρχουν επίσης plug-ins που συγκρίνουν το αρχείο με το μοντέλο.

Conversion Plug-ins: Παίρνουν ένα αποτέλεσμα εξόρυξης και το μετατρέπουν σε μία άλλη μορφή. Π.χ από EPC σε Petri-net. [6,7]

5.2 Χρήση ProM

Μενου του ProM: Όπως παρατηρούμε οι επιλογές Mining, Analysis, Conversion, και Exports, αντιστοιχούν στα ανάλογα plug-ins ρογ περιγράφονται πιο πάνω.

Μετά το άνοιγμα κάποιου αρχείου (log) XML, η οθόνη του ProM είναι όπως φένεται πιο κάτω:



Οθόνη 1

Συγκεκριμένα, όταν επιλέξουμε να ανοίξουμε κάποιο αρχείο XML, εμφανίζεται το

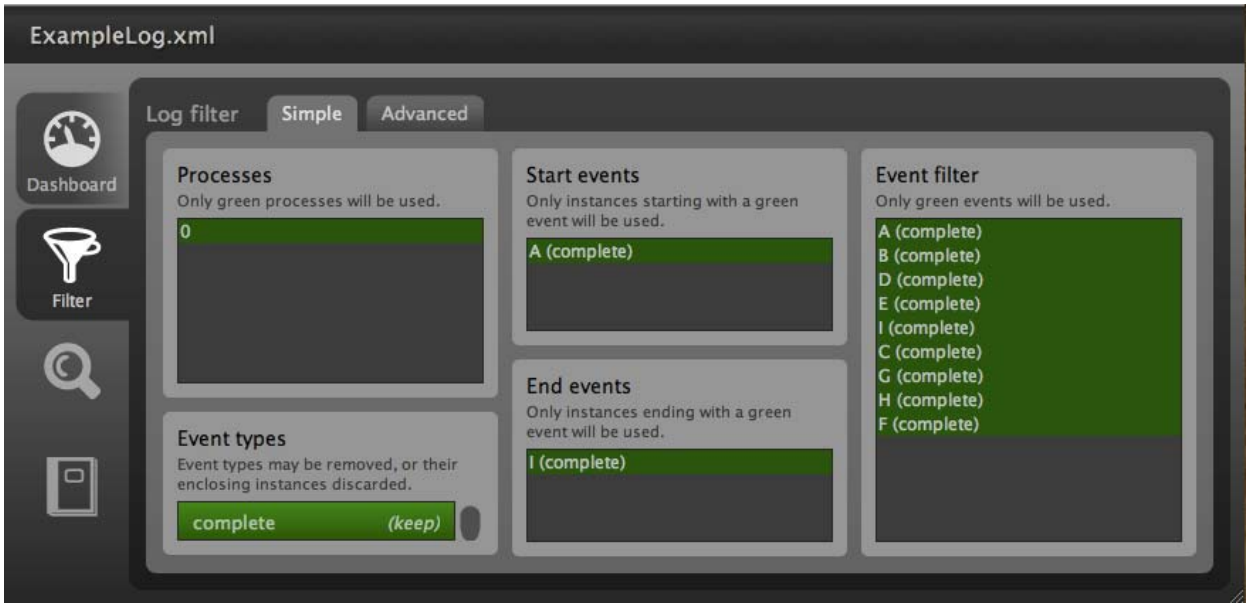
πιο κάτω παράθυρο (*οθόνη 2*), το οποίο μας δίνει διάφορες πληροφορίες σχετικά με το αρχείο. Για το πιο κάτω παράδειγμα οι πληροφορίες που παίρνουμε είναι :

-Στο αρχείο αναφέρονται 1 διαδικασία (process), 5 περιπτώσεις (cases) όπου αυτό σημαίνει ότι έχουμε 5 διαφορετικά στιγμιότυπα της διαδικασίας, όπου το κάθε ένα αντιστοιχεί σε διαφορετική σειρά εκτέλεσης των διεργασιών της διαδικασίας. Έχουμε 31 γεγονότα (events) τα οποία μπορεί να αντιστοιχούν i) σε διεργασία της διαδικασίας, ii) σε περίπτωση (case),ii) σε originator. Υπάρχουν 9 κλάσεις γεγονότων, 1 τύπος γεγονότων και 0 originators, όπου originators είναι τα άτομα τα οποία είναι υπεύθυνα για την εκτέλεση κάποιας εργασίας. Επίσης μας δίνονται πληροφορίες σχετικά με τον αριθμό των γεγονότων που υπάρχουν σε κάθε περίπτωση και τον αριθμό των κλάσεων των γεγονότων που υπάρχουν σε κάθε περίπτωση. Όπως φαίνεται από το συγκεκριμένο παράδειγμα ο ελάχιστος αριθμός γεγονότων σε μια περίπτωση αυτής της διαδικασίας είναι 5 και ο μέγιστος 7. Ο ελάχιστος αριθμός κλάσεων που υπάρχουν σε μια περίπτωση είναι 5 και ο μέγιστος 7. Από το πεδίο Log Info, παρατηρούμε ότι στο συγκεκριμένο αρχείο δεν υπάρχουν πληροφορίες σχετικά με χρονοδιαγράμματα.

Οθόνη 2

Από τα τέσσερα κουμπιά στα αριστερά του συγκεκριμένου παραθύρου, αν επιλέξουμε το δεύτερο που ονομάζεται filter θα πάρουμε την *Οθόνη 3*. Από αυτό το παράθυρο παίρνουμε πληροφορίες σχετικά με τις διαδικασίες που θα

χρησιμοποιηθούν, τον τύπο των γεγονότων, τα γεγονότα έναρξης και τερματισμού καθώς και τα γεγονότα (διεργασίες) που θα χρησιμοποιηθούν. Οι διεργασίες που θα χρησιμοποιηθούν είναι αυτές που βρίσκονται στο ” Event Filter” και είναι επιλεγμένα με πράσινο χρώμα. Σε αυτό το σημείο ο χρήστης μπορεί να επέμβει και να επιλέξει τα γεγονότα που θέλει να χρησιμοποιηθούν.

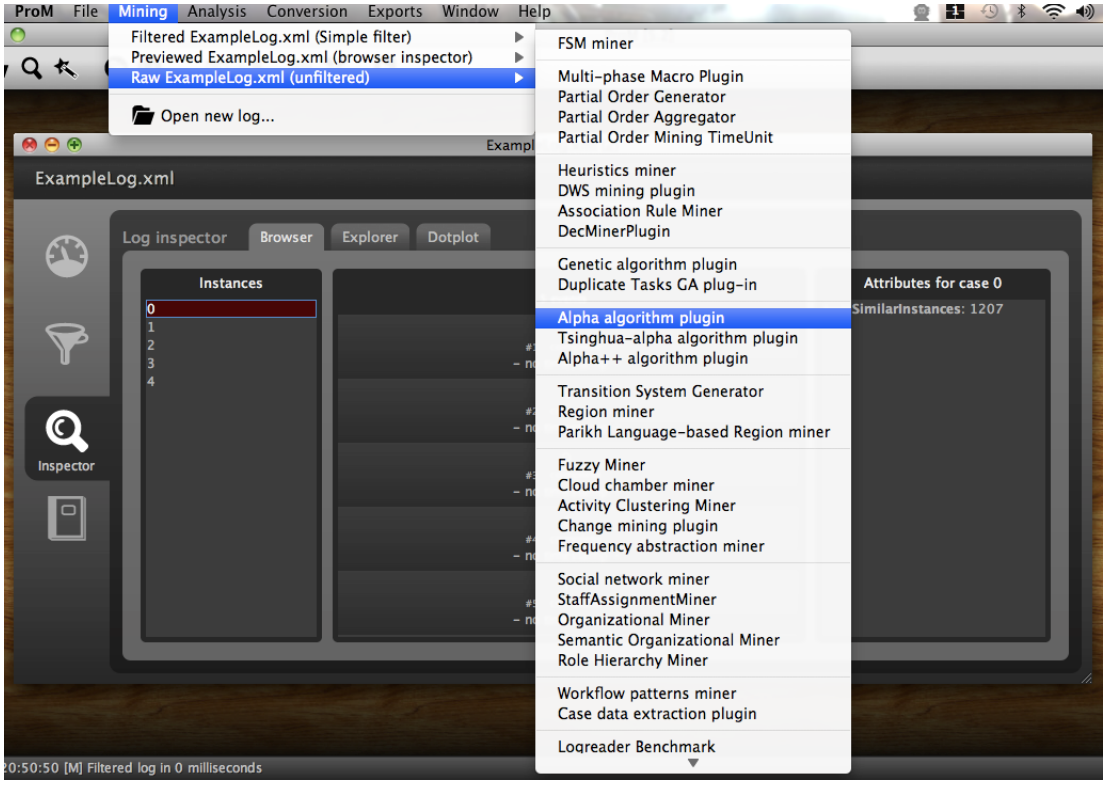


Οθόνη 3

Επιλέγοντας το τρίτο button, που ονομάζεται inspector, μπορούμε να δούμε τα διαφορετικά στιγμιότυπα (περιπτώσεις) της διαδικασίας (*Οθόνη 4*). Μπορούμε επίσης επιλέγοντας ένα από αυτά, να εφρμόσουμε κάποιον αλγόριθμο και να κατασκευάσουμε ένα μοντέλο, ειδικά για το συγκεκριμένο στιγμιότυπο. Στο συγκεκριμένο παράδειγμα έχουμε 5 διαφορετικά στιγμιότυπα.

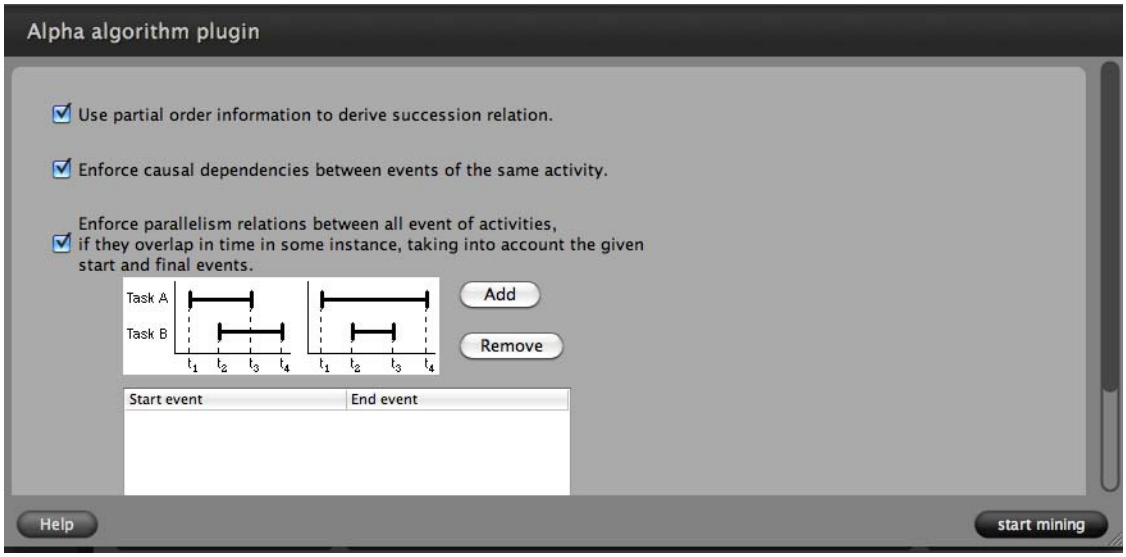
Οθόνη 4

Για να εφαρμόσουμε την εξόρυξη ροής εργασιών από το αρχείο που έχουμε ήδη ανοίξει, επιλέγουμε από το μενού « Mining » , επιλέγουμε το αρχείο μας που είναι ήδη ανοικτό Raw ExampleLog.xml (unfiltered) και εμφανίζεται μία μεγάλη σειρά από plug-ins που αντιπροσωπεύουν αλγόριθμους για εξόρυξη ροών εργασιών. Σε αυτή τη σειρά υπάρχει και plug-in για τον α-αλγόριθμο, το οποίο επιλέγουμε. Αν θέλαμε να τρέξουμε ένα αλγόριθμο για ένα συγκεκρριμένο στιγμιότυπο από την *Οθόνη 4,* τότε από το « Mining », θα επιλέγαμε Previewed ExampleLog.xml(browser inspector). Αν πάλι θέλαμε στην οθόνη 3 φιλτράρουμε το αρχείο, επιλέγωντας εμείς συγκεκριμένες διεργασίες που θέλουμε να εκτελεστούν, από το « Mining », θα επιλέγαμε Filtered ExampleLog.xml (Simple filter).



