ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΗΛΕΚΤΡΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΩΝ ΔΙΑΤΑΞΕΩΝ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΑΠΟΦΑΣΕΩΝ

Επίλυση προβλημάτων χρονοπρογραμματισμού με χρήση αυτομάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μιχαήλ Ασπραδάκης

Επιβλέπων: Δημήτριος Ασκούνης
Αναπληρωτής Καθηγητής ΕΜΠ

Αθήνα, Ιούλιος 2013

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΗΛΕΚΤΡΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΩΝ ΔΙΑΤΑΞΕΩΝ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΑΠΟΦΑΣΕΩΝ

Επίλυση προβλημάτων χρονοπρογραμματισμού με χρήση αυτομάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μιχαήλ Ασπραδάκης

Επιβλέπων: Δημήτριος Ασκούνης
Αναπληρωτής Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την _____

_____ _____ _____

Αθήνα, Ιούλιος 2013

ΠΕΡΙΛΗΨΗ

Στις μέρες μας οι αλλαγές που σημειώνονται σε παγκόσμια κλίμακα δεν αφορούν μόνο την οικονομική κατάσταση αλλά τείνουν να επηρεάζουν σε μεγάλο βαθμό τη λειτουργία τόσο των μικρών όσο και των μεγαλύτερων σε κλίμακα επιχειρήσεων και βιομηχανιών παραγωγής. Οι εταιρείες που καταφέρνουν να προσαρμοστούν στις νέες συνθήκες έχουν τη πιθανότητα να κερδίσουν ένα ανταγωνιστικό πλεονέκτημα. Για πολλές εταιρείες η διαδικασία προσαρμογής στο νέο περιβάλλον εκλαμβάνεται ως προσπάθεια εξεύρεσης νέων τρόπων οργάνωσης και διοίκησης της παραγωγής τους. Τα ολοένα και υψηλότερα επίπεδα εισχώρησης υπολογιστικών συστημάτων σε όλες τις επιχειρησιακές διαδικασίες μπορούν να καταστήσουν τις εταιρείες περισσότερο αποδοτικές και ευπροσάρμοστες στα νέα δεδομένα και κατ' επέκταση να τους προσφέρουν το ανταγωνιστικό πλεονέκτημα.

Το πρόβλημα του χρονοπρογραμματισμού της παραγωγικής διαδικασίας αποτελεί ένα διαχρονικό σημείο αναφοράς στον κλάδο της διοίκησης παραγωγικών συστημάτων, το οποίο τα τελευταία χρόνια έχει αποκτήσει μια επιπλέον διάσταση, αυτή της παράλληλης βελτιστοποίησης του κόστους, όπως επιβάλλεται εξαιτίας της παγκόσμιας οικονομικής αστάθειας. Με τον όρο χρονοπρογραμματισμός παραγωγικής διαδικασίας εννοούμε την εύρεση του κατάλληλου προγράμματος το οποίο θα συντονίζει τα τεχνολογικά μέσα που απαιτούνται, την κατανομή των πόρων και την παραγωγική διαδικασία, με απώτερο σκοπό τον ελάχιστο χρόνο παραγωγής και το ελάχιστο δυνατό κόστος παραγωγής, που θα καθιστά την επιχείρηση ικανή να «επιβιώσει» μέσα σε ένα άκρως ανταγωνιστικό περιβάλλον που έχει θέσει ως προτεραιότητα την ταχύτητα, την ποιότητα και την αξιοπιστία των σύγχρονων εταιρειών.

Σκοπός της παρούσας διπλωματικής εργασίας είναι η εύρεση βέλτιστης λύσης σε ένα πρόβλημα χρονοπρογραμματισμού σε σύστημα παραγωγής κατά παραγγελία (job-shop), που παράλληλα, εκτός από την οργάνωση των παραγωγικών διεργασιών στο λιγότερο δυνατό χρόνο, θα συνυπολογίζει και θα εξασφαλίζει και το ελάχιστο δυνατό κόστος παραγωγής. Αυτή η μεθοδολογία εφαρμόστηκε και στην πράξη, χρησιμοποιώντας το εργαλείο UPPAAL σε κάποια παραδείγματα χρονοπρογραμματισμού παραγωγής. Έτσι, καταφέραμε να βρούμε κάθε φορά το βέλτιστο (ή σχεδόν το βέλτιστο) ολικό χρόνο διεκπεραίωσης ορισμένων εργασιών σε πολύ καλό χρόνο υπολογισμού και με μειωμένο κόστος.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

χρονοπρογραμματισμός παραγωγής, σύστημα παραγωγής κατά παραγγελία, χρονισμένα αυτόματα, επίλυση & βελτιστοποίηση προβλημάτων χρονοπρογραμματισμού, μείωση κόστους ολοκλήρωσης εργασιών

ABSTRACT

Nowadays, the changes occuring at global scale, do not only apply to the economic situation, but also tend to greatly infuence the operations and manufacturing industries. Companies that manage to adapt to new circumstances have the chance to gain a competitive advantage. For many companies, the process of adapting to new environments is perceived as an attempt to find new ways of organization and management of production. The increasingly higher penetration of computer systems into all business processes can make companies more efficient and adaptable to new circumstances and thus offer them the competitive advantage.

The problem of scheduling the production process represents a timeless benchmark in the field of production management systems, which has gained an extra dimension in recent years, that of parallel optimazation of costs as required by the global economic instability. The term production scheduling process means finding the right program that will coordinate the technological tolls required, allocation of resources and production process, with the ultimate aim of a minimum production time and minimum cost of production, which makes the company able to "survive" in a highly competitive environment, that has prioritized the speed, quality and reliability of modern companies.

The purpose of this thesis is to find the optimal solution to a problem in scheduling production system on demand (job-shop) that except the organization of production processes in the shortest possible time, it will also include and ensure the lowest possible production costs. This methodology is applied in practice, using the tool UPPAAL in some production scheduling examples.

KEYWORDS

production scheduling, production system on demand, timed automata, cost reduction, solving & optimizing scheduling problems

ΠΡΟΛΟΓΟΣ

Η παρούσα διπλωματική εκπονήθηκε στη σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου, στα πλαίσια των δραστηριοτήτων του εργαστηρίου Συστημάτων Αποφάσεων και Διοίκησης. Η ολοκλήρωση της διπλωματικής εργασίας πραγματοποιήθηκε τον Ιούλιο του 2013.

Το αντικείμενο της διπλωματικής εργασίας αφορά στην επίλυση και βελτιστοποίηση προβλημάτων χρονοπρογραμματισμού με τη χρήση αυτομάτων.

Θέλω να ευχαριστήσουμε θερμά τον Αναπληρωτή Καθηγητή και Επιβλέποντα της παρούσας διπλωματικής εργασίας κ. Δ. Ασκούνη για την ευκαιρία που μου έδωσε να εργαστώ σε ένα σύγχρονο αντικείμενο, για τη βοήθεια και καθοδήγησή του κατά τη διάρκεια της εργασίας, όπως επίσης και για τις γνώσεις που μετέδωσε στα πλαίσια των μαθημάτων του.

Θα ήθελα επίσης να ευχαριστήσω τον υπ. Διδάκτορα και αγαπητό φίλο Δημήτρη Πανόπουλο, για την πολύτιμη βοήθειά του και την άψογη συνεργασία μας κατά τη διάρκεια αυτής της προσπάθειας.

Τέλος, τον κ. Βελιβασάκη για τα δεδομένα από το εργοστάσιο της Candia Strom, που συνέβαλαν ώστε η υλοποίηση της διπλωματικής εργασίας να ελεγχθεί σε πραγματικά δεδομένα και καταστάσεις.

Μιχάλης Ασπραδάκης

Ιούλιος 2013

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΔΟΜΗ

Η *οργανωτική δομή* που εφαρμόστηκε για την οργάνωση και την εκπόνηση της παρούσας διπλωματικής εργασίας είναι η εξής:

Στο **1ο Κεφάλαιο** γίνεται αναφορά στο θεωρητικό πλαίσιο που διέπει τη διοίκηση των συστημάτων παραγωγής. Αρχικά λοιπόν παρουσιάζονται βασικές έννοιες της παραγωγής, του παραγωγικού συστήματος, της οργάνωσης και της διοίκησης παραγωγικών συστημάτων, ενώ παρατίθεται και το βασικό πρόβλημα που αντιμετωπίζουν οι επιχειρήσεις. Η ανάγκη για την οργάνωση και τη σωστή διοίκηση των συστημάτων παραγωγής παρουσιάζεται μέσω μια σύντομης αναφοράς στην ιστορική εξέλιξη της διοίκησης παραγωγής. Στη συνέχεια του κεφαλαίου παρατίθενται οι τύποι των παραγωγικών συστημάτων που καταγράφονται έως τώρα, πως λειτουργούν και ποιον στόχο εξυπηρετούν, ενώ παρουσιάζεται και η σχέση των κυριότερων συστημάτων με τον όγκο παραγωγής και την ποικιλία παραγόμενων προϊόντων. Το 1ο Κεφάλαιο συνεχίζεται με την παράθεση των βασικών παραμέτρων της παραγωγικής διαδικασίας, οι οποίες είναι οι προβλέψεις, τα αποθέματα και ο προγραμματισμός των απαιτούμενων υλικών. Ο τελευταίος περιλαμβάνει και την έννοια του χρονικού προγραμματισμού της παραγωγής, που αποτελεί και το αντικείμενο εργασίας. Στην τελευταία υποενότητα του κεφαλαίου δίνεται σαφής εικόνα για την σημασία του χρονοπρογραμματισμού στα παραγωγικά συστήματα, το πρόβλημα που εντοπίζεται στην προσπάθεια του χρονικού προγραμματισμού και τις ενδεχόμενες περιοχές λύσεων του προβλήματος, έπειτα από σωστή επιλογή των κριτηρίων αξιολόγησης. Η πρώτη ενότητα ολοκληρώνεται με τις κυριότερες υποθέσεις του προβλήματος του χρονοπρογραμματισμού.

Το **2ο κεφάλαιο** της διπλωματικής αποτελεί μια εισαγωγή στο πρόβλημα του χρονοπρογραμματισμού σε περιβάλλον εργασίας κατά παραγγελία, δηλαδή σε σύστημα job-shop. Αρχικά δίνεται ο ορισμός του προβλήματος job-shop και καταγράφονται οι κυριότερες μέθοδοι αναπαράστασης του, προς διευκόλυνση της επίλυσης του προβλήματος. Η επόμενη ενότητα του κεφαλαίου αναφέρεται στις κυριότερες μεθόδους βελτιστοποίησης, που χρησιμοποιούνται για την «εξάλειψη» των βασικών περιοριστικών παραγόντων του χρονοπρογραμματισμού παραγωγής. Βασική υποενότητα του 2ου κεφαλαίου αποτελεί η εισαγωγή στην έννοια των χρονισμένων αυτομάτων, αλλά και τον τρόπο που μπορούν να συμπεριληφθούν και να υπολογιστούν στα χρονισμένα αυτόματα οι έννοιες του χρόνου και του κόστους, και οι κατάλληλες μεταβλητές που οι παραπάνω έννοιες συνεπάγονται ώστε να επιτευχθεί ο αρχικός στόχος της εργασίας που αποτελεί και βασικό σκοπό ενός ολοκληρωμένου χρονοπρογραμματισμού παραγωγής. Γίνεται επίσης αναφορά στη Μέθοδο Κρίσιμης Διαδρομής (CPM), που χρησιμοποιήθηκε για να αντιμετωπίσει τη σχέση και την αλληλεπίδραση χρόνου και κόστους.

Το **3ο Κεφάλαιο** αποτελείται από δύο βασικά μέρη. Το πρώτο μέρος παρέχει πληροφορίες για το εργαλείο μοντελοποίησης του προβλήματος, που επιλέχθηκε για τη συγκεκριμένη εφαρμογή. Πρόκειται για το εργαλείο μοντελοποίησης UPPAAL, το οποίο ουσιαστικά είναι ένα ολοκληρωμένο περιβάλλον εργαλείων για τη μοντελοποίηση, την επικύρωση και την επαλήθευση των συστημάτων πραγματικού χρόνου που μοντελοποιούνται ως δίκτυα

χρονισμένων αυτομάτων, εμπλουτισμένα με τύπους δεδομένων. Καταγράφονται τα βασικά εργαλεία που συνιστούν το περιβάλλον εργασίας του προγράμματος, καθώς και πως μπορεί να εφαρμοστεί στα χρονισμένα αυτόματα. Οι εκφράσεις που χρησιμοποιούνται, και τα βασικά μέρη όπως τα ρολόγια, οι μεταβάσεις, οι θέσεις αλλά και οι μέθοδοι επαλήθευσης που παρέχονται από το ίδιο το εργαλείο, συνιστούν το θεωρητικό υπόβαθρο για το επόμενο μέρος του κεφαλαίου, το οποίο αποτελεί και τη μοντελοποίηση του προβλήματος. Σε αυτό το μέρος, σχεδιάζονται τα δομικά στοιχεία του μοντέλου που χρησιμοποιείται και εκτελείται δοκιμή ορθής λειτουργίας.

Το **4ο Κεφάλαιο** περιέχει δύο αντιπροσωπευτικά πειράματα που χρησιμοποιείται το μοντέλο του 3ου Κεφαλαίου. Το πρώτο πείραμα χρησιμοποιεί τυχαίες τιμές, ενώ το δεύτερο πείραμα στηρίζεται σε πραγματικά δεδομένα από το εργοστάσιο παραγωγής και εμπορίας στρωμάτων της Candia Strom.

Τέλος στο **5ο Κεφάλαιο** υπάρχουν τα συμπεράσματα από τη διπλωματική εργασία, όπως επίσης προτάσεις και προοπτικές.

Κεφάλαιο 1

Παραγωγικά Συστήματα & Χρονοπρογραμματισμός

1.1 Εισαγωγή

Η οικονομική αστάθεια των τελευταίων ετών τόσο σε επίπεδο χώρας όσο και σε παγκόσμιο άλλαξε τα μέχρι τώρα δεδομένα στη διοίκηση και λειτουργία των βιομηχανικών μονάδων, που πλέον αντιμετωπίζουν προβλήματα ανταγωνιστικότητας, παραγωγής και αξιοπιστίας στον τομέα των εργασιακών και πελατειακών σχέσεων. Καθίσταται πλέον επιτακτική ανάγκη οι εταιρίες και οι μονάδες παραγωγής να προσαρμοστούν στις τεχνολογικές εξελίξεις και στις νέες συνθήκες, ώστε να αποκτήσουν το συγκριτικό πλεονέκτημα επιβίωσης σε μια δύσκολη αγορά.

Η μετάβαση αυτή στην παραγωγική διαδικασία διεκπεραιώνεται μέσω του χρονοπρογραμματισμού παραγωγής στα πλαίσια της διοίκησης παραγωγικών συστημάτων. Παρά το γεγονός ότι τα αντικείμενα παραγωγής είναι διαφορετικά για τις διάφορες παραγωγικές βιομηχανίες, τα βασικά ιδιαίτερα χαρακτηριστικά του παραγωγικού προγράμματος είναι όμοια και απαιτούν την εύρεση ενός βέλτιστου προγράμματος για την κατανομή των πόρων με ένα συγκεκριμένο παραγωγικό στόχο, όπως ο συντομότερος χρόνος παραγωγής, το ελάχιστο παραγωγικό κόστος, η έγκαιρη διεκπεραίωση της παραγωγής κλπ. και ορισμένους σχετικούς περιοριστικούς παράγοντες, όπως είναι ο απαιτούμενος χρόνος επεξεργασίας για κάθε εργασία, η προτεραιότητα του κάθε προϊόντος, η εφαρμογή και η εκμετάλλευση του εξοπλισμού, ο ρυθμός παραγωγής, οι ομάδες παραγωγής, οι ημερομηνίες παράδοσης των προϊόντων κλπ. Ο εκάστοτε στόχος και οι περιοριστικοί παράγοντες εξαρτώνται από τα χαρακτηριστικά της κάθε επιχείρησης και παραγωγικής διαδικασίας.

Το ουσιαστικό πρόβλημα που έχει προκύψει από την εισχώρηση των πληροφοριακών συστημάτων στη διοίκηση της παραγωγικής διαδικασίας είναι ότι παρά το γεγονός ότι έχουν επιλυθεί προβλήματα εργασιακής έρευνας, αποδοτικότητας και προσαρμογής στα νέα δεδομένα, αδυνατούν να προσαρμοστούν στις συνεχόμενες αλλαγές και στις αυξανόμενες απαιτήσεις της σύγχρονης παραγωγικής διαδικασίας. Η λύση για την εξοικονόμηση χρόνου, αλλά και τη μείωση του κόστους παραγωγής δίδεται μέσω ενός ολοκληρωμένου συστήματος επιλογής που θα χρησιμοποιεί τον πλέον κατάλληλο αλγόριθμο, τόσο για την επίλυση του προβλήματος παραγωγής, όσο και για την εξαγωγή του αποτελέσματος χρονοπρογραμματισμού.

1.2 Διοίκηση Παραγωγικών Συστημάτων

1.2.1 Βασικές έννοιες

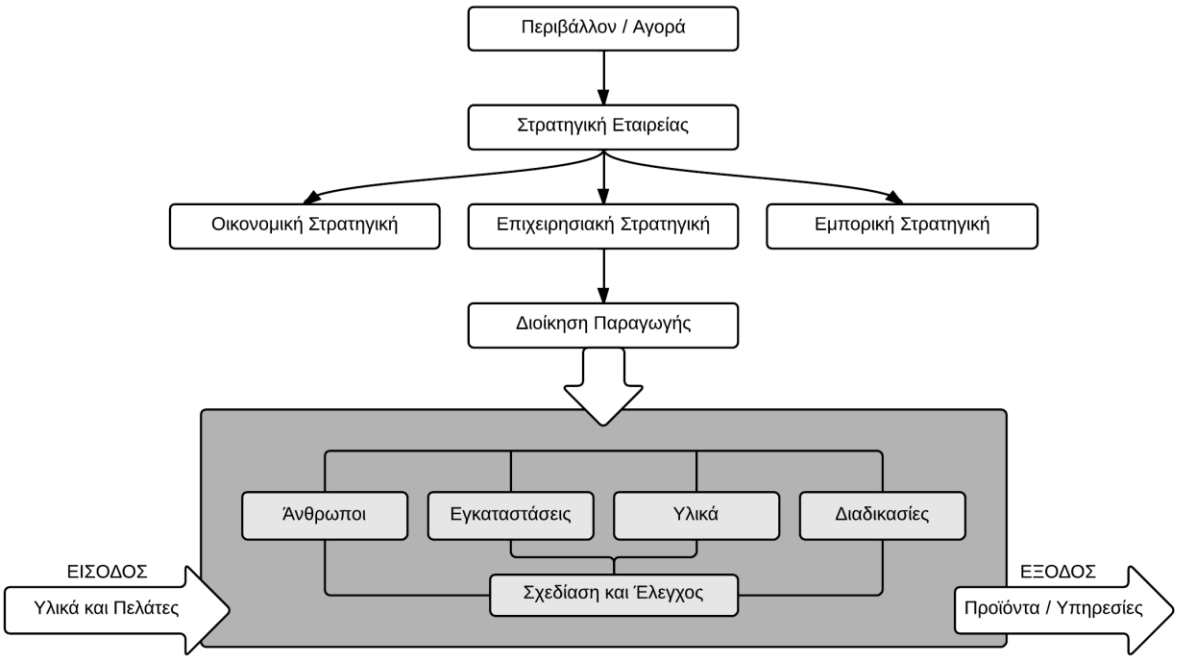
● **Παραγωγή:** Παραγωγή είναι κάθε οργανωμένη δραστηριότητα που αποσκοπεί στην αύξηση της αξίας ή της χρησιμότητας υλικών πραγμάτων ή στην παροχή υπηρεσιών, με την ανάλωση κάποιων πόρων (υλικών, εργασίας κλπ.)

● **Παραγωγικό σύστημα:** Παραγωγικό σύστημα είναι κάθε σύστημα, δηλαδή κάθε οργανωμένο σύνολο στοιχείων που παράγει προϊόντα ή υπηρεσίες.

● **Οργάνωση & Διοίκηση Παραγωγής:** Ως Οργάνωση & Διοίκηση Παραγωγής (ΟΔΠ) ορίζουμε τον σχεδιασμό, προγραμματισμό, λειτουργία και βελτίωση της παραγωγικής διαδικασίας με την οποία κάποιοι πόροι μετατρέπονται σε προϊόντα ή υπηρεσίες.

[Εργαστήριο Συστημάτων Αποφάσεων και Διοίκησης http://academics.epu.ntua.gr]

● **Διοίκηση Παραγωγικών Συστημάτων:** Η διοίκηση παραγωγικών συστημάτων ασχολείται κυρίως με την εξασφάλιση της παραγωγής των προϊόντων ή υπηρεσιών στις ποσότητες που απαιτούνται, σύμφωνα με ορισμένες ποιοτικές προδιαγραφές, σε συγκεκριμένες προθεσμίες και με το μικρότερο δυνατό κόστος, λαμβάνοντας υπόψη τους περιορισμούς και τις απαγορεύσεις που προέρχονται από το περιβάλλον. Οι αποφάσεις που λαμβάνονται στα πλαίσια της οργάνωσης ενός παραγωγικού συστήματος διακρίνονται σε στρατηγικές τακτικές και λειτουργικές. Οι πρώτες αφορούν θέματα σχεδιασμού παραγωγικών συστημάτων και έχουν μακροπρόθεσμες επιπτώσεις στο σύστημα, ενώ οι τακτικές και λειτουργικές αφορούν στην οργάνωση και τον έλεγχο της λειτουργίας του, με τις τακτικές να έχουν βραχυπρόθεσμες επιπτώσεις και τις λειτουργικές άμεσες σε επίπεδο καθημερινής λειτουργίας. Απαραίτητη προϋπόθεση για ένα επιτυχημένο σύστημα διοίκησης είναι η ικανοποίηση των γενικών στόχων της επιχείρησης/οργανισμού και των ειδικών στόχων του παραγωγικού συστήματος. Οι στόχοι αυτοί σχετίζονται με την βέλτιστη χρήση των παραγωγικών πόρων, γεγονός που σημαίνει ελαχιστοποίηση του κόστους και βελτιστοποίηση της ποιότητας των προϊόντων με ταυτόχρονη τήρηση των προθεσμιών παράδοσής τους. Η διαδικασία που ακολουθείται στη διοίκηση ενός συστήματος παραγωγής απεικονίζεται στο Σχήμα 1.1.



Σχήμα 1.1

1.3 Ιστορική Εξέλιξη

Η ανάγκη για τη διοίκηση παραγωγικών συστημάτων ήταν αποτέλεσμα της εξέλιξης των ανθρώπινων κοινωνιών και του πολιτισμού. Αρχικά, η παραγωγή αναπτύχθηκε σε μικρά εργαστήρια και οικοτεχνίες, κυρίως για να εξυπηρετήσει τις ανάγκες ιδιοκατανάλωσης και σπανιότερα το ανταλλακτικό εμπόριο. Κατά την περίοδο της Αναγέννησης διαμορφώνεται η βάση για τη μεγάλη Βιομηχανική Επανάσταση του 18^{ου} αιώνα με την πρόοδο που σημειώθηκε στις φυσικές επιστήμες και την τεχνολογία. Η οικογενειακή επιχείρηση αντικαθίσταται από το εργοστάσιο και επιτυγχάνεται η μαζική παραγωγή προϊόντων που δίνει νέα ώθηση στο εμπόριο. Η δημιουργία εργοστασίων γεννά προβλήματα οργάνωσης και διεύθυνσης της παραγωγής. Μέσα στο εργοστάσιο δημιουργούνται πλέον ιεραρχικές σχέσεις ελέγχου, αποδίδοντας αρμοδιότητες προγραμματισμού και ελέγχου της παραγωγής σε εξειδικευμένο προσωπικό, ενώ ο απλός εργαζόμενος έχει μόνο εκτελεστικό ρόλο. Έτσι διαμορφώνονται στο τέλος του 19^{ου} αιώνα νέες αρχές οργάνωσης παραγωγής και διατυπώνονται από τον Taylor το 1895, οι οποίες στηρίζονται στην αντικατάσταση του τεχνίτη-εργάτη από τον ανειδίκευτο εργάτη, που ο ρόλος του στην παραγωγική διαδικασία περιορίζεται στην επαναληπτική εκτέλεση κάποιων στοιχειωδών κινήσεων στις οποίες αναλύεται η διαδικασία αυτή και έτσι τελικά επιτυγχάνεται η φθηνή εργασία, σύμφωνα με τις αρχές της Επιστημονικής Διοίκησης.

Το 1912 επιτυγχάνεται από τον Ford η πρώτη Αλυσίδα Παραγωγής στο εργοστάσιο, εξοικονομώντας εργατικό δυναμικό, όσον αφορά στη μεταφορά υλικού και αφιερώνεται το σύνολο του χρόνου στην ίδια την παραγωγή, εισάγεται ο κοινός ρυθμός εργασίας των

ανθρώπων στη γραμμή παραγωγής και προωθείται η συστηματική χρήση μηχανών που όπου είναι εφικτό αντικαθιστούν τους εργάτες και εξασφαλίζουν υψηλότερη παραγωγικότητα.

Στα μέσα του 20ού αιώνα νέες τεχνολογίες αναπτύσσονται και εντάσσονται στην παραγωγή. Πιο συγκεκριμένα εισάγεται η ρομποτική, συστήματα CAD/CAM, έμπειρα συστήματα, συστήματα CIM όπου όλες οι λειτουργίες της παραγωγής υποστηρίζονται από μία βάση δεδομένων που αφορούν τη σχεδίαση των προϊόντων και τη βιομηχανοποίηση, καθώς και από άλλα προηγμένα συστήματα. Αρχίζει η αυτοματοποίηση όλων των παραγωγικών λειτουργιών και η μείωση της ανθρώπινης φθοράς που μετατρέπεται πλέον σε διανοητική και νευρική καταπόνηση.

1.4 Τύποι Παραγωγικών Συστημάτων

Η κατηγοριοποίηση των συστημάτων παραγωγής γίνεται βάσει τον αριθμό των μηχανών, τη ροή των υλικών και των προϊόντων, τη διάταξη των τμημάτων που απαρτίζουν τις παραγωγικές μονάδες, το επίπεδο αυτοματοποίησης, καθώς και την ευελιξία του συστήματος. Έτσι γίνεται η παρακάτω διάκριση:

- **Single Machine Shop**: Το πρόβλημα του προγραμματισμού μιας μηχανής ή ενός πόρου, είναι η διαδικασία της ανάθεσης μιας ομάδας καθηκόντων σε ένα μόνο μηχάνημα ή έναν πόρο. Τα καθήκοντα διατάσσονται έτσι ώστε ένα ή πολλά αποδοτικά μέτρα να μπορούν να βελτιστοποιηθούν. Το πρόβλημα της μιας μηχανής σπάνια συνίσταται πλέον. [Jeffrey W. Herrmann, 2007]
- **Parallel Machine Shop:** Σε αυτή την περίπτωση μερικές διεργασίες πρέπει να υποβληθούν σε επεξεργασία από δυο παράλληλα συνδεδεμένες μηχανές, ταυτόχρονα [Lin S-F. , Chen J-Y, 2002]. Οι μηχανές αυτές μπορεί να είναι:

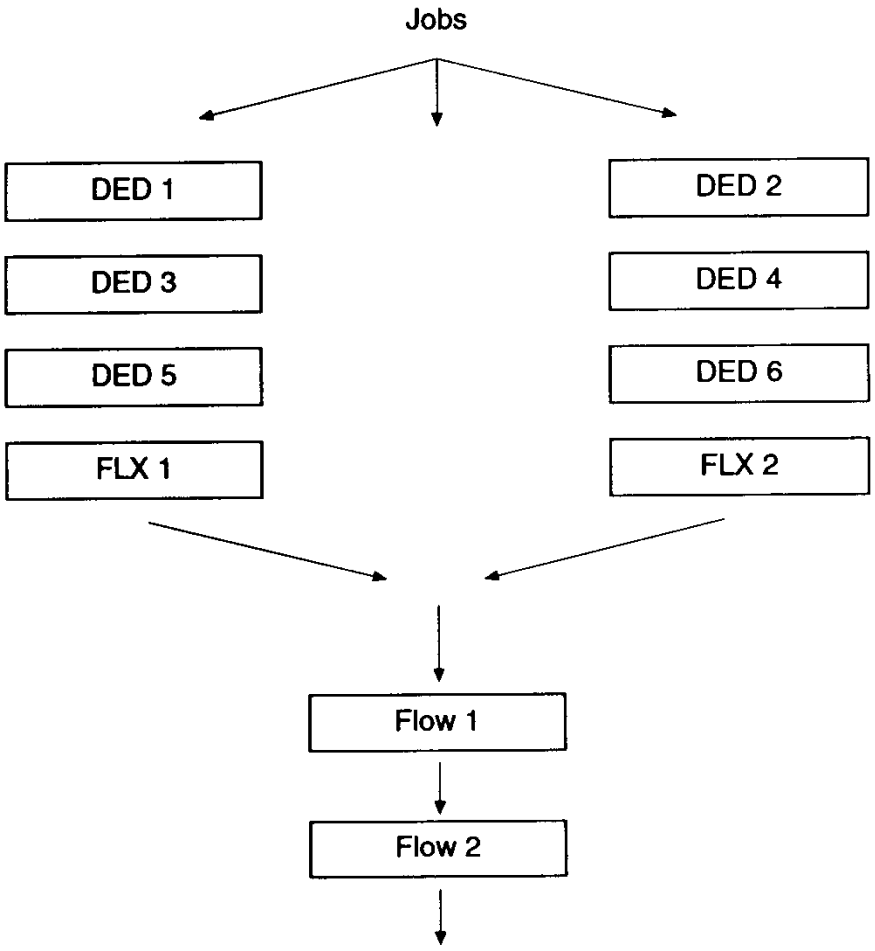
    - **<u>Ίδιες μηχανές, συνδεδεμένες παράλληλα.</u>** Στη συγκεκριμένη περίπτωση όταν απαιτείται η επεξεργασία μιας εργασίας, αυτό μπορεί να γίνει σε όποια μηχανή είναι διαθέσιμη. Το πρόβλημα σε αυτήν τη περίπτωση μπορεί να περιγραφεί ως εξής: Υπάρχουν n εργασίες J_i (i = 1, . . ., n) με χρόνους επεξεργασίας p_i (i = 1, . . ., n) που πρέπει να υποστούν επεξεργασία σε m ίδιες μηχανές M_1, . . ., M_m που είναι συνδεδεμένες παράλληλα. Στόχος του χρονοπρογραμματισμού είναι η ελαχιστοποίηση μιας αντικειμενικής συνάρτησης.

    - **<u>Μηχανές με διαφορετική ταχύτητα, συνδεδεμένες παράλληλα (ομοιόμορφες μηχανές).</u>** Σε αυτή τη περίπτωση οι μηχανές που συνδέονται παράλληλα έχουν διαφορετικές ταχύτητες. Αυτό μπορεί π.χ. να οφείλεται στο ότι κάποια μηχανή είναι παλαιότερης τεχνολογίας (και συνεπώς βραδύτερη) από μια άλλη. Αν οι μηχανές έχουν τις ίδιες ταχύτητες τότε αυτό το περιβάλλον είναι όμοιο με το προηγούμενο. Το εξεταζόμενο πρόβλημα διατυπώνεται ως εξής: Θεωρούμε n εργασίες, J_i (i = 1, . . ., n) που πρέπει να υποστούν επεξεργασία σε m παράλληλες μηχανές M_j (j = 1, . . ., m). Οι μηχανές έχουν διαφορετικές ταχύτητες s_j (j = 1, . . ., m). Κάθε εργασία J_i έχει μια συγκεκριμένη απαίτηση επεξεργασίας p_i (i = 1, . . ., n). Η εκτέλεση μιας εργασίας J_i σε μια μηχανή M_j

απαιτεί p_i / s_j μονάδες χρόνου. Αν τεθεί s_j = 1 για j = 1, . . ., m προκύπτουν m παράλληλες όμοιες μηχανές.

- **<u>Ασυσχέτιστες μηχανές, συνδεδεμένες παράλληλα</u>**. Το περιβάλλον αυτό είναι μια γενίκευση του προηγούμενου. Υπάρχουν διαφορετικές μηχανές συνδεδεμένες παράλληλα. Μια μηχανή μπορεί να επεξεργαστεί μια εργασία με μια συγκεκριμένη ταχύτητα. Αν οι ταχύτητες των μηχανών είναι **ανεξάρτητες** των εργασιών, τότε αυτό το περιβάλλον είναι όμοιο με το προηγούμενο. Το προς εξέταση πρόβλημα ορίζεται ακολούθως: Έστω n ανεξάρτητες εργασίες i = 1, . . ., n που πρέπει να υποστούν επεξεργασία σε m μηχανές. Ο χρόνος επεξεργασίας κάθε εργασίας i στην μηχανή M_j είναι p_{ij} (i = 1, . . ., n; j = 1, . . ., m). Αυτό το μοντέλο είναι μια γενίκευση του μοντέλου των ομοιόμορφων μηχανών, αν τεθεί p_{ij} = p_i / s_j. Και σε αυτή τη περίπτωση στόχος είναι η ελαχιστοποίηση μιας αντικειμενικής συνάρτησης [Γεωργόπουλος, 2004].

- **Flow Shop:** Σε αυτά τα συστήματα, η παραγωγή εξειδικεύεται σε ένα περιορισμένο αριθμό τυποποιημένων προϊόντων που παράγονται σε αντίστοιχες γραμμές παραγωγής και προορίζονται για ευρεία κατανάλωση. Οι διεργασίες περνούν από συγκεκριμένα στάδια επεξεργασίας, τα οποία μπορεί να είναι διακριτά, όπως σε μια εταιρεία επεξεργασίας τροφίμων, ή συνεχή, όπως σε ένα διυλιστήριο. Η πορεία των διεργασιών μέσα στο σύστημα είναι η ίδια για κάθε προϊόν, ακολουθώντας μια νοητή ευθεία γραμμή. Σε ορισμένα συστήματα του τύπου flow shop, αν μια διεργασία δεν χρειάζεται επεξεργασία σε μια συγκεκριμένη μηχανή, μπορεί να την παρακάμψει. Τα συστήματα αυτά είναι γνωστά ως non–permutation ή general flow shop και θεωρούνται τα πιο αντιπροσωπευτικά flow shop συστήματα, γιατί αποτελούν μια καλή αναπαράσταση των πρακτικών προβλημάτων. Άλλα συστήματα flow shop δεν επιτρέπουν αυτή τη παράκαμψη, λειτουργώντας με τον κανόνα FIFO (First-In-First-Out) και ονομάζονται permutation flow shop. Μια γενίκευση των συστημάτων είναι τα flexible ή compound ή hybrid flow shops, τα οποία συνίστανται σε ένα αριθμό σταδίων σε σειρά με ένα αριθμό μηχανών συνδεδεμένων παράλληλα σε κάθε στάδιο (συνδυασμός μηχανών συνδεδεμένων παράλληλα και flow shop). Οι διεργασίες υφίστανται επεξεργασία σε κάθε στάδιο σε οποιαδήποτε από τις παράλληλες μηχανές. Οι ουρές μεταξύ των διάφορων σταδίων συνήθως λειτουργούν με βάση τον κανόνα FIFO. Ο παραπάνω τύπος συστήματος συναντάται συνήθως στις βιομηχανίες καλλυντικών, τροφίμων και στις υφαντουργίες. Στα συστήματα diverging flexible flow shop, κάθε στάδιο έχει τουλάχιστον όσες μηχανές είχε και το προηγούμενό του.



Σχήμα 1.2: Γραμμή παραγωγής flow shop [http://www.emeraldinsight.com]

- **Job Shop:** Τα συστήματα job shop είναι συνήθως μικρά συστήματα παραγωγής που χειρίζονται παραγωγικές διεργασίες, ειδικά επί παραγγελία όπως μικρού έως μεσαίου μεγέθους παραγγελίες των πελατών ή εργασίες που αφορούν παραγωγή σε παρτίδες. Οι προδιαγραφές της κάθε παραγγελίας τίθενται από τον πελάτη, και εφόσον είναι διαφορετικές ανά παραγγελία, η γραμμή παραγωγής είναι επίσης διαφορετική για κάθε μια, ανεξάρτητα από το αν θα χρησιμοποιηθεί ένα συγκεκριμένο πλήθος παραγωγικών μονάδων. Οι μηχανές κατηγοριοποιούνται στην παραγωγή ανάλογα με τη φύση των δεξιοτήτων και τις τεχνολογικές διαδικασίες που συμπεριλαμβάνονται, εξασφαλίζοντας έτσι ευελιξία στο σύστημα παραγωγής εφόσον οι εργασίες μπορούν να πραγματοποιηθούν σε περισσότερες από μια μηχανές.

  Υπάρχουν διάφορες κατηγορίες συστημάτων job shop. Σε ένα τυπικό σύστημα παραγωγής τύπου classic ή closed job shop, κάθε παραγγελία είναι μοναδική και έχει ένα μοναδικό δρομολόγιο. Οι διάφορες διεργασίες εκτελούνται με τη σειρά σαν μια μεγάλη παρτίδα από διάφορα τμήματα που προωθούνται μαζί διαμέσου του συστήματος παραγωγής.

Ένα σύστημα παραγωγής που παράγει προϊόντα τα οποία μπορούν να αποθηκευτούν ώστε να εξυπηρετήσουν μια μελλοντική ζήτηση και δεν απευθύνονται αποκλειστικά σε συγκεκριμένες παραγγελίες, είναι γνωστό ως open shop. Μπορεί δηλαδή να υπάρχουν πολλοί πελάτες που ζητούν τα ίδια (ή σχεδόν τα ίδια) προϊόντα και έτσι έχει νόημα η αποθήκευση τελικών προϊόντων ή η εκτροπή δραστηριοτήτων που ήταν προορισμένες για ένα πελάτη, προς χάρη ενός άλλου πελάτη μεγαλύτερης προτεραιότητας (multi–use parts).

Μια άλλη κατηγορία είναι τα flexible job shops (job shop με ίδιες μηχανές), τα οποία αποτελούν μια γενίκευση των απλών job shops. Εδώ, κάθε παραγωγική μονάδα αντικαθίσταται από έναν αριθμό μηχανών συνδεδεμένων παράλληλα. Όταν μια διεργασία, στη διάρκεια του δρομολογίου της, φτάσει σε αυτή τη παραγωγική μονάδα με τις παράλληλες μηχανές, μπορεί να υποστεί επεξεργασία σε οποιαδήποτε από τις μηχανές. Το παραπάνω περιβάλλον είναι πολύ συνηθισμένο στη βιομηχανία ημιαγωγών.

Τέλος, σε μια πιο πολύπλοκη θεώρηση του συστήματος, μια διεργασία μπορεί να “επισκεφτεί” μια συγκεκριμένη μηχανή αρκετές φορές κατά τη διάρκεια του δρομολογίου της. Τότε λέμε ότι αυτά τα συστήματα υπόκεινται σε επανακυκλοφορία (recirculation). Τα πλεονεκτήματα του συγκεκριμένου συστήματος παραγωγής είναι:

- Υψηλή ευελιξία παραγωγής
- Μεγάλη ευελιξία στη μηχανική των προϊόντων
- Υψηλή ευελιξία επέκτασης (μηχανήματα εύκολα να προστεθούν ή υποκατασταθούν)
- Υψηλή ελαστικότητα όγκου παραγωγής
- Χαμηλή «αχρηστία»
- Υψηλή αντοχή σε μηχανικά λάθη

Τα μειονεκτήματα από την άλλη μεριά είναι ο πολύ δύσκολος προγραμματισμός της παραγωγής και η μικρή εκμετάλλευση της παραγωγικής ικανότητας. [Καρόπουλος 2005/V. Vinoda, R. Sridharan, 2011]

- **Assembly Systems:** Ένα τέτοιο σύστημα αποτελεί το επιστέγασμα για την υλοποίηση του προϊόντος, όπου τα συστατικά μέρη και υποσυγκροτήματα ενσωματώνονται μαζί για να σχηματίσουν τα τελικά προϊόντα. Καθώς η ποικιλία των προϊόντων αυξάνεται λόγω της μετάβασης από τη μαζική παραγωγή στη μαζική προσαρμογή, τα συστήματα συναρμολόγησης πρέπει να σχεδιάζονται κατάλληλα ώστε να μπορούν να χειριστούν τέτοια υψηλή ποικιλία. Εδώ υπάρχει ένας περιορισμένος αριθμός διαφορετικών τύπων προϊόντων και το σύστημα πρέπει να παράγει μια δεδομένη ποσότητα από κάθε τύπο προϊόντος. Η κίνηση των διεργασιών σε ένα τέτοιο σύστημα ελέγχεται συχνά από ένα σύστημα ελέγχου, το οποίο θέτει περιορισμούς στους χρόνους έναρξης των διεργασιών στις διάφορες μηχανές. [S. J. Hu, et al, 2011]

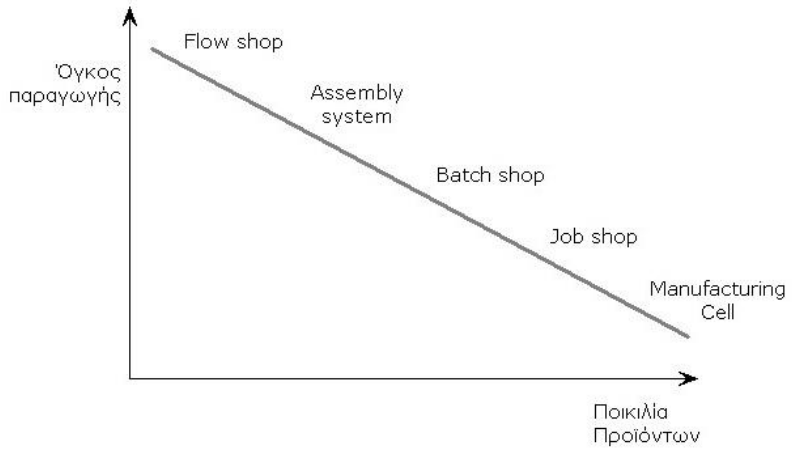
- **Batch Shop Systems:** Είναι το σύστημα παραγωγής κατά το οποίο επιτυγχάνεται η παραγωγή τμημάτων των προϊόντων σε συγκεκριμένες παρτίδες εξασφαλίζοντας έτσι μεγάλες οικονομίες κλίμακας για παραγωγή μεγάλων ποσοτήτων πανομοιότυπων προϊόντων.

- **Manufacturing Cells:** Είναι σύνολα μηχανημάτων που ομαδοποιούνται από τα προϊόντα ή εξαρτήματα που παράγουν, σε ένα «ισχνό» περιβάλλον παραγωγής. Το σύστημα αυτό χρησιμοποιείται στην κυτταρική έννοια κατασκευής, η οποία είναι διακριτή από το παραδοσιακό λειτουργικό σύστημα παραγωγής, στο οποίο όλες οι παρόμοιες μηχανές ομαδοποιούνται. Η χρήση των κυττάρων παραγωγής βελτιώνει τη ροή του υλικού και είναι ιδιαίτερα κατάλληλο για την παραγωγή των παρτίδων, ακόμα και σε σχετικά χαμηλές ποσότητες. Μια από τις προκλήσεις της εφαρμογής σε ένα κυψελοειδές σύστημα παραγωγής είναι η πραγματική δημιουργία των κυττάρων παραγωγής. Αν τα ίδια μηχανήματα απαιτούνται σε διαφορετικά κελιά, τότε κάτι τέτοιο μπορεί να οδηγήσει σε υψηλότερες κεφαλαιακές απαιτήσεις. Ωστόσο, τα οφέλη των κυττάρων παραγωγής, όπως η αύξηση της παραγωγικότητας, καλύτερη ανταπόκριση στις συνθήκες της αγοράς και η ικανότητα παραγωγής εξατομικευμένων προϊόντων σε μικρές ποσότητες, αντισταθμίζει τα παραπάνω μειονεκτήματα.

- **Multiprocessor Task System:** Σε ένα σύστημα multiprocessor task οι διεργασίες απαιτούν επεξεργασία από μια ή περισσότερες μηχανές ταυτόχρονα.

- **Multipurpose Machine Shop:** Σε ένα τέτοιο σύστημα οι εργασίες μπορούν να πραγματοποιηθούν σε οποιαδήποτε μηχανή των προκαθορισμένων υποομάδων του συνόλου των μηχανών. Σε αυτή τη περίπτωση έχουμε ένα αριθμό πολυχρηστικών (multipurpose) μηχανών, δηλαδή μηχανών οι οποίες μπορεί να είναι εξοπλισμένες με διαφορετικά εργαλεία και είναι ικανές να διεκπεραιώσουν διαφόρων ειδών εργασίες. Μια μηχανή μπορεί να επεξεργαστεί μια διεργασία, μόνο αν είναι εξοπλισμένη με τα κατάλληλα εργαλεία.

- **JIT (Just–In–Time) Systems:** Η πολιτική των συστημάτων αυτών είναι τα τελικά προϊόντα να παράγονται ακριβώς τη στιγμή που πρέπει να παραχθούν και διέπει όλο το σύστημα από την στιγμή που αγοράζονται οι πρώτες ύλες μέχρι την στιγμή που το προϊόν φτάνει στο τελικό στάδιο επεξεργασίας. Στα συστήματα αυτά στόχος είναι η μείωση στα κόστη αποθήκευσης των προϊόντων (ενδιάμεσων και τελικών), η υψηλότερη ποιότητα των τελικών προϊόντων, η αυξημένη ικανότητα προσαρμογής στις απαιτήσεις των πελατών και τελικά η ελαχιστοποίηση του συνολικού κόστους κατασκευής. Απαραίτητη προϋπόθεση σε αυτή την περίπτωση είναι οι παραγωγοί να γνωρίζουν με ακρίβεια τη ζήτηση του προϊόντος στην αγορά [Καρόπουλος 2005]. Ένα καλό παράδειγμα θα ήταν ένας κατασκευαστής αυτοκινήτων που λειτουργεί με πολύ χαμηλά επίπεδα αποθεμάτων, που θα στηρίζονται στην αλυσίδα εφοδιασμού τους για να παραδώσει τα εξαρτήματα που χρειάζονται για την κατασκευή αυτοκινήτων. Τα εξαρτήματα που απαιτούνται για την κατασκευή των αυτοκινήτων δεν φθάνουν πριν ούτε μετά τους χρειάζονται, αλλά φτάνουν ακριβώς όπως απαιτούνται.



Σχήμα 1.3: Σχέση μεταξύ των κυριότερων συστημάτων παραγωγής, του όγκου παραγωγής και της ποικιλίας των προϊόντων που μπορούν να παραχθούν [Καρόπουλος 2005]

1.5 Παράμετροι Συστημάτων Παραγωγής

1.5.1 Προβλέψεις

Οι προβλέψεις και κατά συνέπεια οι μέθοδοι προβλέψεων θεωρούνται σαφέστατα αναγκαίο και αναπόσπαστο κομμάτι το συστημάτων παραγωγής και αυτό γιατί τα μεγέθη που αφορούν τη ζήτηση και την κατανάλωση των προϊόντων, τα απαιτούμενα αποθέματα υλικών, κεφαλαίων, ανθρωπίνου δυναμικού, καθώς και τον απαιτούμενο μηχανολογικό εξοπλισμό, δεν μπορούν να αποδοθούν με ακρίβεια. Ο προγραμματισμός και ο έλεγχος της παραγωγής, ειδικότερα, απαιτούν εκτιμήσεις όσον αφορά την ποσότητα και το χρόνο που αναμένεται να ζητηθεί το προϊόν ενός παραγωγικού συστήματος. Οι εκτιμήσεις αυτές θα χρησιμοποιηθούν για την κατάρτιση των προγραμμάτων παραγωγής, προμήθειας πρώτων υλών, απασχόλησης ανθρώπινου δυναμικού κλπ. Τα προγράμματα αυτά θα είναι τόσο περισσότερο αποτελεσματικά, σε σχέση με το σκοπό του παραγωγικού συστήματος, όσο περισσότερο αξιόπιστες είναι οι σχετικές προβλέψεις.

Πολλές είναι οι μέθοδοι που έχουν αναπτυχθεί για τη διενέργεια προβλέψεων και χρησιμοποιούνται για τη λήψη αποφάσεων σε ποικίλες συνθήκες. Η επιλογή της κατάλληλης κάθε φορά μεθόδου, η εγκατάσταση και χρήση της και η ερμηνεία των αποτελεσμάτων της είναι μερικά από τα προβλήματα που αντιμετωπίζονται στην πρακτική αξιοποίηση των μεθόδων αυτών. Μερικές ενδεικτικές μέθοδοι προβλέψεων στα συστήματα παραγωγής είναι:

- Η μέθοδος παλινδρόμησης
- Ο κινούμενος μέσος όρος
- Η εκθετική εξομάλυνση
- Η προσαρμοστική εξομάλυνση
- Η πρόγνωση της κατανομής πιθανότητας

- Η εκτίμηση Bayes
- Μοντέλα με εξαρτημένες παρατηρήσεις

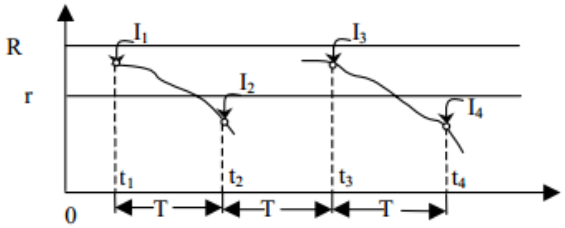
Γενικά, ο βασικός παράγοντας που καθορίζει την επιλογή της μεθόδου προβλέψεων είναι το είδος των αποφάσεων που θα ληφθούν βάσει των προβλέψεων που θα προκύψουν. Εκτός από τον παράγοντα αυτό, η επιλογή της κατάλληλης μεθόδου καθορίζεται από ένα σύνολο ειδικότερων παραγόντων, στους οποίους περιλαμβάνονται η ζητούμενη μορφή της πρόβλεψης, η περίοδος και ο ορίζοντας πρόβλεψης, το κόστος της μεθόδου, η επιζητούμενη ακρίβεια, η απλότητα και η ευκολία εφαρμογής και τα διαθέσιμα στοιχεία. Η σημασία των προβλέψεων στην πράξη είναι μεγάλη, αφού κάθε σπουδαία απόφαση στις επιχειρήσεις, στρατηγικού ή τακτικού χαρακτήρα, βασίζεται σε μεγάλο βαθμό σε αυτές. Έχει σημασία, λοιπόν, η επιλογή της σωστής μεθόδου (ή συνδυασμού μεθόδων) πρόβλεψης.

1. 5. 2 Αποθέματα

Απόθεμα παραγωγής είναι το σύνολο των υλικών που βρίσκονται στο στάδιο της επεξεργασίας ή περιμένουν επεξεργασία. Η δημιουργία των αποθεμάτων μπορεί να είναι είτε σχεδιασμένη, είτε να αποτέλεσμα διαφόρων παραγόντων (κακός προγραμματισμός, ύπαρξη σημείου μποτιλιαρίσματος). Αποθέματα μπορούν να δημιουργούνται και στην περίπτωση συστημάτων που παράγουν υπηρεσίες, όταν διατίθεται κατάλληλη τεχνολογία, όπως στην περίπτωση ενός προγράμματος λογισμικού που αποθηκεύεται σε cd.

Η διαχείριση των αποθεμάτων πρώτων υλών, ενδιάμεσων και τελικών προϊόντων, αποτελεί σημαντική λειτουργία σε ένα παραγωγικό σύστημα για πολλούς λόγους: από τη μια μεριά, τα αποθέματα δεσμεύουν ένα μεγάλο ποσοστό του κεφαλαίου κίνησης των επιχειρήσεων. Επίσης δεσμεύουν ένα σημαντικό μέρος του διατιθέμενου χώρου σε μία επιχείρηση, ενώ η προμήθεια, η φύλαξη, η συντήρηση, η ασφάλιση και, γενικά, η διαχείριση των αποθεμάτων κοστίζουν. Από την άλλη μεριά, με τη διατήρηση αποθεμάτων τελικών προϊόντων μπορούμε να αποσυνδέσουμε το παραγωγικό σύστημα από τις διακυμάνσεις της ζήτησης, αφού μία αύξηση της ζήτησης σε κάποια περίοδο θα μπορεί να αντιμετωπιστεί με υπάρχοντα αποθέματα, χωρίς δηλαδή αντίστοιχη αύξηση της παραγωγής κατά την περίοδο αυτή. Η ύπαρξη επαρκών πρώτων υλών και ενδιάμεσων αποθεμάτων, άλλωστε, εξασφαλίζει τη συνεχή, χωρίς διακοπές, τροφοδότηση του παραγωγικού συστήματος, τη διατήρηση της ομαλής ροής της παραγωγής, την ανεξάρτητη λειτουργία μεταξύ των παραγωγικών σταδίων, την αύξηση του ρυθμού παραγωγής και την ελάττωση του βιομηχανικού κόστους. Γενικά, το πρόβλημα της διαχείρισης των αποθεμάτων μπορεί να οριστεί ως πρόβλημα εξισορρόπησης, συνήθως μέσα σε συνθήκες αβεβαιότητας, μεταξύ του κόστους έλλειψης και του κόστους πλεονάσματος αποθεμάτων πρώτων υλών, ενδιάμεσων και τελικών προϊόντων ενός παραγωγικού συστήματος. Εξαιτίας της σημασίας της λειτουργίας της διαχείρισης αποθεμάτων, στα πλαίσια της Επιχειρησιακής Έρευνας έχει αναπτυχθεί η Θεωρία Αποθεμάτων, που εξετάζει συστηματικά τα σχετικά προβλήματα που συνδέονται με τη δημιουργία και τη διαχείριση των αποθεμάτων, ενώ διατίθενται σχετικά συστήματα λογισμικού που βοηθούν στη συστηματική παρακολούθηση και τον έλεγχο των αποθεμάτων. Οι στρατηγικές που ακολουθούνται για τον έλεγχο των αποθεμάτων είναι:

- Στρατηγική περιοδικού ελέγχου



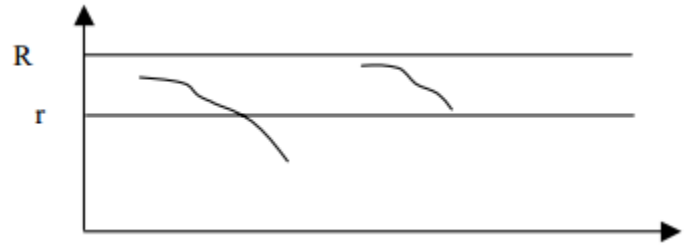
  - T: περίοδος ελέγχου
  - r: σημείο παραγγελίας
  - R: αποθεματικός έλεγχος

  Στην παρατήρηση j η ποσότητα παραγγελίας είναι

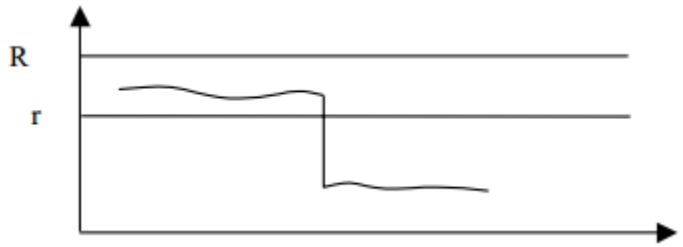
$$Q_j = \begin{cases} 0, & I_j > r \\ R - I_j, & I_j \le r \end{cases}.$$

  Αν r = R τότε σε κάθε σημείο ελέγχου Qj = R – Ij

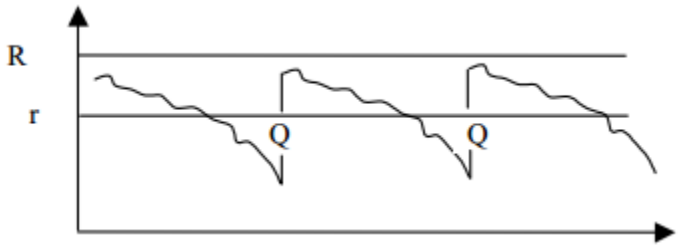
- Συνεχής στρατηγική



Εδώ T → 0 και Q (t) = R− I (t). Θα έπρεπε να γράψουμε Q (t) = R− r αλλά υπάρχει περίπτωση απότομης (βηματικής) βύθισης:



- Στρατηγική δεδομένης ποσότητας



[Φίλης Γιάννης 2006]

1.5.3 Προγραμματισμός απαιτούμενων υλικών

Τα συστήματα προγραμματισμού απαιτούμενων υλικών (ΠΑΥ) αποτελούν μια ξεχωριστή κατηγορία συστημάτων διαχείρισης αποθεμάτων. Τα συστήματα αυτά αφορούν τη διαχείριση υλικών που είναι απαραίτητα για την εκτέλεση του προγράμματος παραγωγής και είτε παραγγέλλονται σε εξωτερικούς προμηθευτές είτε κατασκευάζονται από το ίδιο το παραγωγικό σύστημα. Η ζήτηση για αυτά είναι εσωτερική, προέρχεται δηλαδή από το ίδιο το σύστημα και είναι εξαρτημένη και ασυνεχής. Ο καθορισμός των ποσοτήτων και του χρόνου που πρέπει να είναι διαθέσιμες, στηρίζεται στις απαιτήσεις για υλικά, που καθορίζει συγκεκριμένα το σύστημα παραγωγής και όχι σε προβλέψεις.

Ο στόχος των συστημάτων ΠΑΥ δεν είναι άλλος από το βασικό στόχο όλων των συστημάτων διαχείρισης αποθεμάτων, η εξασφάλιση των ποσοτήτων των απαιτούμενων υλικών ώστε να είναι διαθέσιμα στους χρόνους που χρειάζονται χωρίς να δημιουργούνται καταστάσεις υποαποθέματος, οπότε υπάρχει κίνδυνος να διακοπεί η παραγωγική διαδικασία. Παράλληλα πρέπει να αποφεύγεται η δημιουργία υπεραποθέματος, αφού συνεπάγεται δέσμευση κεφαλαίων, δαπάνες αποθήκευσης κλπ. Ένα τέτοιο σύστημα πρέπει να δίνει απάντηση στα ερωτήματα «πόσο να παραγγελθεί» και «πότε να παραγγελθεί» για κάθε υλικό που χρησιμοποιείται ως εισροή στην παραγωγική διαδικασία.

- **Συγκεντρωτικός Προγραμματισμός Παραγωγής:** Ο Συγκεντρωτικός Προγραμματισμός Παραγωγής είναι η δραστηριότητα με την οποία καθορίζεται το πρόγραμμα (πλάνο) της παραγωγής συγκεντρωτικά, δηλαδή για το σύνολο των προϊόντων ενός παραγωγικού συστήματος, για ένα σύνολο περιόδων. Το συγκεντρωτικό πρόγραμμα παραγωγής περιλαμβάνει τις μεσοπρόθεσμες αποφάσεις της διοίκησης για τις τιμές των βασικών μεγεθών της παραγωγής. Τα μεγέθη αυτά είναι το συνολικό ύψος της παραγωγής, της απασχόλησης και των αποθεμάτων, συνήθως σε μηνιαία βάση, που τίθενται ως στόχοι για ένα μεσοπρόθεσμο ορίζοντα προγραμματισμού (π.χ. 12 μηνών).

  Τα δεδομένα που απαιτούνται για την κατάρτιση του συγκεντρωτικού προγράμματος παραγωγής είναι η δυναμικότητα του συστήματος, η προβλεπόμενη ζήτηση στον ορίζοντα προγραμματισμού για κάθε περίοδο (συνήθως κάθε μήνα), τα υπάρχοντα αποθέματα και οι γενικοί στόχοι και κριτήρια προγραμματισμού που θέτει η διοίκηση.

Απαραίτητη είναι και η γνώση των στοιχείων που αφορούν κυρίως το κόστος της παραγωγής (κόστος εργασίας, κόστος αποθεματοποίησης, κόστος υποαποθέματος, μεταβλητό κόστος παραγωγής, κόστος μεταβολών στο επίπεδο απασχόλησης). Με βάση αυτά τα δεδομένα μπορούν καταρχήν να διαμορφωθούν πολλά εναλλακτικά συγκεντρωτικά προγράμματα παραγωγής.

Το συγκεντρωτικό πρόγραμμα παραγωγής αποτελεί το πλαίσιο, μέσα στο οποίο οργανώνεται και αναπτύσσεται η παραγωγική δραστηριότητα ενός συστήματος. Περιλαμβάνει ένα σύνολο στόχων που τίθενται για το σύστημα και αφορούν την παραγωγή, την απασχόληση και τα αποθέματα για κάθε περίοδο μέσα στον ορίζοντα προγραμματισμού. Οι στόχοι αυτοί είναι παράλληλα και περιορισμοί του συστήματος, όσον αφορά στην παραγωγική λειτουργία. Ακόμα, οι στόχοι αυτοί αποτελούν έμμεσα στόχους και περιορισμούς και των άλλων λειτουργιών, όπως της χρηματοοικονομικής λειτουργίας ή της λειτουργίας των προμηθειών. Τα επιμέρους προγράμματα αυτών των λειτουργιών πρέπει να καταρτίζονται μέσα στο πλαίσιο που θέτει το συγκεντρωτικό πρόγραμμα παραγωγής.

- **Χρονικός Προγραμματισμός Παραγωγής**: Εκτός από το στρατηγικό πρόβλημα του μακροπρόθεσμου σχεδιασμού της δυναμικότητας κάθε συστήματος παραγωγής, τίθεται το πρόβλημα του προγραμματισμού σε μεσοπρόθεσμη και βραχυπρόθεσμη βάση των διατιθέμενων πόρων (ανθρώπινο δυναμικό, μηχανολογικός εξοπλισμός, οικονομικοί πόροι), ώστε τα συστήματα να εκπληρώσουν τους στόχους τους, ανταποκρινόμενα στη ζήτηση των προϊόντων τους. Στην προηγούμενη παράγραφο περιγράφηκε το ζήτημα του μεσοπρόθεσμου συγκεντρωτικού προγραμματισμού παραγωγής, το οποίο αφορά βασικά τα συστήματα flow-shop. Στην παράγραφο αυτή περιγράφουμε το ζήτημα του (βραχυχρόνιου) χρονικού προγραμματισμού παραγωγής, που αφορά τα συστήματα flow-shop και job-shop.

  Σε κάθε τέτοιο πρόβλημα, με βάση τη διαθέσιμη δυναμικότητα, τις απαιτήσεις για παραγωγή προϊόντων και διάφορους τεχνολογικούς και άλλους περιορισμούς, ζητείται η καλύτερη δυνατή τιμή των μεταβλητών απόφασης, δηλαδή οι τιμές αυτών που αντιστοιχούν στην καλύτερη δυνατή τιμή μιας συνάρτησης κόστους. Έτσι από ένα σύνολο εφικτών προγραμμάτων ζητείται το καλύτερο (αν και συχνά ο καθορισμός του είναι ανέφικτος, οπότε αρκεί να βρεθεί ένα «καλό» πρόγραμμα).

  Οι απαιτήσεις για παραγωγή προϊόντων μεταφράζονται μέσω των πινάκων υλικών, των προβλέψεων και των παραγγελιών των πελατών σε απαιτήσεις για παραγωγικούς πόρους (ανθρώπινο δυναμικό, μηχανήματα, σειρά επεξεργασιών, χρόνοι παραγωγής). Οι περιορισμοί αφορούν τη δυναμικότητα, την ακολουθία των δραστηριοτήτων που ορίζει η υπάρχουσα τεχνολογία, τις απαιτήσεις για συντήρηση των μηχανών. Οι μεταβλητές απόφασης αφορούν το μέγεθος μιας παρτίδας παραγωγής (πόσα κομμάτια ανά παρτίδα), τη φόρτωση των μηχανών (ποια παραγγελία εκτελείται σε ποια μηχανή), τη σειρά εκτέλεσης των παραγγελιών κλπ. Τέλος, η συνάρτηση κόστους αφορά την πλήρωση κάποιων κριτηρίων που μπορεί να αναφέρονται στην εξυπηρέτηση των

πελατών, στο συνολικό κόστος λειτουργίας, στην αξιοποίηση της διαθέσιμης δυναμικότητας κλπ. Έτσι ένα πρόγραμμα είναι καλύτερο από ένα άλλο αν το πρώτο ικανοποιεί σε μεγαλύτερο βαθμό τα κριτήρια που έχουν τεθεί.

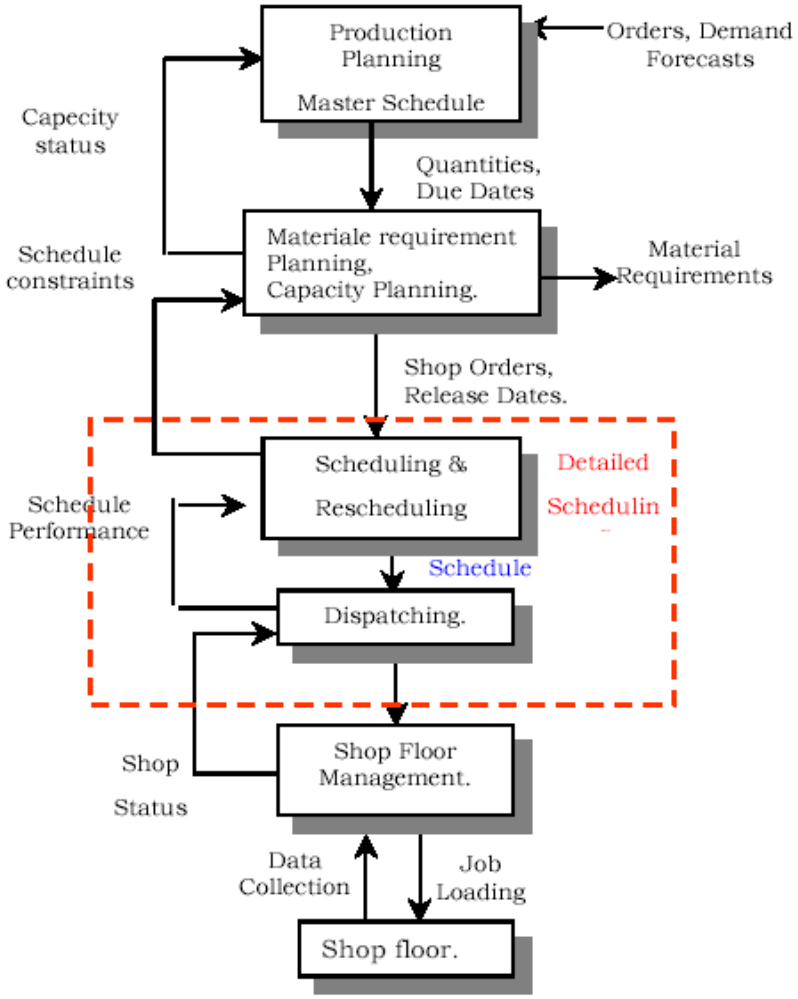
- **Προγραμματισμός Έργων:** Τα προβλήματα προγραμματισμού και οργάνωσης της εκτέλεσης ενός έργου έχουν να κάνουν με το μεγάλο πλήθος των επιμέρους δραστηριοτήτων, από την εκτέλεση και τη διαπλοκή των οποίων εξαρτάται η ολοκλήρωση του ίδιου του έργου. Οι δραστηριότητες αυτές συνδέονται μεταξύ τους με τεχνολογικές, φυσικές, οικονομικές ή άλλες σχέσεις προτεραιότητας, δηλαδή του τύπου «προηγείται-έπεται», ενώ υπόκεινται σε διάφορους περιορισμούς, π.χ. λόγω διαθέσιμων πόρων ή υπάρχοντος θεσμικού πλαισίου, που πρέπει να ληφθούν υπόψη κατά τον προγραμματισμό τους. Τα παραπάνω προβλήματα αποκτούν ιδιαίτερη σημασία λόγω της κλίμακας των έργων, του κόστους κατασκευής τους, του ρόλου τους στην οικονομική και κοινωνική ζωή (όπως ένα λιμάνι ή ένα φράγμα) κλπ. Το ζητούμενο σε τέτοια προβλήματα μπορεί να είναι η ελαχιστοποίηση του συνολικού χρόνου εκτέλεσης του έργου, η ελαχιστοποίηση του συνολικού κόστους, η ελαχιστοποίηση του κόστους για ένα δεδομένο ολικό χρόνο, η ελαχιστοποίηση του χρόνου εκτέλεσης για ένα δεδομένο κόστος, η ελαχιστοποίηση των πόρων που αδρανούν. Οι κύριες μέθοδοι που έχουν αναπτυχθεί για την επίλυση των παραπάνω προβλημάτων είναι η μέθοδος της κρίσιμης διαδρομής (CPM) και η τεχνική αξιολόγησης και αναθεώρησης προγράμματος (PERT). [Δίπλας – Τσακίρης, 2004]

1.6 Χρονοπρογραμματισμός Παραγωγής

1.6.1 Σημασία χρονοπρογραμματισμού στα συστήματα παραγωγής

Ως «χρονοπρογραμματισμός» ορίζεται *«η κατανομή δεδομένων πόρων στη διάρκεια του χρόνου με σκοπό την ολοκλήρωση ενός συνόλου εργασιών»* [Baker, 1974]. Η διαδικασία του χρονικού προγραμματισμού αποτελεί βασικό κομμάτι πλέον στις βιομηχανίες παραγωγής-κατασκευών, αλλά και σε αυτές της παροχής υπηρεσιών. Στο σημερινό ανταγωνιστικό περιβάλλον της αγοράς, ο αποτελεσματικός προγραμματισμός εργασιών είναι σημαντικός παράγοντας στην επιβίωση μιας επιχείρησης. Οι εταιρείες πρέπει να ανταποκριθούν σε πολλές διορίες που τίθενται από τους πελάτες τους, αλλιώς θα χάσουν την αξιοπιστία τους. Ταυτόχρονα θα πρέπει να προγραμματίσουν τις ενέργειές τους έτσι ώστε να χρησιμοποιούν τους πόρους τους με αποτελεσματικό τρόπο.

Ο χρονικός προγραμματισμός περιλαμβάνει τις διαδικασίες εκείνες που απαιτούνται για να διασφαλιστεί η ολοκλήρωση του έργου στον προαποφασισμένο χρόνο, με το αρχικά ορισμένο κόστος και στην ζητούμενη ποιότητα [Project Management Institute, 2008].



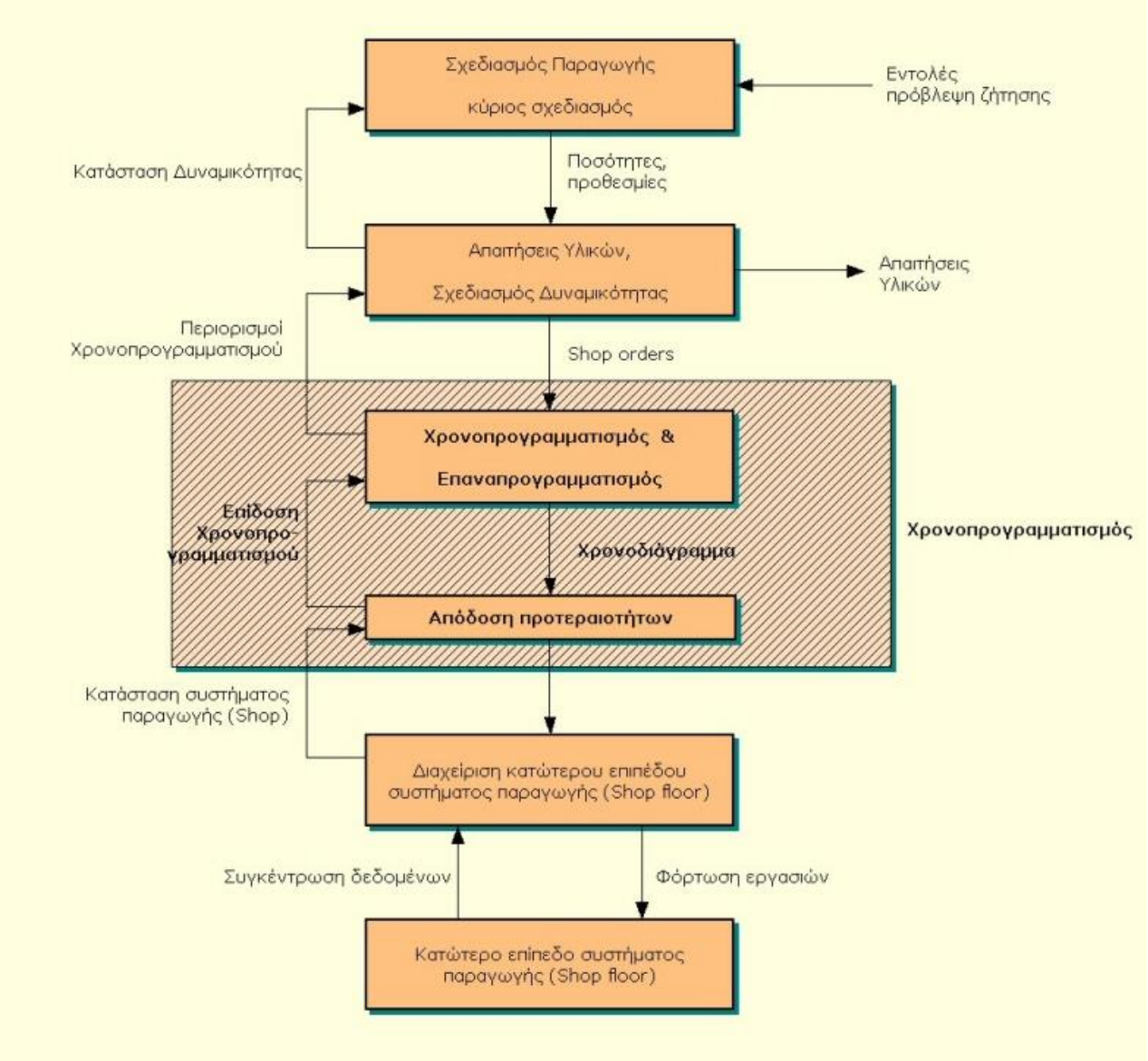
Σχήμα 1.4: Ένα τυπικό σύστημα παραγωγής [Λεραντζής 2004]

Οι διαδικασίες αυτές συνοψίζονται ως εξής:

- Καθορισμός των δραστηριοτήτων που απαρτίζουν το έργο.
- Σχέσεις προτεραιότητας μεταξύ των δραστηριοτήτων
- Καθορισμός των απαιτούμενων πόρων για την επίτευξη των στόχων
- Εκτίμηση της διάρκειας των δραστηριοτήτων του έργου
- Σχεδιασμός χρονοπρογράμματος
- Έλεγχος του έργου

Επομένως, κατά το χρονικό προγραμματισμό καθορίζεται η χρονική αλληλουχία των επί μέρους δραστηριοτήτων, η χρονική κατανομή του παραγωγικού δυναμικού και η ροή των υλικών που θα χρησιμοποιηθούν στο έργο. Ο χρονικός προγραμματισμός αφορά την δημιουργία χρονοπινάκων (ημερομηνίες έναρξης, ημερομηνίες ολοκλήρωσης, διάρκειες δραστηριοτήτων).

Τα χρονοπρογράμματα είναι βασικά εργαλεία εργασίας για τον προγραμματισμό, την αξιολόγηση και τον έλεγχο του προγράμματος Το σύνολο των χρονοπινάκων αποτελεί το πρόγραμμα του. [Καλοειδάς 2012]



Σχήμα 1.5 [Καρόπουλος 2005]

1.6.2 Το πρόβλημα του χρονοπρογραμματισμού παραγωγής

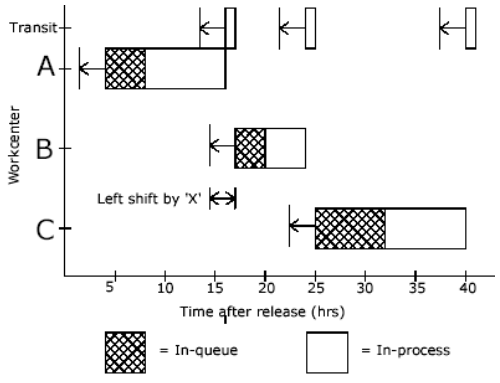
Ένας τυπικός ορισμός που έχει δοθεί για την περιγραφή του γενικού προβλήματος είναι ο παρακάτω:

Ένα σύνολο J αποτελούμενο από n εργασίες (οι όροι εργασίες και παραγγελίες/προϊόντα έχουν κοινή σημασία) $J = \{J_1, J_2, ..., J_n\}$ πρέπει να υποστούν επεξεργασία από ένα σύνολο m διαθέσιμων μηχανών, $M = \{M_1, M_2, ..., M_m\}$. Κάθε εργασία αντιστοιχεί σε κάποια παραγγελία

από το σύνολο O, όπου O = {O_1, O_2, ..., O_O}. Για την ολοκλήρωση του συνόλου των λειτουργιών κάθε εργασίας απαιτείται η επίσκεψη ενός συγκεκριμένου υποσυνόλου των μηχανών. Η αλληλουχία, σύμφωνα με την οποία κάθε εργασία θα επισκεφτεί το σύνολο των μηχανών που της αντιστοιχεί μπορεί να είναι ορισμένο ή όχι. Η επεξεργασία της εργασίας J_j στη μηχανή M_i καλείται λειτουργία (operation) OP_{ij}. Σε κάθε λειτουργία αντιστοιχίζεται ο χρόνος επεξεργασίας t_{ij}. Για το σύνολο των εργασιών ορίζονται οι χρόνοι διαθεσιμότητας r_j (ready time, release time) που δηλώνουν τη χρονική στιγμή, από την οποία και μετά είναι διαθέσιμες οι εργασίες για επεξεργασία ή/και οι χρόνοι ολοκλήρωσης d_j (due date) που ορίζουν τη χρονική στιγμή μέχρι την οποία θα πρέπει να έχει ολοκληρωθεί το αργότεροι το σύνολο των λειτουργιών τους από τις μηχανές που τους αντιστοιχούν. Με βάση τα παραπάνω, ως χρονοπρογραμματισμός θεωρείται η χρονική ανάθεση των εργασιών στις μηχανές. Το πρόβλημα του χρονοπρογραμματισμού αναφέρεται στην εύρεση βέλτιστων πλάνων χρονοδρομολογήσεων σύμφωνα με προκαθορισμένα κριτήρια. [Καρόπουλος 2005]

1.6.3 Περιοχές λύσεων του προβλήματος

Σ' αυτή την παράγραφο θα οριστούν κάποιοι όροι και θα εξηγηθούν οι κύριες περιοχές του διαστήματος λύσεων του προβλήματος χρονοπρογραμματισμού προκειμένου να προσδιοριστούν οι ιδιότητες όλων των λύσεων. Ένα πραγματοποιήσιμο πρόγραμμα (feasible schedule) είναι ένα πρόγραμμα που δεν αντιβαίνει τους περιορισμούς του προβλήματος. Ας θεωρήσουμε το πρόγραμμα που παρουσιάζεται στον χάρτη Gantt στο σχήμα παρακάτω. Δοσμένης της σειράς των λειτουργιών σε μία μηχανή, μία τοπική μετακίνηση προς τα αριστερά (local left shift) σε ένα πραγματοποιήσιμο πρόγραμμα υφίσταται όταν μία λειτουργία μπορεί να μετακινηθεί προς τα αριστερά για να αρχίσει νωρίτερα και παράλληλα να διατηρηθεί το εφικτό του προγράμματος (feasibility).

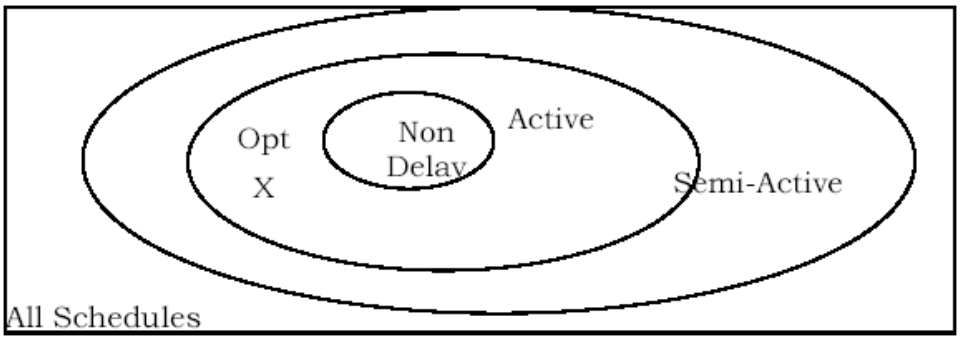


Σχήμα 1.6: Προσαρμογή μετακίνησης προς τα αριστερά (left shift adjustment)

Ένα πρόγραμμα χαρακτηρίζεται ως ημι-ενεργό (semi-active) αν δεν μπορεί να πραγματοποιηθεί τοπική μετακίνηση προς τα αριστερά και να διατηρηθεί το εφικτό. Σε ένα ημι-ενεργό πρόγραμμα καμία λειτουργία δεν μπορεί να ξεκινήσει νωρίτερα αν δεν μεταβληθούν οι σειρές των λειτουργιών στη μηχανή. Μία γενική μετακίνηση προς τα αριστερά (global left shift) συμβαίνει

όταν μία λειτουργία μπορεί να μετακινηθεί ένα διάστημα νωρίτερα στο πρόγραμμα και να διατηρηθεί το εφικτό, χωρίς να καθυστερήσουν άλλες εργασίες. Το σύνολο των προγραμμάτων στα οποία δεν μπορεί να γίνει γενική μετακίνηση προς τα αριστερά ονομάζονται ενεργά προγράμματα (active schedules). Ανάμεσα στα ενεργά προγράμματα βρίσκονται τα βέλτιστα αλλά χρειάζεται ένας επιπλέον μηχανισμός για να ψάξει σ' αυτό το διάστημα λύσεων ώστε να βρει τη βέλτιστη λύση.