Οθόνη 5

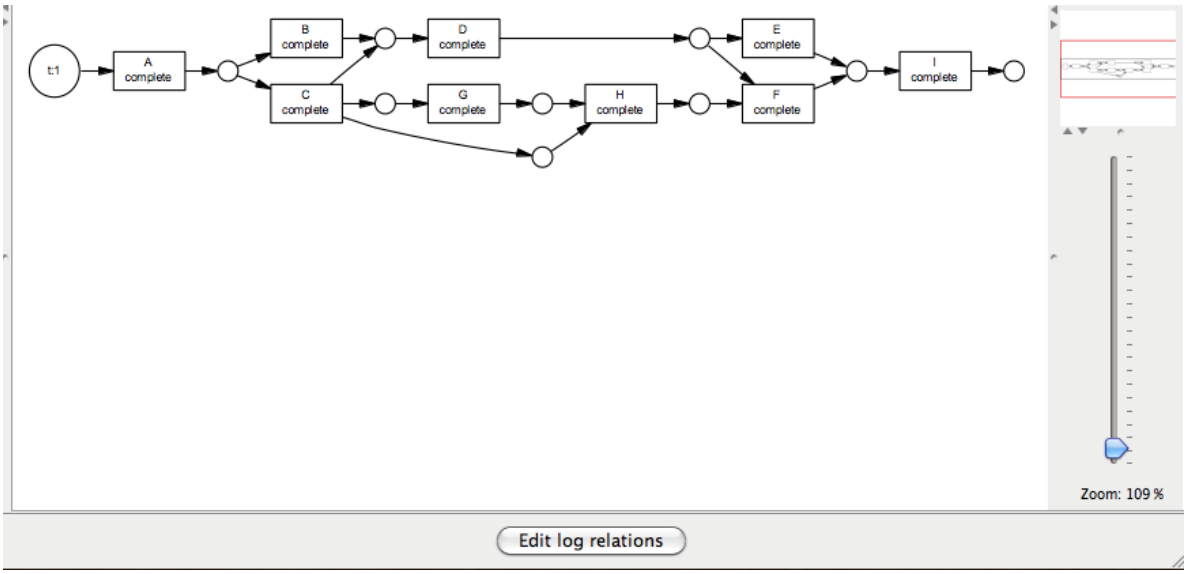
Επιλέγοντας το τον α-αλγόριθμο ξέρουμε ότι το αποτέλεσμα που θα μας δώσει είναι το Petri-net. Όταν το επιλέξουμε εμφανίζεται η *Οθόνη 6.*



Οθόνη 6

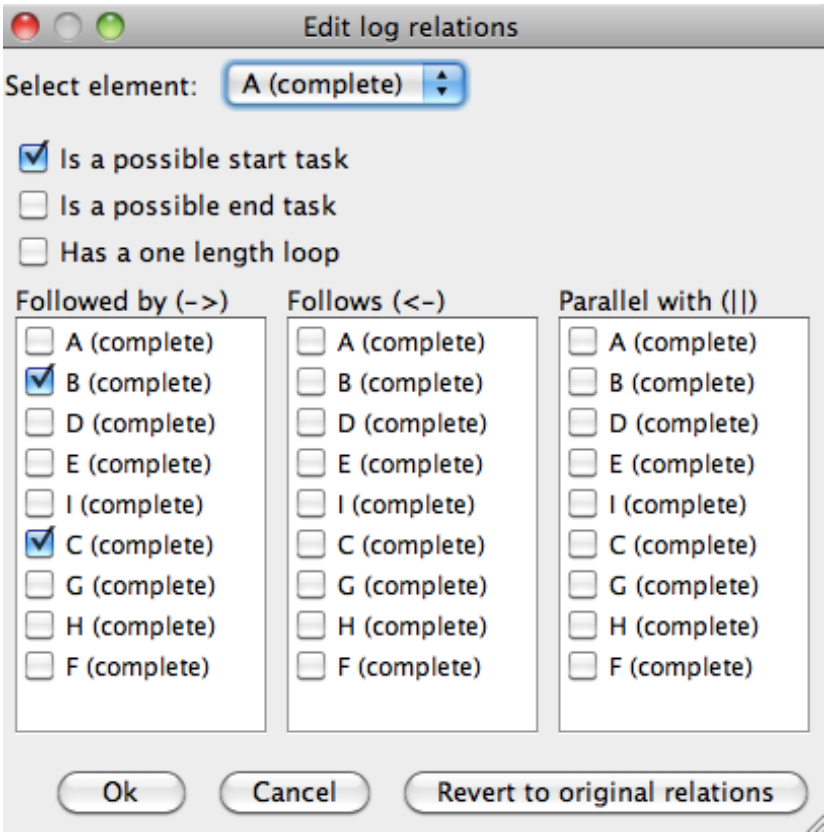
Οι διεργασίες που υπάρχουν στα αρχεία μπορεί να μην είναι πάντα του ίδιου τύπου. Στο συγκεκριμένο παράδειγμα έχουμε μόνο ένα τύπο διεργασιών στο αρχείο και ο τύπος αυτός είναι “complete”. Οι τροποποιήσεις που μπορούν να γίνουν στην

Οθόνη 6 έχουν να κάνουν με το χειρισμό των διεργασιών που έχουν διαφορετικό τύπο. Από το πιο πάνω παράθυρο επιλέγουμε το κουμπί start mining για να ξεκινήσει η διαδικασία εξόρυξης και κατασκευής του Petri-net. Στη συνέχεια εμφανίζεται το επόμενο παράθυρο στο οποίο εμφανίζεται το αποτέλεσμα.



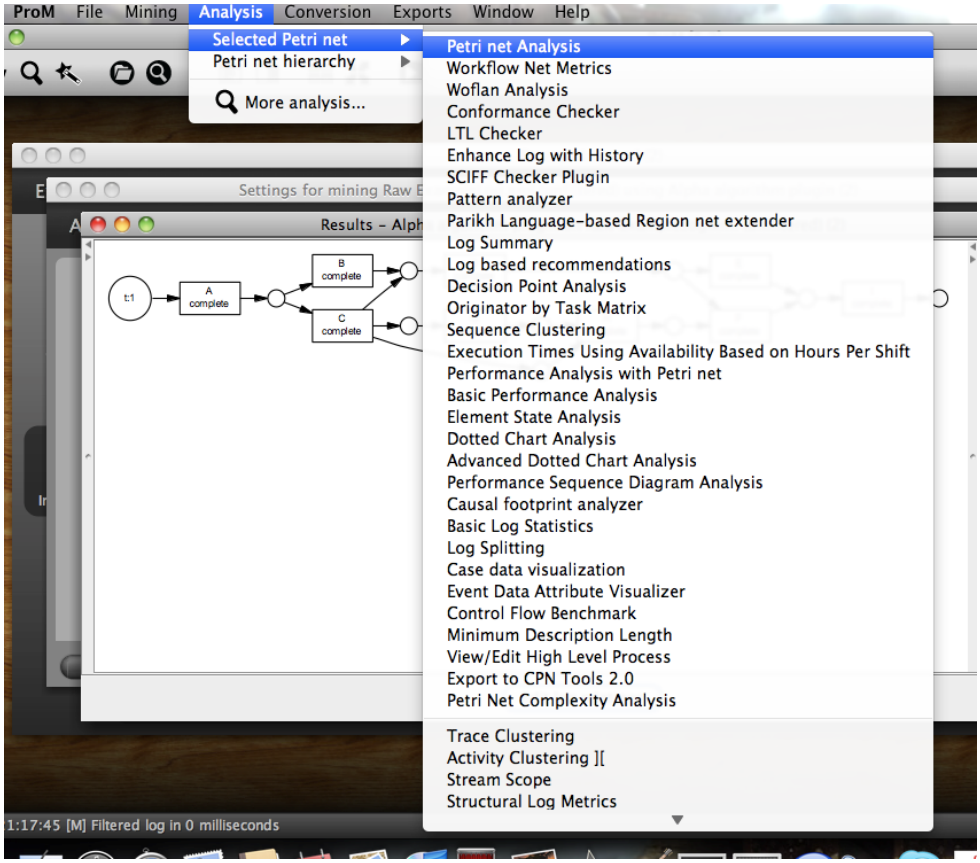
Οθόνη 7

Το πιο πάνω αποτέλεσμα έχει κατασκευαστεί με βάση τον α-αλγόριθμο. Μπορούμε πατώντας στο κουμπί « Edit log relations » να το επεξεργαστούμε και να αλλάξουμε τις σχέσεις ανάμεσα στις διεργασίες στο αρχείο, με πολύ απλό τρόπο. Όταν πατήσουμε το « Edit log relations », εμφανίζεται η οθόνη *Οθόνη 8*. Οι αλλαγές που κάνουμε στις σχέσεις, εμφανίζονται αμέσως στο Petri-net.



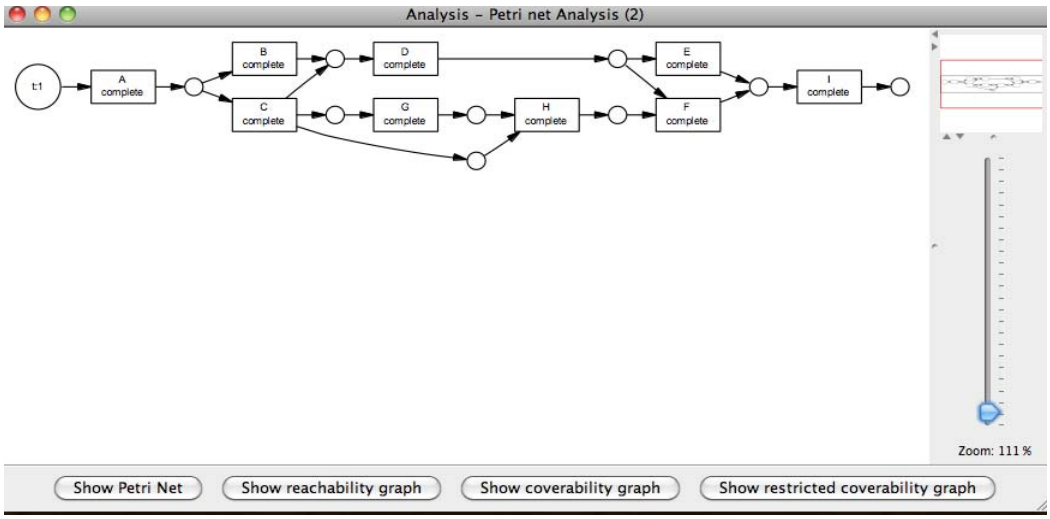
Οθόνη 8

Αφού έχει ολοκληρωθεί η εξόρυξη της ροής εργασιών, μπορούμε τώρα να χρησιμοποιήσουμε τα plug-ins για την ανάλυση των αποτελεσμάτων. Στο ποιο κάτω παράδειγμα χρησιμοποιούμε το Petri-net Analysis (*Οθόνη 9*).