Ένα πρόγραμμα χαρακτηρίζεται non-delay όταν είναι ενεργό και όταν καμία μηχανή δεν μένει μη απασχολημένη για κάποιο διάστημα αλλά επεξεργάζεται συνεχώς κάποια λειτουργία. Η βέλτιστη λύση προέρχεται από τέτοια προγράμματα. Υπάρχουν όμως περιπτώσεις λύσεων όπου οι μηχανές μένουν αχρησιμοποίητες για κάποιο διάστημα και παρόλα αυτά δίνουν καλύτερα αποτελέσματα από non-delay προγράμματα για κάποια κριτήρια. Στο σχήμα 1.7 φαίνονται όλες οι δυνατές λύσεις και η σχέση μεταξύ τους.



Σχήμα 1.7 – Ταξινόμηση προγραμμάτων (schedules classification) [Λεραντζής 2004]

1.6.4. Κύρια κριτήρια αξιολόγησης προγραμμάτων (Primary Output Measures)

Στον χρονοπρογραμματισμό μπορούμε να χρησιμοποιήσουμε ένα ευρύ φάσμα κριτηρίων. Το κριτήριο είναι αυτό που καθορίζει εάν ένα πρόγραμμα είναι κατάλληλο ή όχι. Μπορεί ένα πρόγραμμα σύμφωνα με ένα κριτήριο να είναι ακατάλληλο ενώ το ίδιο πρόγραμμα βάσει ενός άλλου κριτηρίου να είναι πολύ καλό.

Τα διάφορα κριτήρια ή αλλιώς οι αντικειμενικές συναρτήσεις για προβλήματα χρονοπρογραμματισμού παραγωγής είναι τα ακόλουθα:

- Χρόνος ολοκλήρωσης (**completion time**) makespan (C_{max}). Με τον όρο makespan καλούμε τη διάρκεια στην οποία όλες οι λειτουργίες όλων των εργασιών έχουν ολοκληρωθεί. Το πρόγραμμα με το μικρότερο makespan λαμβάνεται συνήθως ως το πρόγραμμα με τη μεγαλύτερη χρησιμότητα. Η βασική ιδέα πίσω από αυτό είναι πως ολοκληρώνοντας το δοσμένο σύνολο δραστηριοτήτων νωρίτερα θα επιτρέψει νέες δραστηριότητες να ξεκινήσουν νωρίτερα. Αυτή η ιδέα είναι μία πολύ καλή προσέγγιση

στην ειδική περίπτωση όπου διαθέτουμε μία μηχανή και μία χρήσιμη σκέψη για πιο γενικά προβλήματα. Παρόλα αυτά, το makespan δεν λαμβάνει υπόψη του τις ημερομηνίες παράδοσης, κάτι που στη βιομηχανία σε πραγματικές συνθήκες είναι παράγοντας υψηλής προτεραιότητας.

- Ο χρόνος ροής (**flowtime**). Ο χρόνος ροής μετρά την ανταπόκριση του συστήματος στις διάφορες απαιτήσεις υπηρεσιών. Ο χρόνος ροής είναι το χρονικό διάστημα μεταξύ της άφιξης μίας εργασίας στο σύστημα και της αναχώρησης από αυτό.

- Καθυστέρηση (**lateness**) το χρονικό διάστημα μία συγκεκριμένη εργασία βρίσκεται στο σύστημα παραγωγής καθυστέρηση Η καθυστέρηση μετρά την ικανότητα προσαρμογής ενός προγράμματος σε μία δοσμένη ημερομηνία παράδοσης. Η καθυστέρηση «ανταμείβει» τις εργασίες που γίνονται νωρίτερα και «τιμωρεί» αυτές που αργοπορούν. Ο χρόνος ροής και η καθυστέρηση είναι στην πραγματικότητα δύο μορφές του ίδιου μέτρου και καταλήγουν στην ίδια συμπεριφορά χρονοπρογραμματισμού.

- Σταθμισμένη καθυστέρηση (weighted lateness, L_{wt}): Η σταθμισμένη καθυστέρηση είναι ένα μέτρο της διαφοράς μεταξύ την αυστηρής ημερομηνίας παράδοσης και της πραγματικής ημερομηνίας ολοκλήρωσης των εργασιών. Στην περίπτωση της καθυστερημένης παράδοσης η τιμή της είναι θετική και στην περίπτωση της παράδοσης σε νωρίτερη χρονική στιγμή από αυτή της ημερομηνίας παράδοσης το μέτρο αυτό έχει τιμή αρνητική. Στη συνέχεια η τιμή αυτή πολλαπλασιάζεται με ένα συντελεστή βάρους. Είναι ένα δημοφιλές μέτρο της ποιότητας του χρονοπρογραμματισμού. Είναι πολύ ισχυρό μέτρο με την έννοια ότι οι βέλτιστες λύσεις που βρίσκουμε στηριζόμενοι σε αυτό συχνά παράγουν καλές λύσεις για προβλήματα με άλλες αντικειμενικές συναρτήσεις.

- Ο χρόνος βραδύτερης περάτωσης (**tardiness),** η καθυστέρηση μίας εργασίας εάν αυτή είναι θετική. Αν η καθυστέρηση δεν είναι θετική, ο χρόνος βραδύτερης περάτωσης είναι μηδέν. Ο χρόνος βραδύτερης περάτωσης έχει να κάνει με το γεγονός ότι σε πολλές περιπτώσεις συγκεκριμένα κόστη μπορεί να προέρχονται από μία θετική καθυστέρηση ενώ μία αρνητική καθυστέρηση δεν αποφέρει πραγματικά οφέλη. Αυτό είναι ένα καλό μέτρο της χρησιμότητας ενός προγράμματος σε πραγματικούς όρους.

- Σταθμισμένος χρόνος βραδύτερης περάτωσης (weighted tardiness, T_{wt}): Σε πολλές περιπτώσεις ο χρόνος βραδύτερης περάτωσης αποτελεί μία καλή αντικειμενική συνάρτηση. Αυτό το μέτρο είναι το ίδιο με την σταθμισμένη καθυστέρηση εκτός από το ότι στις νωρίτερα ολοκληρωμένες εργασίες θέτει την τιμή μηδέν. Με αυτό τον τρόπο ανταποκρίνεται καλύτερα στις ανάγκες του πραγματικού πελάτη. Σε πολλές πρακτικές καταστάσεις είναι το πιο κατάλληλο μέτρο της απόδοσης ενός προγράμματος μια και λαμβάνει υπόψη του το κόστος που συνεπάγεται μία καθυστερημένη παράδοση αλλά δεν αντισταθμίζει με κέρδος τις νωρίτερες παραδόσεις εργασιών από τη στιγμή που αυτό δεν είναι αναγκαίο. Παρά το ότι είναι ένα μέτρο αρκετά χρήσιμο, στη βιβλιογραφία θα βρούμε λίγες περιπτώσεις που να χρησιμοποιείται ως μέτρο απόδοσης.

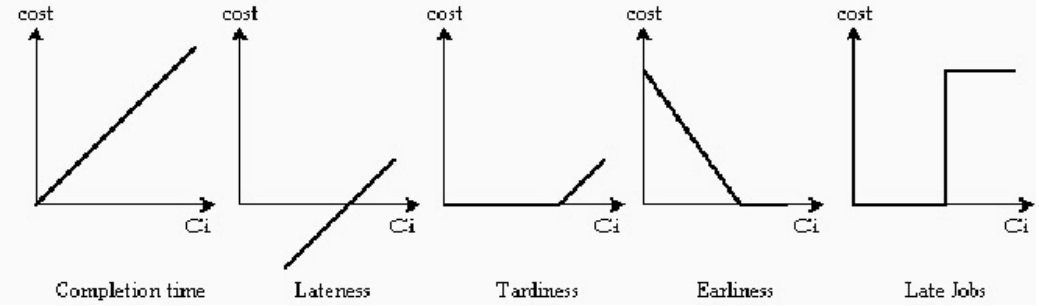
- Μέγιστος χρόνος βραδύτερης περάτωσης (maximum tardiness, T_{max}): Η ελαχιστοποίηση του μέγιστου χρόνου βραδύτερης περάτωσης είναι σημαντικό μέτρο όταν οι πελάτες ανέχονται τις μικρές καθυστερήσεις αλλά γίνονται σταδιακά όλο και πιο ανήσυχοι και πιεστικοί όσο αυξάνονται αυτές. Η ελαχιστοποίηση της μέγιστης καθυστέρησης είναι αντίστοιχο πρόβλημα και μπορεί να χρησιμοποιηθεί ως βοηθητικό για τη λύση άλλων προβλημάτων.

  Παρακάτω, το C_i δηλώνει το χρόνο ολοκλήρωσης της εργασίας J_i. Τα περισσότερα κλασικά κριτήρια χρονοπρογραμματισμού λαμβάνουν υπόψη τους μία ημερομηνία παράδοσης δ_i για κάθε εργασία. Η βραδύτερη περάτωση T_i της εργασίας J_i ορίζεται ως $T_i = max\ (0,\ C_i - \delta_i)$. Η μεταβλητή U_i χρησιμοποιείται για να δηλώσει μία μονάδα «ποινής» (penalty) για κάθε καθυστερημένη εργασία. Η U_i ισούται με 0 όταν $C_i \leq \delta_i$ και με 1 σε άλλη περίπτωση.

- Ο χρόνος νωρίτερης περάτωσης **(earliness),** το αντίστροφο της καθυστέρησης μίας εργασίας, εάν αυτή είναι αρνητική. Εάν η καθυστέρηση είναι θετική ο χρόνος νωρίτερης περάτωσης είναι μηδέν. Ο χρόνος νωρίτερης περάτωσης αντιστοιχεί σε κάποιες περιπτώσεις όπου μπορεί να υπάρξει κόστος όταν μία δραστηριότητα ολοκληρώνεται πριν την ημέρα παράδοσής της (το κόστος αποθεμάτων ενδέχεται να είναι αρκετά υψηλό). Το κριτήριο χρονοπρογραμματισμού διατυπώνεται είτε ως άθροισμα είτε ως μέγιστο. Ένας συντελεστής βάρους w_i μπορεί να δοθεί σε κάθε εργασία δίνοντας ιδιαίτερη βαρύτητα σε κάποιες από αυτές. Σημειώνουμε τα παρακάτω γνωστά κριτήρια βελτιστοποίησης:

  - Makespan: $F = C_{max} = \max C_i$
  - Σταθμισμένος συνολικός χρόνος ροής (total weighted flow time): $F = \sum w_i C_i$
  - Μέγιστος χρόνος βραδύτερης περάτωσης (maximum tardiness): $F = T_{max} = \max T_i$
  - Σταθμισμένος συνολικός χρόνος βραδύτερης περάτωσης (total weighted tardiness): $F = \sum w_i T_i$
  - Σταθμισμένος συνολικός αριθμός αργοπορημένων εργασιών (total weighted number of late jobs): $F = \sum w_i U_i$ [Λεραντζής 2004, Γιαλεδάκη Ελένη 2005]

  Το σχήμα 1.8 δείχνει τη σχέση που υπάρχει μεταξύ του κόστους και των μέτρων που αναφέρθηκαν προηγουμένως κατά τη διάρκεια του χρόνου



Σχήμα 1.8: Συνάρτηση κόστους των κύριων μέτρων αξιολόγησης

1.7 Υποθέσεις Προβλήματος Χρονοπρογραμματισμού

Σε αυτή την παράγραφο αναφέρονται οι κύριες υποθέσεις που αφορούν το πρόβλημα χρονοπρογραμματισμού παραγωγής όπως διαμορφώθηκαν τις τελευταίες τέσσερις δεκαετίες, μετά από έρευνα στις περιοχές της επιχειρησιακής έρευνας και του χρονοπρογραμματισμού. Αυτές οι υποθέσεις αποτελούν τα θεμέλια των περισσοτέρων θεωρητικών τεχνικών χρονοπρογραμματισμού. Στην αρχή και στο τέλος του προβλήματος χρονοπρογραμματισμού το σύστημα παραγωγής είναι άδειο. Όλες οι εργασίες είναι διαθέσιμες για επεξεργασία τη χρονική στιγμή μηδέν. Οι μηχανές είναι ο σημαντικότερος και καθοριστικός παράγοντας της παραγωγής.

Το ανθρώπινο δυναμικό είναι «άφθονο» (plentiful) και οι ικανότητές του δεν αποτελούν περιορισμό στη παραγωγή. Με άλλα λόγια, η τεχνογνωσία και το πλήθος των εργαζομένων δεν λαμβάνονται υπόψη. Κάθε εργασία πηγαίνει σε κάθε μηχανή το πολύ μία φορά. Όλες οι πληροφορίες σχετικά με τους χρόνους επεξεργασίας, τη δρομολόγηση και τις ημερομηνίες παράδοσης είναι αιτιοκρατικές (deterministic) κα γνωστές εκ των προτέρων τη χρονική στιγμή μηδέν του προβλήματος. Όλοι οι χρόνοι ετοιμασίας (set-up times) είναι ανεξάρτητοι από τη σειρά των εργασιών και συμπεριλαμβάνονται στους χρόνους επεξεργασίας. Δεν υπάρχουν παράλληλες μηχανές, δηλαδή διαθέτουμε μόνο μία από κάθε είδος μηχανής. Μία λειτουργία μίας εργασίας μπορεί να ολοκληρωθεί μόνο σε μία μηχανή. Όλες οι λειτουργίες μόλις αρχίσουν, ολοκληρώνονται. Δηλαδή δεν μπορούν να διακοπούν και να συνεχίσουν κάποια άλλη χρονική στιγμή. Οι διακοπές των μηχανών (machines breakdowns) δεν λαμβάνονται υπόψη. Θεωρούμε ότι οι μηχανές είναι αξιόπιστες. Δεν είναι διαθέσιμες υπερωρίες (overtime). Κάθε μηχανή μπορεί να επεξεργαστεί και να ολοκληρώσει μία εργασία μία συγκεκριμένη χρονική στιγμή. Κάθε εργασία έχει μία μόνο διαδρομή, δηλαδή οδηγείται προς μία μόνο κατεύθυνση. Όλες οι εργασίες είναι της ίδιας σημαντικότητας ή προτεραιότητας. Στην πραγματικότητα πολλά από τα παραπάνω σημεία δεν ισχύουν στα περισσότερα προβλήματα που έχουν να αντιμετωπίσουν οι επιχειρήσεις.

Κεφάλαιο 2

Σύγχρονες Προσεγγίσεις Επίλυσης Προβλημάτων Χρονοπρογραμματισμού

2.1 Εισαγωγή

Το πρόβλημα job-shop είναι ένα από τα πιο δημοφιλή προβλήματα στη θεωρία προγραμματισμού παραγωγής. Από τη μια μεριά είναι πολύ απλό στην κατανόηση, από την άλλη όμως είναι αντιπροσωπευτικό του γενικότερου τομέα, αφού καταδεικνύει τη δυσκολία της συνδυαστικής βελτιστοποίησης. Η δυσκολία είναι τόσο θεωρητική όσο και πρακτική (π.χ. το πρόβλημα 10 εργασιών και 10 μηχανών όπως προτάθηκε από τους Fisher και Thompson το οποίο παρέμεινε άλυτο για σχεδόν 25 χρόνια, παρά την εντεταμένη έρευνα που έγινε πάνω στον τομέα). Κάθε εργασία είναι μια αλυσίδα λειτουργιών και κάθε μηχανή μπορεί να επεξεργαστεί μοναδική λειτουργία τη φορά. Επιπλέον, σχεδόν μια λειτουργία από κάθε εργασία μπορεί να καταστεί επεξεργάσιμη οποιαδήποτε στιγμή και οι λειτουργίες δεν μπορούν να διακοπούν εφόσον έχουν ήδη αρχίσει. Η κύρια πρόταση είναι να αποδοθούν στις λειτουργίες χρόνοι έναρξης, έτσι ώστε ο μέγιστος χρόνος ολοκλήρωσης να είναι ελάχιστος. Γενικά υπάρχουν δύο τύποι περιορισμών στο πρόβλημα job-shop: προτεραιότητα (χρονική) και διαζευκτική (πόροι–πηγές). Οι περιορισμοί που αφορούν την προτεραιότητα δηλώνουν ότι μερικές λειτουργίες πρέπει να ολοκληρωθούν προτού αρχίσουν κάποιες άλλες. Οι διαζευκτικοί περιορισμοί εκφράζουν το γεγονός ότι δύο λειτουργίες που απαιτούν την ίδια πηγή πόρων δεν μπορούν να πραγματοποιηθούν ταυτόχρονα. Ο αντικειμενικός στόχος είναι να δημιουργηθεί ένα πρόγραμμα σχεδιασμού που θα διευκρινίζει το χρόνο που πού κάθε τομέας εργασίας πρόκειται να ξεκινήσει και ποιους πόρους θα χρησιμοποιήσει, ώστε να ικανοποιούνται όλοι οι περιορισμοί και παράλληλα να αξιοποιείται ο ελάχιστος δυνατός χρόνος.

[Combining Neural Networks and CLP for Production Scheduling, DIMITRIOS PANOPOULOS, DIMITRIOS PTOCHOS, ATHINA OIKONOMOU, DIMITRIOS ASKOUNIS, JOHN PSARRAS]

2.2 Το Πρόβλημα

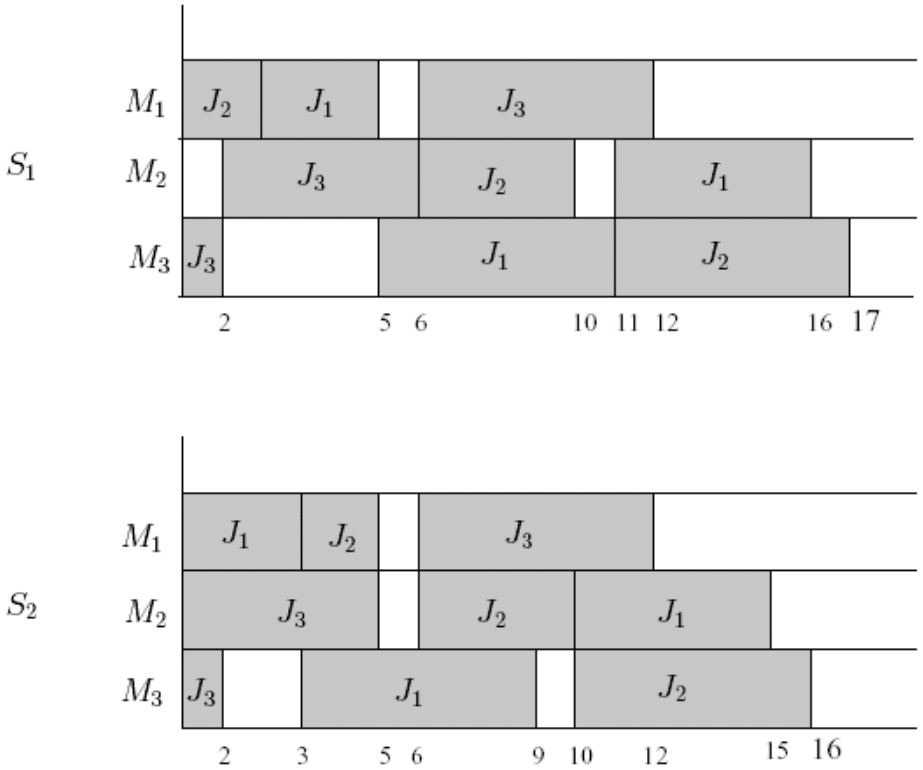
Ένα πρόβλημα τύπου job-shop αποτελείται από ένα πεπερασμένο αριθμό **J** = $\{J^1, \ldots, J^n\}$ εργασιών που πρέπει να πραγματοποιηθούν σε ένα πεπερασμένο αριθμό **M** = $\{m_1, \ldots, m_m\}$ μηχανών. Κάθε εργασία J^i είναι μια πεπερασμένη σειρά βημάτων που πρέπει να εκτελεστούν το ένα μετά το άλλο, όπου κάθε βήμα είναι ένα ζεύγος της μορφής (m, d), $m \in M$ και $d \in N$, δείχνοντας την απαιτούμενη χρήση της μηχανής m για διάρκεια χρόνου d. Κάθε μηχανή μπορεί

να πραγματοποιήσει το πολύ ένα βήμα σε κάποια συγκεκριμένη στιγμή και έτσι εξαιτίας των προηγούμενων περιορισμών, το πολύ ένα βήμα κάθε εργασίας μπορεί να πραγματοποιείται σε κάθε χρονική στιγμή. Βήματα, των οποίων η επεξεργασία έχει ξεκινήσει, δεν μπορούν να αντικατασταθούν από άλλα. Ο σκοπός είναι να προσδιοριστούν οι χρόνοι έναρξης για κάθε βήμα προκειμένου να ελαχιστοποιηθεί ο συνολικός χρόνος διεκπεραίωσης όλων των εργασιών, δηλαδή ο χρόνος κατά τον οποίο εκτελείται το τελευταίο βήμα. Το πρόβλημα αυτό είναι γενικότερα γνωστό ως J || C_{max} όπου C_{max} είναι ο μέγιστος συνολικός χρόνος διεκπεραίωσης, λεγόμενος και 'makespan'.

Παράδειγμα: Έστω M = {m_1, m_2, m_3} και τρεις εργασίες J^1, J^2, J^3 που πρέπει να δρομολογηθούν σε αυτές τις μηχανές. Η εργασία J^1 αποτελείται από 3 βήματα, το πρώτο διαρκεί 3 μονάδες χρόνου και πρέπει να εκτελεστεί στη μηχανή m_1, το δεύτερο διαρκεί 6 μονάδες χρόνου κ. ο. κ.

- J^1: (m_1, 3), (m_3, 6), (m_2, 5)
- J^2: (m_1, 2), (m_2, 4), (m_3, 7)
- J^3: (m_3, 1), (m_2, 5), (m_1, 6)

Στο σχήμα 2.1 φαίνονται 2 διαφορετικοί προγραμματισμοί των εργασιών στις μηχανές, οι S_1 και S_2. Η διάρκεια του S_1 είναι 17 ενώ του S_2 είναι 16, που είναι και το βέλτιστο. Οι 2 προγραμματισμοί παρουσιάζονται σε ένα διάγραμμα Gantt, όπου φαίνεται η εξέλιξη κάθε εργασίας στις μηχανές.



Σχήμα 2.1 - Οι 2 προγραμματισμοί των εργασιών, S1 και S2, σε ένα διάγραμμα Gantt [Δίπλας –Τσακίρης 2004]

2.3 Μέθοδοι Αναπαράστασης Προβλήματος Job-Shop

Οι μέθοδοι που ακολουθούνται για την αναπαράσταση λύσεων του προβλήματος job-shop είναι οι εξής:

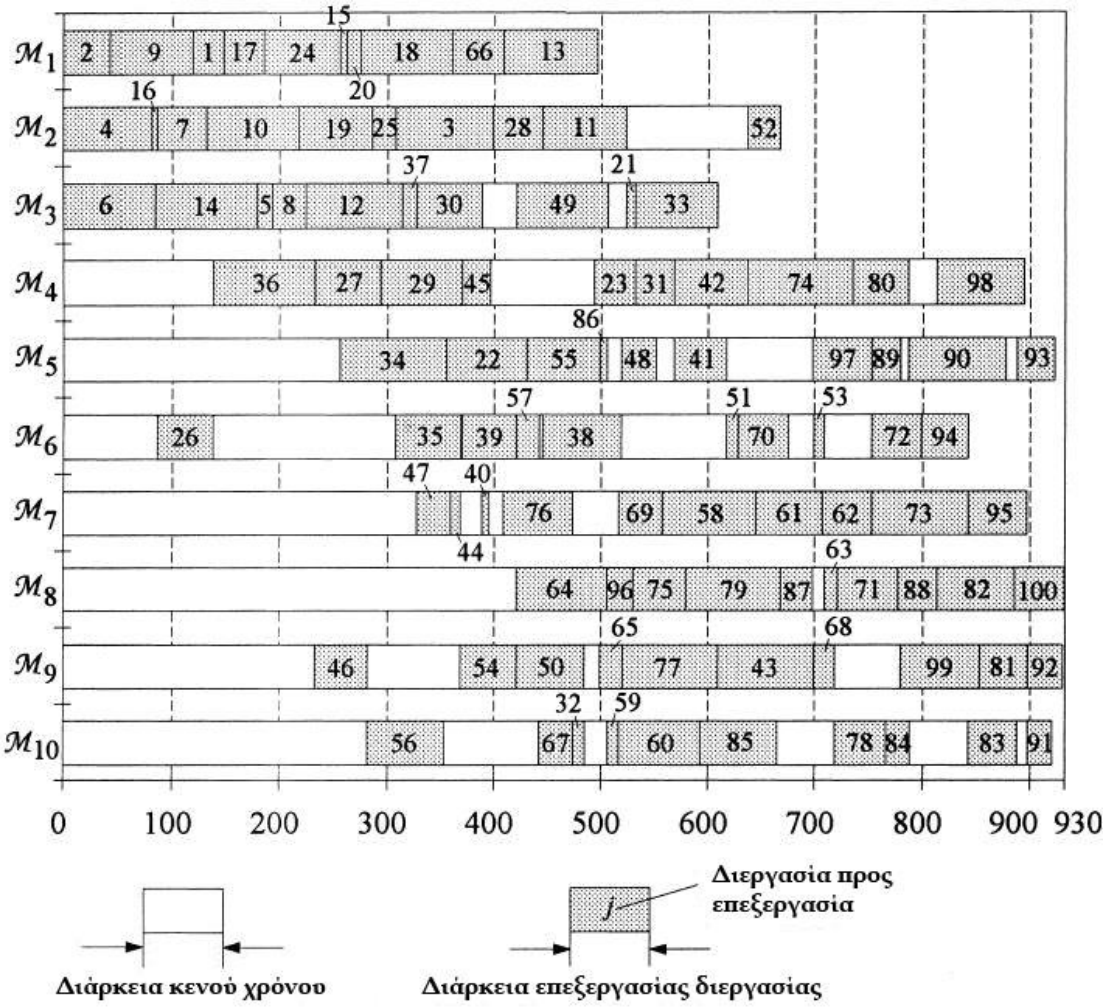
- operation-based representation
- job-based representation
- preference list-based representation
- job pair relation-based representation
- priority rule-based representation
- disjunctive graph-based representation
- completion time-based representation
- machine-based representation
- random keys-based representation
- Gantt-based representation

Στη συνέχεια της ενότητας αναλύονται οι μέθοδοι που στηρίζονται στα διαγράμματα Gantt και στους disjunctive γράφους.

2.3.1 Διάγραμμα Gantt

Ένα πρόγραμμα μπορεί να απεικονιστεί σε ένα διάγραμμα Gantt. Οι μηχανές απεικονίζονται στον κάθετο άξονα και ο χρόνος στον οριζόντιο. Για κάθε μηχανή οι λειτουργίες που εκτελεί απεικονίζονται με μια μπάρα κατά μήκος του άξονα του χρόνου όπου φαίνεται η εξέλιξη κάθε εργασίας στις μηχανές.

Μολονότι το διάγραμμα που περιγράφτηκε από τους Gantt (1919), Clark (1922) και Porter (1968) παραδοσιακά αποτέλεσε την πιο δημοφιλή μέθοδο αναπαράστασης της λύσης ενός προβλήματος χρονοπρογραμματισμού εργασιών, ο Blazewicz (1996) έδειξε ότι το διασπαστικό γραφικό μοντέλο (disjunctive graph model) G = {N, A, E} των Roy και Sussmann (1964) είναι πλέον αυτό που επικρατεί. [Γιαλεδάκη Ελένη 2005]



Σχήμα 2.2 - Διάγραμμα Gantt της βέλτιστης λύσης του FT 10

2.3.2 Αναπαράσταση διαζευκτικού γραφήματος (Disjunctive graph representation)

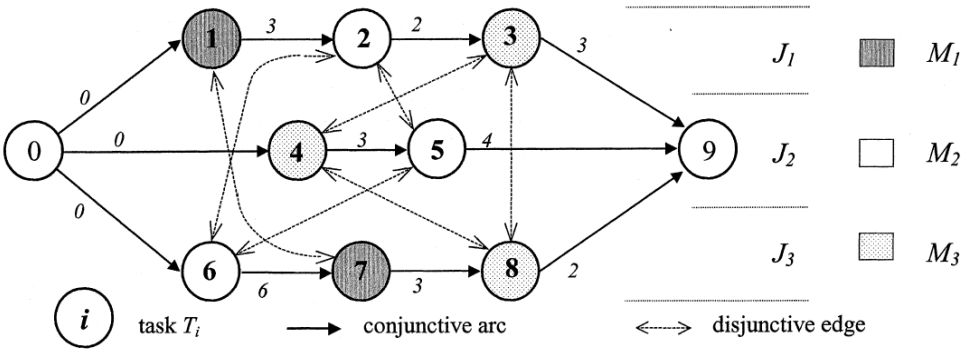
Το διαζευκτικό γράφημα που προτείνεται από τον Roy και Sussman είναι ένα από τα πιο δημοφιλή μοντέλα που χρησιμοποιούνται για να περιγράφει περιπτώσεις του προβλήματος προγραμματισμού σε περιβάλλον job-shop, που στην πραγματικότητα είναι ένα από τα πιο διευρυμένα σε επίπεδο έρευνας, πρόβλημα προγραμματισμού. Το διαζευκτικό γράφημα είναι ένα κατευθυνόμενο γράφημα G = (V, C U D). Το V δηλώνει ένα σύνολο κορυφών που αντιστοιχούν σε τομείς εργασίας. Το σετ περιλαμβάνει δύο επιπλέον κορυφές: μια πηγή και ένα νεροχύτη, τα οποία αντιπροσωπεύουν την έναρξη και το τέλος του προγράμματος. Η πηγή είναι ισοδύναμη με ένα εικονικό T_0 καθήκον που προηγείται όλων των άλλων καθηκόντων και ο νεροχύτης με ένα εικονικό T_{n+1}, πετυχαίνοντας όλα τα άλλα καθήκοντα. Και οι δύο εργασίες έχουν μηδενικό χρόνο επεξεργασίας. Το C είναι ένα σύνολο τόξων που τα οποία αντανακλούν τους περιορισμούς προτεραιότητας, συνδέοντας αρχικά δύο διαδοχικά καθήκοντα της ίδιας εργασίας. Οι μη προσδιορισμένες διαζευκτικές ακμές που ανήκουν στο σύνολο D συνδέουν

αμοιβαία καθήκοντα τα οποία απαιτούν την ίδια μηχανή για την εκτέλεσή τους. Κάθε τόξο είναι επισημασμένο με θετικό βάρος ίσο προς το χρόνο επεξεργασίας του έργου, όπου το τόξο αρχίζει. Το πρόβλημα προγραμματισμού σε περιβάλλον job-shop απαιτεί μια βέλτιστη σειρά εργασιών σχετικών με τις μηχανές, με αποτέλεσμα ένα χρονοδιάγραμμα με το ελάχιστο μήκος. Στο διαζευκτικό μοντέλο, αυτό είναι ισοδύναμο με το να επιλέγεται ένα τόξο σε κάθε διάζευξη. Με τον καθορισμό κατευθύνσεων σε όλες τις διαζευκτικές ακμές, η σειρά εκτέλεσης όλων των αντικρουόμενων καθηκόντων που απαιτούν την ίδια μηχανή, προσδιορίζεται και ένα πλήρες πρόγραμμα αποκτάται. Επιπλέον, το προκύπτον γράφημα πρέπει να είναι ακυκλικό και αν είναι βέλτιστο, το μήκος του μακρύτερου μονοπατιού από την πηγή στον νεροχύτη είναι ελάχιστο. Αυτή η μεγαλύτερη διαδρομή καθορίζει το makespan, είναι η διάρκεια της μακρύτερης αλυσίδας των καθηκόντων σε ένα πρόβλημα job-shop

Παράδειγμα: Ένα job shop αποτελείται από ένα σύνολο τριών μηχανών M = {M1, M2, M3} και ένα σύνολο τριών εργασιών J = {J1, J2, J3} που περιγράφονται από τις ακόλουθες σειρές εργασιών J1: T1 → T2 → T3, και J2: T4 → T5, J3: T6 → T7 → T8. Για κάθε εργασία Ti η απαιτούμενη μηχανή M (Ti) και ο χρόνος επεξεργασίας pi παρουσιάζονται στο σχήμα 2.3. Το διαζευκτικό γράφημα για τη δεδομένη περίσταση παρουσιάζεται στο σχήμα 2.4.

T_i	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
$M(T_i)$	M_1	M_2	M_3	M_3	M_2	M_2	M_1	M_3
p_i	3	2	3	3	4	6	3	2

Σχήμα 2.3



Σχήμα 2.4

[Jacek Blazewicz, Erwin Pesch, Malgorzata Sterna, The disjunctive graph machine representation of the job shop scheduling problem]

2.4 Μέθοδοι Βελτιστοποίησης

2.4.1 Αποδοτικές Μέθοδοι (efficient methods)

Ένας αποδοτικός αλγόριθμος λύνει ένα δεδομένο πρόβλημα βέλτιστα με απαίτηση που αυξάνει πολυωνυμικά με το μέγεθος των δεδομένων εισόδου. Αυτές οι μέθοδοι κατασκευάζουν απλά μία βέλτιστη λύση από τα δεδομένα του προβλήματος ακολουθώντας ένα δεδομένο και απλό σετ κανόνων, οι οποίοι καθορίζουν τη σειρά επεξεργασίας των διεργασιών.

2.4.2 Μαθηματικές Διατυπώσεις (mathematical formulations)

Διάφοροι ερευνητές έχουν συμφωνήσει πως τα προβλήματα χρονοπρογραμματισμού εργασιών μπορούν να λυθούν βέλτιστα χρησιμοποιώντας μεθόδους μαθηματικού προγραμματισμού. Μία από τις πιο γνωστές μεθόδους είναι η μέθοδος μεικτού ακέραιου γραμμικού προγραμματισμού MIP (Mixed Integer Linear Programming). Μία ακόμη μέθοδος που ανήκει στις μεθόδους των μαθηματικών διατυπώσεων είναι η Langrangian Relaxation (LR), η οποία μάλιστα θεωρείται και από τις πιο σημαντικές σε αυτή την κατηγορία μεθόδων επίλυσης. Πιο συγκεκριμένα, πρόκειται για μια μαθηματική προγραμματιστική τεχνική κατάλληλη για βελτιστοποίηση προβλημάτων με περιορισμούς (constrained optimization). Όμοια με τον μηχανισμό των τιμών σε μια αγορά, η μέθοδος αντικαθιστά τους “σκληρούς” περιορισμούς (π.χ. περιορισμούς στην διαθεσιμότητα των μηχανών) με την πληρωμή ορισμένων τιμών (δηλαδή των πολλαπλασιαστών Lagrange) βασιζόμενη στην “ζήτηση” για τη χρήση μιας μηχανής σε κάθε μονάδα χρόνου. Το αρχικό πρόβλημα μπορεί με αυτό τον τρόπο να αναλυθεί σε πολλά μικρότερα και ευκολότερα στην επίλυση υπο-προβλήματα. Τότε χρησιμοποιείται ο κατευθυνόμενος προς τα πίσω δυναμικός προγραμματισμός (backward dynamic programming) για την επίλυση αυτών των υπο-προβλημάτων, όποτε επιβάλλονται άλλοι περιορισμοί. Αφού αυτά τα υπο-προβλήματα επιλυθούν, οι πολλαπλασιαστές προσαρμόζονται βασιζόμενοι στο βαθμό παραβίασης των περιορισμών και ακολουθώντας ξανά τους μηχανισμούς της αγοράς. Τα υπο-προβλήματα τότε επιλύονται ξανά με βάση την καινούρια ομάδα πολλαπλασιαστών και η διαδικασία επαναλαμβάνεται. [JIHUA WANG et al, 1997]. Με μαθηματικούς όρους, η “διπλή συνάρτηση” μεγιστοποιείται κατά την διαδικασία της αλλαγής της τιμής των πολλαπλασιαστών, όπου οι τιμές της διπλής συνάρτησης είναι τα κατώτατα όρια για το βέλτιστο εφικτό κόστος. Όταν οι περιορισμοί χαλαρώνουν (relax) λόγω των πολλαπλασιαστών, οι λύσεις των ξεχωριστών υπο-προβλημάτων, όταν τοποθετούνται μαζί, μπορεί να μην συνιστούν έναν εφικτό χρονοπρογραμματισμό. Συνεπώς χρησιμοποιείται μια απλή ευρετική μέθοδος (θα αναλυθούν στη συνέχεια) προς το τέλος της διαδικασίας αλλαγής των πολλαπλασιαστών ώστε να παραχθούν εφικτοί χρονοπρογραμματισμοί που θα ικανοποιούν όλους τους περιορισμούς. Η ποιότητα όλων των εφικτών χρονοπρογραμματισμών μπορεί να εκτιμηθεί ποσοτικά συγκρίνοντας τα κόστη τους με το μεγαλύτερο κάτω όριο που μας δίνει η διπλή συνάρτηση. [Brucker P, 2001] [Kaskavelis C., Caramanis M., 1998] [Φερλέμης Νικόλαος, 2012]

2.4.3 Τεχνικές Branch and Bound (Branch and Bound techniques)

Η τεχνική Branch and Bound (BB ή B & B) είναι ένας γενικός αλγόριθμος για την εύρεση βέλτιστης λύσης διαφόρων προβλημάτων βελτιστοποίησης, ειδικά σε περιπτώσεις διακριτών και συνδυαστικών βελτιστοποιήσεων. Ένας Branch and Bound αλγόριθμος αποτελείται από μια

συστηματική απαρίθμηση όλων των υποψήφιων λύσεων, όπου τα μεγάλα υποσύνολα μη «καρποφόρων» υποψήφιων λύσεων που απορρίπτονται μαζικά, με τη χρήση άνω και κάτω εκτιμώμενων ορίων της υπό βελτιστοποίησης ποσότητας. Η μέθοδος προτάθηκε για πρώτη φορά από τον A. H. Land και A. G. Doig το 1960 για διακριτό προγραμματισμό [A.H.Land et al, 1960][Moore, 1966][Jaulin et al, 2001][Hansen, 1992]. Οι αλγόριθμοι Branch and Bound χρησιμοποιούν μία δυναμικά κατασκευασμένη δενδρική μορφή ως έναν τρόπο να αναπαραστήσουν τον χώρο λύσεων όλων των δυνατών αλληλουχιών των διεργασιών, όλων των δυνατών δηλαδή, λύσεων του προβλήματος. Η έρευνα ξεκινά από την κορυφή του δένδρου και μια ολοκληρωτική επιλογή έχει γίνει όταν έχουμε πια φτάσει σε ένα από τα χαμηλότερα επίπεδα του δένδρου. Κάθε κόμβος σε ένα επίπεδο p του δένδρου, αναπαριστά μια μερική διαδοχή p διεργασιών. Η συγκεκριμένη μέθοδος χρησιμοποιείται για την επίλυση συνδυαστικών προβλημάτων βελτιστοποίησης τα οποία είναι και αρκετά χρονοβόρα. Βασίζεται στην ιδέα της έξυπνης απαρίθμησης όλων των εφικτών λύσεων. [Brucker P, 2001]

Προκειμένου να διευκολυνθεί η συγκεκριμένη περιγραφή, υποθέτουμε ότι ο στόχος είναι να βρεθεί η ελάχιστη τιμή της συνάρτησης $f(x)$, όπου x κυμαίνεται πάνω από ένα συγκεκριμένο σύνολο S των παραδεκτών ή υποψήφιων λύσεων (ο χώρος αναζήτησης ή εφικτή περιοχή). Σημειώνεται ότι μπορεί κανείς να βρει τη μέγιστη τιμή της $f(x)$, με την εύρεση του ελάχιστου της $g(x) = -f(x)$. Ένας Branch and Bound αλγόριθμος απαιτεί δύο εργαλεία. Το πρώτο είναι μια διαδικασία διαχωρισμού που, δεδομένου ενός συνόλου υποψηφίων λύσεων S, επιστρέφει δύο ή περισσότερα μικρότερα σύνολα $S_1, S_2, \ldots$ των οποίων η ένωση καλύπτει S. Σημειώστε ότι το ελάχιστο της $f(x)$ είναι πάνω από S, τότε $\min\{v_1, v_2, \ldots\}$ όπου καθένα v_i είναι το ελάχιστο εσωτερικό. Αυτό το βήμα ονομάζεται διακλάδωση (branching), αφού εκάστοτε αναδρομική εφαρμογή της καθορίζει μια δομή δέντρου (το δέντρο αναζήτησης) του οποίου οι κόμβοι είναι τα υποσύνολα της S. Το δεύτερο εργαλείο είναι μια διαδικασία που υπολογίζει άνω και κάτω όρια για την ελάχιστη τιμή της $f(x)$ μέσα σε δεδομένο υποσύνολο S. Αυτό το βήμα ονομάζεται οριοθέτηση (bounding).