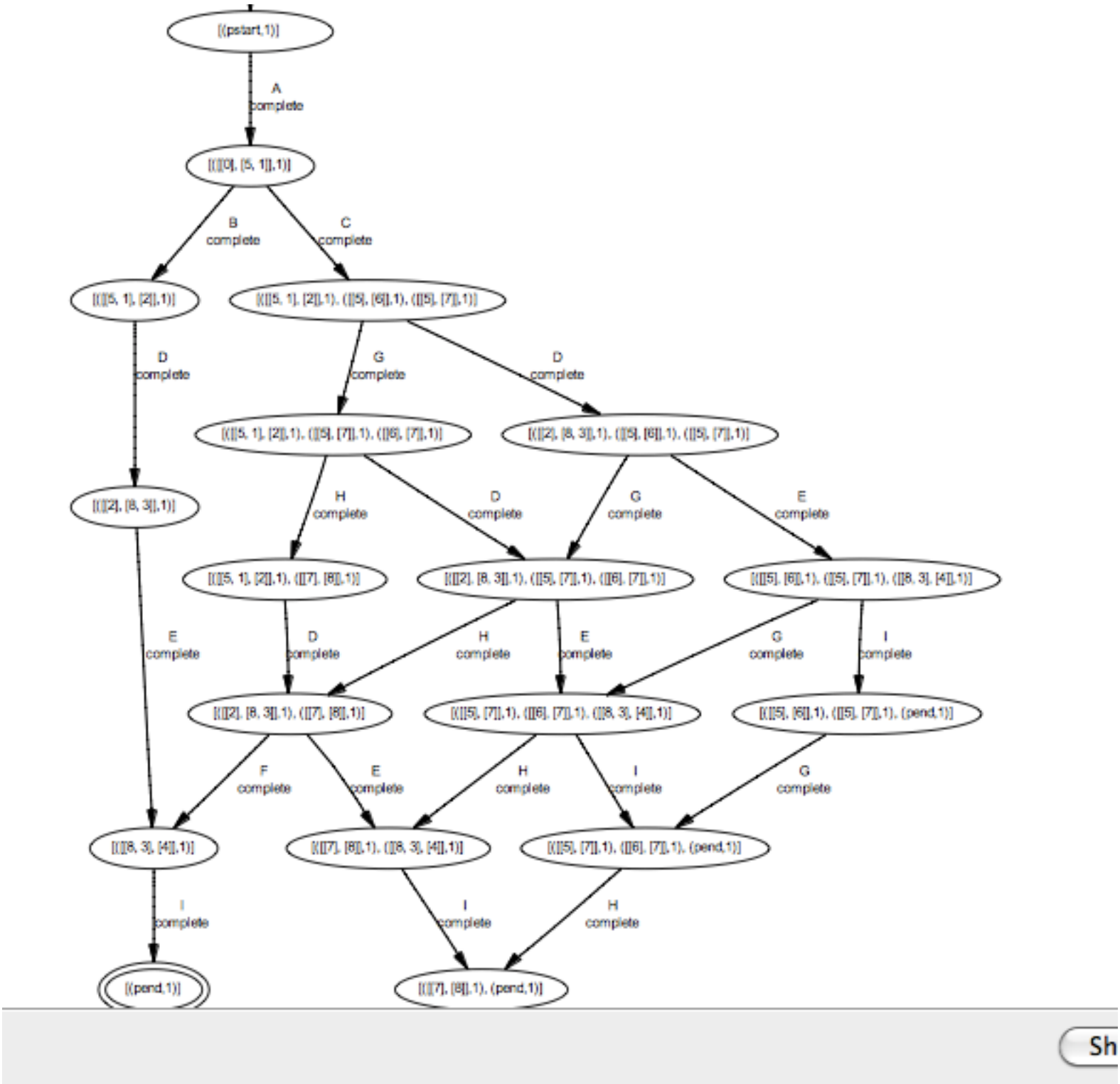
Οθόνη 9

Επιλέγοντας αυτό το plug-in ανοίγει το πιο κάτω παράθυρο (*Οθόνη 10*), το οποίο μας εμφανίζει το Petri-net και μας δίνει τη δυνατότητα εμφάνισης του reachability graph, coverability graph και restricted coverability graph.



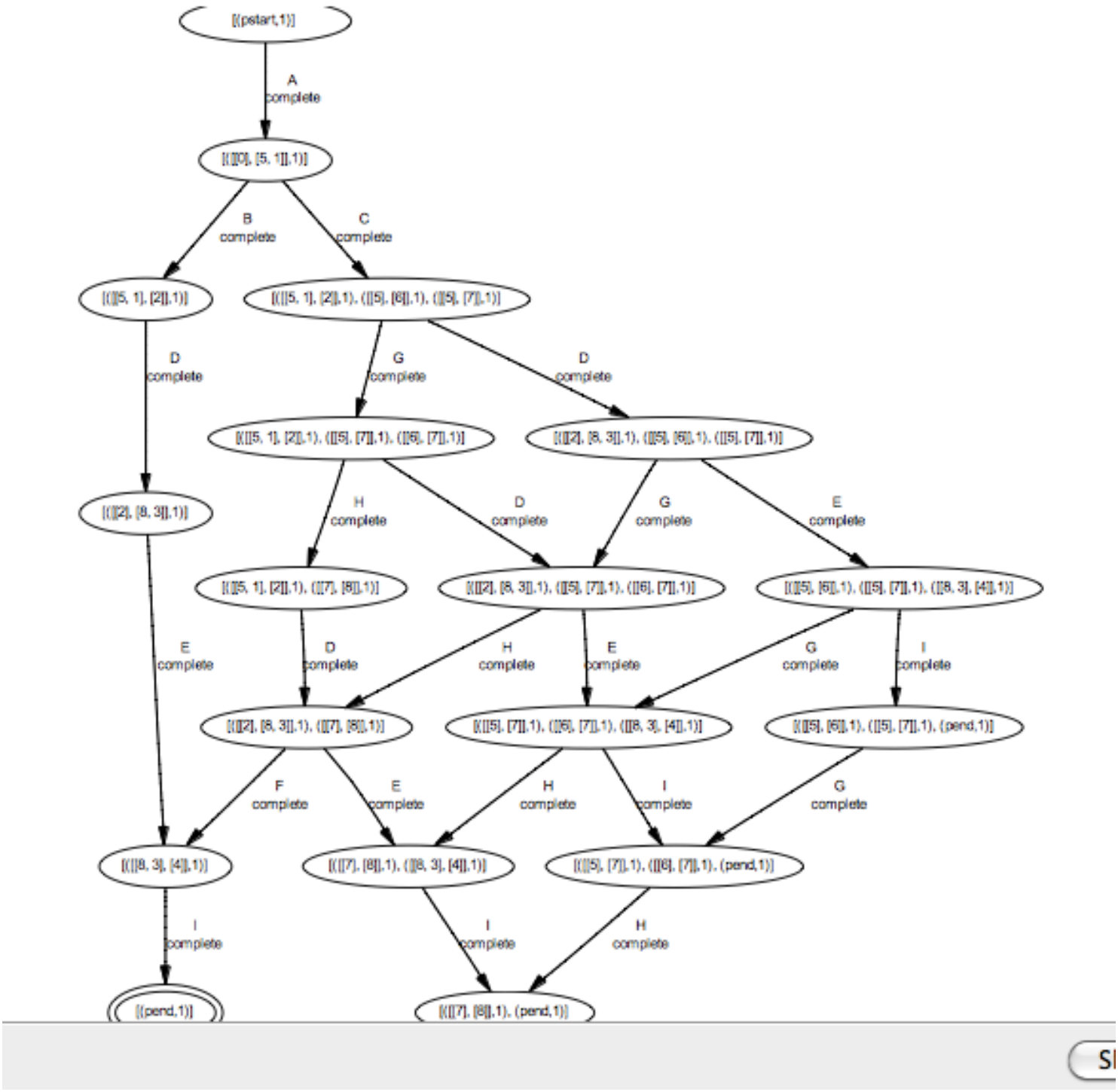
Οθόνη 10

Αν επιλέξουμε να εμφανίσουμε το reachability graph, παίρνουμε το πιο κάτω αποτέλεσμα (*Οθόνη 11*).



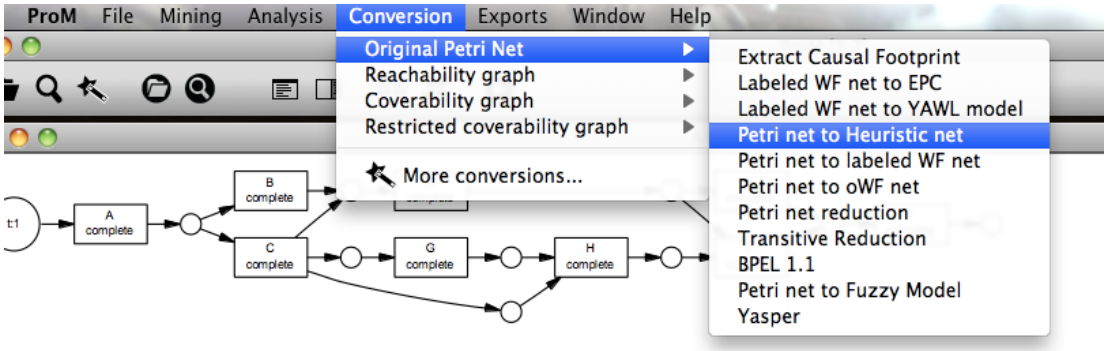
Οθόνη 11

Αν επιλέξουμε να εμφανίσουμε το coverability graph, τότε θα πάρουμε το πιο κάτω αποτέλεσμα (*Οθόνη 12*).



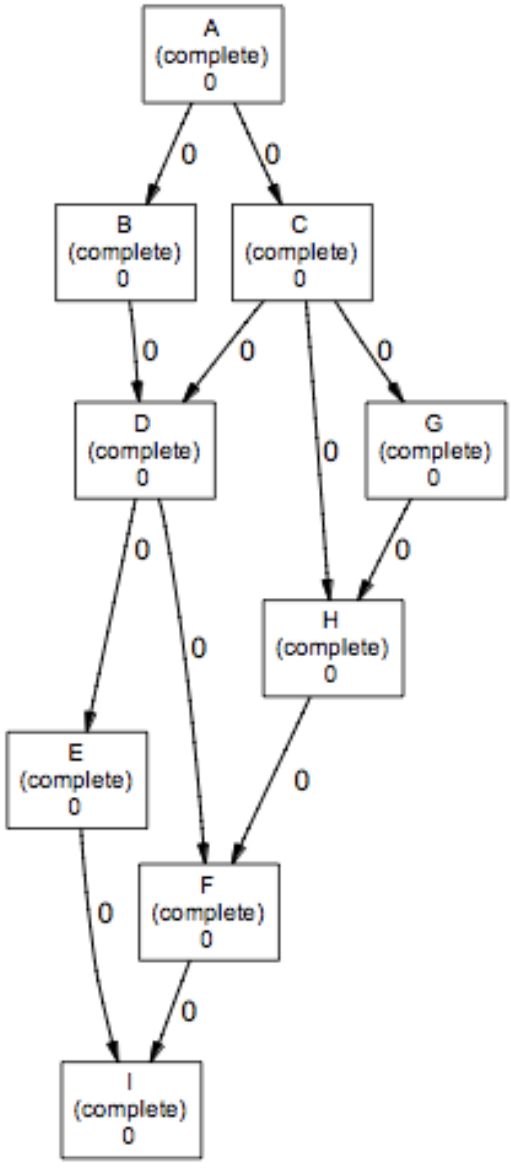
Οθόνη 12

Ας δούμε τώρα ένα παράδειγμα Conversion plug-in. Από το μενού επιλογών, επιλέγουμε Conversion. Στη συνέχεια διαλέγουμε την επιλογή «Petri-net to Heuristic net ».



Οθόνη 13

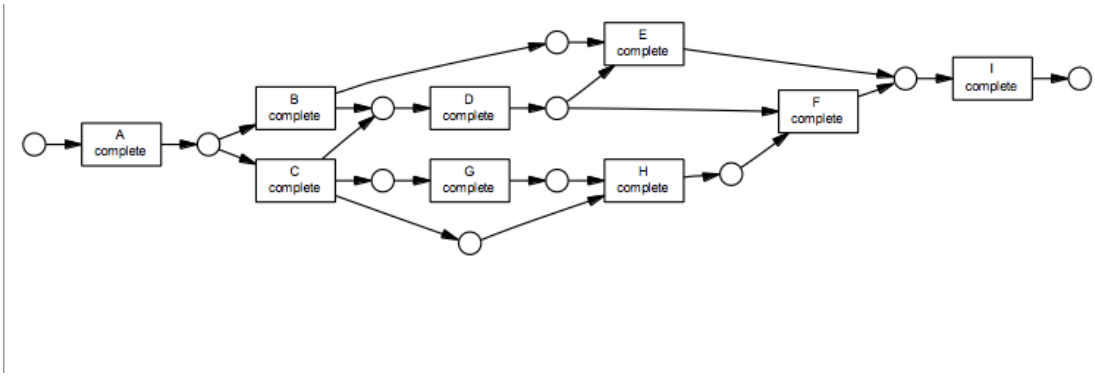
Το heuristic net που παίρνουμε είναι:



Σχήμα 5.1

Ας δούμε όμως πιο αποτέλεσμα θα πάρουμε αν χρησιμοποιήσουμε το ίδιο αρχείο εκτελέσεων, αλλά να χρησιμοποιήσουμε διαφορετικούς αλγόριθμους εξόρυξης.

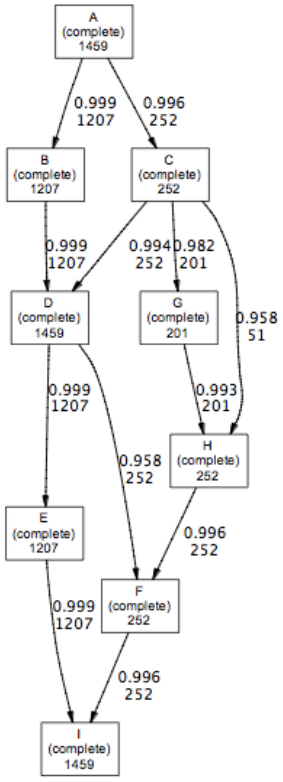
Ο αλγόριθμος α++ θα μας δώσει:



Σχήμα 5.2

Παρατηρούμε ότι και αυτός ο αλγόριθμος μας κατασκευάζει petri-net, όχι όμως το ίδιο με αυτό που μας έδωσε ο α-αλγόριθμος.

Ο heuristic miner μας δίνει το πιο κάτω δίκτυο:



Σχήμα 5.3

5.3 Εφαρμογή α-αλγόριθμου σε αρχείο δικής μας κατασκευής

Το αρχείο που κατασκευάσαμε περιέχει μια διαδικασία και τις πιο κάτω εκτελέσεις:

ABCDE, όπου ανάφερα ότι υπάρχουν 1207 παρόμοια στιγμιότυπα.

ABDCE, 145 παρόμοια στιγμιότυπα.

AFG, 56 παρόμοια στιγμιότυπα.

Το αρχείο είναι ως εξής:

```
<?xml version="1.0" encoding="UTF-8"?>
<WorkflowLog xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="WorkflowLog.xsd"
description="Test log for conformance analysis">
<Source program="name: , desc: , data: {program=none}">
<Data>
<Attribute name="program">name: , desc: , data:
{program=none}</Attribute>
</Data>
</Source>
<Process id="0" description="">

<ProcessInstance id="0" description="">
<Data>
<Attribute name="numSimilarInstances">1207</Attribute>
</Data>
<AuditTrailEntry>
<WorkflowModelElement>A</WorkflowModelElement>
<EventType>complete</EventType>
</AuditTrailEntry>
<AuditTrailEntry>
<WorkflowModelElement>B</WorkflowModelElement>
<EventType>complete</EventType>
</AuditTrailEntry>
<AuditTrailEntry>
<WorkflowModelElement>C</WorkflowModelElement>
<EventType>complete</EventType>
</AuditTrailEntry>
<AuditTrailEntry>
<WorkflowModelElement>D</WorkflowModelElement>
<EventType>complete</EventType>
</AuditTrailEntry>
<AuditTrailEntry>
<WorkflowModelElement>E</WorkflowModelElement>
<EventType>complete</EventType>
</AuditTrailEntry>
</ProcessInstance>

<ProcessInstance id="1" description="">
<Data>
<Attribute name="numSimilarInstances">145</Attribute>
</Data>
<AuditTrailEntry>
<WorkflowModelElement>A</WorkflowModelElement>
<EventType>complete</EventType>
</AuditTrailEntry>
```

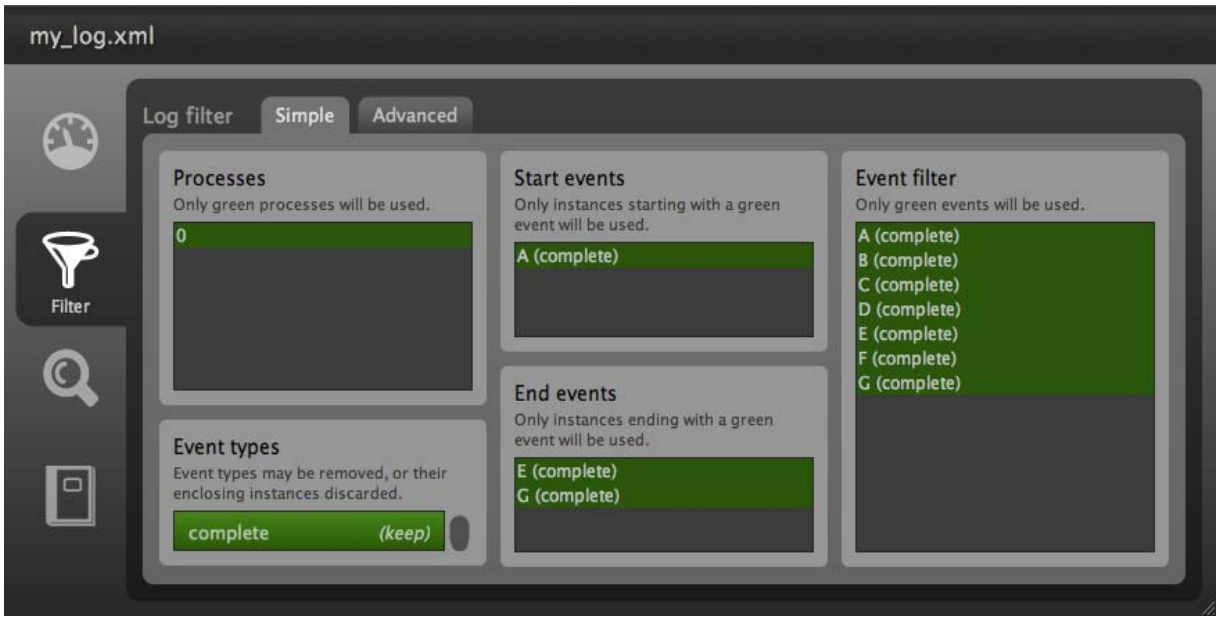
```
<AuditTrailEntry>
<WorkflowModelElement>B</WorkflowModelElement>
<EventType>complete</EventType>
</AuditTrailEntry>
<AuditTrailEntry>
<WorkflowModelElement>D</WorkflowModelElement>
<EventType>complete</EventType>
</AuditTrailEntry>
<AuditTrailEntry>
<WorkflowModelElement>C</WorkflowModelElement>
<EventType>complete</EventType>
</AuditTrailEntry>
<AuditTrailEntry>
<WorkflowModelElement>E</WorkflowModelElement>
<EventType>complete</EventType>
</AuditTrailEntry>
</ProcessInstance>

<ProcessInstance id="2" description="">
<Data>
<Attribute name="numSimilarInstances">56</Attribute>
</Data>
<AuditTrailEntry>
<WorkflowModelElement>A</WorkflowModelElement>
<EventType>complete</EventType>
</AuditTrailEntry>
<AuditTrailEntry>
<WorkflowModelElement>F</WorkflowModelElement>
<EventType>complete</EventType>
</AuditTrailEntry>
<AuditTrailEntry>
<WorkflowModelElement>G</WorkflowModelElement>
<EventType>complete</EventType>
</AuditTrailEntry>
</ProcessInstance>
```

Όταν ανοίξουμε το αρχείο εμφανίζεται η *Οθόνη 14*. Παρατηρούμε στα key data, ότι μία διαδικασία υπάρχει στο αρχείο, 3 περιπτώσεις οι οποίες αντιστοιχούν στις 3 διαφορετικές εκτελέσεις, 18 γεγονότα τα οποία αντιστοιχούν στον συνολικό αριθμό εκτελέσεων όλων των διεργασιών. Οι 7 κλάσεις γεγονότων αντιστοιχούν στις 7 διαφορετικές διεργασίες ABCDEFG, και ο τύπος των γεγονότων είναι 1 και είναι ”complete”.

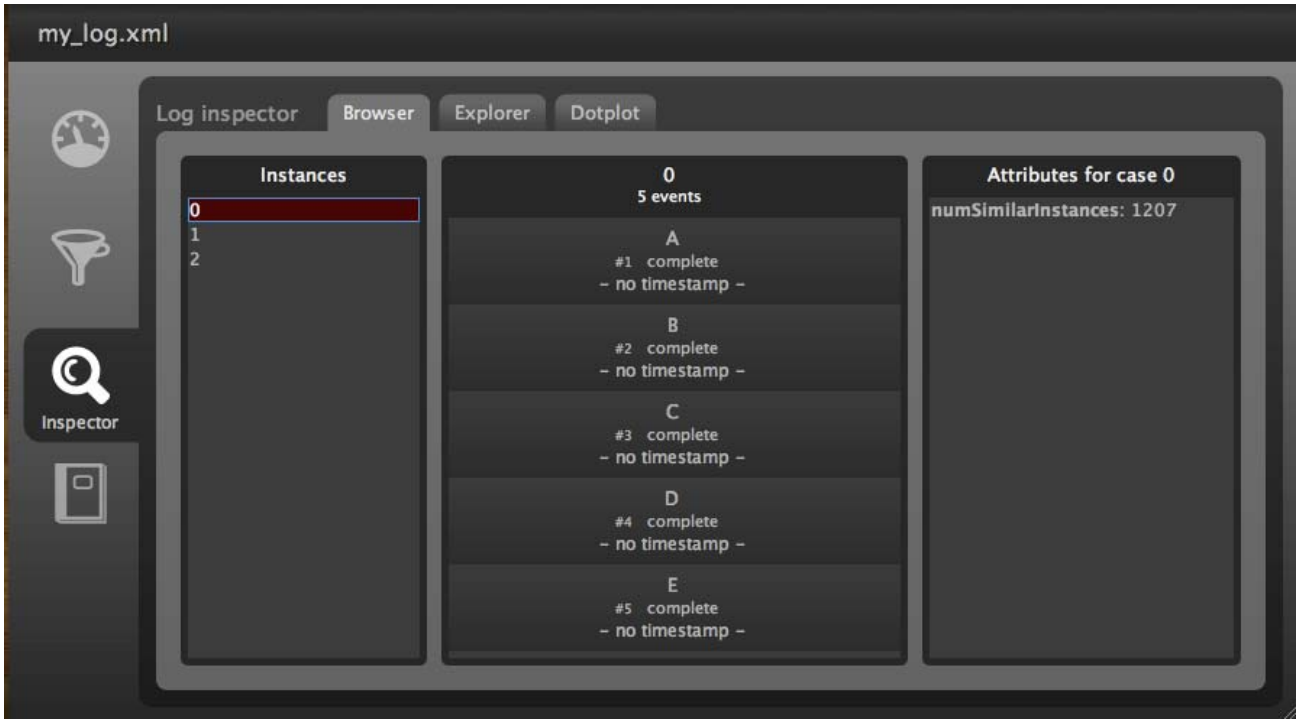
Οθόνη 14

Αν επιλέξουμε το Filter εμφανίζεται η πιο κάτω οθόνη η οποία μας δίνει πληροφορίες σχετικά με τα γεγονότα, που στο συγκεκριμένο παράδειγμα είναι μόνο οι διεργασίες που ανήκουν οι στη διαδικασία.



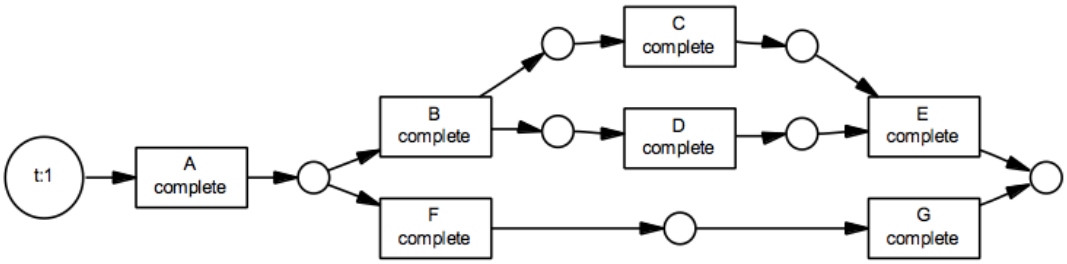
Οθόνη 15

Άν επιλέξουμε το Inspector μπορούμε να δούμε τις 4 διαφορετικές περιπτώσεις (εκτελέσεις) που υπάρχουν στο αρχείο. Εκτός από να δούμε αυτές τις περιπτώσεις, μπορουμε επίσης επιλέγωντας μία από αυτές να κατασκευάσουμε το μοντέλο της κάθε περίπτωσης ξεχωριστά.



Οθόνη 16

Όταν τρέξουμε τον α-αλγόριθμο , το Petri-net που παίρνουμε είναι το πιο κάτω (*Σχήμα 5.4*). Στο πρώτο απλό παράδειγμα στο οποίο κατασκευάσαμε Petri-net μέσα από ένα μικρό πίνακα με εκτελέσεις είχαμε το ίδιο ακριβώς αποτέλεσμα *Σχήμα 1.8*.