Η βασική ιδέα του αλγορίθμου BB είναι: αν το κατώτερο όριο A για κάποιο κόμβο του δένδρου (σύνολο των υποψηφίων λύσεων) είναι μεγαλύτερο από το ανώτερο όριο B για κάποιο άλλο κόμβο, τότε μπορεί να απορριφθεί με ασφάλεια από την αναζήτηση. Αυτό το βήμα ονομάζεται κλάδεμα, και συνήθως υλοποιείται με τη διατήρηση μιας καθολικής μεταβλητής m (μοιράζεται ανάμεσα σε όλους τους κόμβους του δέντρου) που καταγράφει το ελάχιστο άνω όριο μεταξύ όλων των υπο-περιοχών που εξετάστηκαν μέχρι σήμερα. Κάθε κόμβος του οποίου το χαμηλότερο όριο είναι μεγαλύτερο από m, μπορεί να απορρίπτεται. Η αναδρομή σταματά όταν το τρέχον σύνολο υποψηφίων έχει μειωθεί σε ένα μόνο στοιχείο, ή όταν το άνω όριο για το σύνολο ταιριάζει με το κατώτερο όριο. Είτε έτσι είτε αλλιώς, κάθε στοιχείο του θα είναι ένα ελάχιστο της συνάρτησης.

[A.H.Land et al, 1960] [Moore, 1966][Jaulin et al, 2001][Hansen, 1992]

2.5 Προσεγγιστικές Μέθοδοι

Αν και οι προσεγγιστικές μέθοδοι βελτιστοποίησης αδυνατούν να παράγουν ακριβείς λύσεις, πλεονεκτούν σε μεγάλο βαθμό σε προβλήματα μεγαλύτερου μεγέθους και αυτό γιατί μπορούν να προσεγγίσουν ικανοποιητικά λύσεις που είναι πολύ κοντά στις βέλτιστες, με ένα μέτριο χρόνο υπολογισμού. Η σημασία των προσεγγιστικών μεθόδων επίλυσης των προβλημάτων χρονοπρογραμματισμού υποδεικνύεται από τους Glover και Greenberg (1989) που υποστηρίζουν ότι η κατευθυνόμενη αναζήτηση ενός «δένδρου» δεν είναι ουσιαστικά ικανοποιητική για συνδυαστικά και δύσκολα προβλήματα.

2.5.1 Μέθοδος Beam Search

Η μέθοδος Beam Search είναι ένας ευρετικός branch and bound αλγόριθμος που επιστρέφει μια λύση, αναζητώντας μόνο κάποιες ελπιδοφόρες επιμέρους λύσεις. Σε κάθε επίπεδο στο δέντρο branch and bound, επιλέγεται μόνο ένας προκαθορισμένος αριθμός κόμβων για διακλάδωση, το υπόλοιπο του δέντρου μονίμως απορρίπτεται. Έτσι, ο αριθμός των κόμβων που πρόκειται να αξιολογηθεί είναι πολυωνυμική στο μέγεθος του προβλήματος. Σε κάθε επίπεδο, ο αριθμός των κόμβων φυλάσσονται για περαιτέρω διακλάδωση. Ο αριθμός των «ελπιδοφόρων» κόμβων ονομάζεται πλάτος δέσμης, β. Η συνάρτηση αξιολόγησης που καθορίζει τους β καλείται συνάρτηση αξιολόγησης πορείας. [Selin Bilgin Ozpeynirci, Meral Azizog˘lu, 2009]

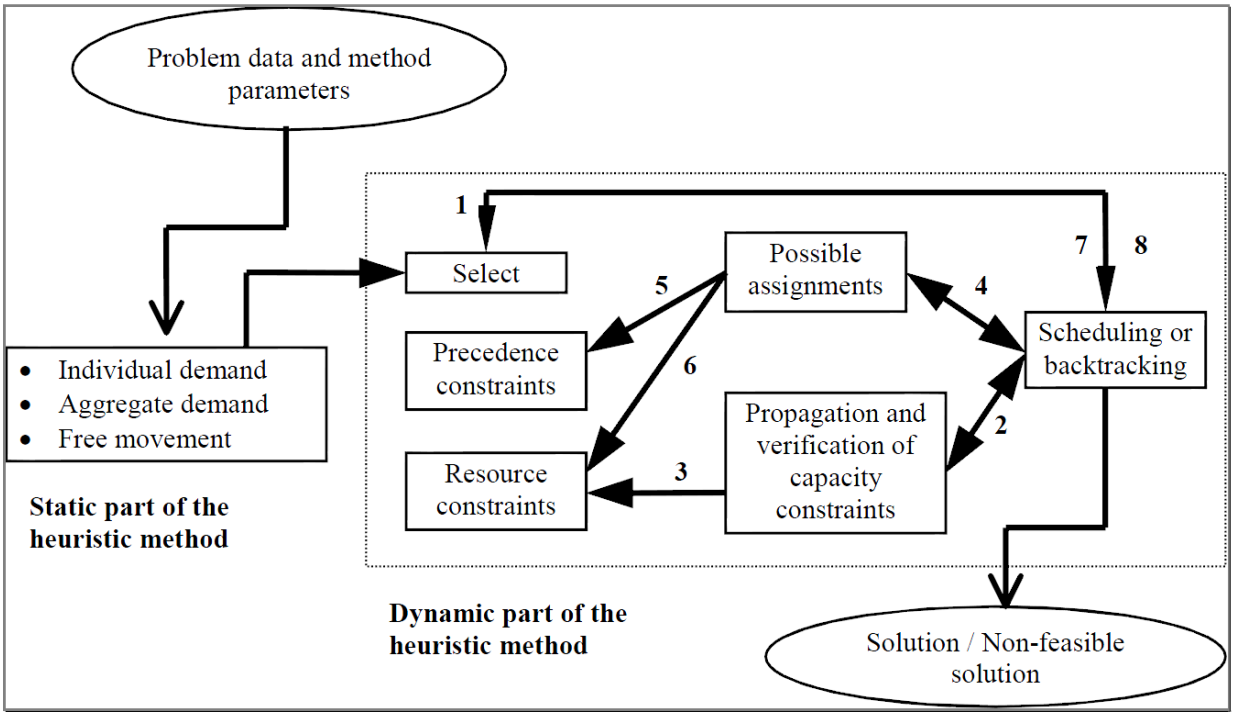
Ένα απλό παράδειγμα μιας χοντρικής πρόβλεψης είναι η εξής: υπολογίζουμε τη συνεισφορά του μερικού χρονοπρογραμματισμού στον αντικειμενικό στόχο και στους χρόνους προθεσμίας των διεργασιών που είναι απρογραμμάτιστες. Με βάση αυτές τις τιμές, οι κόμβοι σε ένα συγκεκριμένο επίπεδο μπορούν να συγκριθούν με αυτούς ενός άλλου επιπέδου και είναι δυνατό να γίνει μια συνολική αξιολόγηση. Κάθε φορά που ένας κόμβος πρέπει να υποβληθεί σε λεπτομερή αποτίμηση, όλες οι διεργασίες που δεν έχουν ακόμα προγραμματιστεί, προγραμματίζονται σύμφωνα με ένα σύνθετο κανόνα απόδοσης προτεραιότητας (dispatching rule). Ένας τέτοιος χρονοπρογραμματισμός μπορεί να παραχθεί γρήγορα, εφόσον απαιτεί μόνο ταξινόμηση.

Το αποτέλεσμα του είναι μια ένδειξη του εγγενούς δυναμικού του κόμβου. Αν ως αποτέλεσμα έχουμε ένα μεγάλο αριθμό διεργασιών, οι κόμβοι μπορεί να φιλτραριστούν εξετάζοντας πρώτα ένα μερικό χρονοπρογραμματισμό ο οποίος λαμβάνεται προγραμματίζοντας μόνο ένα υποσύνολο των διεργασιών που απομένουν με βάση ένα dispatching rule. Αυτός ο διευρυμένος μερικός χρονοπρογραμματισμός μπορεί να αποτιμηθεί και κατόπιν, βασιζόμενοι στην τιμή του, μπορούμε να αποφασίσουμε αν ένας κόμβος θα απορριφθεί ή όχι. Ένας κόμβος που διατηρείται μπορεί να αναλυθεί πιο λεπτομερειακά με το να προγραμματίζουμε όλες τις διεργασίες του που απομένουν, με τον σύνθετο dispatching rule. Η τιμή του αντικειμενικού στόχου αυτού του χρονοπρογραμματισμού αναπαριστά τότε ένα άνω όριο για τον καλύτερο χρονοπρογραμματισμό ανάμεσα στους απόγονους αυτού του κόμβου.

[Brucker P, 2001], [Jayamohan M. S., Rajendran C. 2000] [Φερλέμης 2012]

2.5.2 Ευρετικές Μέθοδοι (Heuristics)

Οι ευρετικές μέθοδοι χρησιμοποιούν μια σιωπηρή αναπαράσταση ενός χρονοδιαγράμματος στο οποίο κάθε γονίδιο στο χρωμόσωμα παριστάνει μια ευρετική για να χρησιμοποιηθεί σε κάθε στάδιο της δημιουργίας ενός χρονοδιαγράμματος (Emma Hart, Peter Ross). Μια ευρετική διαδικασία αναζήτησης (heuristic search procedure) μπορεί να προκύψει από τον συνδυασμό ευρετικών μεθόδων με αλγόριθμους αναζήτησης. Ο αλγόριθμος αναζήτησης, αναμφίβολα καθορίζει ένα διάστημα μερικών λύσεων του προβλήματος και το εξερευνά ωσότου βρεθεί μια αποδεκτή ή βέλτιστη λύση. Ο μηχανισμός που χρησιμοποιείται για να εκτελεστεί η αναζήτηση είναι επαναληπτικός και αρκετά απλός. Ο αλγόριθμος αρχίζει από μια προφανή ή από μια ήδη γνωστή μερική λύση και καθορίζει τις αλλαγές που μπορούν να γίνουν σε αυτή τη μερική λύση. Η ευρετική γνώση χρησιμοποιείται για να αποτιμηθούν οι πιθανές αλλαγές καθώς και η ποιότητα των μερικών λύσεων που προκύπτουν. Αυτή η γνώση βοηθά κατόπιν στην επιλογή εκ των μερικών λύσεων που προκύπτουν αυτών που αξίζουν να τροποποιηθούν διαδοχικά. Σε κάθε βήμα της διαδικασίας επίλυσης του προβλήματος, ο αλγόριθμος ενημερώνει πολλά μέρη της πληροφορίας στην οποία η ευρετική μέθοδος αναφέρεται και αλλάζει την πορεία της αναζήτησης, αποκλείοντας τις μερικές λύσεις από τις οποίες δεν είναι δυνατό να εξασφαλιστεί ολοκληρωμένη λύση ή μη ολοκληρωμένη λύση καλύτερη από ήδη γνωστές λύσεις [RajendranC., Ziegler H. 1999] [Φερλέμης 2012]. Στο σχήμα 2.5 απεικονίζεται η διαδικασία που ακολουθείται στην εφαρμογή των ευρετικών μεθόδων.



Σχήμα 2.5 [A. Garrido, M. A. Salido, F. Barber, M. A. López]

2.5.3 Μέθοδοι Τοπικής Αναζήτησης (Local Search Methods)

Μια σημαντική κατηγορία των βελτιωτικών αλγόριθμων είναι οι *μέθοδοι τοπικής αναζήτησης*. Μια τέτοια διαδικασία που βασίζεται σε τοπική αναζήτηση, σε αντίθεση με μια γενική διαδικασία αναζήτησης (global search procedure) δεν εγγυάται βέλτιστη λύση. Επιχειρεί να βρει μια καλύτερη λύση από την ήδη υπάρχουσα, μέσα από αναζήτηση στην *γειτονική περιοχή* (*neighborhood*) του υπάρχοντος χρονοπρογραμματισμού. Δυο χρονοπρογραμματισμοί λέμε ότι γειτνιάζουν αν ο ένας μπορεί να ληφθεί διαμέσου καλά ορισμένης τροποποίησης του άλλου [Brucker P, 2001]. Σε κάθε επανάληψη, μια μέθοδος τοπικής αναζήτησης επιτελεί μια αναζήτηση στη γειτονική περιοχή της λύσης και αξιολογεί τις γειτονικές λύσεις. Η διαδικασία είτε δέχεται είτε απορρίπτει μια υποψήφια λύση σύμφωνα με ένα δεδομένο κριτήριο αποδοχής – απόρριψης. Κάποιος μπορεί να συγκρίνει τις διάφορες μεθόδους τοπικής αναζήτησης σύμφωνα με τα ακόλουθα σχεδιαστικά κριτήρια:

- Την αναπαράσταση του χρονοπρογραμματισμού που χρειάζεται για την διαδικασία: Ένας χρονοπρογραμματισμός που αφορά μια απλή μηχανή χωρίς διακοπές (non – preemptive) μπορεί να καθοριστεί από τον απλό συνδυασμό (permutation) n διεργασιών. Ένας χρονοπρογραμματισμός για ένα πρόβλημα του τύπου job shop χωρίς διακοπές, μπορεί να καθοριστεί από m διαδοχικούς αλφαριθμητικούς χαρακτήρες όπου κάθε ένας αναπαριστά ένα συνδυασμό n επιμέρους εργασιών σε μια συγκεκριμένη μηχανή. Με βάση αυτές τις πληροφορίες, είναι δυνατό να υπολογιστούν οι χρόνοι έναρξης και ολοκλήρωσης όλων των επιμέρους εργασιών. Εντούτοις, όταν επιτρέπονται οι διακοπές, η μορφή της αναπαράστασης του χρονοπρογραμματισμού γίνεται σημαντικά πιο περίπλοκη.

- Τον σχεδιασμό της γειτονικής περιοχής: Η δομή της γειτονικής περιοχής είναι ένα πολύ σημαντικό μέρος μιας μεθόδου τοπικής αναζήτησης. Για μια απλή μηχανή, η γειτονική περιοχή ενός συγκεκριμένου χρονοπρογραμματισμού μπορεί να καθοριστεί απλά ως το σύνολο των χρονοπρογραμματισμών που μπορούν να ληφθούν αφού πρώτα εφαρμοστεί μια απλή ανταλλαγή μεταξύ γειτονικών ζευγαριών. Μια μεγαλύτερη γειτονική περιοχή για ένα χρονοπρογραμματισμό μιας απλής μηχανής μπορεί να οριστεί με το να ληφθεί αυθαίρετα μια διεργασία από τον χρονοπρογραμματισμό και να τοποθετηθεί σε άλλη θέση. Η γειτονική περιοχή ενός χρονοπρογραμματισμού σε ένα πιο πολύπλοκο περιβάλλον μηχανών είναι συνήθως πιο περίπλοκη.

- Την διαδικασία αναζήτησης στην γειτονική περιοχή: Η διαδικασία αναζήτησης μέσα σε μια γειτονική περιοχή μπορεί να γίνει με διαφορετικούς τρόπους. Ένας απλός τρόπος είναι να επιλέξουμε τυχαία κάποιους χρονοπρογραμματισμούς στη γειτονική περιοχή, να τους αποτιμήσουμε και να αποφασίσουμε ποιον θα αποδεχτούμε. Μπορούμε επίσης να εφαρμόσουμε μια πιο οργανωμένη αναζήτηση και να επιλέξουμε κατόπιν τους χρονοπρογραμματισμούς που εμφανίζονται ελπιδοφόροι.

- Το κριτήριο αποδοχής – απόρριψης: το κριτήριο αποδοχής – απόρριψης είναι συνήθως το σχεδιαστικό μέρος που διαφοροποιεί σημαντικά μια μέθοδο τοπικής αναζήτησης. Για παράδειγμα, η διαφορά ανάμεσα σε δυο γνωστές μεθόδους τοπικής αναζήτησης

(simulated annealing and tabu – search) έγκειται κυρίως στην διαφοροποίηση των κριτηρίων αυτών. Στην μεν μέθοδο simulated annealing, το κριτήριο αποδοχής – απόρριψης βασίζεται σε στοχαστική διαδικασία, ενώ στη δε μέθοδο tabu – search το κριτήριο βασίζεται σε ντετερμινιστική διαδικασία. [Brucker P, 2001] [Φερλέμης 2012]

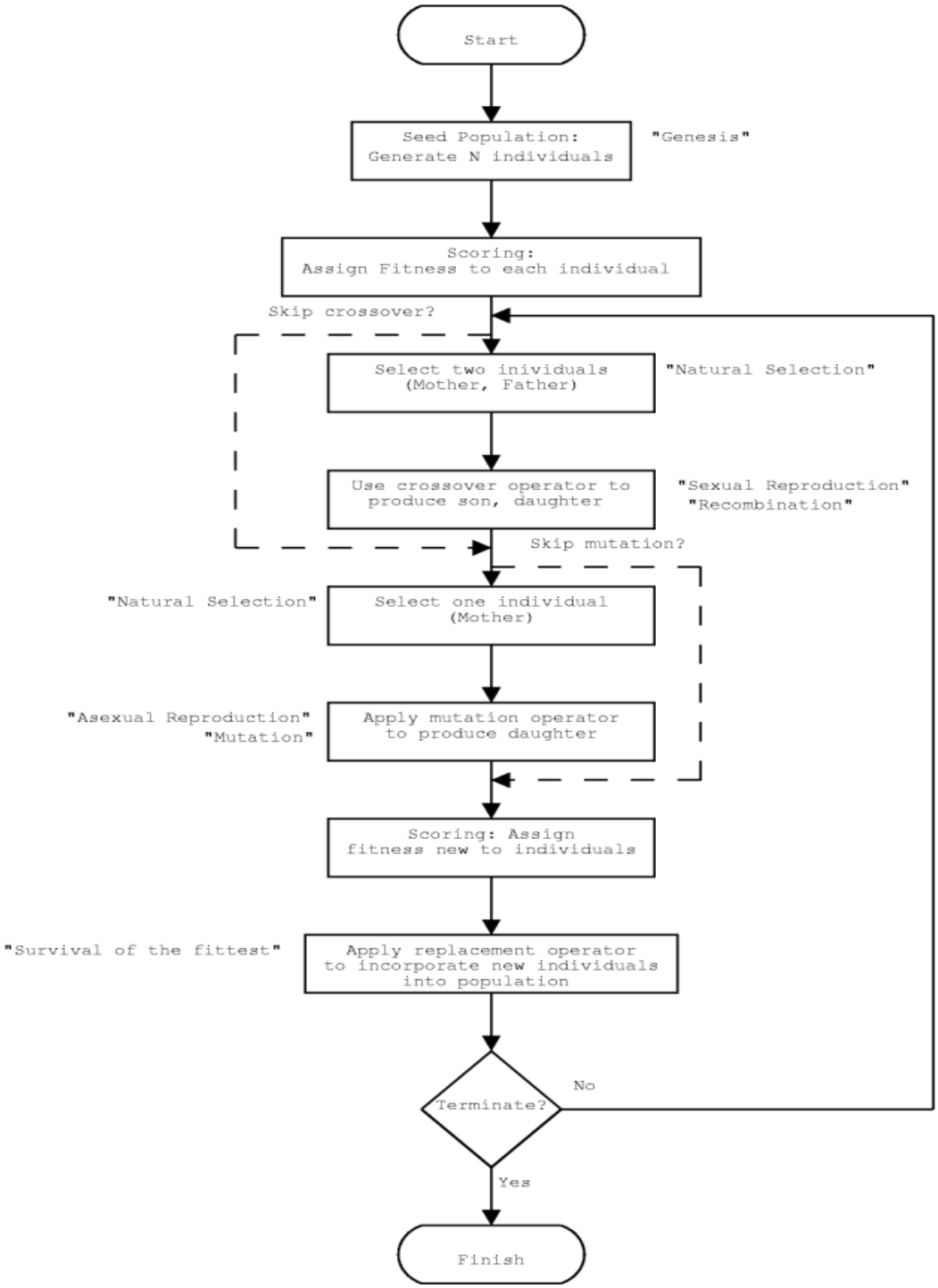
2.5.4 Γενετικοί Αλγόριθμοι (Genetic Algorithms)

Στον τομέα της επιστήμης των υπολογιστών της τεχνητής νοημοσύνης, ένας γενετικός αλγόριθμος (GA) είναι μια ευρετική αναζήτηση που μιμείται τη διαδικασία της φυσικής εξέλιξης. Η ευρετική (ονομάζεται επίσης μερικές φορές metaheuristic) συνήθως χρησιμοποιείται για να παράγει χρήσιμες λύσεις για τη βελτιστοποίηση και την αναζήτηση προβλημάτων. Οι γενετικοί αλγόριθμοι ανήκουν στην ευρύτερη κατηγορία των εξελικτικών αλγορίθμων (EA), οι οποίες παράγουν λύσεις σε προβλήματα βελτιστοποίησης με τη χρήση τεχνικών που εμπνέονται από τη φυσική εξέλιξη, όπως η κληρονομικότητα, η μετάλλαξη και επιλογή. Οι γενετικοί αλγόριθμοι βρίσκουν εφαρμογή στη βιοπληροφορική, φυλογενετική, υπολογιστική επιστήμη, την τεχνολογία, τα οικονομικά, τη χημεία, την κατασκευή, μαθηματικά, φυσική και σε άλλους τομείς.

Οι γενετικοί αλγόριθμοι, όταν εφαρμόζονται στον χρονοπρογραμματισμό, θεωρούν τις ακολουθίες ή τους διάφορους χρονοπρογραμματισμούς ως *άτομα (individuals)* ή μέλη ενός *πληθυσμού (population).* Κάθε άτομο χαρακτηρίζεται από την *υγεία του (fitness).* Η υγεία ενός ατόμου μετράται με την συνδυασμένη τιμή της αντικειμενικής συνάρτησης. Η διαδικασία δουλεύει επαναληπτικά και κάθε επανάληψη είναι μια *γενιά (generation).* Ο πληθυσμός μιας γενιάς συνίσταται από άτομα που επιζούν από την προηγούμενη γενιά συν τους νέους χρονοπρογραμματισμούς ή τα *παιδιά (children)* από την προηγούμενη γενιά. Το μέγεθος του πληθυσμού συνήθως παραμένει σταθερό από τη μια γενιά στην άλλη.

Τα παιδιά γεννώνται μέσω αναπαραγωγής και μεταβολής των ατόμων που ήταν μέρος της προηγούμενης γενιάς (οι *γονείς – parents*). Τα άτομα αναφέρονται ενίοτε και ως “χρωμοσώματα”. Σε ένα περιβάλλον με πολλές μηχανές, ένα χρωμόσωμα μπορεί να συνίσταται από υπό – χρωμοσώματα, κάθε ένα από τα οποία μπορεί να περιέχει την πληροφορία που αφορά την σειρά των διεργασιών σε μια μηχανή. Μια μεταβολή στο χρωμόσωμα ενός γονέα μπορεί να είναι ισοδύναμη με μια αμοιβαία ανταλλαγή γειτονικών ζευγαριών στην αντίστοιχη σειρά. Σε κάθε γενιά, οι περισσότερες διαδικασίες που καθορίζουν την σύνθεση της επόμενης γενιάς μπορεί να είναι σύνθετες και συνήθως να εξαρτώνται από το επίπεδο υγείας των ατόμων της παρούσας γενιάς. [Φερλέμης. 2012]

Ένα τυπικό διάγραμμα της εξέλιξης της εφαρμογής ενός γενετικού αλγορίθμου απεικονίζεται στο σχήμα 2.6.



Σχήμα 2.6 [http://gaul.sourceforge.net/]

2.5.5 Τεχνητή Νοημοσύνη και Νευρωνικά δίκτυα

Ο όρος **τεχνητή νοημοσύνη** (Artificial Intelligence) αναφέρεται στον κλάδο της επιστήμης υπολογιστών ο οποίος ασχολείται με τη σχεδίαση και την υλοποίηση υπολογιστικών

συστημάτων που μιμούνται στοιχεία της ανθρώπινης συμπεριφοράς τα οποία υπονοούν έστω και στοιχειώδη ευφυΐα: μάθηση, προσαρμοστικότητα, εξαγωγή συμπερασμάτων, κατανόηση από συμφραζόμενα, επίλυση προβλημάτων κλπ. Ο Τζον Μακάρθι όρισε τον τομέα αυτόν ως «επιστήμη και μεθοδολογία της δημιουργίας νοούντων μηχανών». Η τεχνητή νοημοσύνη αποτελεί σημείο τομής μεταξύ πολλών πεδίων όπως της επιστήμης υπολογιστών, της ψυχολογίας, της φιλοσοφίας, της νευρολογίας, της γλωσσολογίας και της επιστήμης μηχανικών, με στόχο τη σύνθεση ευφυούς συμπεριφοράς, με στοιχεία συλλογιστικής, μάθησης και προσαρμογής στο περιβάλλον, ενώ συνήθως εφαρμόζεται σε μηχανές ή υπολογιστές ειδικής κατασκευής.

Διαιρείται στη συμβολική τεχνητή νοημοσύνη, η οποία επιχειρεί να εξομοιώσει την ανθρώπινη νοημοσύνη αλγοριθμικά χρησιμοποιώντας σύμβολα και λογικούς κανόνες υψηλού επιπέδου, και στην υποσυμβολική τεχνητή νοημοσύνη, η οποία προσπαθεί να αναπαράγει την ανθρώπινη ευφυΐα χρησιμοποιώντας στοιχειώδη αριθμητικά μοντέλα που συνθέτουν επαγωγικά νοήμονες συμπεριφορές με τη διαδοχική αυτοοργάνωση απλούστερων δομικών συστατικών («συμπεριφορική τεχνητή νοημοσύνη»), προσομοιώνουν πραγματικές βιολογικές διαδικασίες όπως η εξέλιξη των ειδών και η λειτουργία του εγκεφάλου («υπολογιστική νοημοσύνη»), ή αποτελούν εφαρμογή στατιστικών μεθοδολογιών σε προβλήματα τεχνητής νοημοσύνης.

Η διάκριση σε συμβολικές και υποσυμβολικές προσεγγίσεις αφορά τον χαρακτήρα των χρησιμοποιούμενων εργαλείων, ενώ δεν είναι σπάνια η σύζευξη πολλαπλών προσεγγίσεων (διαφορετικών συμβολικών, υποσυμβολικών, ή ακόμα συμβολικών και υποσυμβολικών μεθόδων) κατά την προσπάθεια αντιμετώπισης ενός προβλήματος. Με βάση τον επιθυμητό επιστημονικό στόχο η τεχνητή νοημοσύνη κατηγοριοποιείται σε άλλου τύπου ευρείς τομείς, όπως επίλυση προβλημάτων, μηχανική μάθηση, ανακάλυψη γνώσης, συστήματα γνώσης κλπ. Επίσης υπάρχει επικάλυψη με συναφή επιστημονικά πεδία όπως η μηχανική όραση, η επεξεργασία φυσικής γλώσσας, η ρομποτική κλπ [Stuart Russel et al, Βλαχάβας κ.α.]. Δύο βασικές μεθοδολογίες της τεχνητής νοημοσύνης μπορούν να βρουν εφαρμογή στην επίλυση του δεδομένου προβλήματος, το οποίο μελετάμε: η προσέγγιση που ικανοποιεί δεδομένους περιορισμούς *(constraint satisfaction – CS)* και η μέθοδος των νευρωνικών δικτύων.

Το **νευρωνικό δίκτυο** είναι ένα δίκτυο από απλούς υπολογιστικούς κόμβους (νευρώνες, νευρώνια), διασυνδεδεμένους μεταξύ τους. Είναι εμπνευσμένο από το Κεντρικό Νευρικό Σύστημα (ΚΝΣ), το οποίο προσπαθεί να προσομοιώσει. Οι νευρώνες είναι τα δομικά στοιχεία του δικτύου. Κάθε τέτοιος κόμβος δέχεται ένα σύνολο αριθμητικών εισόδων από διαφορετικές πηγές (είτε από άλλους νευρώνες, είτε από το περιβάλλον), επιτελεί έναν υπολογισμό με βάση αυτές τις εισόδους και παράγει μία έξοδο. Η εν λόγω έξοδος είτε κατευθύνεται στο περιβάλλον, είτε τροφοδοτείται ως είσοδος σε άλλους νευρώνες του δικτύου.

Υπάρχουν τρεις τύποι νευρώνων: οι νευρώνες εισόδου, οι νευρώνες εξόδου και οι υπολογιστικοί νευρώνες ή κρυμμένοι νευρώνες. Οι νευρώνες εισόδου δεν επιτελούν κανέναν υπολογισμό, μεσολαβούν απλώς ανάμεσα στις περιβαλλοντικές εισόδους του δικτύου και στους υπολογιστικούς νευρώνες. Οι νευρώνες εξόδου διοχετεύουν στο περιβάλλον τις τελικές αριθμητικές εξόδους του δικτύου. Οι υπολογιστικοί νευρώνες πολλαπλασιάζουν κάθε είσοδό τους με το αντίστοιχο συναπτικό βάρος και υπολογίζουν το ολικό άθροισμα των γινομένων. Το

άθροισμα αυτό τροφοδοτείται ως όρισμα στη συνάρτηση ενεργοποίησης, την οποία υλοποιεί εσωτερικά κάθε κόμβος. Η τιμή που λαμβάνει η συνάρτηση για το εν λόγω όρισμα είναι και η έξοδος του νευρώνα για τις τρέχουσες εισόδους και βάρη.

Εάν x_{ki} είναι η i-οστή είσοδος του k νευρώνα, w_{ki}: το i-οστό συναπτικό βάρος του k νευρώνα και $\phi(\cdot)$ η συνάρτηση ενεργοποίησης του νευρωνικού δικτύου, τότε η έξοδος y_k του k νευρώνα δίνεται από την εξίσωση:

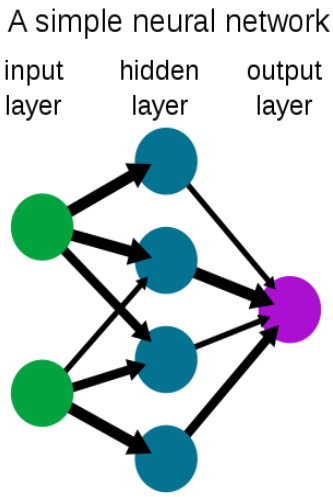
$$y_k = \phi\left(\sum_{i=0}^{N} x_{ki} w_{ki}\right)$$

Στον k-οστό νευρώνα υπάρχει ένα συναπτικό βάρος w_{k0} με ιδιαίτερη σημασία, το οποίο καλείται πόλωση ή κατώφλι (bias, threshold). Η τιμή της εισόδου του είναι *πάντα* η μονάδα, $x_{k0} = 1$. Εάν το συνολικό άθροισμα από τις υπόλοιπες εισόδους του νευρώνα είναι μεγαλύτερο από την τιμή αυτή, τότε ο νευρώνας ενεργοποιείται. Εάν είναι μικρότερο, τότε ο νευρώνας παραμένει ανενεργός. Η ιδέα προέκυψε από τα βιολογικά νευρικά κύτταρα.

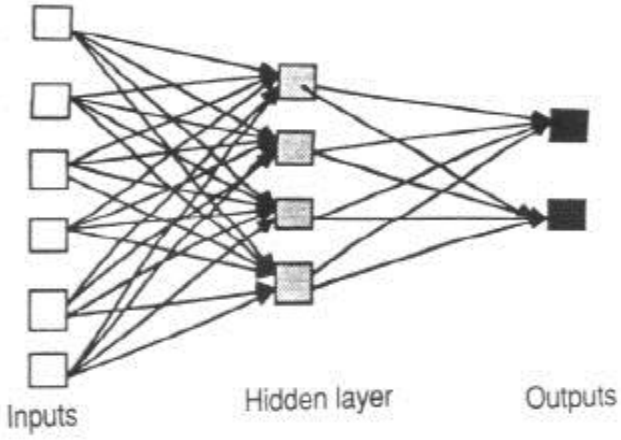
Όπως είναι φανερό, οι αριθμοί οι οποίοι συναποτελούν το διάνυσμα εισόδου (κάθε στοιχείο του διανύσματος τροφοδοτείται κατά τη λειτουργία του δικτύου σε έναν νευρώνα εισόδου), αλλά και οι αριθμοί οι οποίοι συναποτελούν το διάνυσμα εξόδου (κάθε στοιχείο του οποίου εμφανίζεται, μετά το πέρας του ολικού υπολογισμού, σε έναν νευρώνα εξόδου), περιγράφουν χαρακτηριστικά του προς επίλυση προβλήματος. Συνήθως αυτό που μας ενδιαφέρει είναι το δίκτυο να απεικονίζει με ορθό τρόπο διανύσματα εισόδου σε κατάλληλα διανύσματα εξόδου, το πρόβλημα δηλαδή είναι η υλοποίηση μίας συνάρτησης πολλαπλών μεταβλητών, κατά κανόνα περίπλοκης και με άγνωστο ακριβή τύπο. Τέτοιες απεικονίσεις έχουν εφαρμογή σε ποικιλία τομέων της επιστήμης και της τεχνολογίας, αφού λειτουργούν ως αριθμητικά μοντέλα για πολλά διαφορετικά ζητήματα. Το ίδιο δίκτυο μπορεί να υλοποιήσει άπειρες διαφορετικές απεικονίσεις, μία για κάθε διαφορετική επιλογή συνόλου συναπτικών βαρών.

Το κύριο χαρακτηριστικό των νευρωνικών δικτύων είναι η εγγενής ικανότητα μάθησης. Ως μάθηση μπορεί να οριστεί η σταδιακή βελτίωση της ικανότητας του δικτύου να επιλύει κάποιο πρόβλημα (π.χ. η σταδιακή προσέγγιση μίας συνάρτησης). Η μάθηση επιτυγχάνεται μέσω της εκπαίδευσης, μίας επαναληπτικής διαδικασίας σταδιακής προσαρμογής των παραμέτρων του δικτύου (συνήθως των βαρών και της πόλωσής του) σε τιμές κατάλληλες ώστε να επιλύεται με επαρκή επιτυχία το προς εξέταση πρόβλημα. Αφού ένα δίκτυο εκπαιδευτεί, οι παράμετροί του συνήθως «παγώνουν» στις κατάλληλες τιμές και από εκεί κι έπειτα είναι σε λειτουργική κατάσταση. Το ζητούμενο είναι το λειτουργικό δίκτυο να χαρακτηρίζεται από μία ικανότητα γενίκευσης: αυτό σημαίνει πως δίνει ορθές εξόδους για εισόδους καινοφανείς και διαφορετικές από αυτές με τις οποίες εκπαιδεύτηκε.

Παραδείγματα νευρωνικών δικτύων:



[el.wikipedia.org]



[http://biophysics.biol.uoa.gr]

2.6 Κανόνες Απόδοσης Προτεραιοτήτων (dispatching rules)

Οι κανόνες απόδοσης προτεραιοτήτων (dispatching rules) έχουν λάβει πολλή προσοχή από τους ερευνητές κατά τις τελευταίες δεκαετίες [Blackstone1982, Oliver & Chandrasekharan, 1997 Panwalkar & Wafik, 1977]. Σε γενικές γραμμές, κάθε φορά που μια μηχανή έχει απελευθερωθεί, μια δουλειά με την υψηλότερη προτεραιότητα στην ουρά έχει επιλεγεί για να υποβληθεί σε επεξεργασία σε μια μηχανή ή το κέντρο εργασίας. [Joc Cing Tay, Nhu Binh Ho, 2008].

Μερικοί από αυτούς τους κανόνες είναι:

- **Service in random order (SIRO) rule**. Σύμφωνα με αυτό τον κανόνα, όποτε μια μηχανή καθίσταται διαθέσιμη, η επόμενη διεργασία επιλέγεται στη τύχη, ανάμεσα από αυτές που αναμένουν για επεξεργασία. Δεν γίνεται προσπάθεια για βελτιστοποίηση κανενός αντικειμενικού στόχου

- **Earliest release date first (ERD) rule.** Αυτός ο κανόνας είναι κατά κάποιο τρόπο ισοδύναμος με τον κανόνα first-come-first-served. Κατά μια έννοια ελαχιστοποιεί τις αποκλίσεις στους χρόνους αναμονής των διεργασιών σε μια μηχανή.

- **Earliest due date first (EDD) rule.** Όποτε μια μηχανή καθίσταται διαθέσιμη, η διεργασία με τον νωρίτερο χρόνο προθεσμίας επιλέγεται για να επεξεργαστεί. Αυτός ο κανόνας τείνει να ελαχιστοποιεί την μέγιστη καθυστέρηση (maximum lateness) ανάμεσα στις διεργασίες που αναμένουν για επεξεργασία. Στην πράξη, σε ένα περιβάλλον με μια απλή μηχανή, με n διεργασίες διαθέσιμες την χρονική στιγμή 0, ο κανόνας EDD όντως ελαχιστοποιεί την μέγιστη καθυστέρηση.

- **Minimum slack first (MS) rule.** Σύμφωνα με τον κανόνα, οι εργασίες διατάσσονται σύμφωνα με το κριτήριο max (dj – pj – t, 0) όπου dj είναι ο χρόνος προθεσμίας, pj ο χρόνος επεξεργασίας και t ο τρέχων χρόνος. Λόγω αυτού του ορισμού συνεπάγεται πως σε κάποια χρονική στιγμή η εργασία j μπορεί να έχει μεγαλύτερη προτεραιότητα από τη εργασία k ενώ σε κάποια επόμενη χρονική στιγμή μπορεί να έχουν την ίδια προτεραιότητα.

- **Weighted shortest processing time first (WSPT) rule.** Όποτε μια μηχανή καθίσταται διαθέσιμη, η διεργασία με τον μεγαλύτερο λόγο βάρους (w_j) προς χρόνο επεξεργασίας (p_j) προγραμματίζεται για τη συνέχεια. Έτσι οι διεργασίες προγραμματίζονται με φθίνουσα σειρά w_j / p_j. Ο κανόνας αυτός τείνει να μειώνει το σταθμισμένο (weighted) άθροισμα των χρόνων ολοκλήρωσης, δηλ. το $\Sigma w_j C_j$. Σε περιβάλλον με μια απλή μηχανή με n διεργασίες διαθέσιμες την χρονική στιγμή 0, αυτό όντως ισχύει. Όταν όλα τα βάρη είναι ίσα, ο κανόνας WSPT ανάγεται στον κανόνα **shortest processing time first (SPT).**

- **Longest processing time first (LPT) rule.** Αυτός ο κανόνας διατάσσει τις διεργασίες σε φθίνουσα σειρά χρόνων επεξεργασίας. Όταν έχουμε μηχανές σε παράλληλη διάταξη αυτός ο κανόνας τείνει να εξισορροπήσει το φόρτο εργασίας σε κάθε μηχανή. Η λογική που ακολουθεί είναι η εξής: είναι επωφελές να κρατάμε τις διεργασίες με μικρούς χρόνους επεξεργασίας για αργότερα γιατί αυτές οι διεργασίες είναι χρήσιμες προς το τέλος για την εξισορρόπηση του φόρτου των μηχανών. Αφού καθορίσουμε την ανάθεση των διεργασιών στις μηχανές, οι διεργασίες σε οποιαδήποτε μηχανή μπορούν να αλλάξουν ξανά σειρά χωρίς να επηρεαστεί η ισορροπία του φόρτου των μηχανών.

- **Shortest setup time first (SST) rule.** Όποτε μια μηχανή καθίσταται διαθέσιμη, ο κανόνας αυτός επιλέγει για επεξεργασία την διεργασία με τον συντομότερο χρόνο προετοιμασίας (setup time).

- **Critical path (CP) rule.** Ο κανόνας αυτός χρησιμοποιείται όταν οι διεργασίες υπόκεινται σε περιορισμούς προτεραιότητας (precedence constraints). Από τον αντίστοιχο γράφο που εκφράζει αυτούς τους περιορισμούς, επιλέγεται η διεργασία που είναι στη κεφαλή της μακρύτερης αλληλουχίας των χρόνων επεξεργασίας.

- **Largest number of successors (LNS) rule.** Ο κανόνας αυτός μπορεί να χρησιμοποιηθεί επίσης όταν οι διεργασίες υπόκεινται σε περιορισμούς προτεραιότητας. Επιλέγει την διεργασία που έχει τον μεγαλύτερο αριθμό διεργασιών που να την ακολουθούν.

  [Brucker P, 2001] [Jayamohan M. S., Rajendran C. 2000] [Φερλέμης 2012]

2.7 Εισαγωγή στα Χρονισμένα Αυτόματα

ΟΡΙΣΜΟΣ 1: Χρονισμένα Αυτόματα.

Χρονισμένο αυτόματο είναι ένα διάνυσμα A = (Q, C, s, f, Δ} όπου το Q είναι ένα πεπερασμένο σύνολο από καταστάσεις, το C είναι ένα πεπερασμένο σύνολο από χρονικές μεταβλητές (ρολόγια) και το Δ είναι μια σχέση μεταβολών που αποτελείται από στοιχεία της μορφής (q, φ, ρ, q΄) όπου q και q΄ είναι καταστάσεις, $\rho \subseteq C$ και φ (η συνθήκη ελέγχου της μεταβολής) είναι ένας αλγεβρικός συνδυασμός τύπων της μορφής (CεI), για ορισμένα ρολόγια C και ορισμένα διαστήματα I ακέραια περιορισμένα. Οι καταστάσεις s και f είναι η αρχική και η τελική κατάσταση αντίστοιχα.