Σχήμα 5.4

Κεφάλαιο 6

Δίκτυα Προτιμήσεων CP-nets

Σε αυτό το κεφάλαιο θα μελετήσουμε τα δίκτυα προτιμήσεων CP-nets. Είναι δίκτυα τα οποία χρησιμοποιούνται για την αναπαράσταση προτιμήσεων. Χρησιμοποιούνται σε αρκετά παραδείγματα για την αναπαράσταση προτιμήσεων κάποιων ατόμων και είναι χρήσιμα για την αναπαράσταση των διαφόρων προτιμήσεων των χρηστών σε διάφορες εφαρμογές. Επίσης θα προσπαθήσουμε να συσχετίσουμε αυτά τα δίκτυα με την εξόρυξη ροής εργασιών, προτείνωντας έναν τρόπο όπου μέσα από ένα αρχείο εκτελέσεων, θα μπορούμε να κατασκευάσουμε το CP-net.

Η συλλογή πληροφοριών σχετικά με τις προτιμήσεις χρηστών σε διάφορους τομείς, είναι μία επίπονη διαδικασία και οι αναλυτές αποφάσεων έχουν κατασκευάσει κάποιες τεχνικές έτσι ώστε να γίνει αυτοματοποίηση αυτής της διαδικασίας. Τα CP-nets έχουν προταθεί σαν ένα απλό εργαλείο για την γραφική αναπαράσταση των

προτιμήσεων, ορίζοντας σχέσεις ανάμεσα στις προτιμήσεις, χρησιμοποιώντας υποθετικές προτάσεις προτιμήσεων που λέγονται ceteris paribus.

Η σημασιολογία των υποθετικών (conditional) ceteris paribus προτάσεων, απαιτεί ότι για ένα χαρακτηριστικό ενδιαφέροντος Α, ένα άτομο καθορίζει πια άλλα χαρακτηριστικά θα επηρεάσουν την προτίμηση του για το Α. Για παράδειγμα, όταν ένα άτομο θέλει να αγοράσει μία τηλεόραση και λέει « μία τηλεόραση 40' θα ήταν καλύτερη από μία 32'», δεν εννοεί ότι μία τηλεόραση 40' είναι πάντα καλύτερη από μία 32' ανεξάρτητα από τα άλλα χαρακτηριστικά τους. Αυτό που θέλει να πει είναι αν έχει να επιλέξει ανάμεσα σε δύο τηλεόρασης που η κύρια διαφορά τους είναι το μέγεθος και δεν έχουν σημαντικές διαφορές στα υπόλοιπα χαρακτηριστικά, θα επιλέξει αυτήν των 40'. Οι προτάσεις αυτού του είδους είναι πολύ φυσικές και εμφανίζονται σε πολλές εφαρμογές. Συγκεκριμένα μια τέτοια πρόταση έχει την εξής μορφή: Το a1 προτιμείται σε σχέση με το a2, όταν ισχύει το b1 και c2. [8,9]

6.1 Σχέσεις Προτιμήσεων και αναπαράσταση CP-net

6.1.1 Σχέσεις προτιμήσεων:

Υποθέτουμε ότι έχουμε ένα σύνολο καταστάσεων S. Για κάθε κατάσταση υπάρχει ένα σύνολο πράξεων (actions) τις οποίες μπορούμε να εκτελέσουμε. Κάθε πράξη όταν εκτελείται έχει ένα συγκεκριμένο αποτέλεσμα (outcome). Το σύνολο των *outcomes* συμβολίζεται με O. Η κατάταξη των προτιμήσεων είναι η εφαρμογή κάποιων σχέσεων ανάμεσα στα αποτελέσματα (outcomes).

- $o_1 \geq o_2$ δηλώνει ότι το o_1 προτιμάται το ίδιο ή περισσότερο από το o_2.
- $o_1 > o_2$ δηλώνει ότι το o_1 προτιμάται περισσότερο από το o_2.
- $o_1 \sim o_2$ δηλώνει ότι δεν υπάρχει προτίμηση ανάμεσα στα δύο, (δηλ. $o_1 \geq o_2$ και $o_2 \geq o_1$).

Ας ορίσουμε το $\mathbf{V}=\{X_1,X_2,\ldots,X_n\}$ σαν ένα σύνολο από μεταβλητές όπου ένα άτομο δηλώνει τις προτιμήσεις του. Κάθε μεταβλητή X_i, συσχετίζεται με ένα σύνολο τιμών τις οποίες μπορεί να πάρει το $Dom(X_i)$. Μία ανάθεση τιμών x σε ένα σύνολο $\mathbf{X} \subseteq$

$\mathbf{V}$ συσχετίζει κάθε μεταβλητή στο $\mathbf{X}$ με μία τιμή από το σύνολο τιμών της Dom(). Αν $\mathbf{X} = \mathbf{V}$ τότε η ανάθεση είναι «πλήρης», διαφορετικά είναι «μερική» ανάθεση. Το σύνολο όλων των αναθέσεων στο $\mathbf{X} \subseteq \mathbf{V}$, το ορίζουμε σαν $Ass(\mathbf{X})$.

Αν x ανάθεση στο σύνολο $\mathbf{X}$ και y ανάθεση στο σύνολο $\mathbf{Y}$ και ισχύει $\mathbf{X} \cap \mathbf{Y} = \emptyset$ ο συνδυασμός των x και y δηλώνεται σαν xy. Οι πλήρεις αναθέσεις αναφέρονται απευθείας στα outcomes όπου κάποιος μπορεί να έχει κάποιες προτιμήσεις. Για κάθε outcome o, δηλώνουμε σαν $o[X]$ την τιμή $x \in Dom(X)$ που ανατέθηκε στη μεταβλητή X από αυτό το outcome.

Ένα σύνολο μεταβλητών $\mathbf{X}$ είναι προτιμησιακά ανεξάρτητο (preferentially independent) από το συμπλήρωμα του $\mathbf{Y}=\mathbf{V}-\mathbf{X}$ αν και μόνο αν (iff) για όλα τα $x_1, x_2 \in Ass(\mathbf{X})$ και $y_1, y_2 \in Ass(\mathbf{X})$, έχουμε:

$$x_1 y_1 \geq x_2 y_1 \quad \text{iff} \quad x_1 y_2 \geq x_2 y_2$$

Η δομή της κατάταξης των προτιμήσεων πάνω σε αναθέσεις του $\mathbf{X}$, όταν οι άλλες μεταβλητές παραμένουν σταθερές, είναι η ίδια ανεξάρτητα από την τιμή που θα πάρουν οι άλλες μεταβλητές. Αν ισχύει η πιο πάνω σχέση λέμε ότι το x_1 προτιμάται από το x_2 ceteris paribus.

Ένα σύνολο μεταβλητών $\mathbf{X}$ είναι υποθετικά προτιμησιακά ανεξάρτητο (conditionally preferentially independent) του $\mathbf{Y}$, όταν έχουμε $\mathbf{X,Y,Z}$ να αποτελούν το $\mathbf{V}$ και δίνεται ανάθεση $\mathbf{z}$ στο $\mathbf{Z}$, αν και μόνο αν για όλα τα $x_1, x_2 \in Ass(\mathbf{X})$ και $y_1, y_2 \in Ass(\mathbf{X})$, έχουμε:

$$x_1 y_1 \mathbf{z} \geq x_2 y_1 \mathbf{z} \quad \text{iff} \quad x_1 y_2 \mathbf{z} \geq x_2 y_2 \mathbf{z}$$

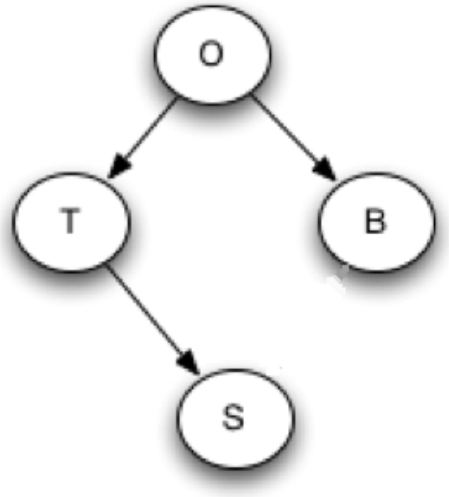
Άρα το $\mathbf{X}$ είναι υποθετικά προτιμησιακά ανεξάρτητο του $\mathbf{Y}$ αν στο $\mathbf{Z}$ ανατεθεί το $\mathbf{z}$. [8]

6.1.2 Αναπαράσταση CP-net

Για κάθε μεταβλητή X_i, ζητούμε από το χρήστη να δώσει ένα σύνολο προγόνων (parent) μεταβλητών $Pa(X_i)$ που μπορούν να επηρεάσουν την προτίμηση του για τις τιμές του X_i. Το X_i είναι προτιμησιακά ανεξάρτητο του $V-(Pa(X_i) \cup \{X_i\})$, έτσι ζητούμε από το χρήστη να δώσει τις προτιμήσεις για τις τιμές του X_i, για όλα τα στιγμιότυπα των μεταβλητών $Pa(X_i)$. Το CP-net είναι με μεταβλητές $\mathbf{V}=\{X_1,X_2,\ldots,X_n\}$ είναι ένας κατευθυνόμενος γράφος G πάνω στο $X_1,X_2,\ldots,X_n$ και όπου κάθε κόμβος έχει ένα υπο-συνθήκη (conditional) πίνακα προτιμήσεων $CPT(X_i)$. [8] . Για την καλύτερη κατανόηση των CP-nets ακολουθεί ένα παράδειγμα.

Παράδειγμα 6.1

Προτιμώ να πηγαίνω στον κινηματογράφο με τους φίλους μου, παρά σε νυχτερινά κέντρα διασκέδασης. Αν πάω κινηματογράφο προτιμώ να φορέσω ένα φανελάκι και παντελόνι, ενώ αν θα πάω σε ένα νυχτερινό κέντρο προτιμώ να φορέσω μία φούστα με μία μπλούζα. Αν θα φορέσω παντελόνι προτιμώ να βάλω χαμηλά παπούτσια , ενώ αν θα φορέσω φούστα προτιμώ να βάλω ψηλά παπούτσια.

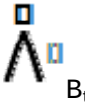


Πίνακας Προτιμήσεων

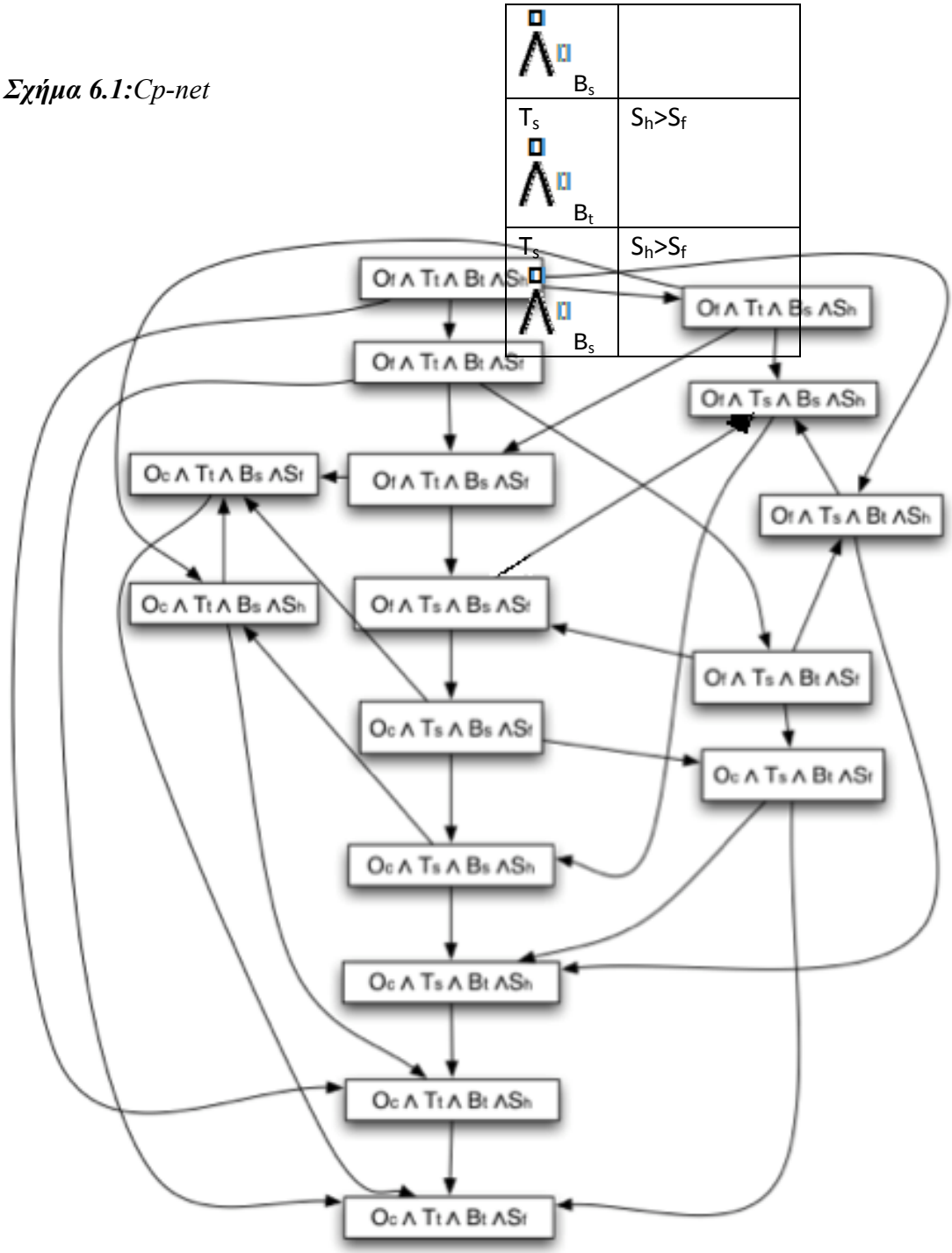
$O_c > O_f$

O_c	$T_t > T_s$
O_f	$T_s > T_t$

O_c	$B_t > B_s$
O_f	$B_s > B_t$

T_t  B_t	$S_f > S_h$
T_t	$S_f > S_h$

Σχήμα 6.1:Cp-net



Σχήμα 6.2:γράφος προτιμήσεων

6.2 Βελτιστο Αποτέλεμα και Σύγκριση αποτελεσμάτων

6.2.1 Βελτιστο Αποτέλεμα (Outcome Optimization)

Μία από τις βασικές ιδιότητες των CP-nets είναι ότι, με βάση ένα δεδομένο CP-net

N, μπορούμε με ευκολία να βρούμε ποιο είναι το βέλτιστο αποτέλεσμα (outcome) ανάμεσα σε όλες τις προτιμήσεις που ικανοποιούν το N. Αυτή την ερώτηση την ονομάζουμε «ερώτηση βελτιστοποίησης αποτελέσματος».

Αλγόριθμος για βελτιστοποίηση αποτελέσματος

Για να βρούμε το βέλτιστο αποτέλεσμα, απλά πρέπει να σαρώσουμε το CP-net από πάνω προς τα κάτω, και για κάθε μεταβλητή να δίνουμε την τιμή η οποία είναι προτιμότερη δεδομένων των στιγμιότυπο των γονέων της. Το δίκτυο είναι σε θέση να μας δώσει μοναδικό βέλτιστο αποτέλεσμα. Μέσα από το επόμενο παράδειγμα εξηγείται πως εφαρμόζεται ο αλγόριθμος. [8]

Παράδειγμα 6.2

Πρόταση : Μου αρέσει να γυμνάζομαι. Προτιμώ να γυμνάζομαι πρωινές ώρες παρά το απόγευμα. Αν θα γυμναστώ το πρωί τότε προτιμώ να πηγαίνω joking στο πάρκο, αν όμως θα γυμναστώ απόγευμα, τότε προτιμώ να πάω στο γυμναστήριο του πανεπιστημίου. Αν θα πάω για joking, προτιμώ να πάω με παρέα ενώ στο γυμναστήριο, μου αρέσει να πηγαίνω μόνος.

T: μεταβλητή για την ώρα της μέρας που προτιμώ να γυμνάζομαι. Παίρνει δύο τιμές, T_m για πρωί και T_e απόγευμα.

P: μεταβλητή για το που και πως θέλω να γυμναστώ. Παίρνει δύο τιμές P_j για joking στο πάρκο, και P_g για γυμναστήριο.

W: μεταβλητή που δηλώνει το πως θέλω να πάω. Παίρνει δύο τιμές W_f αν θέλω να πάω με φίλους, και W_a αν θέλω να πάω μόνος.

Κατασκευάζουμε το CP-net :

$T_m > T_e$

T_m	$P_j > P_g$
T_e	$P_g > P_j$

Pj	$W_f > W_a$
P_g	$W_a > W_f$

Σχήμα 6.3

Η εφαρμογή του αλγόριθμου για την εύρεση του βέλτιστου αποτελέσματος στο πιο πάνω παράδειγμα θα έχει ως εξής: Ξεκινούμε με την μεταβλητή που βρίσκεται ψηλότερα στην ιεραρχία δηλ. την Τ. Για αυτή την μεταβλητή δίνουμε την τιμή την οποία είναι προτιμότερη και είναι T_m. Στην συνέχεια πάμε στην επόμενη μεταβλητή Ρ. Για αυτή την μεταβλητή, δεδομένου ότι στην μεταβλητή γονέα έχει δοθεί η τιμή T_m , τότε η προτιμότερη τιμή για την Ρ είναι η P_j. Τέλος για την μεταβλητή W, η προτιμότερη τιμή δεδομένου ότι για την Ρ δόθηκε η P_j, είναι η W_f.

Άρα το βέλτιστο αποτέλεσμα, με βάση το πιο πάνω CP-net, είναι: T_m P_j W_f. Να γυμναστώ πρωί πηγαίνοντας για joking με παρέα.

6.2.2 Σύγκριση Αποτελεσμάτων

Εκτός από την εύρεση του βέλτιστου αποτελέσματος, μία άλλη διαδικασία η οποία πρέπει να υποστηρίζεται από ένα μοντέλο αναπαράστασης προτιμήσεων, είναι η σύγκριση ανάμεσα στα αποτελέσματα. Συγκεκριμένα, η δυνατότητα ανάμεσα σε δύο αποτελέσματα να μπορούμε να ξεχωρίσουμε ποιο προτιμάται περισσότερο. Υπάρχουν δύο τρόποι, με βάση τους οποίους μπορούμε να συγκρίνουμε δύο αποτελέσματα με βάση ένα CP-net:

1. Dominance Queries (Ερωτήσεις κυριαρχίας) : Δεδομένου ενός CP-net *N* και δύο αποτελεσμάτων ο και ο', ψάχνουμε αν N |= *o* > *o'*. Αν αυτή η σχέση ισχύει, τότε το *o* προτιμάται από το *o*' και λέμε ότι το *o* κυριαρχεί του *o'* ως προς το *N*.

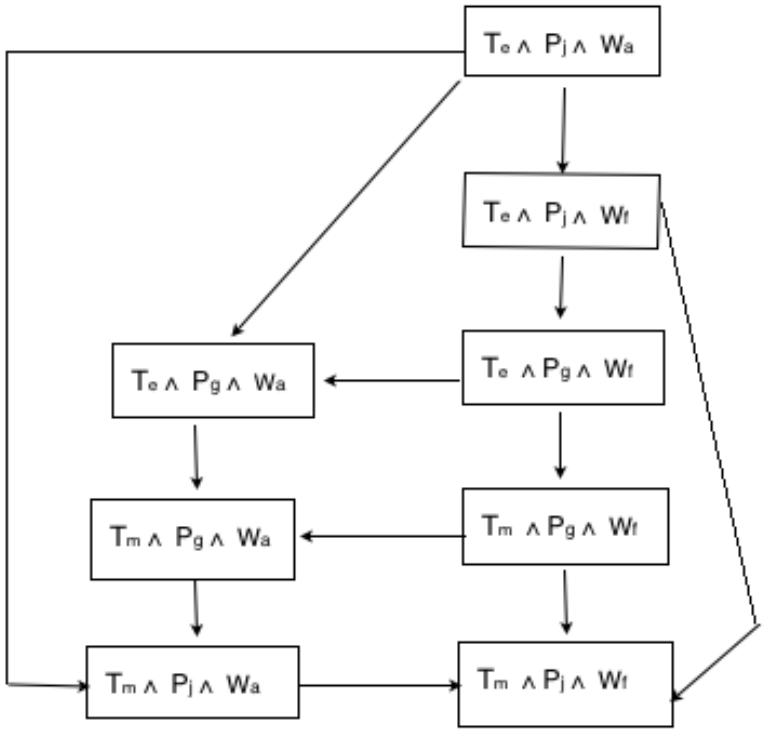
2. Ordering Queries (Ερώτηση Ταξινόμησης) : Δεδομένου ενός CP-net *N* και δύο αποτελεσμάτων *o* και *o'*, ψάχνουμε αν

ισχύει N |≠ *o′* > *o*. Αν αυτό ισχύει, τότε υπάρχει μία προτιμησιακή ταξινόμηση συνεπής με το N, στην οποία *o* > *o′*. Είναι λοιπόν συνέπεια της γνώσης που εκφράζεται από το N, ότι το ο ταξινομείται "over" (πάνω από το) *o′*.

Οι σχέσεις ταξινόμησης, είναι πιο αδύναμες από τις κυρίαρχες σχέσεις. Αν ισχύει N |= *o* > *o′* τότε σίγουρα ισχύει N |≠ *o*′ > *o*. Αν όμως ξέρουμε ότι ισχύει N |≠ *o*′ > *o*, τότε μπορεί ταυτόχρονα να ισχύει και η σχέση N |≠ o > o'. Άρα η σχέση N |≠ *o*′ > *o* δεν σημαίνει πάντα ότι το *o* προτιμάται από το *o′*, αλλά μπορεί να σημαίνει ότι τα δύο αυτά αποτελέσματα δεν μπορούν να συγκριθούν προτιμησιακά, ή ότι έχουν την ίδια σειρά προτίμησης. Παρόλο που οι κυρίαρχες σχέσεις είναι συνήθως πιο χρήσιμες, κάποιες φορές οι σχέσεις ταξινόμησης είναι επαρκής. Υπάρχουν εφαρμογές στις οποίες δεν είναι τόσο απαραίτητο να υπάρχουν κυρίαρχες σχέσεις και οι σχέσεις ταξινόμησης θεωρούνται ικανοποιητικές. [8]

Flipping Sequences

Η σημασιολογία των ceteris paribus, επιτρέπει στην άμεση χρήση των πληροφοριών από το CPT της μεταβλητής X, έτσι ώστε να αλλάξουμε την τιμή της X μέσα σε ένα αποτέλεσμα έτσι ώστε να πετύχουμε ένα καλύτερο αποτέλεσμα. Μπορούμε να φτιάξουμε μία αλυσίδα αποτελεσμάτων στην οποία κάθε φορά αλλάζει η τιμή μίας μεταβλητής, έτσι ώστε να φτάσουμε στο προτιμότερο αποτέλεσμα. Αυτή η αλυσίδα, ονομάζεται flipping sequence. Κατασκευάζοντας τον γράφο προτιμήσεων για ένα CP-net μπορούμε να πάρουμε ένα ή περισσότερα flipping sequences. Από το προηγούμενο παράδειγμα έχουμε το πιο κάτω γράφο προτιμήσεων. [8]



Σχήμα 6.4

Από τον πιο πάνω γράφο προτιμήσεων, τα flipping sequences που παίρνουμε είναι:

$T_eP_jW_a < T_eP_jW_f < T_eP_gW_f < T_mP_gW_f < T_mP_jW_f$

$T_eP_jW_a < T_eP_gW_a < T_mP_gW_a < T_mP_jW_a < T_mP_jW_f$

$T_eP_jW_a < T_mP_jW_a < T_mP_jW_f$

$T_eP_jW_f < T_mP_jW_f$

$T_eP_jW_a < T_eP_jW_f < T_eP_gW_f < T_eP_gW_a < T_mP_gW_a < T_mP_jW_a < T_mP_jW_f$

$T_eP_jW_a < T_eP_jW_f < T_eP_gW_f < T_mP_gW_f < T_mP_gW_a < T_mP_jW_a < T_mP_jW_f$

Σε ένα flipping sequence, κάθε φορά αλλάζει η τιμή μίας μεταβλητής. Σε ένα CP-net, για δύο αποτέλεσμα *o* και *o'*, τα οποία βρίσκονται μαζί σε μία αλυσίδα (flipping sequence) βελτίωσης από το ο προς το *o'*, μπορούμε να συμπεραίνουμε ότι ισχύει η σχέση κυριαρχίας N |= *o'* > *o.* Ενώ αν βρίσκονται σε αλυσίδα χειροτέρευσης από το ο στο ο', τότε ισχύει η σχέση κυριαρχίας N |= *o* > *o'*.