ΟΡΙΣΜΟΣ 2: Εκτίμηση Χρόνου (Clock Valuation).

Η εκτίμηση χρόνου (clock valuation) είναι μια συνάρτηση $V: C \rightarrow R_+$. Ορίζουμε το σύνολο των χρονικών εκτιμήσεων με Η. Μια διαμόρφωση ενός αυτόματου είναι επομένως ένα ζεύγος $(q, v) \in QxH$ το οποίο αποτελείται από μια διακριτή κατάσταση και μια εκτίμηση χρόνου. Κάθε υποσύνολο $\rho \subseteq C$ εισάγει μια συνάρτηση μηδενισμού.

$Reset_\rho: H \rightarrow H$ ορίζεται για κάθε εκτίμηση χρόνου V και κάθε μεταβλητή χρόνου C ως:

$$\text{Reset}_\rho \, v \, (c) \begin{cases} 0 & if \quad c \in \rho \\ v(c) & if \quad c \notin \rho \end{cases}$$

Αυτό σημαίνει ότι η $Reset_\rho$ μηδενίζει όλα τα ρολόγια στο ρ και αφήνει τα υπόλοιπα αμετάβλητα. Χρησιμοποιούμε το 1 για να δηλώσουμε το μοναδιαίο διάνυσμα (1,..., 1) και 0 για το μηδενικό διάνυσμα.

Ένα βήμα σε ένα αυτόματο μπορεί να είναι:

- Διακριτό βήμα: $(q, v) \overset{0}{\rightarrow} (q', v')$ όπου υπάρχει $S = (q, \varphi, \rho, q') \in \Delta$ τέτοιο ώστε το V ικανοποιεί το φ και $v' = Reset_\rho (v)$
- Χρονικό βήμα: $(q, v) \overset{t}{\rightarrow} (q, v+t1)$, $t \in R_+$

Μια εκτέλεση (run) του αυτόματου το οποίο ξεκινάει από μια διαμόρφωση (q_o, v_o) είναι μια πεπερασμένη ακολουθία βημάτων

$$\xi: (q_o, v_o) \overset{t_1}{\rightarrow} (q_1, v_1) \overset{t_1}{\rightarrow} \ldots \overset{t_n}{\rightarrow} (q_n, v_n).$$

Το λογικό μήκος μιας τέτοιας εκτέλεσης είναι n και το μετρικό μήκος είναι $t_1 + t_2 + \ldots + t_n$. Πρέπει να σημειωθεί ότι οι διακριτές μεταβολές δεν καταναλώνουν χρόνο.

Ορισμένες φορές είναι χρήσιμο να διευρύνουμε το χρονισμένο αυτόματο A με ένα επιπρόσθετο ρολόι t το οποίο είναι ενεργό σε κάθε κατάσταση και δεν μηδενίζεται ποτέ. Το αυτόματο που προκύπτει συμβολίζεται με A΄ και ονομάζεται διευρυμένο αυτόματο του A και οι εκτελέσεις του

ονομάζονται διευρυμένες εκτελέσεις του A΄. Καθώς το A δείχνει πάντα τον απόλυτο χρόνο το (q, v, t) είναι δυνατόν να προσεγγιστεί στο A΄ αν το (q, v) μπορεί να προσεγγιστεί το A σε χρόνο t. Σημειώνεται ότι οι μεταβολές "έναρξη" και "λήξη" οι οποίες χρησιμοποιήθηκαν στις προηγούμενες παραγράφους δεν χρησιμοποιούνται στη συνέχεια καθώς στη θέση τους χρησιμοποιείται η χρονική τους διάρκεια η οποία είναι μηδέν. Αν και από τον ορισμό των ρολογιών αυτά εμφανίζονται ομοιόμορφα με παράγωγο 1 σε όλες τις καταστάσεις υπάρχουν καταστάσεις όπου τιμές συγκεκριμένων ρολογιών δεν είναι σημαντικές γιατί σε όλες τις διαδρομές οι οποίες ξεκινούν από αυτές τις καταστάσεις δεν ελέγχονται πριν μηδενιστούν (π. χ το ρολόι X_1 στην κατάσταση $(\overline{p}_1, p_2)$). Λέμε ότι ένα ρολόι είναι ανενεργό σε μια τέτοια κατάσταση και αντί να καταγράψουμε την τιμή του χρησιμοποιούμε το σύμβολο.

ΟΡΙΣΜΟΣ 3: Χρονισμένο αυτόματο για μια εργασία (Timed automaton for a task)

Για κάθε εργασία p ∈ P το αντίστοιχο συνδεόμενο χρονισμένο αυτόματο είναι A = (Q, {c}, I, Δ,

s, f) με Q = {p, p^- p, } όπου η αρχική κατάσταση είναι p^- και η τελική κατάσταση είναι p. Οι σταθερές παραμένουν για το p^- και για p και c<<d (p) το p. Η σχέση μετάβασης Δ αποτελείται από τις δύο μεταπτώσεις:

αρχή :
$$\left(\overline{p}, \bigwedge_{p' \in \Pi(p)} \underline{p'}, \{c\}, p \right)$$

και

τέλος
$$(p, c = d(p), \emptyset, \underline{p}).$$

Σε αυτό το σημείο σημειώνεται ότι το ρολόι είναι ενεργό μόνο στην κατάσταση p, όπου μετρά το χρόνο που έχει παρέλθει από τότε που άρχισε την εκτέλεση ενώ η αξία της τιμής p^- δεν επηρεάζει το μέλλον και ως εκ τούτου δεν χρειάζεται να είναι μέρος της κατάστασης του συστήματος.

ΟΡΙΣΜΟΣ 4: Επόμενα (Successors)

Έστω A = (Q, C, s, f, Δ) ένα χρονικό αυτόματο και (q, Z) μια συμβολική κατάσταση.

- Το χρονικό επόμενο του (q, Z) είναι ένα σύνολο από διαμορφώσεις οι οποίες μπορούν να προσεγγιστούν από το (q, Z) αφήνοντας να περάσει χρόνος:

  $$Post^t (q, Z) = \{ (q, z+r1): z \in Z, r \geq 0\}$$

  Λέμε ότι το (q, Z) είναι χρονικά κλειστό εάν (q, Z) = $Post^t$ (q, Z).

- Τα επόμενα δ-μεταβολής (δ-transition successor) του (q, Z) είναι ένα σύνολο από διαμορφώσεις οι οποίες είναι δυνατό να προσεγγιστούν από το (q, Z) ακολουθώντας τη μεταβολή δ = (q, φ, ρ, q΄) ∈ Δ:

$$Post^t (q, Z) = \{ (q, Reset_p (z)): z \in Z \cap \varphi\}$$

- Τα δ-επόμενα (δ-successors) μιας χρονικά κλειστής συμβολικής κατάστασης (q, Z) είναι ένα σύνολο από διαμορφώσεις οι οποίες είναι δυνατό να προσεγγιστούν από μία δ-μεταβολή η οποία ακολουθείται από πάροδο χρόνου:

$$Succ^{\delta} (q, Z) = Post^t (Post^{\delta} (q, Z)).$$

- Τα επόμενα του (q, Z) είναι το σύνολο όλων των δ-επόμενών του:

$$Succ (q, Z) = \bigcup_{\delta \in \Delta} (Succ^{\delta} (q, Z))$$

Εξοπλισμένοι με αυτές τις λειτουργίες (οι οποίες μετασχηματίζουν τις ζώνες σε ζώνες) μπορούμε να λύσουμε προβλήματα προσεγγισιμότητας για χρονισμένα αυτόματα χρησιμοποιώντας αλγόριθμους ανίχνευσης γράφων οι οποίοι δουλεύουν στο "γράφο εξομοίωσης", ένα γράφο του οποίου οι κόμβοι είναι συμβολικές καταστάσεις. Αυτή η προσέγγιση για την επαλήθευση των χρονισμένων αυτομάτων αρχικά προτάθηκε και εφαρμόστηκε στο πρόγραμμα μοντελοποίησης KRONOS.

Αλγόριθμος (Forward Reachability for Acyclic Timed Automata)

Waiting:={$Post^t${(s,o)}};
while Waiting ≠ ∅ do
    Pick (q, Z) ∈ Waiting;
    For every (q΄, Z΄) ∈ Succ(q, Z);
      Insert (q΄, Z΄) into Waiting;
    Remove (q, Z) from Waiting

Καθώς το αυτόματο είναι μη κυκλικό ο αλγόριθμος θα τερματιστεί ακόμα και αν δεν κρατήσουμε μια λίστα με τις καταστάσεις που έχουν ερευνηθεί. Παρόλα αυτά για λόγους απόδοσης τέτοιοι έλεγχοι είναι σημαντικοί ειδικά για όπου υπάρχουν πολλές διαδρομές οι οποίες οδηγούν στην ίδια διακριτή κατάσταση.

ΟΡΙΣΜΟΣ 5: Προαπαιτούμενα (Predecessors)

Έστω A = (Q, C, s, f, Δ) ένα χρονισμένο αυτόματο και (q, Z) μια συμβολική κατάσταση.

- Το χρονικό προαπαιτούμενο του (q, Z) είναι ένα σύνολο από διαμορφώσεις οι οποίες μπορούν να προσεγγιστούν από το (q, Z) αφήνοντας να περάσει χρόνος:

$$Pre^t (q, Z) = \{ (q, v): v+r1 \in Z, r \geq 0\}$$

  Λέμε ότι το (q, Z) είναι χρονικά κλειστό εάν (q, Z) = Pre^t (q, Z).

- Τα προαπαιτούμενα δ-μεταβολής (δ-transition predecessors) του (q, Z) είναι ένα σύνολο από διαμορφώσεις οι οποίες είναι δυνατό να προσεγγιστούν από το (q, Z) ακολουθώντας τη μεταβολή δ = (q΄, φ, ρ, q) ∈ Δ:

$$Pre^t(q, Z) = \{ (q', v'): v' \in Reset_\rho^{-1} \cap \varphi\}$$

- Τα προαπαιτούμενα του (q, Z) είναι το σύνολο από όλες τις διαμορφώσεις από τις οποίες το (q, Z) μπορεί να προσεγγιστεί από οποιαδήποτε μεταβολή δ η οποία ακολουθείται από την πάροδο του χρόνου:

$$Pre(q, Z) = \bigcup_{\delta \in \Delta} Pre^t(Pre^\delta(q, Z))$$

Αλγόριθμος 5 (Backward Reachability for Acyclic Timed Automata)

```
Waiting:={(f,H)};
while Waiting ≠ ∅ do
    pick (q, Z) ∈ Waiting;
    For every (q΄, Z΄) ∈ Pre(q, Z);
      Insert (q΄, Z΄) into Waiting;
    Remove (q, Z) from Waiting;
End
```

[Λυγούρας –Σκλαβενίτη 2004] [Scheduling with timed automata, Yasmina Abdeddaim, Eugene Asarin, Oded Maler]

2.8 Προσθήκη Χρόνου και Κόστους στα Αυτόματα

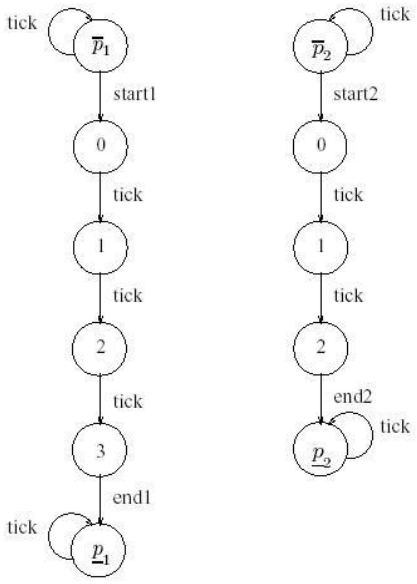
2.8.1 Προσθήκη χρόνου στα αυτόματα

Παρά το γεγονός ότι τα αυτόματα μοντέλα πεπερασμένων καταστάσεων παράγουν μια ποιοτική εικόνα για το ποια λειτουργία προηγείται της άλλης σε ένα σύστημα παραγωγής, μειονεκτούν διότι δεν να προσδιορίσουμε τον ακριβή χρόνο μεταξύ δύο λειτουργιών. Σε πολλές εφαρμογές, όπως είναι ο προγραμματισμός σε πραγματικό χρόνο ή η χρονική ανάλυση κυκλωμάτων, η ορθότητα των συστημάτων εξαρτάται από τις σχετικές ταχύτητες του συστήματος και του περιβάλλοντός του. Πάρα πολλές προτάσεις έγιναν στα τέλη της δεκαετίας του 80 για επέκταση της μεθοδολογίας επαλήθευσης σε μοντέλα ποσοτικού χρόνου και από αυτές το μοντέλο των Alur και Dill φάνηκε πολύ χρήσιμο. Αυτό το μοντέλο είναι αρκετό για να καλύψει όλα τα χρονικά προβλήματα πρακτικού ενδιαφέροντος, αλλά τα βασικά προβλήματα προσεγγισιμότητας για αυτό, μπορούν να λυθούν αλγοριθμικά χρησιμοποιώντας προεκτάσεις των αλγορίθμων αναζήτησης γράφων.

Έστω 2 διαδικασίες P_1 και P_2, με χρόνους εκτέλεσης 3 και 2 μονάδες του χρόνου αντίστοιχα. Κάθε διαδικασία μπορεί να βρίσκεται σε 3 βασικές ‘φάσεις’: μπορεί να περιμένει να ξεκινήσει, μπορεί να είναι ενεργή (δηλαδή να εκτελείται) και τέλος μπορεί να έχει τελειώσει, αφού έχει

περάσει αρκετός χρόνος στην ενεργή φάση. Η κατάσταση της διαδικασίας στην ενεργή φάση καθορίζεται από το ποσό του χρόνου που έχει περάσει από την έναρξή της. Έτσι, μπορούμε να μοντελοποιήσουμε τις διαδικασίες χρησιμοποιώντας τα αυτόματα A_1 και A_2 του σχήματος 2.7. Οι καταστάσεις του A_1 είναι οι $\{\overline{p}_1, 0, 1, 2, 3, \underline{p}_1\}$, όπου $\overline{p}_1$ είναι η αρχική κατάσταση που δηλώνει ότι η διαδικασία είναι ακόμα ανενεργή και $\underline{p}_1$ είναι η τελική κατάσταση. Οι άλλες καταστάσεις αναπαριστούν το ποσό του χρόνου που έχει περάσει στην ενεργή φάση. Το αυτόματο έχει 2 είδη μεταβάσεων, τις άμεσες μεταβάσεις όπως είναι οι 'start' και 'end', οι οποίες δεν διαρκούν καθόλου, και μια ειδική μετάβαση, 'tick', που δείχνει το πέρασμα της μονάδας χρόνου. Οι συμπεριφορές των διαδικασιών είναι οι εκτελέσεις του αυτομάτου που αποτελούνται από ακολουθίες καταστάσεων και από τα 2 είδη μεταβάσεων. Για παράδειγμα, μια συμπεριφορά όπου το P_1 περιμένει μια μονάδα του χρόνου και μετά ξεκινά, δίδεται από την εξής εκτέλεση, όπου η διάρκεια ισούται με τον αριθμό των μεταβάσεων 'tick':

$$\overline{p}_1 \xrightarrow{tick} \overline{p}_1 \xrightarrow{start1} 0 \xrightarrow{tick} 1 \xrightarrow{tick} 2 \xrightarrow{tick} 3 \xrightarrow{end1} \underline{p}_1$$



Σχήμα 2.7 - Τα 2 αυτόματα A_1 και A_2 που αντιστοιχούν στα P_1 και P_2

Στην περίπτωση που 2 ή περισσότερα αυτόματα 'τρέχουν' παράλληλα, το κάθε αυτόματο μπορεί να κάνει τις άμεσες μεταβάσεις του ανεξάρτητα, αλλά οι μεταβάσεις 'tick' γίνονται συγχρονισμένα. Δηλαδή, αν μια διαδικασία κάνει μια τέτοια μετάβαση, τότε και όλες οι άλλες διαδικασίες θα κάνουν την ίδια. Το αποτέλεσμα αυτής της μετάβασης σε μια ενεργή διαδικασία που βρίσκεται σε κατάσταση i είναι να την μετακινήσει στην i + 1.

Στο σχήμα 2.7 φαίνεται το αυτόματο $A = A_1 \| A_2$. Ξεκινά από μια αρχική κατάσταση $(\underline{p}_1, \underline{p}_2)$ όπου οι 2 διαδικασίες περιμένουν. Κάθε διαδικασία μπορεί να αρχίσει να εκτελείται μετά από οποιαδήποτε αριθμό 'ticks' και έτσι όλες οι πιθανές συμπεριφορές του συστήματος αντιστοιχούν στην εκτέλεση του A.

Για παράδειγμα, η συμπεριφορά όπου και οι 2 διαδικασίες περιμένουν 2 μονάδες χρόνου και μετά ξεκινούν ταυτόχρονα είναι η εξής:

$$(\overline{p}_1, \overline{p}_2) \xrightarrow{tick} (\overline{p}_1, \overline{p}_2) \xrightarrow{tick} (\overline{p}_1, \overline{p}_2) \xrightarrow{start1} (0, \overline{p}_2) \xrightarrow{start2} (0,0) \xrightarrow{tick}$$
$$(1,1) \xrightarrow{tick} (2,2) \xrightarrow{end2} (2, \underline{p}_2) \xrightarrow{tick} (3, \underline{p}_2) \xrightarrow{end1} (\underline{p}_1, \underline{p}_2)$$

Το αποτέλεσμα που προκύπτει όταν η P_1 ξεκινά αμέσως και η P_2 2 μονάδες χρόνου αργότερα, φαίνεται ως εξής:

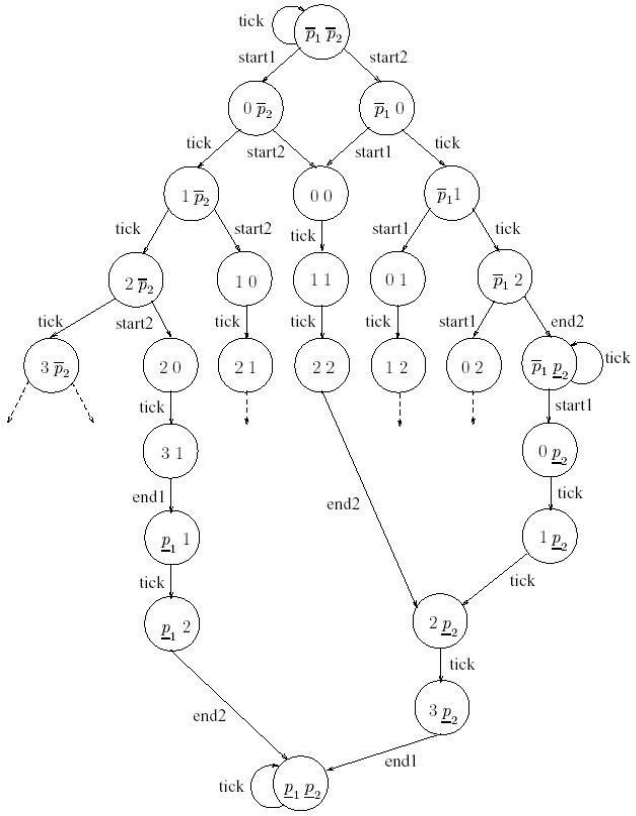
$$(\overline{p}_1, \overline{p}_2) \xrightarrow{start1} (0, \overline{p}_2) \xrightarrow{tick} (1, \overline{p}_2) \xrightarrow{tick} (2, \overline{p}_2) \xrightarrow{start2}$$
$$(2,0) \xrightarrow{tick} (3,1) \xrightarrow{end1} (\underline{p}_1, 1) \xrightarrow{tick} (\underline{p}_1, 2) \xrightarrow{end2} (\underline{p}_1, \underline{p}_2)$$

Αν η P_2 ξεκινά αμέσως και η P_1 1 μονάδα χρόνου αφού τελειώσει η P_2, τότε έχουμε την εξής απόκριση:

$$(\overline{p}_1, \overline{p}_2) \xrightarrow{start2} (\overline{p}_1, 0) \xrightarrow{tick} (\overline{p}_1, 1) \xrightarrow{tick} (\overline{p}_1, 2) \xrightarrow{end2} (\overline{p}_1, \underline{p}_2) \xrightarrow{tick}$$
$$(\overline{p}_1, \underline{p}_2) \xrightarrow{start1} (0, \underline{p}_2) \xrightarrow{tick} (1, \underline{p}_2) \xrightarrow{tick} (2, \underline{p}_2) \xrightarrow{tick} (3, \underline{p}_2) \xrightarrow{end1} (\underline{p}_1, \underline{p}_2)$$

Η διάρκεια των παραπάνω εκτελέσεων, δηλαδή ο χρόνος από την αρχική κατάσταση μέχρι την ($\underline{p}_1$, $\underline{p}_2$), είναι απλά ίση με τον αριθμό των 'ticks' (5, 4 και 3 μονάδες χρόνου αντίστοιχα). Καθώς κάθε διαδικασία μπορεί να επιλέξει την πραγματοποίηση της άμεσης μετάβασης ανεξάρτητα, είναι πιθανοί πολλοί συνδυασμοί τιμών χρόνου, όπου κάθε συνδυασμός αντιστοιχεί και σε μια τέτοια επιλογή. Αυτό μπορεί να οδηγήσει σε τεράστια αύξηση των καταστάσεων καθώς αυξάνεται ο αριθμός των διαδικασιών.

Επιπλέον, κάθε αλλαγή στην κλίμακα χρόνου (επιτρέποντας για παράδειγμα τα γεγονότα να συμβαίνουν σε διαστήματα χρόνου που είναι πολλαπλάσια του ½) θα αυξήσει σημαντικά τον αριθμό των καταστάσεων σε κάθε αυτόματο. Παρόλα αυτά, το πλεονέκτημα αυτής της αναπαράστασης είναι ότι μας επιτρέπει να παραμείνουμε στο γνώριμο πεδίο των αυτομάτων και να εφαρμόσουμε τους συνηθισμένους αλγορίθμους συντομότερου μονοπατιού και αλγορίθμους προσεγγισιμότητας, δίδοντας συντελεστή βαρύτητας 1 για τις μεταβάσεις 'tick' και 0 για τις άμεσες.

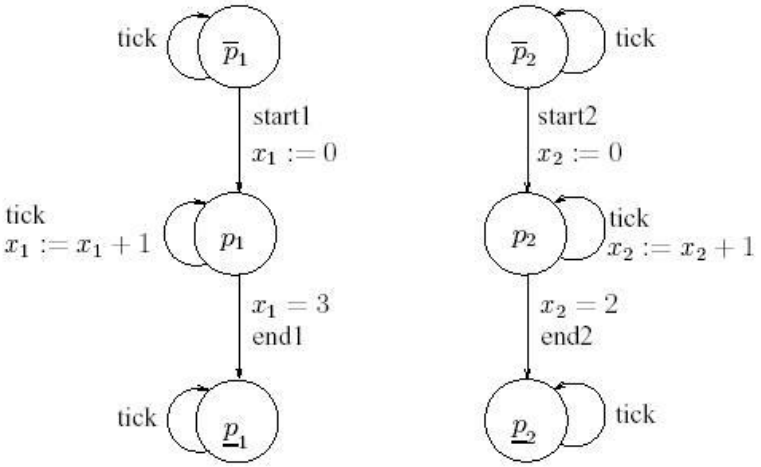


Σχήμα 2.8 - Το συνολικό αυτόματο $A = A_1 \| A_2$

2.8.2 Χρονικές μεταβλητές

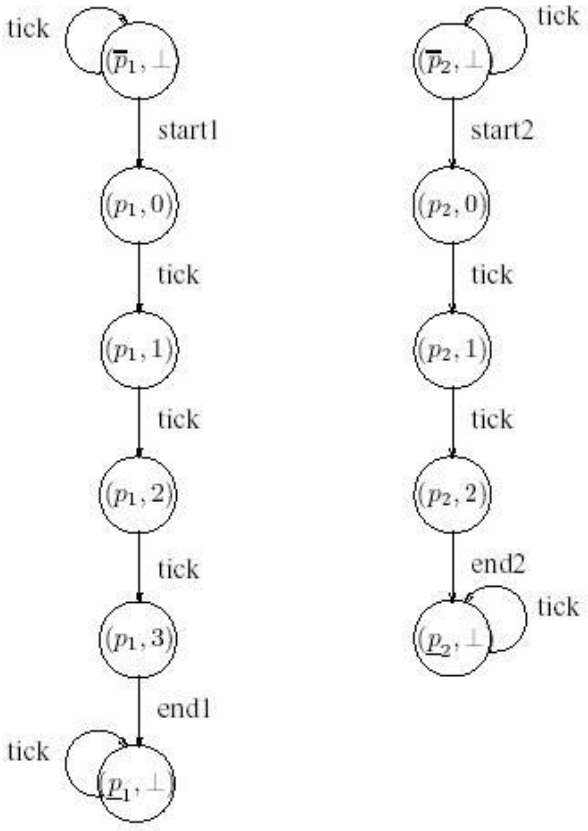
Μια πιο συμπαγής αναπαράσταση αυτών των αυτομάτων μπορεί να γίνει χρησιμοποιώντας κάποιες ειδικές μεταβλητές που δηλώνουν το πέρασμα του χρόνου, αντί να παίρνουμε μέσα από τις καταστάσεις τον χρόνο που έχει περάσει. Αυτές, είναι χρονικές μεταβλητές ή μετρητές που μηδενίζονται όταν εισάγεται μια ενεργή κατάσταση, αυξάνονται κατά μια μονάδα με κάθε 'tick' και η τιμή τους ελέγχεται στις μεταβάσεις που γίνονται από τις ενεργές καταστάσεις. Στην παρακάτω απεικόνιση φαίνεται πώς αυτό γίνεται για τα αυτόματα του σχήματος 2.8, έχοντας προσθέσει τις χρονικές μεταβλητές X_1 και X_2. Το νέο αυτό αυτόματο χαρακτηρίζεται από ένα ζεύγος της μορφής (q, **v**) όπου q είναι μια κατάσταση και **v** μια τιμή για τις μεταβλητές. Τέτοιες χρονικές μεταβλητές μπορούν να κυμαίνονται σε μη-αρνητικές ακέραιες τιμές, και με ειδική τιμή $\perp$ που δηλώνει ότι ο μετρητής δεν είναι ενεργός σε μια κατάσταση. Μια εκτέλεση του A'_1 θα είναι ως εξής:

$$(\overline{p}_1, \perp) \xrightarrow{tick} (\overline{p}_1, \perp) \xrightarrow{atart1} (p_1, 0) \xrightarrow{tick} (p_1, 1) \xrightarrow{tick} (p_1, 2) \xrightarrow{tick} (p_1, 3) \xrightarrow{end1} (\underline{p}_1, \perp)$$



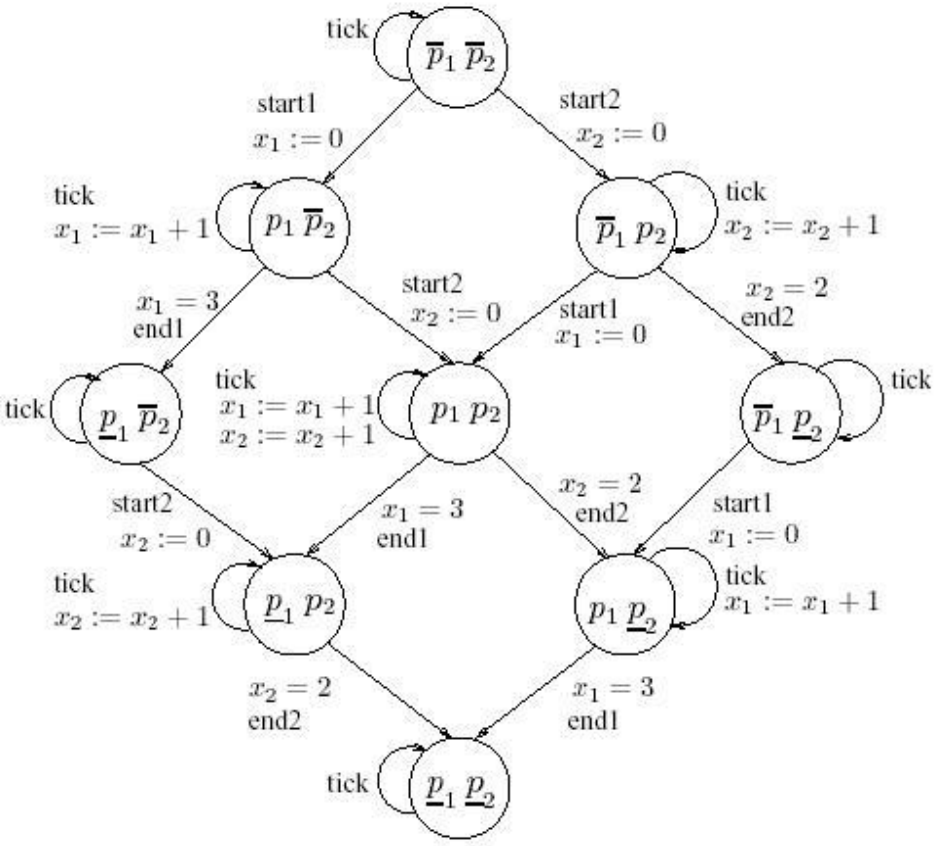
Σχήμα 2.9 - Τα αυτόματα Α΄$_1$ και Α΄$_2$ χρησιμοποιώντας χρονικές μεταβλητές

Πρέπει να σημειωθεί ότι η διαφορά μεταξύ των 2 προσεγγίσεων είναι στον τρόπο σύνταξης. Αν δηλαδή, 'αναπτύξουμε' τα αυτόματα Α΄$_1$ και Α΄$_2$ προσθέτοντας χρονικές τιμές στις καταστάσεις, τότε θα λάβουμε ομοιόμορφα αυτόματα σαν τα A_1 και A_2, όπως φαίνεται άλλωστε και στο σχήμα 2.9.



Σχήμα 2.10 - Τα αυτόματα Α΄$_1$ και Α΄$_2$ αναπτυγμένα με σαφή αναπαράσταση των χρονικών τιμών στις διάφορες καταστάσεις

Έτσι, όταν συνθέτουμε τα A'_1 και A'_2, λαμβάνουμε το συνολικό αυτόματο A' του σχήματος 2.10, το οποίο φαίνεται απλούστερο από εκείνο του αυτομάτου A του σχήματος 2.8. Παρόλα αυτά, η απλούστευση αυτή είναι παραπλανητική. Έστω για παράδειγμα η κατάσταση (p_1, p_2) όπου και οι 2 διαδικασίες είναι ενεργές. Υπάρχουν 2 μεταβάσεις που αφήνουν αυτή την κατάσταση οι οποίες είναι περιορισμένες από τις συνθήκες $X_1 = 3$ και $X_2 = 2$, αντίστοιχα. Η κατάσταση από μόνη της δεν μας πληροφορεί ποια από τις μεταβάσεις θα πραγματοποιηθεί, καθώς αυτό εξαρτάται από τις τιμές των μετρητών, οι οποίες με τη σειρά τους εξαρτώνται από το παρελθόν (όταν δηλαδή οι μετρητές ήταν μηδενισμένοι). Στη χειρότερη περίπτωση, οι αλγόριθμοι προσεγγισιμότητας για το A' ίσως χρειαστούν να το αναπτύξουν σε A.

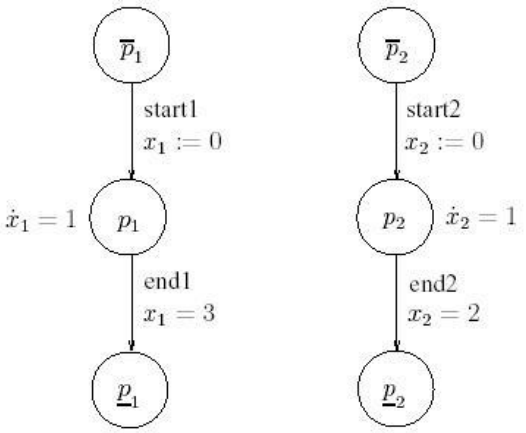


Σχήμα 2.11 - Το συνολικό αυτόματο $A' = A'_1 \| A'_2$

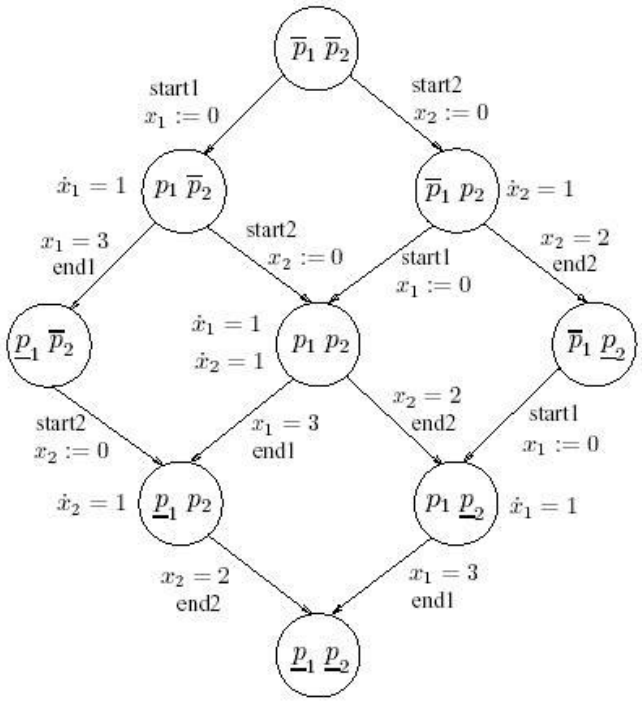
Συμβολική μέθοδος αναπαράστασης όπως η παραπάνω, μπορεί να εφαρμοστεί σε μεταβλητές που ανήκουν σε κάποιο μαθηματικό πεδίο όπως είναι οι ακέραιοι. Αντί να αναπαρασταθεί ένας αριθμός καταστάσεων με αναλυτικό τρόπο, μπορεί να αναπαρασταθεί με τη βοήθεια ενός τύπου. Ας υποθέσουμε, για παράδειγμα, 2 διαδικασίες με διάρκειες d_1 και d_2, αντίστοιχα, τέτοιες ώστε $d_1 < d_2$, οι οποίες εισέρχονται στις ενεργές τους φάσεις 2 μονάδες χρόνου η μια μετά την άλλη. Τότε το σύνολο των τιμών του μετρητή έχει τη μορφή:

$$\{ (2, 0), (3, 1), (4, 2), \ldots. (d_1, d_2 - 2) \}$$

και το μέγεθός του εξαρτάται από το d_1. Ωστόσο ο τύπος $X_1 – X_2 = 2 \text{ ^ } X_1 \leq d_1$ εκφράζει αυτό το σύνολο και το μέγεθός του δεν εξαρτάται από d_1. Στην πραγματικότητα, χαρακτηρίζει τις προσεγγίσιμες καταστάσεις ακόμα και αν δουλεύουμε με απόλυτο χρόνο. Με αυτόν τον τρόπο μπορούμε να επιτρέπουμε να συμβαίνουν γεγονότα οπουδήποτε στον άξονα του πραγματικού χρόνου και να βλέπουμε τους μετρητές διαφορετικά, δηλαδή σαν συνεχείς μεταβλητές που εξελίσσονται με την πρώτη παράγωγο μέσα στις ενεργές καταστάσεις. Αυτά είναι τα χρονισμένα αυτόματα, όπως φαίνονται και στα σχήματα 2.12 και 2.13, τα οποία μπορούν να θεωρηθούν σαν το όριο μιας διαδικασίας που κάνει τα χρονικά βήματα, σχετικά με τις μεταβάσεις ‘tick’, απειροελάχιστα.



Σχήμα 2.12 - 2 χρονισμένα αυτόματα



Σχήμα 2.13 - Το συνολικό αυτόματο του σχήματος 2.11

[Alur, Dill – Automata for modelling Real-Time Systems. Δίπλας-Τσακίρης 2004]

2.8.3 Προσθήκη κόστους στα αυτόματα

Η εκτίμηση και η διαχείριση του κόστους του έργου αποτελεί καθοριστικό παράγοντα τόσο για την ανάληψή του όσο και για την επιτυχή ολοκλήρωσή του. Κόστος του έργου αποτελούν οι δαπάνες που γίνονται και αφορούν το έργο. Περιλαμβάνει το κόστος των πόρων και των υπηρεσιών που συμβάλλουν στην διεκπεραίωση του έργου. Ο πρωταρχικός στόχος του προγραμματισμού είναι η επίτευξη του βέλτιστου κόστους του έργου τηρώντας τους χρονικούς περιορισμούς, έτσι το πρόβλημα της βελτιστοποίησης του έργου ανάγεται στον υπολογισμό της «χρυσής τομής» για την σχέση «χρονική διάρκεια-κόστους έργου». Οι παράγοντες που επηρεάζουν τη διαδικασία βελτιστοποίησης του κόστους ποικίλουν ανάλογα με το είδος του έργου και είναι οι εξής:

Χρονική συμπίεση δραστηριοτήτων. Από τη φύση τους, ορισμένες δραστηριότητες κάποιου έργου μπορούν να υλοποιηθούν ταχύτερα από την προγραμματισμένη διάρκεια τους και άλλες είναι αδύνατον να επιταχυνθούν. Έτσι για δραστηριότητες που με κάποιο τρόπο μπορεί να μειωθεί ο χρόνος υλοποίησής τους λέμε ότι μπορούν να συμπιεστούν ενώ για άλλες που ο απαιτούμενος χρόνος υλοποίησής τους δεν μπορεί να μειωθεί λέμε ότι δεν επιδέχονται συμπίεση. Επιπλέον για κάθε συμπιεστή δραστηριότητα μπορεί να καθοριστεί ο αριθμός των δυνατών χρονικών μονάδων συμπίεσης δεδομένου ότι δεν μπορεί να συμπιέζεται απεριόριστα. Αν η προγραμματισμένη χρονική διάρκεια μιας δραστηριότητας θεωρηθεί ως η κανονική, τότε η χρονική διάρκεια της ίδιας δραστηριότητας κατόπιν μέγιστης συμπίεσης θεωρείται ως η ελάχιστη δυνατή διάρκεια της δραστηριότητας.

Κόστος. Η κατανομή του κόστους κάθε δραστηριότητας στο χρόνο δεν είναι ομοιόμορφη εν γένει. Όταν παραβιάζονται οι χρονικοί περιορισμοί το κόστος δύναται να αυξάνει δραματικά. Για να μπορέσουμε να έχουμε μία πρώτη εκτίμηση του προβλήματος θα θεωρήσουμε την απλούστερη περίπτωση γραμμικής σχέσης κόστους-χρόνου. Θα διακρίνουμε στα πλαίσια της γραμμικής υπόθεσης τα κόστη σε δύο κλάσεις: έμμεσο κόστος, δεν καταγράφεται άμεσα σε κάποια συγκεκριμένη δραστηριότητα του έργου παρόλα αυτά επιβαρύνει το έργο και άμεσο κόστος που περιλαμβάνει τις δαπάνες που γίνονται αποκλειστικά για τις δραστηριότητες.

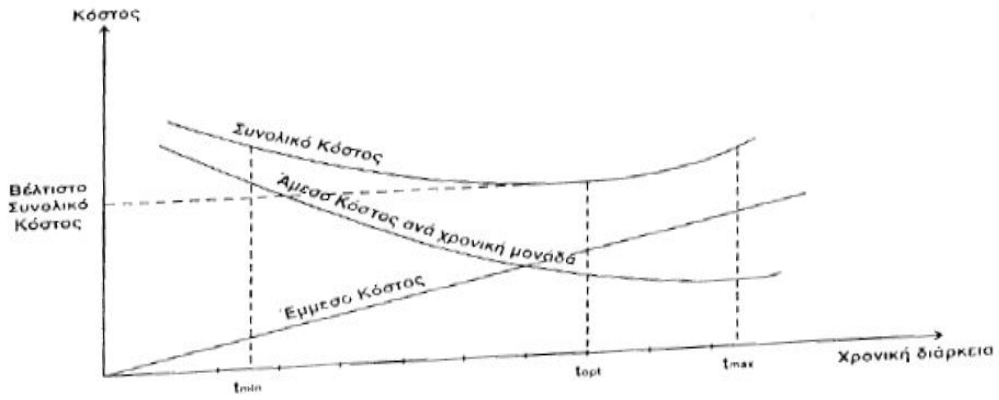
Η μείωση της διάρκειας του έργου μπορεί να απαιτεί περισσότερους πόρους ανά χρονική μονάδα, άρα μεγαλύτερο κόστος. Για τον λόγο αυτό προς τη λήξη του έργου το κόστος συνήθως αυξάνει θεαματικά επειδή γίνεται προσπάθεια απορρόφησης των καθυστερήσεων και επίσπευσης των εργασιών. Επειδή το συνολικό άμεσο κόστος του έργου είναι το άθροισμα του άμεσου κόστους όλων των δραστηριοτήτων του, η προσπάθεια ελέγχου του επικεντρώνεται στον προγραμματισμό και στον σχεδιασμό κάθε επί μέρους δραστηριότητας. Η σχέση «άμεσο κόστος ανά χρονική μονάδα-χρονική διάρκεια δραστηριότητας» είναι αντιστρόφως ανάλογη δηλαδή για να μειωθεί η διάρκεια μιας δραστηριότητας πρέπει ν' αυξηθεί το κόστος των πόρων κατά χρονική μονάδα και στην πράξη λαμβάνεται σαν γραμμική. Έτσι αν θεωρηθεί ότι:

- η ελάχιστη διάρκεια υλοποίησης μιας δραστηριότητας είναι tmin η οποία επιτυγχάνεται όταν διαθέτουμε το επιπλέον από το κανονικό (μέγιστο) κόστος (ανά μονάδα χρόνου) Kmax της συντομότερης διάρκειά της
- η κανονική διάρκεια υλοποίησης μιας δραστηριότητας είναι tmax η οποία επιτυγχάνεται όταν διαθέτουμε το συνηθισμένο (ελάχιστο) κόστος (ανά μονάδα χρόνου) Kmin της

κανονικής της διάρκειας τότε η επίσπευση της δραστηριότητας κατά μια χρονική μονάδα αυξάνει το άμεσο κόστος της κατά

$$\frac{K_{\max} - K_{\min}}{t_{\max} - t_{\min}}$$

το οποίο αποτελεί το κόστος συμπίεσης (compression cost) αυτής της δραστηριότητας. Το έμμεσο κόστος προσδιορίζεται στο ξεκίνημα του έργου ή διαμορφώνεται κατά τη διάρκεια εκτέλεσής του. Στη γενικότερη περίπτωση το έμμεσο κόστος εξελίσσεται γραμμικά, συνεπώς το συνολικό έμμεσο κόστος είναι ανάλογο του χρόνου δηλαδή αυξάνεται ή μειώνεται κατά ένα σταθερό ποσό κάθε φορά που η συνολική διάρκεια του έργου επιμηκύνεται ή συντομεύεται αντίστοιχα κατά μια μονάδα χρόνου. Όμως το συνολικό κόστος της δραστηριότητας διαμορφώνεται τόσο από το έμμεσο όσο και από το άμεσο κόστος. Κατά συνέπεια είναι πιθανό να υπάρχει μια χρονική διάρκεια topt μεταξύ των tmin και tmax για την οποία το συνολικό κόστος είναι ελάχιστο όπως απεικονίζεται στο σχήμα.



Επειδή, όμως, η βέλτιστη χρονική διάρκεια της δραστηριότητας εξαρτάται άμεσα από τη σχέση μεταξύ άμεσου και έμμεσου κόστους, παρατηρούμε τα ακόλουθα: Εφόσον το έμμεσο κόστος υπερτερεί σε αυτή τη σχέση, τότε η βέλτιστη διάρκεια topt θα μετακινείται προς την κατεύθυνση του tmin το οποίο σημαίνει ότι επιδίωξη αποτελεί η συντομότερη υλοποίηση της δραστηριότητας. Αντίθετα, εφόσον υπερτερεί το άμεσο κόστος τότε η βέλτιστη διάρκεια topt θα μετακινείται προς την κατεύθυνση του tmax το οποίο σημαίνει ότι επιδίωξη αποτελεί η βραδύτερη υλοποίηση της δραστηριότητας. [Κάντζαρη Μαρία 2010].

Η μέθοδος CPM (Critical Path Method), η οποία αποκαλείται επίσης και ανάλυση κρίσιμης διαδρομής (CPA, Critical Path Analysis), αναπτύχθηκε γύρω στο 1957 από την εταιρεία Remington Rand Univac, η οποία χρειαζόταν ένα εργαλείο προγραμματισμού και ελέγχου που θα την βοηθούσε να βελτιώσει το χρόνο απόκρισής της, από την παραγωγή ως την πώληση του προϊόντος. Τα πλεονεκτήματα της μεθόδου έγιναν γρήγορα ορατά και τα έξοδα έρευνας αποσβέστηκαν. Η CPM δημιουργήθηκε αρχικά για να αντιμετωπίσει τη σχέση χρόνου κόστους

που προβλημάτιζε πολύ συχνά τους διευθυντές έργου και προέκυπτε από το γεγονός ότι η σχέση ανάμεσα στο χρόνο μέχρι την ολοκλήρωση (time to complete) και το κόστος μέχρι την ολοκλήρωση (cost to complete) είναι εξαιρετικά πολύπλοκη. Συγκεκριμένη η μεθοδολογία της, με συγκεκριμένα βήματα (σε γενικές γραμμές), χαρακτηρίζοντας την ως ντετερμινιστική μεταβλητή • η CPM, αναφορικά με τους παράγοντες που απασχολούν τη διοίκηση, όπως είναι ο χρόνος, το κόστος και η διαθεσιμότητα πόρων, υποθέτει σταθερούς ή καθορισμένους χρόνους (όχι στατιστικά πιθανούς), πράγμα εφαρμόσιμο και χρήσιμο. Αποτελεί αποτελεσματικό εργαλείο εντοπισμού των δραστηριοτήτων, των οποίων η ολοκλήρωση είναι κρίσιμη για την έγκαιρη ολοκλήρωση του έργου.

Η *CPM* προσδιορίζει:

- τη συνολική διάρκεια του έργου
- το βέλτιστο συνδυασμό κόστους διάρκειας
- τις δυνατότητες καθυστέρησης σε ορισμένες δραστηριότητες χωρίς την αύξηση της συνολικής διάρκειας του έργου
- το χρονικό διάστημα χρήσης των πόρων

Στην CPM υπάρχουν δύο χρόνοι ολοκλήρωσης των εργασιών και δύο τιμές για το κόστος. Ο 1ος συνδυασμός χρόνου/κόστους είναι κανονικός (normal). Ο 2ος συνδυασμός προέρχεται από την απόπειρα να επιταχυνθεί η εργασία, προσθέτοντας κάποιους πόρους (σε υπερωρίες, ειδικό εξοπλισμό, περισσότερο εξοπλισμό ή υλικά) και θεωρείται συντομευμένος (crash). Ορίζεται ο λόγος κόστους προς χρόνο που δείχνει το κόστος/μέρα για την επιτάχυνση ενός σχεδίου και παίρνει πάντα αρνητική τιμή διότι το κόστος με την επιτάχυνση αυξάνεται ενώ ο χρόνος μειώνεται. (Λόγος κόστους/χρόνος = συντομευμένο κόστος- κανονικό κόστος/συντομευμένος χρόνος- κανονικός χρόνος).

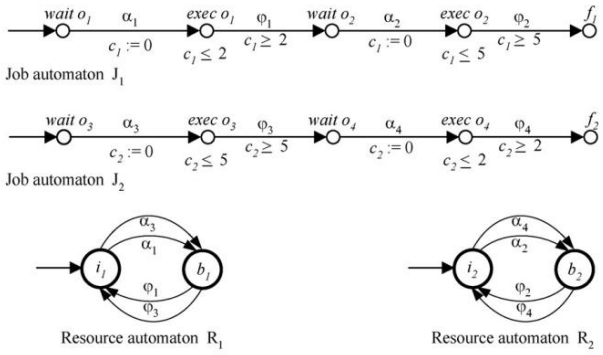
Αρχικά, η Μέθοδος Κρίσιμης Διαδρομής θεωρούνταν μόνο λογικές εξαρτήσεις μεταξύ των τελικών στοιχείων. Έκτοτε, έχει επεκταθεί για να επιτρέψει τον συνυπολογισμό των πόρων σχετικά με την κάθε δραστηριότητα μέσω των αποκαλούμενων διαδικασιών της δραστηριότητας –βασισμένης ανάθεσης πόρων και της ισοπέδωσης πόρων. Ένας πόρος –ισοπεδωμένο πρόγραμμα περιλαμβάνει τις καθυστερήσεις λόγω δυσχέρειας των πόρων, σε περίπτωση αδιαθεσιμότητας του πόρου στον απαραίτητο χρόνο. Και έχει τη δυνατότητα να αναγκάσει μια προηγούμενη κοντύτερη πορεία να γίνει μακρύτερη ή περισσότερο κρίσιμη πορεία των πόρων. Σχετική έννοια αντ' αυτού καλείται κρίσιμη αλυσίδα, η οποία προσπαθεί να προστατεύσει τη δραστηριότητα και τις διάρκειες προγράμματος από τις απρόβλεπτες καθυστερήσεις λόγω των περιορισμών των πόρων.

Από την αλλαγή σχεδίου προγράμματος σε κανονική βάση, η CPM επιτρέπει το συνεχή έλεγχο του προγράμματος, επιτρέποντας στο διευθυντή προγράμματος να ακολουθήσει τις κρίσιμες δραστηριότητες. Τον προειδοποιεί ότι οι μη κρίσιμες δραστηριότητες μπορεί να καθυστερήσουν πέρα από το συνολικό επιπλέον σώμα τους. Δημιουργώντας κατά συνέπεια μια νέα κρίσιμη πορεία, καθυστερώντας την ολοκλήρωση προγράμματος. Επιπλέον, η μέθοδος μπορεί εύκολα να ενσωματώσει τις έννοιες των πιθανολογικών προβλέψεων, χρησιμοποιώντας την τεχνική αξιολόγησης και αναθεώρησης προγράμματος (Program Evaluation and Review Technique) (Pert) και τη μεθοδολογία αλυσίδων γεγονότος (event chain methology). Αυτή την περίοδο,

υπάρχουν διάφορες λύσεις λογισμικού, διαθέσιμες στη βιομηχανία, που χρησιμοποιούν τη μέθοδο CPM. Η μέθοδος που χρησιμοποιείται αυτή την περίοδο περισσότερο από το λογισμικό διαχείρισης του προγράμματος είναι βασισμένη σε μια χειρωνακτική προσέγγιση υπολογισμού που αναπτύσσεται από το Fondahl του Πανεπιστημίου του Stanford.

Ένα πρόγραμμα που παράγεται χρησιμοποιώντας τη μέθοδο κρίσιμης διαδρομής συχνά δεν πραγματοποιείται με ακρίβεια, δεδομένου ότι οι εκτιμήσεις χρησιμοποιούνται για να υπολογίσουν, εάν όταν ένα λάθος γίνεται, αν τα αποτελέσματα της ανάλυσης μπορούν να τα αλλάξουν. Αυτό θα μπορούσε να προκαλέσει διαταραχές στην εφαρμογή του προγράμματος, εάν οι εκτιμήσεις γίνονται στα τυφλά και εάν οι αλλαγές δεν εξετάζονται αμέσως. Εντούτοις, η δομή της ανάλυσης της μεθόδου κρίσιμης διαδρομής είναι τέτοια που η διαφορά που προκαλείται από το αρχικό πρόγραμμα από οποιαδήποτε αλλαγή μπορεί να μετρηθεί. Και ο αντίκτυπος αυτής είτε καλυτερεύει, είτε ρυθμίζεται. Πράγματι, ένα σημαντικό στοιχείο της μεταθανάτιας ανάλυσης του προγράμματος είναι η ως Χτισμένη Κρίσιμη Πορεία (*ABCP*), η οποία αναλύει τις συγκεκριμένες αιτίες και τις επιδράσεις των αλλαγών μεταξύ του προγραμματισμένου προγράμματος και του ενδεχόμενου προγράμματος όπως πραγματικά εφαρμόζεται.

Παρακάτω δίνεται ένα παράδειγμα προγραμματισμού σε περιβάλλον job-shop χρησιμοποιώντας χρονισμένα αυτόματα υπολογίζοντας και την παράμετρο του κόστους (timed priced automata). Έστω δύο εργασίες J_1 και J_2 και δύο πηγές R_1 και R_2. Η εργασία J_1 αποτελείται από δύο διεργασίες o_1 και o_2 όπου η πρώτη προηγείται της δεύτερης. Ομοίως η εργασία J_2 αποτελείται από τις διεργασίες o_3 και o_4 που η o_3 προηγείται της o_4. Οι λειτουργίες o_1 και o_3 εκτελούνται στην πηγή R_1, με επεξεργασία διάρκειας 2 και 5 αντιστοίχως. Οι λειτουργίες o_2 και o_4 εκτελούνται στην πηγή R_2, με διάρκεια επεξεργασίας 5 και 2 αντιστοίχως. Το μοντέλο για το παραπάνω πρόβλημα απεικονίζεται στο ακόλουθο σχήμα.



Κάθε εργασία αντιπροσωπεύεται από μια εργασία-αυτόματο που ονομάζεται J_1 και J_2, και κάθε πόρος αντιπροσωπεύεται από έναν πόρο-αυτόματο που ονομάζεται R_1 και R_2. Κάθε λειτουργία μιας εργασίας αντιπροσωπεύεται από δύο θέσεις και συγκεκριμένα την θέση *αναμονή*, όπου η λειτουργία είναι σε αναμονή για να εκτελεστεί στην αντίστοιχη θέση που καλείται *εκτέλεση* πόρων, όπου η λειτουργία καταλαμβάνει το αντίστοιχο πόρο για την αντίστοιχη διάρκεια της διαδικασίας. Η θέση αναμονή και εκτέλεση της λειτουργίας o_1 επισημαίνονται ως αναμονή o_1 και exec o_1, αντιστοίχως. Μια πρόσθετη θέση f_1 ορίζεται να υποδείξει τον τερματισμό της εργασίας J_1.

Κάθε δουλειά έχει ένα αυτόματο ρολόι για να υποδεικνύει το χρονοδιάγραμμα των εργασιών. Μεταβάσεις που αντιπροσωπεύουν την έναρξη μιας επιχείρησης και το φινίρισμα μιας επιχείρησης έχουν ετικέτες που ονομάζονται α και φ αντίστοιχα, με το δείκτη λειτουργίας ως κατάληξη. Ο περιορισμός της προτεραιότητας μεταξύ των επιχειρήσεων o_1 και o_2, αντιπροσωπεύεται από την επισημασμένη μετάβαση μεταξύ των θέσεων exec o_1 και *wait* o_2, εξασφαλίζοντας έτσι ότι η O_2 μπορεί να αρχίσει μόνο μετά την εκτέλεση της λειτουργίας o_1. Το ρολόι c_1 είναι συνδεδεμένο στην εργασία J_1 για να μετρά τη διάρκεια των αντίστοιχων εργασιών. Κάθε μετάβαση που αντιπροσωπεύει την έναρξη μιας επιχείρησης περιέχει τη δράση της ρυθμίσεως του ωρολογίου της αντίστοιχης εργασίας για να διαμορφώσει τη διάρκεια επεξεργασίας της επιχείρησης. Η έναρξη της εκτέλεσης της λειτουργίας o_1 καταλαμβάνοντας τον πόρο R_1, αντιπροσωπεύεται από τη μετάβαση (wait o_1, true, $α_1$, c_1, exec o_1) και την ολοκλήρωση της εκτέλεσης της επιχείρησης με την απελευθέρωση του πόρου.

Το R1 αντιπροσωπεύεται από τη μετάβαση (exec o_1, $c_1 \geq 2$, $φ_1$, ∅, wait o_2). Κάθε exec θέση μιας επιχείρησης αποτελείται από μια αναλλοίωτη κατάσταση, η οποία αναφέρει ότι η θέση exec πρέπει να είναι ενεργή μόνο για να παρέλθει διάρκεια η οποία είναι μικρότερη ή ίση με την δεδομένη διάρκεια επεξεργασίας της αντίστοιχης λειτουργίας. Κάθε μετάβαση που αντιπροσωπεύει το φινίρισμα μιας επιχείρησης αποτελείται από μια σταθερή κατάσταση στο ρολόι αποτίμησης του κόστους, το οποίο αναφέρει ότι η λειτουργία πρέπει να εκτελείται για τουλάχιστον δεδομένη τη διάρκεια επεξεργασίας πριν αρχίζει η επόμενη λειτουργία. Η αναλλοίωτη $c_1 \leq 2$ στη θέση exec o_1 αναγκάζει το αυτόματο J_1, γ να αφήσει τη θέση exec o_1 εφόσον η διάρκεια λειτουργίας των 2 μονάδων έχει λήξει. Η σταθερά $c_1 \geq 2$ σχετικά με την τελική μετάβαση της λειτουργίας o_1 εξασφαλίζει ότι η κατάσταση, με βάση την προϋπόθεση ότι η λειτουργία εκτελείται για ακριβώς 2 μονάδες χρόνου, είναι ικανοποιητική. Κάθε πόρος διαμορφώνεται από ένα ξεχωριστό πόρο αυτόματο, που αποτελείται από δύο περιοχές και συγκεκριμένα, σε αδράνεια (i_1, i_2) και απασχολημένος (b_1, b_2). Για κάθε λειτουργία που μπορεί να εκτελεστεί σε έναν πόρο, υπάρχει μια μετάβαση από το ρελαντί σε απασχολημένος και η μετάβαση από την πολυάσχολη στο ρελαντί Η μετάβαση από την αδράνεια στο απασχολημένο, αντιπροσωπεύει την κατανομή των πόρων για να ξεκινήσει την εκτέλεση μιας λειτουργίας και η μετάβαση από απασχολημένο σε αδράνεια αντιπροσωπεύει την απελευθέρωση του πόρου, εφόσον η αντίστοιχη λειτουργία έχει τερματιστεί. Οι δράσεις για τους πόρους (κατανομή και απελευθέρωση) εκτελούνται από το συγχρονισμό των αντίστοιχων μεταβάσεων των πόρων με τα αυτόματα εργασίας. Για το υπό εξέταση παράδειγμα, είναι η κατάσταση αδράνειας και οι πολυσύχναστες περιοχές του πόρου R_1 εκπροσωπούνται από i_1 και b_1, αντίστοιχα. Η κατανομή των πόρων R_1 κατά την έναρξη της λειτουργίας o_1 διαμορφώνεται με το συγχρονισμό των μεταβάσεων wait o_1 exec o_1 και i_1 έως b_1με την ετικέτα συγχρονισμού a_1. Ομοίως, η απελευθέρωση του πόρου R1 μετά την ολοκλήρωση της λειτουργίας o_1, διαμορφώνεται με το συγχρονισμό των μεταβάσεων exec o_1 wait o_2 και b_1 i_1 με την ετικέτα συγχρονισμού $φ_1$.

[Subanatarajan Subbiaha, et al, 2009]

Κεφάλαιο 3

Μοντελοποίηση του Προβλήματος με Χρονισμένα Αυτόματα

3.1 Το Εργαλείο Μοντελοποίησης UPPAAL

Το εργαλείο μοντελοποίησης UPPAAL είναι ένα ολοκληρωμένο περιβάλλον εργαλείων για τη μοντελοποίηση, την επικύρωση και την επαλήθευση των συστημάτων πραγματικού χρόνου που μοντελοποιούνται ως δίκτυα χρονισμένων αυτομάτων, εμπλουτισμένα με τύπους δεδομένων (ακέραιοι, πίνακες, κ.λπ.). Κάθε ένα από αυτά τα αυτόματα συνιστά μια διεργασία (process) και το δίκτυό τους περιγράφει συστήματα πραγματικού χρόνου. Πρόκειται για συστήματα των οποίων η ορθότητα δεν εξαρτάται μόνο από την έξοδο αλλά και από το χρόνο κατά τον οποίο η έξοδος αυτή παράγεται [Paul Pettersson – Modelling and Verification of Real-Time Systems Using Timed-Automata, Δίπλας-Τσακίρης 2004].

Το εργαλείο έχει αναπτυχθεί σε συνεργασία μεταξύ του Τμήματος Πληροφορικής στο Πανεπιστήμιο της Ουψάλα, στη Σουηδία και στο Τμήμα Επιστήμης Υπολογιστών στο Πανεπιστήμιο Aalborg της Δανίας [http://www.uppaal.org/]. Στις τυπικές περιοχές εφαρμογής του UPPAAL περιλαμβάνονται ελεγκτές πραγματικού χρόνου και πρωτόκολλα επικοινωνίας, όπου οι προδιαγραφές χρόνου θεωρούνται κρίσιμες [http://www.uppaal.com/]. Η πρώτη έκδοση του Uppaal κυκλοφόρησε το 1995. Από τότε το εργαλείο βρίσκεται σε διαρκή ανάπτυξη. Τα πειράματα και οι βελτιώσεις περιλαμβάνουν δομές δεδομένων, η μερική μείωση της τάξης, μείωση της συμμετρίας, μια κατανεμημένη έκδοση του Uppaal, καθοδηγούμενη και με ελάχιστο κόστος [Gerd Behrmann, Alexandre David, and Kim G. Larsen, A Tutorial on Uppaal].

Η σχεδίαση του εργαλείο Uppaal έγινε με βάση την αποδοτικότητα και την ευκολία της χρήσης. Ο περιορισμός στην ανάλυση προσεγγισιμότητας υπήρξε αποφασιστικός για την αποδοτικότητα της μονάδας ελέγχου του μοντέλου του UPPAAL. Ένας άλλος σημαντικός παράγοντας για την αποδοτικότητα υπήρξε η εφαρμογή της τεχνικής της επαλήθευσης κατά τη διάρκεια της εξερεύνησης του χώρου καταστάσεων (on-the-fly verification) που συνδυάζεται με τη συμβολική τεχνική που περιορίζει προβλήματα επαλήθευσης σε προβλήματα χειρισμού και επίλυσης απλών περιορισμών [Johan Bengtsson, Kim Larsen, Fredik Larsson, Paul Pettersson, Wang Yi: UPPAAL – a Tool Suite for Automatic Verification of Real-Time Systems. / Paul Pettersson – Modelling and Verification of Real-Time Systems Using Timed-Automata].

Το δεύτερο κριτήριο σχεδιασμού του UPPAAL υπήρξε η φιλικότητα προς το χρήστη. Για να διευκολυνθεί η εύρεση λαθών στις περιγραφές των συστημάτων, η μονάδα που ευθύνεται για την επαλήθευση των προς εξέταση ιδιοτήτων (verifier) εφοδιάστηκε με τη δυνατότητα για

υποστήριξη διαγνωστικών διαδρομών (diagnostic traces). Όταν η επαλήθευση μιας συγκεκριμένης ιδιότητας έχει αποβεί επιτυχής ή το αντίθετο, τότε ένα παράδειγμα παράγεται που επιδεικνύει γιατί η ιδιότητα ικανοποιήθηκε ή όχι από το αναλυμένο μοντέλο του συστήματος. Η φιλικότητα προς το χρήστη έχει βελτιωθεί περαιτέρω με την ανάπτυξη γραφικού περιβάλλοντος για την σχεδίαση των διαφόρων συστατικών στοιχείων ενός μοντέλου που υιοθετεί το πρότυπο του δικτύου των χρονισμένων αυτομάτων [Marius Mikucionis, Egle Sasnauskaite – On the fly testing using UPPAAL / Δίπλας-Τσακίρης 2004].

Σαν εργαλείο αποτελείται από δύο βασικά μέρη:

- το Graphical User Interface (GUI, Γραφικό περιβάλλον χρήστη) που εκτελείται στο σταθμό εργασίας του χρήστη, και το
- model-checker engine (Μηχανή ελέγχου μοντέλων), που εκτελείται στον ίδιο υπολογιστή με το περιβάλλον χρήστη, ή μπορεί να πραγματοποιηθεί σε έναν πιο «δυνατό» server [Julián Proenza, The UPPAAL Model Checker].

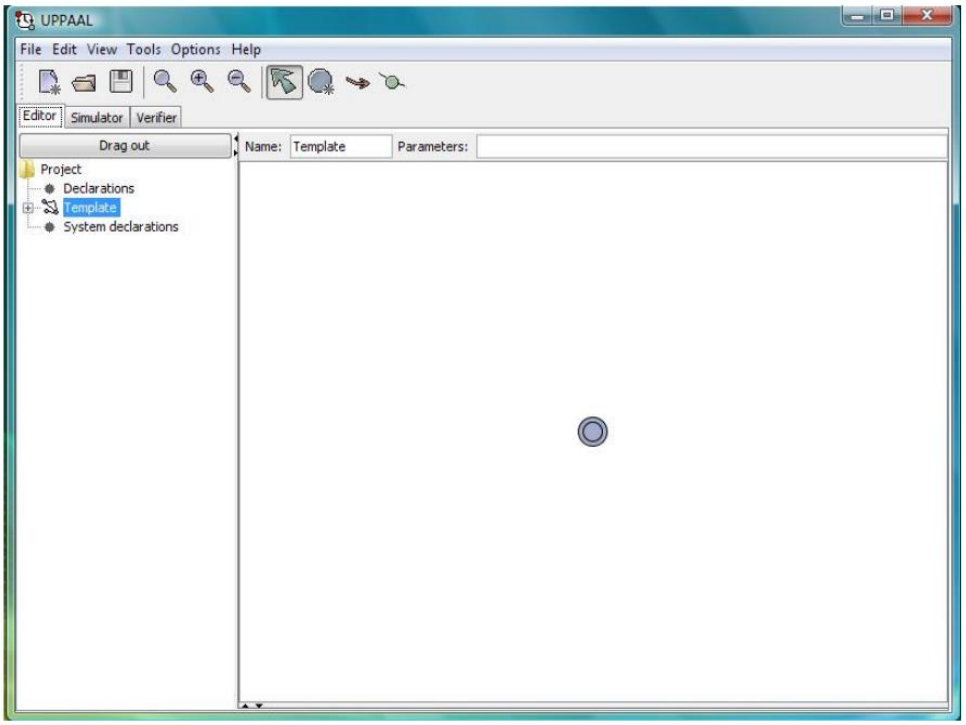
Ο server του UPPAAL παρέχει ένα αποδοτικό υπολογισμό του συμβολικού χώρου του συστήματος κατά τη διαδικασία επαλήθευσης μιας δοσμένης ιδιότητας. Το GUI (επαν) εκκινεί το πρόγραμμα του server κάθε φορά που ο χρήστης αποφασίζει να ανανεώσει τον προσομοιωτή ή την μονάδα επαλήθευσης με ένα νέο μοντέλο συστήματος. Το GUI και ο server επικοινωνούν μέσω μηχανισμού σύνδεσης με TCP/IP sockets. Η σύνδεση αυτή εγκαθίσταται μετά το ξεκίνημα του server.

Το γραφικό περιβάλλον του UPPAAL αποτελείται από τρία κύρια μέρη, προσβάσιμα μέσω τριών καρτελών στο κύριο παράθυρο:

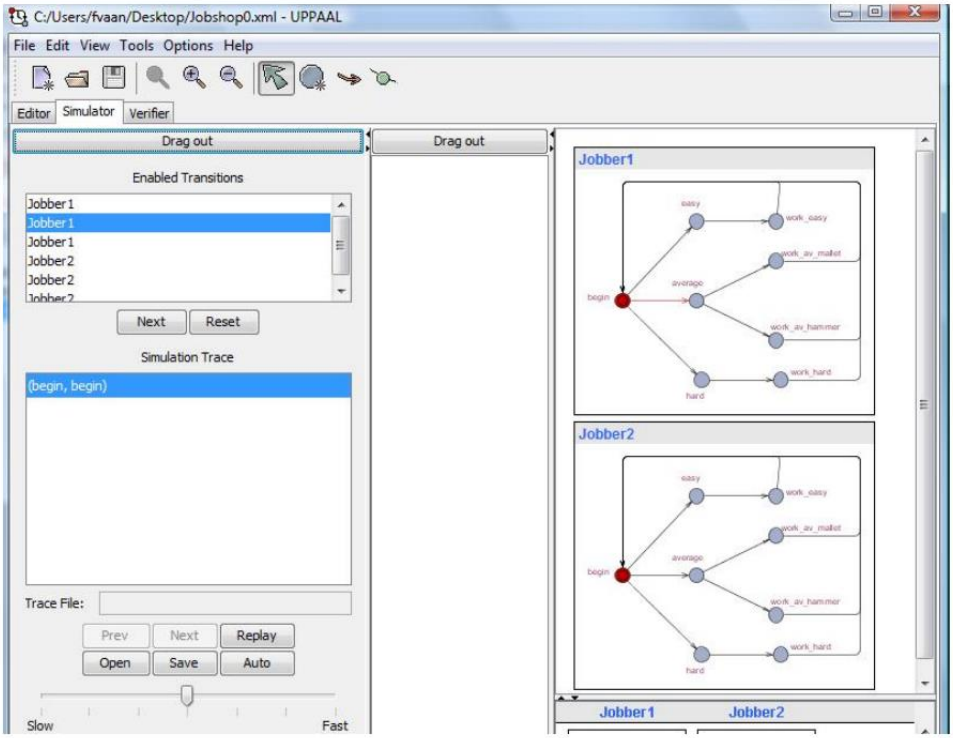
Τη **μονάδα σύνταξης του μοντέλου (system editor)** που μπορεί να χρησιμοποιηθεί για την κατασκευή μοντέλων. Το μοντέλο αποτελείται από ένα φραγμένο αριθμό από παράλληλες διαδικασίες (*concurent processes*), κάθε μία από τις οποίες περιγράφεται από ένα χρονισμένο αυτόματο. Κάθε διαδικασία ενδέχεται να περιλαμβάνει τοπικής εμβέλειας (local) ρολόγια και μεταβλητές, τα οποία μπορούν να χρησιμοποιηθούν μόνο μέσα σε αυτή και μπορεί να κάνει χρήση των γενικής εμβέλειας (global) ρολογιών και μεταβλητών. Οι διαδικασίες επικοινωνούν μεταξύ τους μέσω των μεταβλητών γενικής εμβέλειας και των δυαδικών συγχρονισμών.

Τον **προσομοιωτή**, στον οποί μπορεί ο χρήστης να απεικονίσει τη συμπεριφορά των μοντέλων και τη **μονάδα επαλήθευσης (verifier),** στην οποία μπορεί να αναλυθεί η συμπεριφορά των μοντέλων και δέχεται κατάλληλα μορφοποιημένες ιδιότητες για να επαληθευτούν για ένα συγκεκριμένο μοντέλο αυτομάτων και απεικονίζει το αποτέλεσμα: σωστό ή λάθος αν η ιδιότητα ικανοποιείται ή όχι αντιστοίχως.

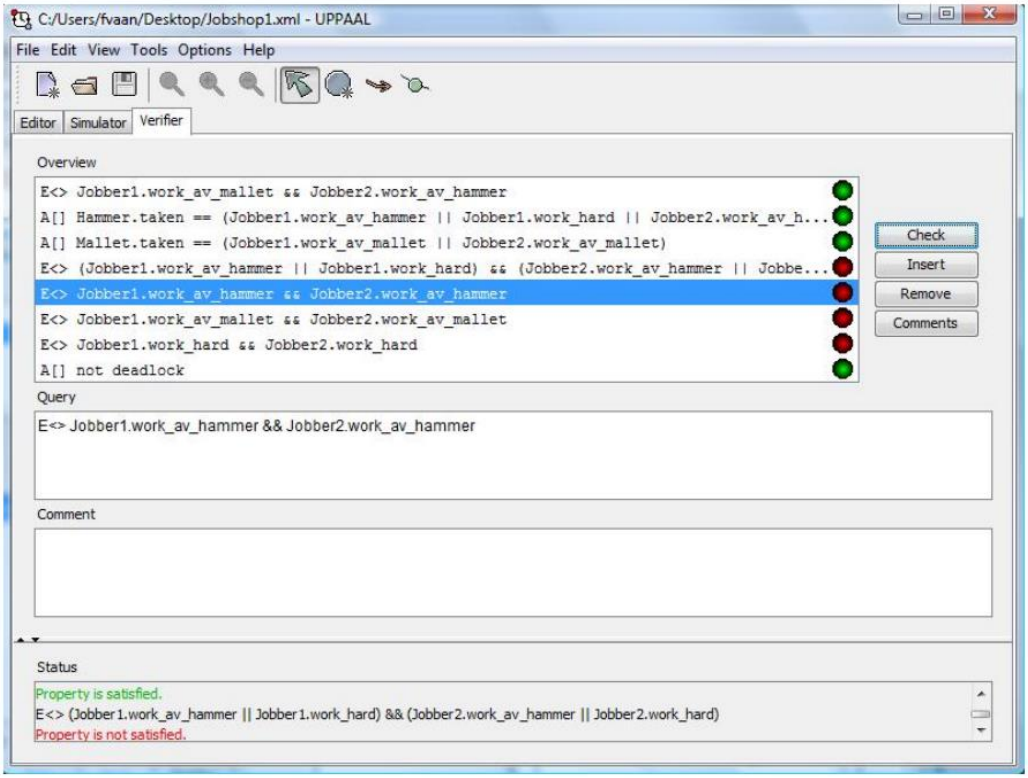
[Frits Vaandrager, *A First Introduction to Uppaal,* Δίπλας-Τσακίρης 2004]



Σχήμα 3.1 - Περιβάλλον του system editor (Frits Vaandrager, A First Introduction to Uppaal)



Σχήμα 3.2 - Περιβάλλον του simulator (Frits Vaandrager, A First Introduction to Uppaal)

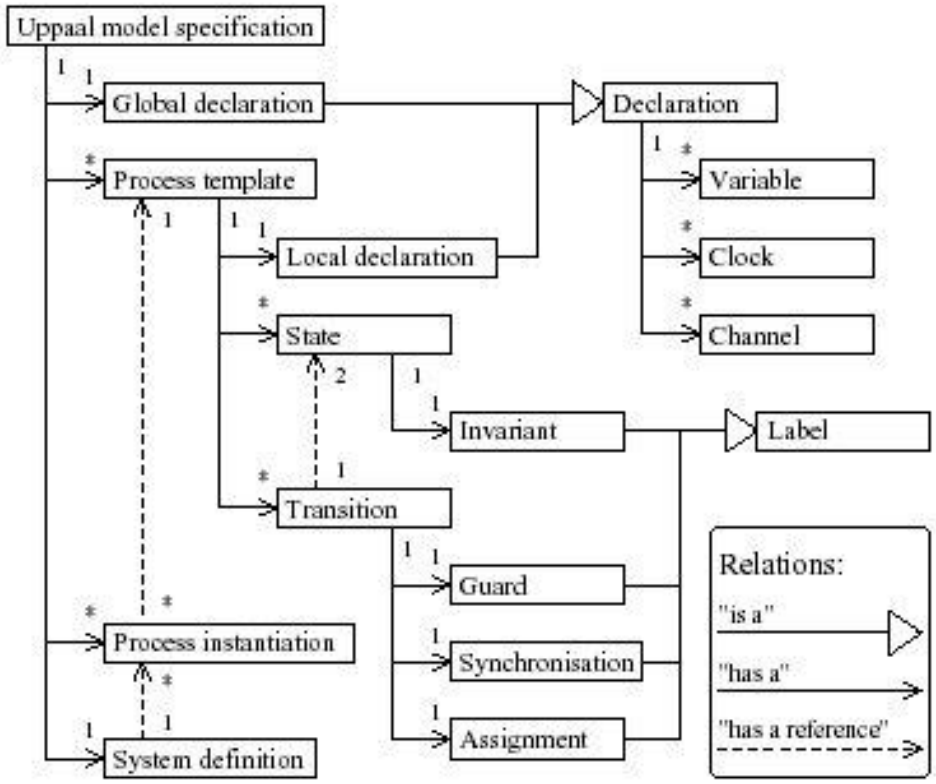


Σχήμα 3.3 - Περιβάλλον του verifier (Frits Vaandrager, A First Introduction to Uppaal)

3.2 Το UPPAAL στα Χρονισμένα Αυτόματα

Στο εργαλείο UPPAAL, τα συστήματα μοντελοποιούνται με τη χρήση χρονισμένων αυτομάτων, τα οποία είναι μηχανές πεπερασμένων καταστάσεων με ρολόγια. Για να παρέχει ένα περισσότερο εκφραστικό μοντέλο και να διευκολύνει το έργο της μοντελοποίησης, το UPPAAL επεκτείνει τα timed automata με γενικότερους τύπους μεταβλητών δεδομένων όπως λογικές (boolean) και ακέραιες μεταβλητές. Ο τελικός σκοπός είναι η μορφοποίηση μια γλώσσας μοντελοποίησης που να είναι όσο το δυνατό περισσότερο συναφής με τις υψηλού επιπέδου και πραγματικού χρόνου γλώσσες προγραμματισμού που περιέχουν διάφορους τύπους δεδομένων. Κάτι τέτοιο ενδεχομένως θα δημιουργούσε προβλήματα λήψης απόφασης (decidability problems) κατά τον έλεγχο του μοντέλου.

Παρακάτω απεικονίζεται μια ολοκληρωμένη προδιαγραφή μοντέλου χρονισμένων αυτομάτων στο UPPAAL.



Σχήμα 3.4 – [Δίπλας –Τσακίρης 2004]

3.2.1 Τα συστατικά του μοντέλου

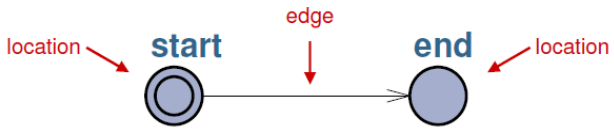
- **Οι εκφράσεις:** Οι εκφράσεις στο UPPAAL μοιάζουν πολύ με τις εκφράσεις γλωσσών προγραμματισμού, όπως η Java και η C++. Είναι δυνατή η χρήση λογικών τελεστών

(and, or, not), ισότητας και ανισότητας, αριθμητικών και εκχώρησης τιμής. Οι τελεστές συνοψίζονται στον εξής πίνακα:

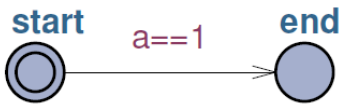
()	Καθορίζει την σειρά αποτίμησης των εκφράσεων	<	Μικρότερο από
[]	Indexing σε πίνακα	<=	Μικρότερο ή ίσο από
.	Lookup operator για πρόσβαση στις local μεταβλητές	==	Τελεστής ισότητας
!	Λογική άρνηση	!=	Τελεστής ανισότητας
++	Αύξηση	>=	Μεγαλύτερο ή ίσο από
--	Μείωση	>	Μεγαλύτερο από
-	Integer subtraction (can also be used as unary negation)	&	Τελεστής and για bits
+	Άθροιση ακεραίων	^	Τελεστής xor για bits
*	Γινόμενο ακεραίων	|	Τελεστής or για bits
/	Διαίρεση ακεραίων	&&	Λογικό and
%	Modulo	||	Λογικό or
<<	Αριστερή μετακίνηση bit	?:	Τελεστής If-then-else
>>	Δεξιά μετακίνηση bit	not	Λογική άρνηση
<?	Ελάχιστο	and	Λογικό and
>?	Μέγιστο	or	Λογικό or
: += -= *= %= &= |= >>= ^=	Τελεστής ανάθεσης τιμής	imply	Λογική συνεπαγωγή

Σχήμα 3.5 – [Δίπλας –Τσακίρης 2004]

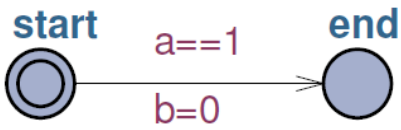
- **Τα ρολόγια** είναι μεταβλητές που μπορούν να αξιολογήσουν σε ένα πραγματικό αριθμό και οποίες μπορούν να ορισθούν σε κάθε αυτόματο προκειμένου να μετρηθεί την πρόοδο του χρόνου. Όλα τα ρολόγια εξελίσσονται με τον ίδιο ρυθμό, προκειμένου να εκπροσωπεί το παγκόσμια πρόοδο του χρόνου. Η πραγματική τιμή ενός ρολογιού μπορεί είτε να δοκιμαστεί ή επαναφερθεί (όχι να ανατεθεί). Δεδομένου ότι το εργαλείο UPPAAL είναι ειδικά σχεδιασμένο για την επαλήθευση των συστημάτων πραγματικού χρόνου, τα ρολόγια αποτελούν ένα θεμελιώδη μέσο μοντελοποίησης και επαλήθευσης. Ένα μοντέλο UPPAAL είναι χτισμένο ως ένα σύνολο ταυτόχρονων διαδικασιών, κάθε μια από τις οποίες είναι γραφικά σχεδιασμένη ως χρονισμένο αυτόματο. Κάθε χρονισμένο αυτόματο αντιπροσωπεύεται από ένα γράφημα το οποίο έχει θέσεις (locations) ως κόμβους και ακμές (edges) ως τόξα ανάμεσα στις θέσεις.



- **Οι μεταβάσεις.** Οι άκρες είναι ενημερωμένες με τους παράγοντες guards (φύλακες), updates (ενημερώσεις), synchronizations (συγχρονισμούς) και selections (επιλογές). Ένας φύλακας είναι μια έκφραση που χρησιμοποιεί τις μεταβλητές και ρολόγια του μοντέλου, ώστε να υποδεικνύει πότε η μετάβαση είναι ενεργοποιημένη. Σημειώνεται εδώ ότι πολλές ακμές μπορούν να ενεργοποιηθούν σε μία συγκεκριμένη χρονική στιγμή, αλλά μόνο μία από αυτές θα δράσει ώστε να οδηγήσει σε διαφορετικές δυνατότητες.