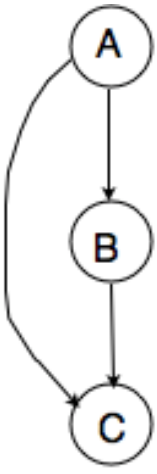
Αν N είναι ένα ακυκλικό CP-net και δεν υπάρχει flipping sequence από το αποτέλεσμα *o* στο *o'*, τότε N |≠ o' > o. Αν N είναι ένα ακυκλικό CP-net και δεν υπάρχει flipping sequence από το αποτέλεσμα *o* στο *o'*, τότε N |≠ *o'* > *o.*

6.3 Κατασκευή CP-net από αρχείο καταγραφής προτιμήσεων

Αφού μελετήσαμε τα CP-nets θα προτείνουμε κάποιες ιδέες με βάση τις οποίες έχοντας ένα αρχείο στο οποίο καταγράφονται προτιμήσεις, να μπορούμε κατασκευάσουμε το CP-net. Σε αυτή την προσέγγιση οι μεταβλητές θα αποτελούν τις διεργασίες, κάθε αλλαγή τιμής της μεταβλητής σε ένα flipping sequence θα αποτελεί εκτέλεση της συγκεκριμένης διεργασίας, η σειρά αλλαγών του flipping sequence θα αντιστοιχεί σε μιά εκτέλεση (στιγμιότυπο της διαδικασίας) στο αρχείο ενώ το το δίκτυο που θα κατασκευαστεί μέσα από το αρχείο θα αντιστοιχεί σε μια διαδικασία.

Ας πάρουμε όμως πρώτα ένα παράδειγμα όπου μας είναι γνωστό το CP-net και να βγάλουμε μέσα από αυτό κάποια συμπεράσματα τα οποία θα μας βοηθήσουν να προτείνουμε τις ιδέες που αναφέραμε.

Παράδειγμα 6.3



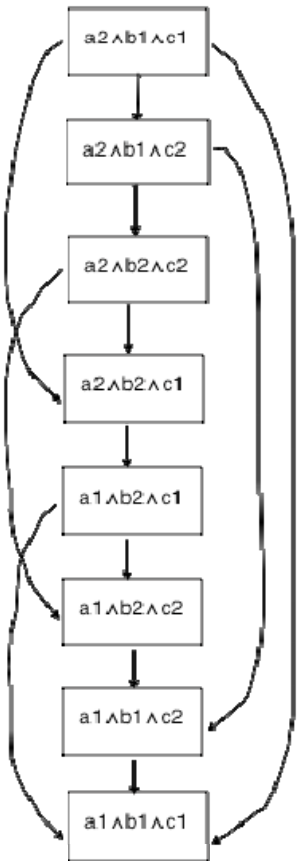
A: a1>a2

B: a1: b1>b2
a2: b2>b1

C: a1∧b1: c1 > c2
a1∧b2:c2 > c1
a2∧b1:c2 > c1
a2∧b2:c1 > c2

Σχήμα 6.5

Σχήμα 6.6



Ο γράφος προτιμήσεων, ξεκινά από το λιγότερο

επιθυμητό αποτέλεσμα και φτάνει στο πιο επιθυμητό. Θεωρούμε κάθε αποτέλεσμα (προτίμηση) σαν μια κατάσταση. Για να πάμε από την μια κατάσταση στην άλλη πρέπει να αλλάξει η τιμή μιας μεταβλητής. Κάθε αλλαγή μεταβλητής την βλέπουμε σαν μια εκτέλεση της συγκεκριμένης διεργασίας. Θέτουμε λοιπόν το γράφο προτιμήσεων σαν το μοντέλο μιας διαδικασίας που μπορεί να μας δώσει τις πιο κάτω εκτελέσεις:

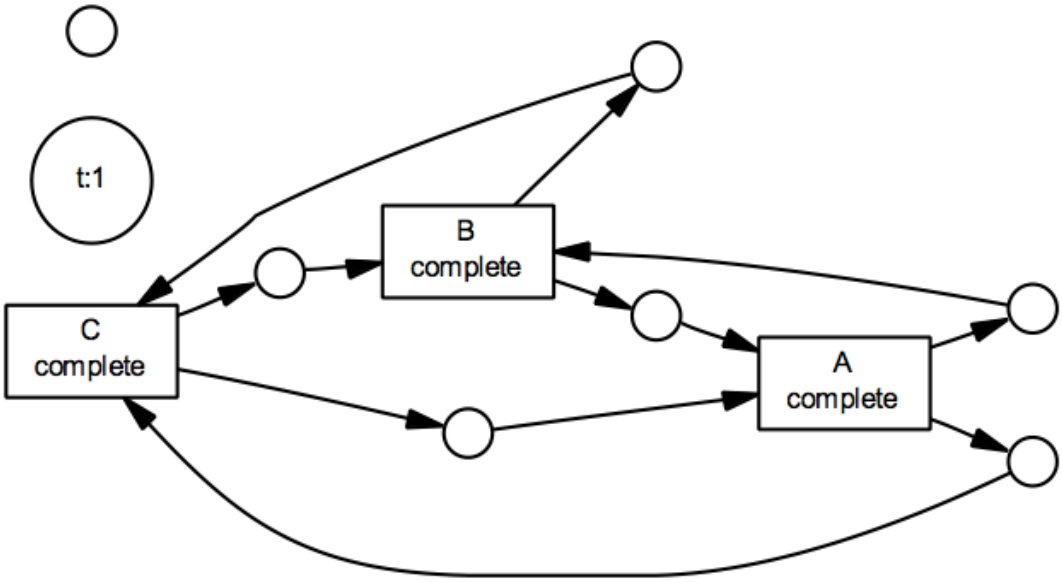
CBCACBC
A
CAC
BACBC
BAB
CBABC
CBCAB

Ας δούμε όμως κάποια συμπεράσματα τα οποία μπορούμε να βγάλουμε μέσα από αυτές τις εκτελέσεις.

- Η μεταβλητή η οποία έχει εκτελεσθεί (αλλάξει τιμή) συνολικά τις περισσότερες φορές είναι η μεταβλητή η οποία είναι χαμηλότερα στην ιεραρχία του CP-net και είναι η C. Ενώ αυτή που εκτελείται τις λιγότερες φορές είναι αυτή που βρίσκεται ψηλότερα στην ιεραρχία του CP-net και είναι η A.
- Αν μια μεταβλητή X εκτελείται για πρώτη φορά και αυτή η μεταβλητή X εξαρτάται από κάποια μεταβλητή Y, τότε υπάρχουν εκτελέσεις στις οποίες παρατηρούμε ότι αν η Y εκτελεστεί αμέσως μετά την X, τότε η X εκτελείται και αμέσως μετα την Y, δηλ. υπάρχει ακολουθία XYX.
- Η εκτέλεση κάθε μεταβλητής, αντιστοιχεί στην αλλαγή της τιμής της από μια τιμή σε μία άλλη προτιμότερη π.χ για μια μεταβλητή X, μία εκτέλεση μπορεί να είναι από X1 σε X2. Αν η X εξαρτάται από κάποια Y μεταβλητή, αυτό σημαίνει ότι για μια συγκεκριμένη τιμή της Y (π.χ Y1), προτιμάται η τιμή X2. Στο παράδειγμα παρατηρούμε ότι όσες φορές αλλάζει η B από b2 σε b1, η τιμή της A είναι α1. Άρα αυτό σημαίνει ότι α1: b1>b2.

Αν σε αυτές τις εκτελέσεις θέλαμε να εφαρμόσουμε τον αλγόριθμο κατασκευής

κατευθυνόμενου γράφου αυτό θα ήταν αδύνατο για τον αλγόριθμο που περιγράψαμε (κεφάλαιο 2.3) αφού θα θεωρεί ότι οι εκτελέσεις περιέχουν κύκλους (υπάρχουν διεργασίες που εμφανίζονται πάνω από μία φορά σε κάθε εκτέλεση). Αν θέλαμε να εφαρμόσουμε τον α-αλγόριθμο το αποτέλεσμα που θα πάρουμε δεν θα είναι το CP-net αφού οι σχέσεις που δημιουργεί ανάμεσα στις διεργασίες δεν είναι οι κατάλληλες. Συγκεκριμένα το δίκτυο που παίρνουμε χρησιμοποιώντας το ProM για να εφαρμόσουμε τον α-αλγόριθμο σέ ένα αρχείο με αυτές τις εκτελέσεις είναι:



Σχήμα 6.7

Όπως βλέπουμε το ***Σχήμα 6.7*** και συγκεκριμένα οι σχέσεις που υπάρχουν ανάμεσα στις διεργασίες (μεταβλητές) δεν είναι αυτές που αναπαριστούσε το αντίστοιχο CPnet (***Σχήμα 6.5***).

Έτσι λοιπόν θα προσπαθήσουμε να προτείνουμε κάποιες τεχνικές έτσι ώστε μέσα από ένα αρχείο που περιέχει προτιμήσεις να μπορούμε να κατασκευάσουμε ένα CP-net. Το αρχείο θεωρούμε ότι για δύο μεταβλητές X και Y οι οποίες μπορούν να εμφανιστούν σε μία εκτέλεση στη μορφή XYX, τότε αυτή η εκτέλεση υπάρχει στο αρχείο. Υποθέτουμε επίσης ότι οι τιμές που μπορεί να πάρει μια μεταβλητή είναι μόνο δύο. Για κάθε flipping sequence το αρχείο περιέχει και την αντίστοιχη εκτέλεση (σειρά αλλαγής μεταβλητών), όπως αυτή που είδαμε στο προηγούμενο

παράδειγμα. Άρα ξεκινούμε με τις εκτελέσεις του αρχείου.

1) Σε κάθε εκτέλεση, ξεκινούμε και πέρνουμε την εκτέλεση κάθε διεργασίας ξεχωριστά. Για κάθε μεταβλητή Χ που εκτελείται, αμέσως μετά ακολουθεί η εκτέλεση μιας δεύτερης διεργασίας Υ, αν αμέσως μετά το Υ εκτελέιται ξανά η Χ, τότε η Χ εξαρτάται από την Υ (δηλ. η Υ είναι γονέας της Χ).

2) Για κάθε μεταβλητή που υπάρχει στο αρχείο, μετράμε τον συνολικό αριθμό εκτελέσεων της. Η μεταβλητή η οποία έχει τον μικρότερο αριθμό εκτελέσεων είναι αυτή που βρίσκεται ψηλότερα στην ιεραρχία και δεν εξαρτάται από κάποια άλλη μεταβλητή. Ομοίως η μεταβλητή με τον μεγαλύτερο αριθμό εκτελέσεων είναι αυτή που βρίσκεται χαμηλότερα στην ιεραρχία.

3) Αφού βρούμε τις εξαρτήσεις ανάμεσα στις μεταβλητές καθώς και την ιεραρχία τους, κατασκευάζουμε το CP-net, με ακμές από την μεταβλητή γονέα προς την μεταβλητή «παιδί».

4) Για κάθε μεταβλητή(έστω Χ μεταβλητή) που υπάρχει στο αρχείο, παίρνουμε για κάθε εκτέλεση της το αντίστοιχο αποτέλεσμα (outcome). Έστω Υ γονέας της Χ. Η τιμή που παίρνει η Χ στο συγκεκριμένο αποτέλεσμα, είναι η τιμή η οποία προτιμάται όταν ισχύει η συγκεκριμένη τιμή της Υ που υπάρχει σε αυτό το αποτέλεσμα. Αν δηλ. το αποτέλεσμα είναι Χ1Υ1, τότε θέτουμε Υ1: Χ1>Χ2.

Θα εφαρμόσουμε αυτά τα βήματα στα αποτέλεσμα που παίρνουμε από το γράφο προτιμήσεων *Σχήμα 6.4* από το παράδειγμα 6.2 για να δούμε αν μπορούμε να το επαληθεύσουμε και να παράξουμε το ίδιο CP-net. Το αρχείο θα περιέχει τις flipping sequences που βγαίνουν από το γράφο προτιμήσεων και τις αντίστοιχες εκτελέσεις.

$T_eP_jW_a < T_eP_jW_f < T_eP_gW_f < T_mP_gW_f < T_mP_jW_f$ (WPTP)

$T_eP_jW_a < T_eP_gW_a < T_mP_gW_a < T_mP_jW_a < T_mP_jW_f$ (PTPW)

$T_eP_jW_a < T_mP_jW_a < T_mP_jW_f$ (TW)

$T_eP_jW_a < T_eP_jW_f < T_mP_jW_f$ (WT)

$T_eP_jW_a < T_eP_jW_f < T_eP_gW_f < T_eP_gW_a < T_mP_gW_a < T_mP_jW_a < T_mP_jW_f$ (WPWTPW)

$T_eP_jW_a < T_eP_jW_f < T_eP_gW_f < T_mP_gW_f < T_mP_gW_a < T_mP_jW_a < T_mP_jW_f$ (WPTWPW)

Εφαρμόζοντας το βήμα 1 στις πιο πάνω εκτελέσεις παίρνουμε ότι:

-Η P εξαρτάται μόνο από την T, άρα η T είναι γονέας της P => Pa(P)=T

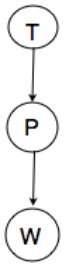
-Η W εξαρτάται μόνο από την P, άρα η P είναι γονέας της W => Pa(W)=P

-Η T δεν εξαρτάται από κάποια άλλη μεταβλητή, άρα δεν έχει γονείς => Είναι προτημισιακά ανεξάρτητη από όλες τις άλλες μεταβλητές.