Μια ενημερωμένη έκδοση (update) είναι μια έκφραση που αξιολογείται το συντομότερο καθώς το αντίστοιχο άκρο βρίσκεται σε δράση. Η αξιολόγηση αλλάζει την κατάσταση του συστήματος.

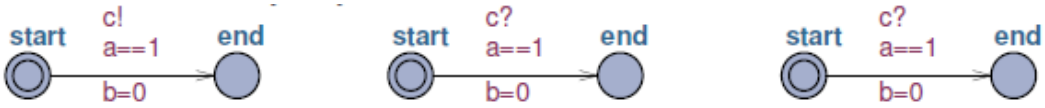


Ο συγχρονισμός είναι ο βασικός μηχανισμός που χρησιμοποιείται για συντονίζει τη δράση των δύο ή περισσότερες διεργασίες. Μοντελοποιεί, για παράδειγμα, την επίδραση των μηνυμάτων. Προκαλεί δύο (ή περισσότερες) διεργασίες να λάβουν μια μετάβαση την ίδια στιγμή. Όταν ένα κανάλι c δηλώνεται, τότε μια διαδικασία θα έχει επισημασμένη την άκρη ως c! Και η άλλη (ες) διαδικασία (ες) ως c?. Υπάρχουν τρία είδη συγχρονισμών, τα οποία περιγράφονται ακολούθως:

- **Regular channel**: δηλώνεται ως chan c. Όταν μια διαδικασία βρίσκεται σε μια θέση από την οποία υπάρχει μια μετάβαση που επισημαίνεται με το c!, ο μόνος τρόπος για τη μετάβαση να είναι ενεργοποιημένη είναι μια άλλη διαδικασία να είναι σε μια θέση από την οποία υπάρχει μετάβαση που επισημαίνεται με c? και το αντίστροφο. Εάν σε μια συγκεκριμένη στιγμή, υπάρχουν διάφοροι τρόποι για να έχουν ζεύγος c! και c?, ένας από αυτούς είναι μη- ντετερμινιστικά επιλεγμένος κατά τον έλεγχο μοντέλου. Η ενημέρωση σε μια άκρη συγχρονισμού για c! εκτελείται πριν από την ενημέρωση σε μια άκρη συγχρονισμού για c?.

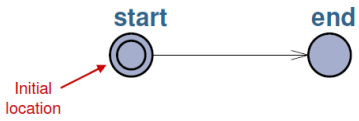
- **Urgent channel:** δηλώνεται ως urgent chan c. Είναι παρόμοιο με το regular channel, εκτός από το γεγονός ότι δεν είναι δυνατό να καθυστερήσει στο σταθμό προέλευσης, αν είναι εφικτό να ενεργοποιήσει το συγχρονισμό πάνω από το regular channel. Αυτό σημαίνει ότι δεν μπορεί να περάσει χρόνος, αλλά μπορεί να εναλλάσσει άλλες μεταβάσεις που δεν απαιτούν χρόνο για να περάσει. Σημειώνεται ότι τα ρολόγια «φύλακες» δεν επιτρέπονται στις άκρες πάνω από ένα urgent channel.

- **Broadcast channel**: δηλώνεται ως broadcast chan c. Όταν μία διεργασία βρίσκεται σε μια θέση από την οποία υπάρχει μια μετάβαση που επισημαίνεται με c! και μία ή περισσότερες διεργασίες είναι σε θέσεις από τις οποίες υπάρχει μια μετάβαση επισημασμένη με c?, όλες οι μεταβάσεις είναι ενεργοποιημένες. Ωστόσο, εάν δεν υπάρχουν διεργασίες στις θέσεις από τις οποίες υπάρχει μια μετάβαση επισημασμένη με c?, η μετάβαση επισημασμένη με c! είναι ενεργοποιημένη ούτως ή άλλως. Παρατηρείται ότι τα ρολόγια «φύλακες» δεν επιτρέπονται στις άκρες που λαμβάνουν πάνω σε ένα broadcast channel. Η ενημέρωση σχετικά με την άκρη εκπομπής εκτελείται πρώτα. Η ενημέρωση σχετικά με τις ακμές που λαμβάνουν εκτελούνται αριστερά προς τα δεξιά, σύμφωνα με τη διάταξη που οι διεργασίες καταχωρήθηκαν κατά τον ορισμό του συστήματος.



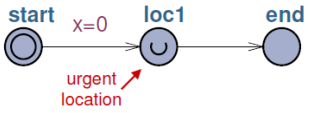
- **Οι θέσεις.** Αναφορικά με τις θέσεις (locations), μπορούν να έχουν ένα προαιρετικό όνομα, το οποίο χρησιμεύει στο να εντοπίζεται η θέση κατά τη διάρκεια του ελέγχου και της τεκμηρίωσης του μοντέλου. Οι θέσεις μπορεί να είναι τριών ειδών:

- **Initial (αρχικές):** Κάθε πρότυπο πρέπει να έχει προετοιμαστεί σωστά, πράγμα που σημαίνει ότι πρέπει να ξεκινήσει σε μια συγκεκριμένη θέση. Ως εκ τούτου, κάθε πρότυπο πρέπει να έχει ακριβώς μία θέση που επισημαίνεται ως αρχική. Οι αρχικές θέσεις που προσδιορίζονται με διπλό κύκλο

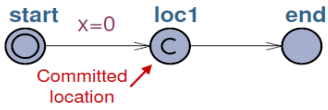


- **Urgent locations**: οι οποίες παγώνουν το χρόνο, δηλαδή δεν επιτρέπεται να παρέλθει χρονικό διάστημα όσο μια διαδικασία βρίσκεται σε εξέλιξη. Η κάθε θέση πρέπει να εγκαταλειφθεί προτού περάσει ο χρόνος. Οι υπόλοιπες μεταβάσεις μπορούν να

πραγματοποιηθούν προηγουμένως εφόσον δεν απαιτούν την παρέλευση χρόνου. Στο μοντέλο, αυτού του είδους οι θέσεις επισημαίνονται με U.

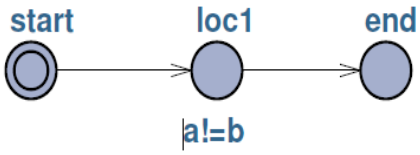


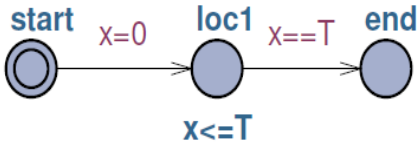
- **Committed locations**, οι οποίες επίσης παγώνουν το χρόνο. Όταν ένα μοντέλο έχει ενεργές μία ή περισσότερες τέτοιες θέσεις τότε καμία άλλη διεργασία, πέρα από αυτές που εγκαταλείπουν τις επισημασμένες θέσεις, δεν επιτρέπεται να ενεργοποιηθεί. Οι συγκεκριμένες θέσεις είναι χρήσιμες στο να δημιουργούν ατομικές ακολουθίες. Μέσα στο μοντέλο επισημαίνονται με C.



- **Normal locations:** όλα τα υπόλοιπα.

Τόσο οι αρχικές (initial) όσο και οι normal locations μπορεί σταθερές, δηλαδή συνθήκες που πρέπει να ικανοποιούνται όσο το αυτόματο παραμένει σε αυτή τη θέση. Οι σταθερές αυτές μπορούν να σχετίζονται είτε με τα ρολόγια είτε με τις μεταβλητές.





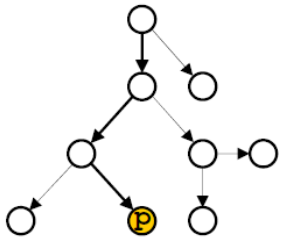
[Julián Proenza, The UPPAAL Model Checker, Systems]

3.2.2 Επαλήθευση στο UPPAAL

Εφόσον έχει χρησιμοποιηθεί ο προσομοιωτής (simulator), για να εξασφαλιστεί ότι το μοντέλο λειτουργεί και συμπεριφέρεται όπως ακριβώς το προς μοντελοποίηση σύστημα (και μερικές φορές για να ανιχνευτούν τυχόν λάθη στον αρχικό σχεδιασμό), η επόμενη φάση είναι να γίνει έλεγχος κατά πόσο επαληθεύονται οι ιδιότητες του συστήματος. Αρχικά λοιπόν θα πρέπει να γίνουν γνωστές αυτές οι ιδιότητες και να «επισημοποιηθούν» και κατά δεύτερον να μεταφραστούν αυτές οι ιδιότητες στη γλώσσα επερωτήσεων του UPPAAL (Uppaal query language). Τα είδη των ιδιοτήτων που μπορούν να ελέγχονται άμεσα χρησιμοποιώντας τα UPPAAL ερωτήματα είναι αρκετά απλά. Οι σχεδιαστές έχουν υιοθετήσει αυτή την προσέγγιση,

αντί να επιτρέπουν σύνθετα ερωτήματα, προκειμένου να βελτιωθεί η απόδοση του εργαλείου. Για το λόγο αυτό, η επαλήθευση των σύνθετων ιδιοτήτων μπορεί να απαιτεί τον έλεγχο πολλών διαφορετικών ερωτημάτων και ακόμη και την προσθήκη στο μοντέλο ειδικά σχεδιασμένων "ελεγμένων αυτομάτων". Οι ακριβείς κατηγορίες ιδιοτήτων που μπορούν να εκφραστούν στη γλώσσα επερωτήσεων του UPPAAL είναι:

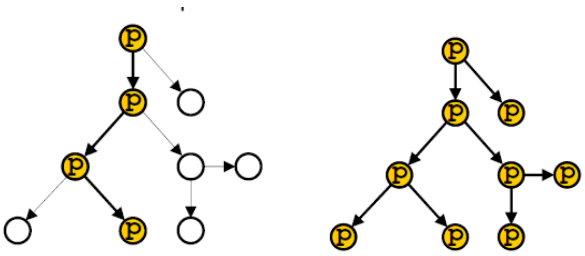
● **Reachability properties:** η συγκεκριμένη κατάσταση παραμένει σε κάποιο δεδομένο σταθμό των ενδεχόμενων συμπεριφορών του μοντέλου. Πάντα εκφράζονται με τον τύπο E < > p “Exists eventually p”, που σημαίνει ότι υπάρχει μια διαδρομή εκτέλεσης όπου το p τελικά παραμένει.



● **Safety properties**: Η κατάσταση παραμένει ίδια σε όλους τους σταθμούς της διαδρομής της εκτέλεσης. Υπάρχουν δύο διαφορετικές πιθανότητες:

E [ ] p “Exists globally p”, που σημαίνει ότι υπάρχει ένα μονοπάτι εκτέλεσης κατά το οποίο το p παραμένει σε όλους τους σταθμούς του μονοπατιού (α)

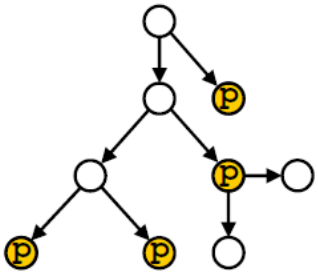
A [ ] p “Always globally p”, όπου για κάθε μονοπάτι εκτέλεσης το p διατηρείται σε όλο το μονοπάτι (β)



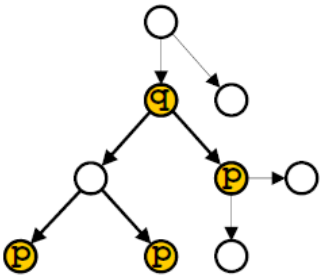
(α) (β)

● **Liveness properties:** μια συγκεκριμένη κατάσταση είναι εγγυημένο ότι θα διατηρηθεί περιστασιακά (σε μια δεδομένη στιγμή). Υπάρχουν και εδώ δύο πιθανότητες

A< > p “Always eventually p” όπου για κάθε μονοπάτι εκτέλεσης το p συγκρατείται μόνο για ένα σταθμό του μονοπατιού.



q→p “q always leads to p”, όπου κάθε μονοπάτι που ξεκινά από ένα σταθμό που παραμένει το q καταλήγει πάντα σε ένα σταθμό που παραμένει το p.



● **Deadlock properties** που είτε είναι δυνατές είτε όχι σε ένα μοντέλο. Κατάσταση **deadlock** επικρατεί εφόσον είναι δυνατό το μοντέλο να εξελίσσεται σε μια διαδοχική στάση χωρίς να περιμένει κάποιο χρονικό διάστημα ή μια μετάβαση μεταξύ θέσεων. Δύο τυπικά παραδείγματα είναι:

- E < > deadlock = “Exists deadlock”
- A [ ] not deadlock = “There is no deadlock

Σημειώνεται εδώ ότι η λέξη «deadlock» μπορεί να χρησιμοποιηθεί στο εσωτερικό οποιασδήποτε έκφρασης επισημοποιώντας μια συγκεκριμένη ιδιότητα. Όταν χρησιμοποιείται έλεγχος μοντέλου για την επαλήθευση ενός συστήματος, έχει συνήθως μελετηθεί αν υπάρχει αδιέξοδο στην μοντέλο ή όχι. Ακόμη και αξιοποιώντας την αφαιρετικότητα ο χώρος κατάστασης μπορεί να «εκραγεί». Υπάρχουν επιλογές ελέγχου που μπορούν να βοηθήσουν. Αν είναι ενεργοποιημένες κάποιες επιλογές, η έξοδος του ελεγκτή μπορεί να το γεγονός ότι η ιδιότητα ίσως είναι ικανοποιητική. Ο ελεγκτής δεν μπορεί να προσδιορίσει την τιμή αληθείας της ιδιότητας λόγω των χρησιμοποιούμενων προσεγγίσεων.

3.3 Εφαρμογή

3.3.1 Μοντελοποίηση του προβλήματος

Στο προηγούμενο κεφάλαιο περιγράφηκαν τρόποι επίλυσης του προβλήματος χρονοπρογραμματισμού, όπως επίσης και βελτιώσεις που λαμβάνουν υπόψη περισσότερα δεδομένα (όπως κόστη, ημερομηνίες), ώστε να υπάρχει μια πιο ρεαλιστική αποτύπωση των διαδικασιών και η προτεινόμενη λύση να αντιπροσωπεύει μια οικονομικότερη λύση.

Στο συγκεκριμένο μοντέλο έχουν ληφθεί υπόψη το κόστος αποθήκευσης σε περίπτωση που οι εργασίες περατωθούν ταχύτερα από το προβλεπόμενο, η ζημιά που προκαλείται (φήμη, ρήτρες, ποιότητα κλπ) εάν καθυστέρηση η περάτωση ή παράδοση κάποιας εργασίας. Για τη μείωση του κόστους, λαμβάνεται επίσης υπόψη το κόστος αδράνειας των μηχανών μεταξύ εργασιών, αλλά και ο συνολικός χρόνος περάτωσης των εργασιών.

3.3.2 Δομικά στοιχεία προγράμματος

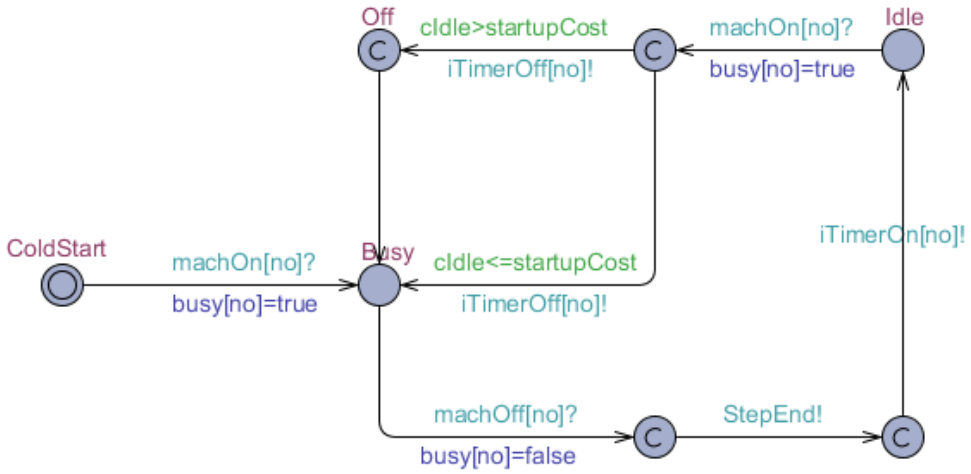
3.3.2.1 Η μηχανή

Για τα δεδομένα του προβλήματος θεωρούμε ότι η μηχανή μπορεί να βρίσκεται ουσιαστικά σε 3 καταστάσεις. Busy (απασχολημένη), Idle (σε ηρεμία) και Off (κλειστή).

Στην μοντελοποίηση βέβαια χρησιμοποιούμε παραπάνω διακριτές καταστάσεις οι οποίες βοηθούν στον καλύτερο έλεγχο της ροής του προγράμματος. Έτσι, ορίζεται επιπλέον η κατάσταση ColdStart η οποία υποδηλώνει την πρώτη εκκίνηση της μηχανής και 3 ακόμα committed καταστάσεις οι οποίες κυρίως βοηθούν να υπάρχει σύγχυση για τη σειρά των πράξεων και των διαδικασιών μέσω των καναλιών συγχρονισμού.

Ανάλογα με τον αριθμό των μηχανών που έχουν δηλωθεί στο System Declarations (Δηλώσεις Συστήματος), το πρόγραμμα πρέπει να δημιουργήσει αντίστοιχο αριθμό εικονικών μηχανών. Αυτό επιτυγχάνεται με την εισαγωγή σταθεράς παραμέτρου (*const int no*) ως αριθμό-ταυτότητα για κάθε μηχανή.

Στο template που ακολουθεί, η παράμετρος *no* θα αντικαθίσταται κάθε φορά από τον αριθμό κάθε μηχανής.



Σχήμα 3.6 – Template μηχανής

Κάθε μηχανή ξεκινά από την κατάσταση ColdStart.

Σημείωση: Η αρχική κατάσταση συμβολίζεται με δύο ομόκεντρους κύκλους στο Uppaal.

Μόλις δοθεί η εντολή (*machOn[no]!*) να ξεκινήσει η μηχανή, τότε αυτή μεταβαίνει στην κατάσταση Busy, ενημερώνοντας πρώτα τη μεταβλητή *busy[no]* σε *true*.

Σημείωση: Ό,τι απεικονίζεται με γαλάζιο χρώμα αφορά σε κανάλι συχρονισμού. Στα κανάλια (chan) ο συγχρονισμός επιτυγχάνεται από την εντολή με το θαυμαστικό (!) στο τέλος προς αυτή με το ερωτηματικό (?) και όχι αντίστροφα.

Η μεταβλητή *busy[no]* χρησιμεύει στην αναγνώριση, από τις άλλες εργασίες που θέλουν να χρησιμοποιήσουν τη μηχανή, ότι είναι απασχολημένη.

Όσο βρίσκεται στην κατάσταση *Busy* αναμένει πότε θα δοθεί η εντολή *machOff[no]!*, ώστε να συγχρονίσει με την *machOff[no]?* και να οδηγηθεί στην αποδέσμευση από τη συγκεκριμένη εργασία.

Μόλις ενεργοποιηθεί η *machOff[no]?*, τότε ενημερώνεται η μεταβλητή *busy[no]* σε *false* και ενεργοποιείται ακαριαία συγχρονισμός του *StepEnd*. Το *StepEnd* δίνει σήμα σε όλες τις εργασίες ότι κάποια μηχανή σταμάτησε τη λειτουργία οπότε μπορεί να χρησιμοποιηθεί από άλλη εργασία.

Η ενδιάμεση κατάσταση χρησιμοποιείται για να ξεχωρίσει τους δύο συγχρονισμούς που πραγματοποιούνται. Εάν χρησιμοποιούσαμε στο ίδιο βήμα το *machOff[no]?* και *StepEnd!* η μηχανή μπορεί να έφευγε από την κατάσταση *Busy* χωρίς να έχει τελειώσει η εργασία, δημιουργώντας πιθανή επικάλυψη (overlapping) μεταξύ των δύο εργασιών.

Σημείωση: Το "C" μέσα στον κύκλο προέρχεται από τη λέξη Committed υποδηλώνοντας ότι η συγκεκριμένη κατάσταση πρέπει να πραγματοποιηθεί ακαριαία, χωρίς να σπαταληθεί χρόνος συστήματος. Εάν δε φύγει από μία committed κατάσταση, το πρόγραμμα δεν μπορεί να συνεχίσει.

Αμέσως επόμενο βήμα είναι ο συγχρονισμός του καναλιού iTimerOn[no]! που δίνει σήμα ότι η μηχανή δεν είναι πλέον απασχολημένη και μετρά εάν συμφέρει η μηχανή να σβήσει (δεδομένου ότι υπάρχει κόστος έναρξης) ή να παραμείνει σε κατάσταση αναμονής μέχρι να ξαναχρειαστεί να χρησιμοποιηθεί.

Ο έλεγχος γίνεται μέσω ενός ρολογιού (clock cIdle;) το οποίο συγκρίνει πόση ώρα πρόκειται να μείνει ανενεργή η μηχανή με την παράμετρο startupCost, που υποδηλώνει ουσιαστικά το κόστος που απαιτείται για να ξεκινήσει η μηχανή μεταφρασμένο σε χρόνο για να είναι εφικτή η σύγκριση.

Η εντολή για να ξαναχρησιμοποιηθεί η μηχανή δίνεται μέσω του καναλιού machOn[no]? και τίθεται αυτόματα η μεταβλητή busy[no]=true, ώστε να φαίνεται απασχολημένη η μηχανή και να μην κληθεί από άλλη εργασία.

Αν και στην πράξη ο έλεγχος -για το αν η μηχανή πρέπει να σβήσει ή όχι για την εξοικονόμηση χρήματος- εκτελείται πριν ξανανοίξει η μηχανή, στην μοντελοποίηση έχουμε χρησιμοποιήσει τον έλεγχο ακριβώς μετά την έναρξη της μηχανής για άλλη εργασία, συνδέοντάς τον με μια ενδιάμεση κατάσταση committed (ώστε να μην υπάρχει σπατάλη χρόνου).

Αυτό, μας δίνει τη δυνατότητα να δημιουργήσουμε ένα πιο απλό μοντέλο μηχανής, το οποίο όμως δεν υπολείπεται σε λειτουργικότητα.

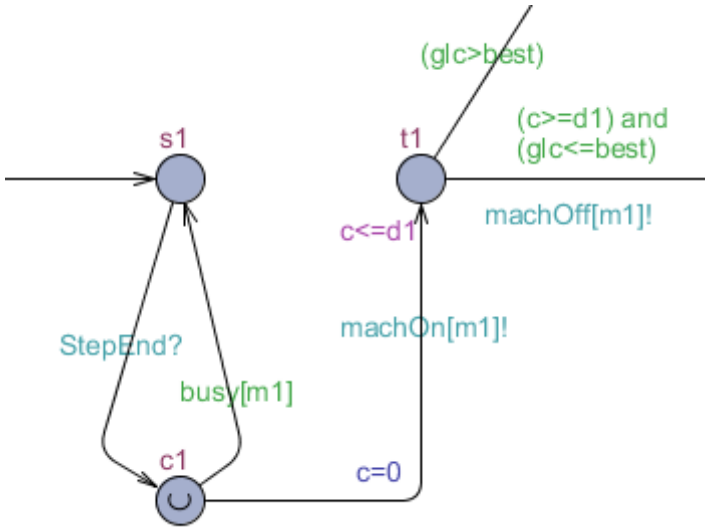
3.3.2.2 Η εργασία

Η κάθε εργασία αποτελεί μια αλληλουχία ενεργειών που πρέπει να γίνουν στις μηχανές, με συγκεκριμένη σειρά ώστε να ολοκληρωθεί.

Για μεγαλύτερη χρηστικότητα και ευκολία κατανόησης και συντήρησης του προγράμματος, επιλέξαμε να χρησιμοποιήσουμε ένα μόνο template για όλες τις εργασίες, στο οποίο θα μεταβάλλεται η σειρά των μηχανών που πρέπει να χρησιμοποιηθούν, ανάλογα με τις δηλώσεις που γίνονται στο System Declarations.

Για παράδειγμα στην παρακάτω εικόνα, το *m1* δεν αντιπροσωπεύει τη μηχανή 1, αλλά θα αντικατασταθεί με διαφορετικό (πιθανά) αριθμό μηχανής για κάθε εργασία. Υποδηλώνει τη δεύτερη μηχανή κατά σειρά που θα χρησιμοποιήσει η κάθε εργασία.

Σημείωση: Η αρίθμηση ξεκινά από το 0, οπότε το m1 συμβολίζει τη δεύτερη μηχανή και όχι την πρώτη.



Θα εξετάσουμε τη βασική λειτουργία της εργασίας όταν χρησιμοποιεί τη δεύτερη κατά σειρά μηχανή. Με αλληλουχίες της παραπάνω διάταξης μπορούμε να δημιουργήσουμε ένα template με όσες μηχανές επιθυμούμε για κάθε εργασία.

Χρησιμοποιώντας τη μηχανή βλέπουμε ότι υπάρχουν τρεις καταστάσεις για να αναπαραστήσουν την πραγματική λειτουργία.

- Στην κατάσταση s (*s1 στην συγκεκριμένη περίπτωση*) ξεκινά το πρόγραμμα να προσπαθεί να χρησιμοποιήσει την *m1*.
- Στην κατάσταση ελέγχου (control) c ελέγχεται εάν η επιθυμητή μηχανή είναι ελεύθερη για χρήση.
- Τέλος, στην κατάσταση t (time), η εργασία περιμένει όσο χρόνο χρειάζεται για να ολοκληρωθεί η χρήση της *m1* και να αποδεσμευτεί.

Αναλυτικά:
Φτάνοντας στην κατάσταση s1, το πρόγραμμα αναμένει να δοθεί σήμα ότι κάποια μηχανή έχει τελειώσει τη λειτουργία της, μέσω του καναλιού *StepEnd*.

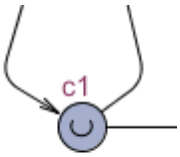
Σημείωση: Έχουμε χρησιμοποιήσει ένα μόνο κανάλι ελέγχου για το τέλος εργασίας των μηχανών, ενώ θα μπορούσε να έχει χρησιμοποιηθεί ξεχωριστό κανάλι για κάθε μηχανή. Με αυτόν τον τρόπο θα μειώναμε τους ελέγχους που πρέπει να πραγματοποιηθούν, αυξάνοντας την ταχύτητα του προγράμματος.

Η υλοποίηση που έχουμε επιλέξει δημιουργεί ένα ευανάγνωστο και λιγότερο περίπλοκο πρόγραμμα, εστιάζοντας στην ουσία του προβλήματος. Στην πράξη οι πόροι που δεσμεύονται είναι ελάχιστοι, χωρίς να επηρεάζουν το τελικό αποτέλεσμα.

Στην κατάσταση c1, ελέγχεται εάν η μεταβλητή *busy[m1]* έχει την τιμή *true*. Εάν ναι, υποδεικνύεται στο πρόγραμμα ότι η μηχανή είναι απασχολημένη και μεταβαίνει πίσω στην κατάσταση s1 αναμένοντας κάποια άλλη μηχανή να τελειώσει τη λειτουργία της.

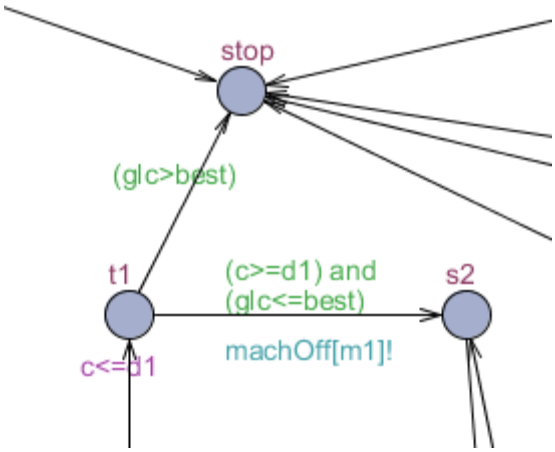
Έχοντας φτάσει ξανά στην κατάσταση c1 και εάν *busy[m1]=false*, το πρόγραμμα μεταβαίνει στην κατάσταση t1, ενημερώνοντας το ρολόι (clock c) με την τιμή 0.

Σημείωση: Το "U" στην κατάσταση c1 υποδηλώνει ότι είναι επείγον (Urgent) να φύγει από την κατάσταση αυτή το πρόγραμμα. Χρησιμοποιείται συνήθως όταν δεν υπάρχουν guards ή εντολές συγχρονισμού που να επιβάλλουν συγκεκριμένη χρονική στιγμή αποχώρησης από την κατάσταση.



Το ρολόι c χρησιμοποιείται ως τοπική μεταβλητή στο template Job για να καταγράφει πόσος χρόνος έχει δαπανηθεί για την ολοκλήρωση των ενεργειών σε κάθε μηχανή.

Παράλληλα ενεργοποιείται συγχρονισμός με το κανάλι machOn[m1] για να ξεκινήσει τη λειτουργία της η μηχανή. Φτάνοντας στην t1 θα παραμείνει σε αυτή την κατάσταση όσο χρόνο έχει ορισθεί από τη σταθερά d1 (const int d1).



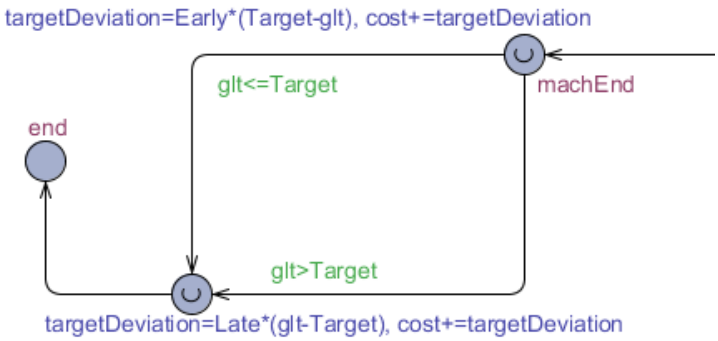
Στο σημείο αυτό πραγματοποιείται έλεγχος εάν ο χρόνος που έχει ήδη δαπανηθεί είναι μεγαλύτερος από ένα άνω όριο που έχει θέσει ο χρήστης (int best;). Το glc (global clock) μετράει τον συνολικό χρόνο από τη στιγμή εκκίνησης, οπότε εάν ξεπεράσει την τιμή best η εργασία οδηγείται στην κατάσταση stop όπου και σταματάει.

Σημείωση: Στην πράξη, για την εύρεση της βέλτιστης λύσης εκτελούμε ένα ερώτημα το οποίο έχει σαν προϋπόθεση όλες οι εργασίες να φτάσουν στην κατάσταση end. Το γεγονός ότι μεταβαίνει στην κατάσταση Stop δίνει τη δυνατότητα στον verifier να καταλάβει ότι η συγκεκριμένη πιθανή λύση δεν πρόκειται να φτάσει στην κατάσταση end, οπότε μεταβαίνει στην επόμενη πιθανή λύση, εξοικονομώντας υπολογιστικό χρόνο.

Στη συγκεκριμένη υλοποίηση έχουμε θέσει πολύ υψηλή τιμή στη μεταβλητή best ώστε να εξεταστούν όλες οι ενδεχόμενες λύσεις και το πρόγραμμα να προτείνει διαδρομή βάσει αυτών.

Δεδομένου λοιπόν ότι έχει παρέλθει ο απαιτούμενος χρόνος d1 και πως ο συνολικός χρόνος glc είναι μικρότερος από αυτόν που έχει ορισθεί από τον χρήστη, δίδεται εντολή στη μηχανή m1 να σβήσει. (machOff[m1]!).

Η ίδια διαδικασία εκτελείται και για τις υπόλοιπες μηχανές, ώσπου η εργασία φτάνει στην κατάσταση machEnd.



Στην κατάσταση αυτή θα γίνει έλεγχος εάν υπάρχει απόκλιση από το χρονικό όριο που είχε τεθεί για την ολοκλήρωση της παραγγελίας (const int Target).

Στην περίπτωση που η εργασία τελειώσει νωρίτερα από το αναμενόμενο, τότε υπάρχει κάποιο κόστος αποθήκευσης ανάλογο της χρονικής απόκλισης από τον στόχο. Οπότε, ορίζουμε μια μεταβλητή targetDeviation στην οποία θα αποθηκεύεται το κόστος της χρονικής απόκλισης.

Δίνονται δύο εντολές:

- targetDeviation=Early*(Target-glt)
- cost+=targetDeviation

όπου Early η παράμετρος που δηλώνει το κόστος αποθήκευσης για κάθε προϊόν ανά μονάδα χρόνου, Target η σταθερά που δηλώνει τον αναμενόμενο χρόνο ολοκλήρωσης και glt (global time) ο συνολικός χρόνος από την έναρξη του προγράμματος.

Σημείωση: Οι μεταβλητές glc και glt έχουν την ίδια τιμή. Η μόνη διαφορά είναι ότι η πρώτη είναι τύπου ρολογιού, ενώ η δεύτερη είναι ακέραια μεταβλητή. Τα ρολόγια μπορούν να χρησιμοποιηθούν για συγκρίσεις μεταξύ τους και με ακεραίους, αλλά μόνο οι ακέραιες μεταβλητές μπορούν να χρησιμοποιηθούν για αριθμητικές πράξεις.

Η πρώτη εντολή αποθηκεύει στην μεταβλητή targetDeviation το γινόμενο της χρονικής απόκλισης με το κόστος αποθήκευσης ανά μονάδα χρόνου, ενώ η δεύτερη εντολή αποθηκεύει το αποτέλεσμα στην μεταβλητή cost.

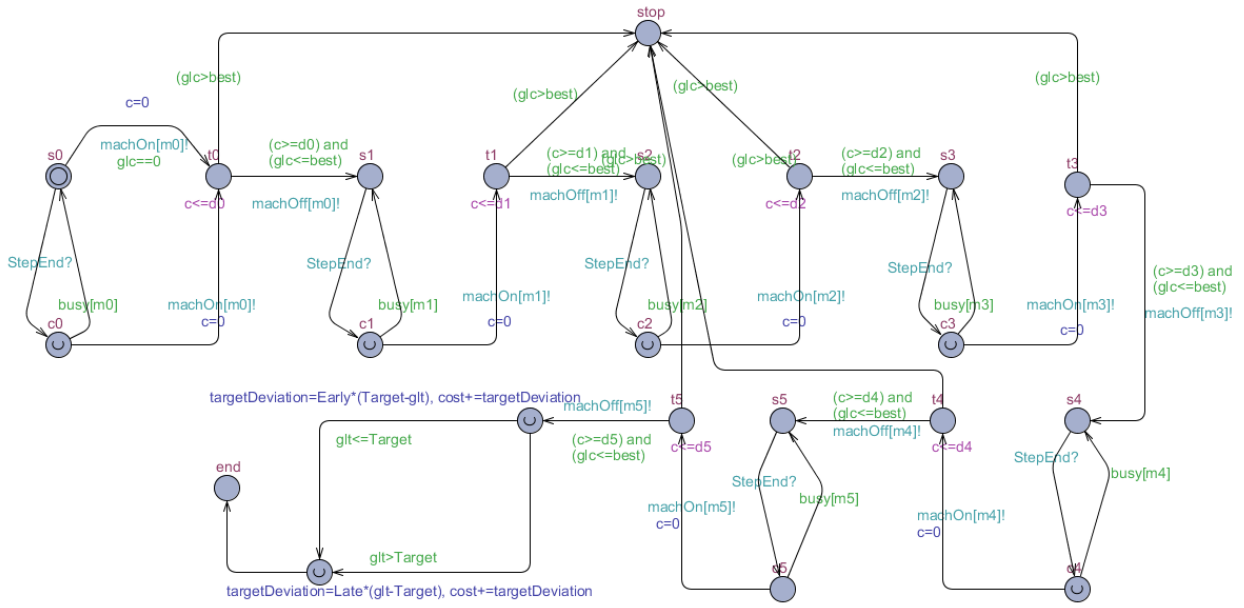
Σημείωση: Η μεταβλητή cost είναι ενσωματωμένη του προγράμματος Uppaal Cora και χρησιμοποιείται για την εισαγωγή κόστους στα χρονισμένα αυτόματα. Ο λόγος που χρειάζεται ενδιάμεση μεταβλητή (targetDeviation), αντί να χρησιμοποιηθεί κατευθείαν η εντολή cost+=Early(Target-glt), είναι λόγω συντακτικού του Uppaal Cora που δε μπορεί να πραγματοποιήσει αριθμητική πράξη και μετά να την καταχωρήσει στην μεταβλητή cost.*

Αντίστοιχα στην περίπτωση που η εργασία τελειώσει πια αργά από το αναμενόμενο, τότε υπάρχει κάποια ποινή που επιβάλλεται στο εργοστάσιο. Συνήθως αυτή προβλέπεται από τη σύμβαση με τους προμηθευτές. Στο κόστος αυτό θα μπορούσε να περιληφθεί η πιθανή ζημιά των προϊόντων σε περίπτωση που έχουν μικρή διάρκεια ζωής, ή και ακόμη η ζημιά στη φήμη της επιχείρησης.

Οι μόνες αλλαγές που χρειάζονται στην περίπτωση βραδείας περάτωσης της εργασίας είναι στη μεταβλητή Late, που δηλώνει το κόστος απόκλισης ανά μονάδα χρόνου και στον υπολογισμό χρόνου απόκλισης, ώστε να μη βγαίνει αρνητικός αριθμός.

- targetDeviation=Late*(glt-Target)

Όταν υπολογιστεί και αυτό το κόστος, η εργασία οδηγείται στο τέλος (κατάσταση end), μέσω μιας ενδιάμεσης επείγουσας (urgent) κατάστασης.



Σχήμα 3.7: Το συνολικό template της εργασίας.
[βασισμένο στο template που παρουσιάστηκε στη διπλωματική Δίπλα-Τσακίρη, 2004]

3.3.2.3 Μετρητές

Χρησιμοποιούμε 3 μετρητές χρόνου/κόστους για να μπορούμε εύκολα να προσαρμόσουμε το μοντέλο ανάλογα με τα δεδομένα. Σε κάθε υλοποίηση, μπορούμε να επιλέξουμε να μη χρησιμοποιήσουμε κάποιον από τους μετρητές, απλά θέτοντας μηδενική τιμή στις μεταβλητές.

Ακόμη και αν δεν έχουμε ακριβείς μετρήσεις για τα κόστη, προσαρμόζοντας τις τιμές των μετρητών είναι σαν να θέτουμε “βάρη” ανάλογα με τη σημαντικότητα. Αυτό, σε συνδυασμό με τις δηλώσεις σταθερών παραμέτρων των μηχανών και των εργασιών μας δίνει πλήρη ευελιξία και ένα ευρύ φάσμα μοντέλων.

Cost Timer

Ο μετρητής κόστους προσθέτει κάποια τιμή στο συνολικό κόστος σε κάθε χρονικό βήμα (5 στο παράδειγμα). Χρησιμοποιείται ώστε να περιοριστεί ο συνολικός χρόνος εκτέλεσης των εργασιών, αφού όταν επιτυγχάνεται βελτίωση ως προς αυτή τη μέτρηση (χωρίς να λαμβάνονται υπόψη άλλοι μετρητές ή παράγοντες που αυξάνουν το κόστος), σημαίνει ότι έχει βρεθεί η βέλτιστη λύση όσον αφορά στον συνολικό χρόνο.