Εφαρμόζοντας το βήμα 2 στις πιο πάνω εκτελέσεις παίρνουμε ότι:

-Η W εκτελείται 9 φορές, η P 8 φορές και η T 6 φορές. Άρα η W βρίσκεται χαμηλότερα στην ιεραρχία, η P είναι δεύτερη στην ιεραρχία και η T είναι πρώτη στην ιεραρχία.

Εφαρμόζοντας το βήμα 3 παίρνουμε το CP-net *Σχήμα 6.7*, το οποίο είναι το ίδιο με αυτό που μας δοθηκε αρχικά. Άρα η τεχνική αυτή επαληθεύεται και για αυό το παράδειγμα.



Σχήμα 6.8

Στο βήμα 4, για την T μεταβλητή, παρατηρούμε ότι κάθε φορά που αλλάζει τιμή παίρνει την τιμή που είναι προτιμότερη ανεξάρτητα από τις τιμές των άλλων μεταβλητών και είναι Tm, άρα Tm>Te. Για την τιμή P παίρνουμε τα πιο κάτω αποτελέσματα όταν γίνει αλλαγή της τιμής της: $T_eP_gW_f$, $T_mP_jW_f$, $T_eP_gW_a$, $T_mP_jW_a$.

αφού ξέρουμε ότι η P εξαρτάται μόνο από την T, συμπεράνουμε ότι για Te: Pg>Pj ενώ για Tm:Pj>Pg. Για την τιμή W παίρνουμε τα πιο κάτω αποτελέσματα όταν αλλαγή της τιμής της: $T_eP_jW_f$, $T_mP_jW_f$, $T_eP_gW_a$, $T_mP_gW_a$. Αφού ξέρουμε ότι η W εξαρτάται μόνο από την P, συμπεράνουμε ότι για Pj: Wf>Wa ενώ για Pg: Wa>Wf.

6.4 Κατασκευή Cp-net με τη χρήση ProM

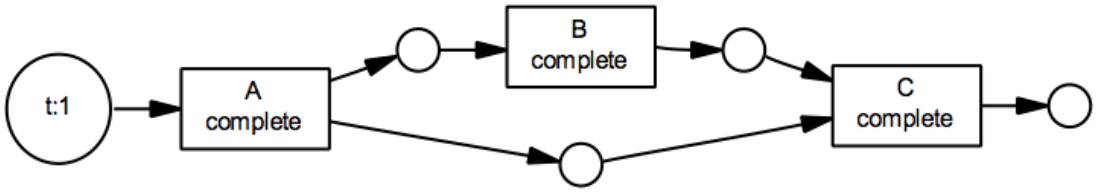
Ένας τρόπος να κατασκευάσουμε ένα δίκτυο το οποίο αντιστοιχεί στο CP-net με τη χρήση του α-αλγόριθμου στο ProM. Αρχικά θα εφαρμόσουμε το πρώτο βήμα της προηγούμενης προσέγγισης για να βρούμε τις σχέσεις ανάμεσα στις μεταβλητές.
Συγκεκριμένα σε κάθε εκτέλεση όταν βρίσκουμε ακολουθία της μορφής XYX, θα εισάγουμε στο αρχείο μια νέα εκτέλεση YX, αν δεν υπάρχει ήδη. Ο λόγος είναι ότι μόνο από τις ακολουθίες αυτής της μορφής μπορούμε να βρούμε τις εξαρτήσεις ανάμεσα στις διεργασίας. Έτσι από μια ακολουθία XYX, αφού το X εμφανίζεται δύο φορές αν την δώσουμε σαν είσοδο στο αρχείο ο α-αλγόριθμος ξέρουμε ότι θα δημιουργήσει τις σχέσεις $X>_WY$ και $Y>_WX$ => $X||_WY$, άρα ότι οι δύο διεργασίες είναι ανεξάρτητες. Έτσι αφαιρούμε την πρώτη εμφάνιση στο X και εισάγουμε την εκτέλεση YX στο αρχείο.

Για το προηγούμενο παράδειγμα από την πρώτη εκτέλεση παίρνουμε την υποακολουθία PTP, άρα θα βάλουμε σαν πρώτη εκτέλεση στο αρχείο “TP” και από την πέμπτη εκτέλεση παίρνουμε την υποακολουθία WPW άρα θα έχουμε ακόμα μια εκτέλεση στο αρχείο την “PW”. Το δίκτυο που θα πάρουμε εφαρμόζοντας τον α-αλγόριθμο χρησιμοποιώντας το ProM είναι:

Σχήμα 6.9

Παρατηρούμε ότι αυτό το Petri-net στο *Σχήμα 6.9,* αντιστοιχεί στο CP-net *Σχήμα 6.3,* άρα είναι το αποτέλεσμα που θέλαμε να πάρουμε.

Αν εφαρμόσουμε αυτή την τεχνική και στο παράδειγμα 6.3, οι είσοδοι θα είναι "AB", "AC", "BC". Εφαρμόζοντας τον α-αλγόριθμο θα πάρουμε το *Σχήμα 6.10.*



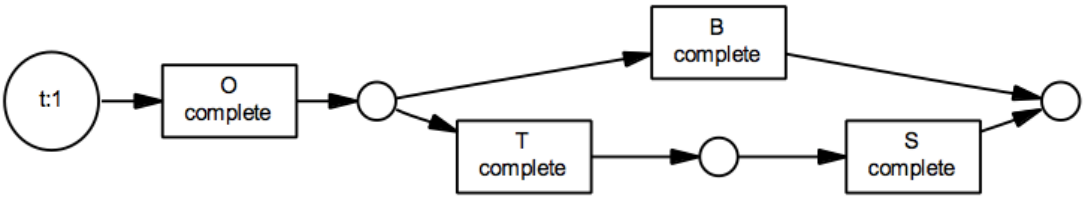
Σχήμα 6.10

Παρατηρούμε ότι και αυτό το petri-net αντιστοιχεί στο CP-net του παραδείγματος από το οποίο πήραμε τις εκτελέσεις (παράδειγμα 6.3).

Από το παράδειγμα 6.1 κάποιες από τις εκτελέσεις που μπορούμε να πάρουμε είναι:

SBOB
SBTOTB
SBTOSBTS
STSOTS

Από την πρώτη εκτέλεση παίρνουμε BOB άρα θα προσθέσουμε στο αρχείο την "OB", από την δεύτερη παίρνουμε TOT και προσθέτουμε στο αρχείο "OT" και από την τέταρτη παίρνουμε STS, άρα θα προσθέσουμε και την "TS". Το αποτέλεσμα του α-αλγόριθμου το βλέπουμε στο *Σχήμα 6.11* και είναι ένα δίκτυο στο οποίο οι εξαρτήσεις ανάμεσα στις διεργασίες είναι οι ίδιες με το CP-net στο παράδειγμα 6.1.



Σχήμα 6.11

Κεφάλαιο 7
Συμπεράσματα

Οι μέθοδοι εξόρυξης ροής εργασιών που υπάρχουν, όπως είδαμε από το εργαλείο ProM είναι αρκετοί, τρεις από τους οποίους μελετήσαμε στην παρούσα διπλωματική.

Από τους αλγόριθμους που έχουμε μελετήσει παρατηρήσαμε ότι έχουν αρκετές διαφορές. Όπως είδαμε κάθε αλγόριθμος μπορεί να χειριστεί διαφορετικά ένα πρόβλημα και ίσως δώσει και διαφορετικό αποτέλεσμα. Όσο πιο περίπλοκες είναι οι εκτελέσεις στο αρχείο, τόσο πιο έντονα φαίνονται οι διαφορές των αλγορίθμων καθώς και τα μειονεκτήματα και πλεονεκτήματα τους.

Επίσης έχουμε δει ότι συγκεκριμένες κατασκευές τις οποίες ο αλγόριθμος κατασκευής κατευθυνόμενου γράφου και ο α-αλγόριθμος δεν μπορούν να εξορύξουν σωστά σύμφωνα με τις εκτελέσεις του αρχείου. Ο α-αλγόριθμος που είναι ένας κλασσικός αλγόριθμος στον τομέα της εξόρυξης ροών εργασιών έχει επεκταθεί σε άλλους αλγόριθμους και κάθε επέκταση του στοχεύει στην επίλυση συγκεκριμένου προβλήματος. Ωστόσο η μέθοδος δύο βημάτων που κατασκευάζει αρχικά ένα σύστημα μεταβάσεων και στη συνέχεια το petri-net, μπορεί να επιλύσει πολλά από τα προβλήματα άλλων αλγορίθμων αφού δεν χειρίζεται όλα τα προβλήματα με τον ίδιο τρόπο.

Στα πλαίσια αυτής της διπλωματικής μελετήσαμε και το εργαλείο ProM, το οποίο είναι ένα πολύ εύχρηστο και ταυτόχρονα χρήσιμο εργαλείο για εξόρυξη ροών εργασιών. Χρησιμοποιεί πάνω από 20 αλγόριθμους για εξόρυξη, καθώς επίσης και για ανάλυση των αποτελεσμάτων, ενώ αρκετά σημαντικό είναι το γεγονός ότι υπάρχει η δυνατότητα σύγκρισης των αποτελεσμάτων διαφορετικών αλγορίθμων. Η εισαγωγή ενός νέου αλγόριθμου σε αυτό το εργαλείο είναι πολύ εύκολη.

Τέλος προσπαθήσαμε να συσχετίσουμε τα δίκτυα προτιμήσεων CP-nets με τις μεθόδους εξόρυξης ροής εργασιών. Παρατηρήσαμε ότι παίρνοντας εκτελέσεις από γράφους προτιμήσεων δεν ήταν δυνατόν να κατασκευάσουμε το CP-net χρησιμοποιώντας αλγόριθμους που ενσωματώνει το ProM, όπως ο α-αλγόριθμος. Ο λόγος είναι ότι αυτές οι εκτελέσεις μπορεί να περιέχουν πολλές εμφανίσεις της ίδιας διαδικασίας και τα αποτελέσματα που πήραμε δεν ήταν επιθυμητά. Έτσι προτείναμε κάποιες ιδέες, ώστε μέσα από ένα αρχείο προτιμήσεων να μπορούμε να κατασκευάσουμε το κατάλληλο CP-net. Είδαμε ότι αυτά τα δεδομένα χρειάζονταν ειδική μεταχείριση και προτείναμε επίσης ένα τρόπο με τον οποίο αν τα επεξεργαστούμε, μπορούμε να τα δώσουμε σαν είσοδο στο ProM και εφαρμόζοντας τον α-αλγόριθμο να κατασκευάσουμε ένα δίκτυο το οποίο αντιστοιχεί στο σωστό δίκτυο προτιμήσεων.

Βιβλιογραφία

[1] R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Work- flow Logs. *In Sixth International Conference on Extending Database Technology*, pages 469–483, 1998..

[2] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. Elsevier.

[3] A.K.A. de Medeiros, W.M.P. van der Aalst, and A.J.M.M. Weijters. Workflow Mining: Current Status and Future Directions.

[4] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management.

[5] Tadao Murata. Petri Nets : Properties, Analysis and Applications.

[6] B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM Framework: A New Era in Process Mining Tool Support.

[7] W.M.P. van der Aalst and C.W. Gunther. Finding Structure in Unstructured Processes: The Case for Process Mining (invited paper).

[8] Craig Boutilier, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, David Poole. CP-nets: A Tool for Representing and Reasoning with Conditional Ceteris Paribus Preference Statements.

[9] Yannis Dimopoulos and Loizos Michael and Fani Athienitou. Ceteris Paribus Preference Elicitation with Predictive Guarantees.

[10] Wil M.P. van der Aalst, V. Rubin, B.F. van Dongen, E. Kindler, and C.W. Gunther. A Two-Step Approach using Transition Systems and Regions.

[11] B.F. van Dongen, A.K. Alves de Medeiros, and L.Wen. Process Mining: Overview and Outlook of Petri Net Discovery Algorithms.