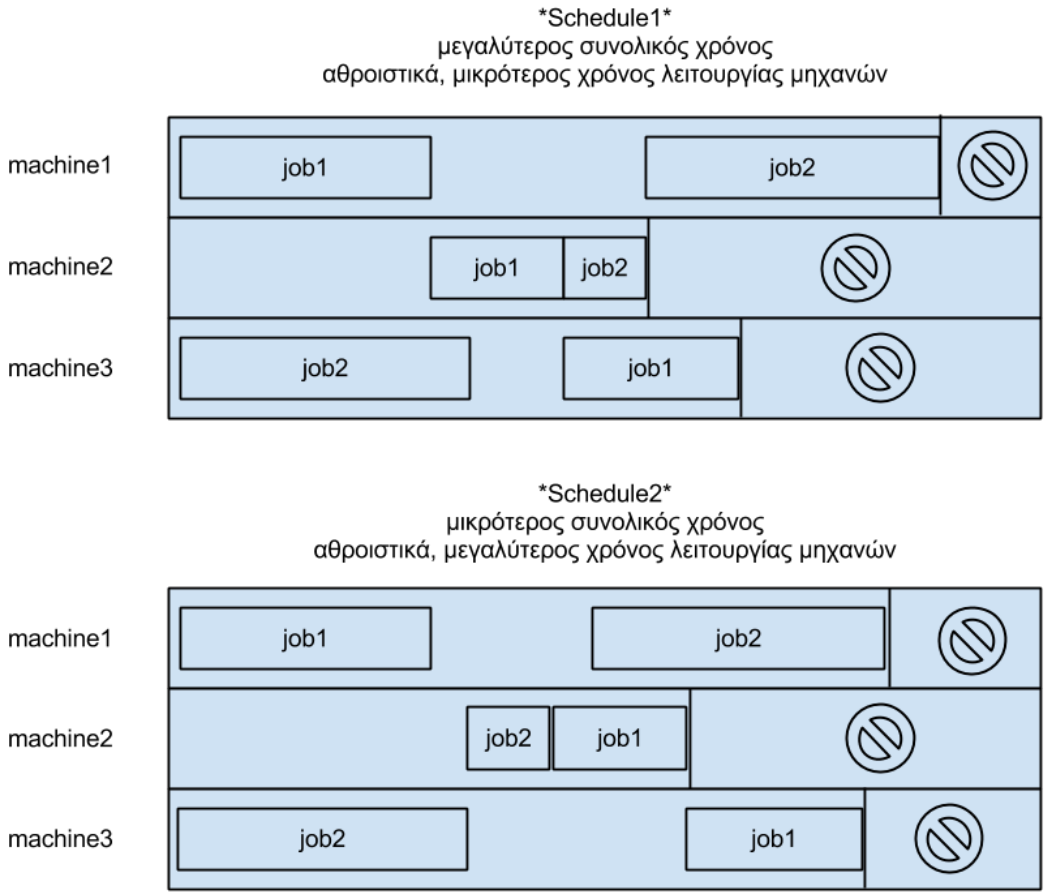
Η τιμή που αυξάνει σε κάθε χρονική μονάδα το κόστος ουσιαστικά δεν έχει σημασία εάν δεν υπάρχουν άλλοι παράγοντες, αφού

*συνολικό κόστος = κόστος ανά χρονική μονάδα * συνολικός χρόνος*

άρα το κόστος ανά μονάδα ως απλός παράγοντας δεν επηρεάζει τον τρόπο βελτίωσης του συνολικού κόστους.

Παρόλα αυτά όμως, αλλάζοντας την τιμή κόστους ανά χρονική μονάδα, όταν υπάρχουν και άλλοι παράγοντες που επηρεάζουν το κόστος, αλλάζει τη βαρύτητα που δίνεται στον συνολικό χρόνο εκτέλεσης έναντι των άλλων παραμέτρων.

Για παράδειγμα, στο παρακάτω σχήμα φαίνεται πώς εάν ο συνολικός χρόνος λειτουργίας των μηχανών είναι σημαντικότερος έναντι του συνολικού χρόνου λειτουργίας, τότε βελτιστοποιώντας μόνο ως προς τον συνολικό χρόνο, μπορεί να απορρίπταμε υλοποιήσεις που θα είχαν ακόμη μικρότερο κόστος.



Σχήμα 3.8

Στο σχήμα 3.8 γίνεται ενδεικτική αναπαράσταση προβλήματος 3 μηχανών και 2 εργασιών. Στην πράξη όσο αυξάνονται οι μηχανές και οι εργασίες, τόσο οι διάφορες παράμετροι επηρεάζουν το συνολικό κόστος και άρα η βαρύτητα που δίνεται σε κάθε παράμετρο έχει μεγαλύτερη σημασία και επίδραση στο συνολικό αποτέλεσμα.

Timer

Ο απλός μετρητής χρόνου σχεδιάστηκε για να καλύψει μια έλλειψη του προγράμματος μοντελοποίησης. Το Uppaal διαχωρίζει πλήρως τον χρόνο από το κόστος και δε δίνει τη δυνατότητα να αντιστοιχιστούν χρονικές τιμές (ρολογιών) στη μεταβλητή κόστους. Στην πράξη το κόστος (πχ αποθήκευσης) είναι άρρηκτα συνδεδεμένο με τον χρόνο.

Σχεδιάζοντας έναν μετρητή ο οποίος μετατρέπει κάθε χρονική μονάδα σε μονάδα κόστους, λύνει αυτό το πρόβλημα. Έπειτα ο χρόνος αυτός πολλαπλασιάζεται με το κόστος αποθήκευσης (ή αργοπορίας) ανά μονάδα χρόνου και το αποτέλεσμα είναι το συνολικό κόστος που αναζητούμε.

Σχήμα 3.9 – Timer

Αρχική κατάσταση θεωρείται αυτή με τους δύο ομόκεντρους κύκλους. Το clock c είναι ένα τοπικό ρολόι που χρησιμοποιείται για να εξασφαλίζει ότι για κάθε χρονικό βήμα (του global clock glc) θα αυξάνεται κατά μια μονάδα και το glt.

Σημειώνεται πως το glt είναι καθολική ακέραια μεταβλητή.

Περιγραφή λειτουργίας χρονομετρητή

Το ρολόι c ξεκινάει με την τιμή 0 (προεπιλογή σε όλα τα ρολόγια). Ξεκινώντας από την αρχική κατάσταση υπάρχει περιορισμός c<=1 (invariant). Αυτό σημαίνει ότι δε μπορεί να περιμένει στην κατάσταση αυτή για τιμή μεγαλύτερη του c από 1. Άρα οι πιθανές τιμές αποχώρησης είναι 0, 1.

Χρησιμοποιώντας και την προϋπόθεση (guard) c>0, εξασφαλίζουμε ότι το πρόγραμμα θα περιμένει ακριβώς 1 χρονική μονάδα στην κατάσταση αυτή πριν αποχωρήσει.

Αποχωρώντας από την αρχική κατάσταση αυξάνεται το glt (που είχε αρχική τιμή 0) κατά μια μονάδα.

Αντίστοιχα στην επόμενη κατάσταση χρησιμοποιείται invariant c<=2 και guard c>1 ώστε να διασφαλιστεί ότι θα περιμένει στην κατάσταση αυτή ακριβώς μια χρονική μονάδα.

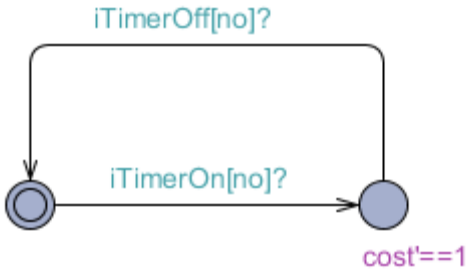
Αποχωρώντας από την κατάσταση αυτή αυξάνεται επίσης κατά μια μονάδα το glt και το τοπικό ρολόι c μηδενίζεται ώστε να ξανακάνει τη διαδικασία αυτή ξανά και ξανά, έως ότου οι εργασίες φτάσουν στην κατάσταση end.

Σημείωση: Στην πράξη υπάρχει αμελητέα απόκλιση της τιμής του glt και του glc το οποίο μπορεί να οφείλεται σε κάποιες στις committed καταστάσεις στις οποίες περνάει το πρόγραμμα, χωρίς να αυξάνεται η τιμή των ρολογιών. Η διαφορά αυτή δεν επηρεάζει τα αποτελέσματα, αφού το glc χρησιμοποιείται μόνο για τον συνολικό χρόνο εκτέλεσης, ενώ η glt για τον υπολογισμό κόστους αποθήκευσης ή αργοπορίας.

Idle Timer

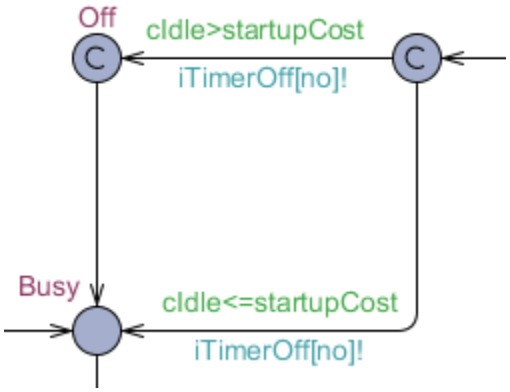
Ο μετρητής αυτός χρησιμοποιείται για να υπολογίζει τον χρόνο που κάθε μηχανή μένει ανενεργή και εάν χρειάζεται αποφασίζει το πρόγραμμα εάν συμφέρει να σβήσει τη μηχανή ή όχι.

Στην ουσία πρόκειται για ένα σύνολο μετρητών, ένα για κάθε μηχανή που χρησιμοποιείται (6 στην συγκεκριμένη περίπτωση).



Σχήμα 3.10 – Idle Timer

Η λειτουργία του είναι σχετικά απλή. Κάθε φορά που δίνεται εντολή από κάποια μηχανή να ενεργοποιηθεί ο Idle Timer, συγχρονίζει το κανάλι iTimerOn[no]? και το πρόγραμμα μεταβαίνει στην επόμενη κατάσταση, όπου το κόστος αυξάνεται κατά μια μονάδα (cost'==1) για κάθε χρονική μονάδα που παραμένει στην κατάσταση.



Αντίστοιχα, όταν δοθεί η εντολή να σβήσει ο Idle Timer τότε μεταβαίνει πάλι στην αρχική κατάσταση σταματώντας να αυξάνει το κόστος.

Είναι προφανές πως όσο περισσότερο παραμένει μια μηχανή σε κατάσταση Standby χωρίς να εκτελεί εργασία, το κόστος αυξάνεται, με μικρότερο βέβαια ρυθμό από ότι αυξάνεται όταν η μηχανή είναι απασχολημένη. Στην πράξη δε χρειάζεται να υπολογίσουμε και να συγκρίνουμε πόσο χρόνο παραμένει ενεργή κάθε μηχανή, αφού δεδομένων των παραγγελιών όλες οι μηχανές θα είναι απασχολημένες συνολικά για τον ίδιο χρόνο.

Στην περίπτωση που υπάρχουν παράλληλες μηχανές (δηλαδή διαφορετικές μηχανές που μπορούν να εκτελέσουν την ίδια εργασία) τότε θα πρέπει να υπολογίζεται ο συνολικός χρόνος που είναι απασχολημένη κάθε μηχανή, αφού λόγω ηλικίας, τεχνολογίας ή άλλων συνθηκών είναι πιθανό να εκτελεί ταχύτερα ή πιο οικονομικά κάποια εργασία.

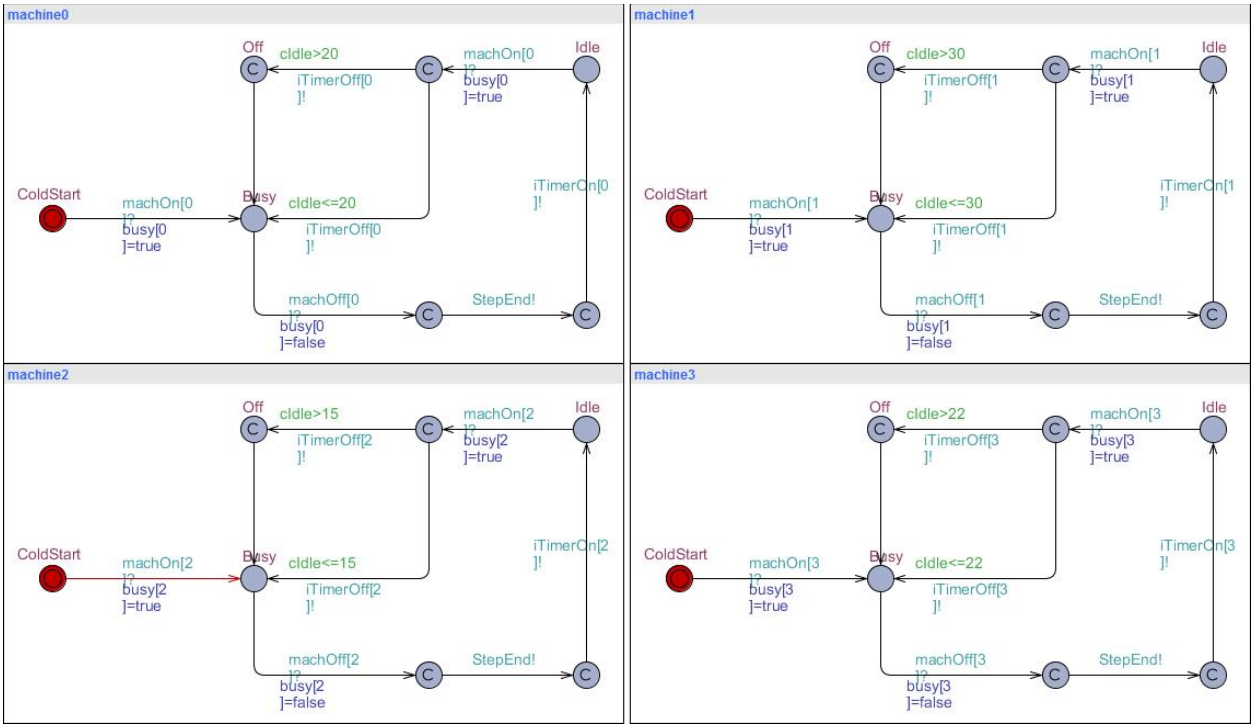
Στο συγκεκριμένο μοντέλο δεν εξετάζεται αυτό το ενδεχόμενο.

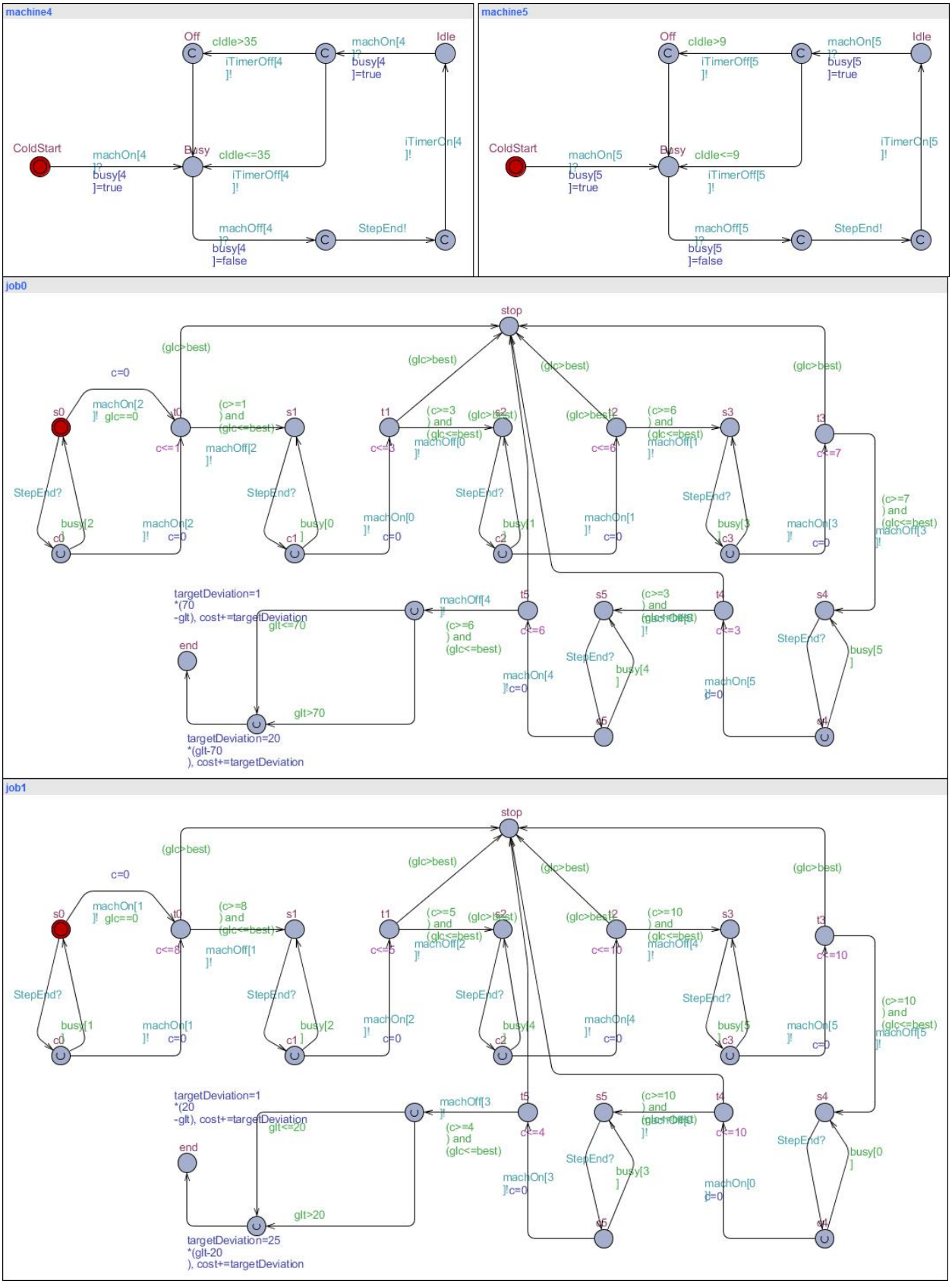
3.3.3 Προσομοίωση

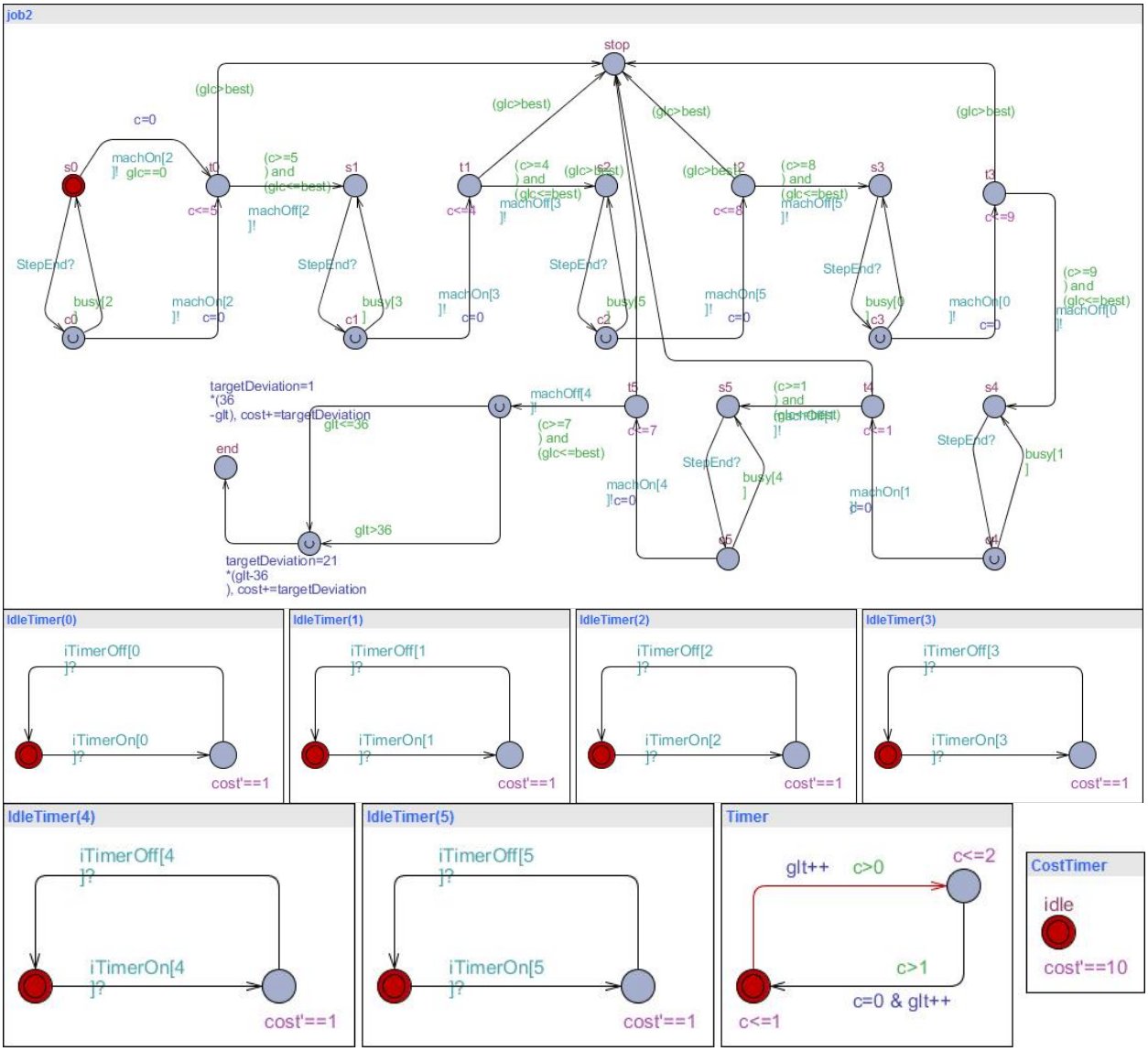
Χρησιμοποιώντας τον Simulator του Uppaal μπορούμε να ελέγξουμε τη ροή του προγράμματος με γραφική αναπαράσταση, γραμμική ροή, αλλά και ζωντανή ενημέρωση των μεταβλητών, των καταστάσεων και των μεταβάσεων. Υπάρχει επίσης δυνατότητα για εισαγωγή τυχαίων μεταβάσεων στο σύστημα για να εξεταστεί η συμπεριφορά του και να εντοπιστούν πιθανά κενά ή περιορισμοί.

Γραφική αναπαράσταση

Εδώ φαίνεται η αρχική κατάσταση κάθε στοιχείου του προγράμματος, βάσει του αντίστοιχου template. Όπως φαίνεται από τα σχήματα έχουν χρησιμοποιηθεί 6 μηχανές για 3 εργασίες. Επίσης στη γραφική αναπαράσταση φαίνονται και οι 6 μετρητές αναμονής, όπως επίσης και ο μετρητής χρόνου και συνολικού κόστους.



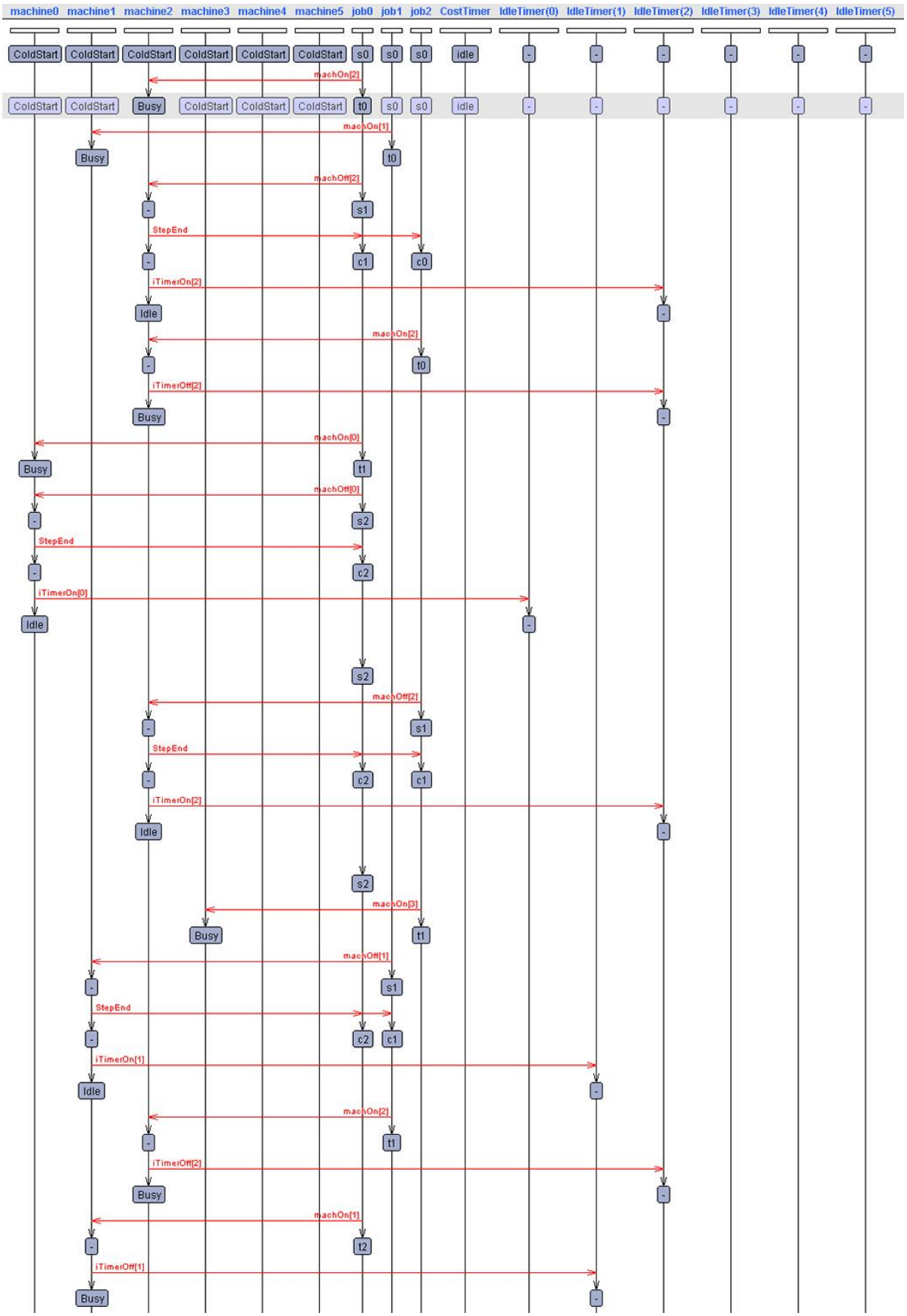

machine4
Off
cIdle>35
machOn[4]?
Idle
iTimerOff[4]!
busy[4]=true
ColdStart
machOn[4]?
Busy
cIdle<=35
busy[4]=true
iTimerOn[4]!
iTimerOff[4]!
machOff[4]?
StepEnd!
busy[4]=false
machine5
Off
cIdle>9
machOn[5]?
Idle
iTimerOff[5]!
busy[5]=true
ColdStart
machOn[5]?
Busy
cIdle<=9
iTimerOn[5]!
busy[5]=true
iTimerOff[5]!
machOff[5]?
StepEnd!
busy[5]=false
job0
stop
(glc>best)
c=0
s0
machOn[2]! glc==0
t0
(c>=1) and (glc<=best)
s1
c<=1
machOff[2]!
t1
(c>=3) and (glc<=best)
t2
(c>=6) and (glc<=best)
s3
t3
c<=7
machOff[0]!
machOff[1]!
StepEnd?
busy[2]
machOn[2]!
busy[0]
machOn[0]!
busy[1]
machOn[1]!
busy[3]
machOn[3]!
(c>=7) and (glc<=best)
machOff[3]!
targetDeviation=1 *(70 -glt), cost+=targetDeviation
machOff[4]!
t5
s5
t4
s4
(c>=6) and (glc<=best)
c<=6
c<=3
busy[4]
machOn[4]!c=0
machOn[5]!c=0
busy[5]
glt<=70
glt>70
end
targetDeviation=20 *(glt-70), cost+=targetDeviation
job1
stop
(glc>best)
c=0
s0
machOn[1]! glc==0
t0
(c>=8) and (glc<=best)
s1
c<=8
machOff[1]!
t1
(c>=5) and (glc<=best)
t2
(c>=10) and (glc<=best)
s3
t3
c<=10
machOff[2]!
machOff[4]!
StepEnd?
busy[1]
machOn[1]!
busy[2]
machOn[2]!
busy[4]
machOn[4]!
busy[5]
machOn[5]!
(c>=10) and (glc<=best)
machOff[5]!
targetDeviation=1 *(20 -glt), cost+=targetDeviation
machOff[3]!
t5
s5
t4
s4
(c>=4) and (glc<=best)
c<=4
busy[3]
machOn[3]!c=0
machOn[0]!c=0
busy[0]
glt<=20
glt>20
end
targetDeviation=25 *(glt-20), cost+=targetDeviation

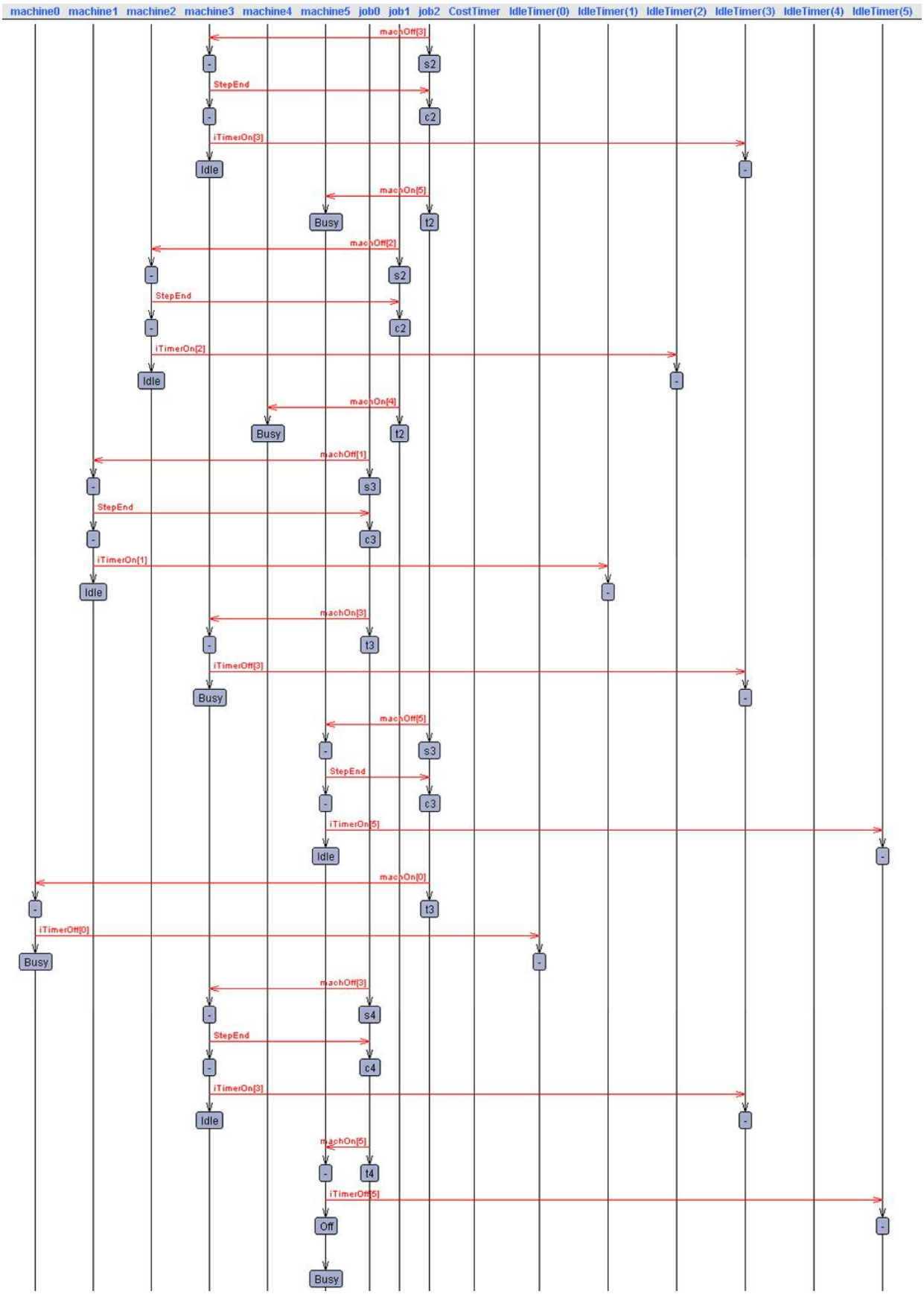


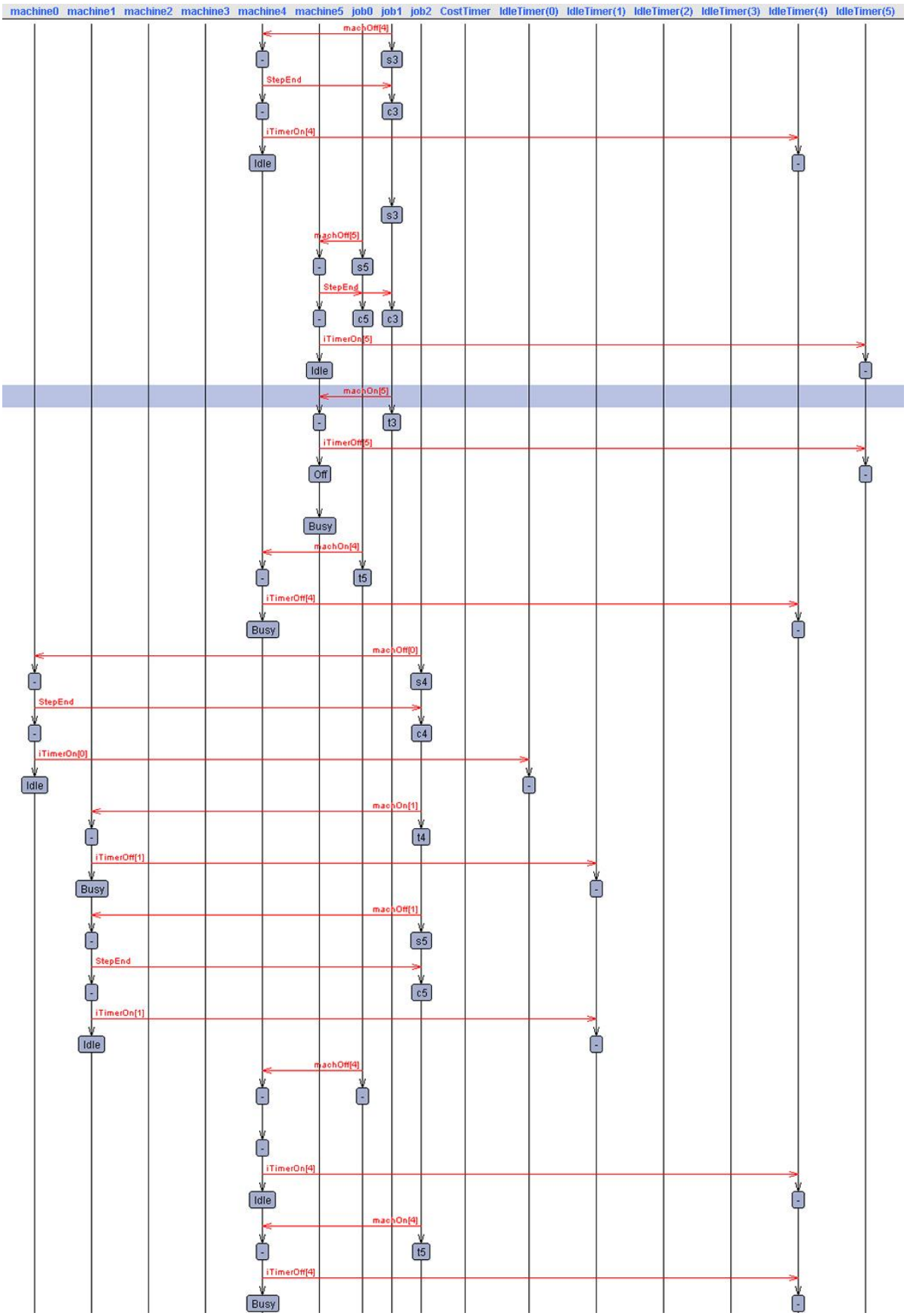
Σε κάθε μονάδα χρόνου γίνεται κόκκινη η κατάσταση που είναι ενεργή σε κάθε στοιχείο, ή η μετάβαση που εκτελείται.

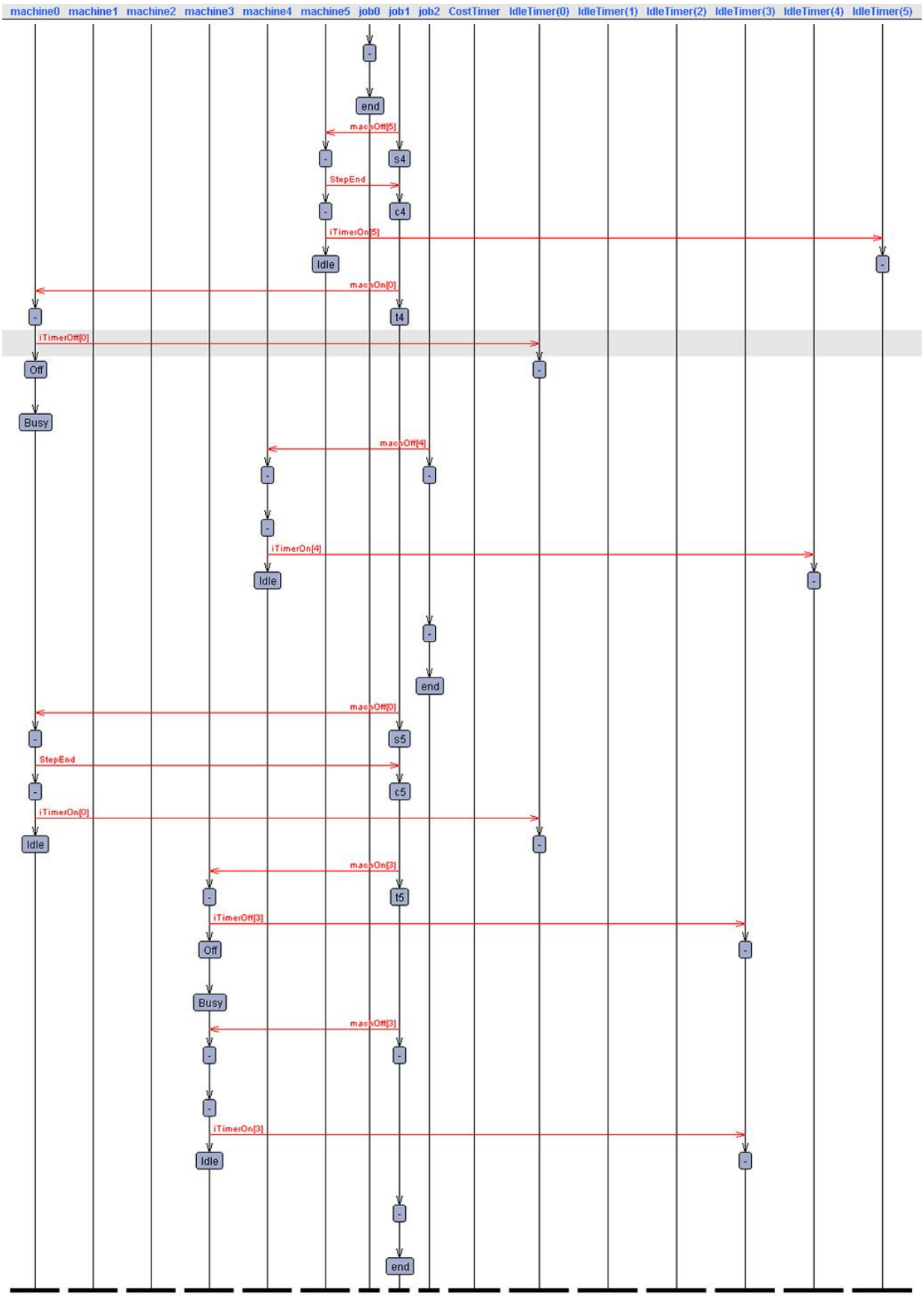
Γραμμική ροή

Στα παρακάτω σχήματα φαίνεται η ροή του προγράμματος για τυχαίες μεταβάσεις, σύμφωνα με την ενσωματωμένη λειτουργία random του Uppaal. Όπως φαίνεται τελικά όλες οι εργασίες φτάνουν στην κατάσταση end.


machine0
machine1
machine2
machine3
machine4
machine5
job0
job1
job2
CostTimer
IdleTimer(0)
IdleTimer(1)
IdleTimer(2)
IdleTimer(3)
IdleTimer(4)
IdleTimer(5)
ColdStart
Busy
Idle
s0
s1
s2
t0
t1
t2
c0
c1
c2
machOn[2]
machOn[1]
machOff[2]
StepEnd
iTimerOn[2]
machOn[0]
machOff[0]
iTimerOn[0]
machOn[3]
machOff[1]
iTimerOn[1]
iTimerOff[2]
iTimerOff[1]


machine0
machine1
machine2
machine3
machine4
machine5
job0
job1
job2
CostTimer
IdleTimer(0)
IdleTimer(1)
IdleTimer(2)
IdleTimer(3)
IdleTimer(4)
IdleTimer(5)
machOff[3]
s2
StepEnd
c2
iTimerOn[3]
Idle
machOn[5]
Busy
t2
machOff[2]
s2
StepEnd
c2
iTimerOn[2]
Idle
machOn[4]
Busy
t2
machOff[1]
s3
StepEnd
c3
iTimerOn[1]
Idle
machOn[3]
t3
iTimerOff[3]
Busy
machOff[5]
s3
StepEnd
c3
iTimerOn[5]
Idle
machOn[0]
t3
iTimerOff[0]
Busy
machOff[3]
s4
StepEnd
c4
iTimerOn[3]
Idle
machOn[5]
t4
iTimerOff[5]
Off
Busy


machine0
machine1
machine2
machine3
machine4
machine5
job0
job1
job2
CostTimer
IdleTimer(0)
IdleTimer(1)
IdleTimer(2)
IdleTimer(3)
IdleTimer(4)
IdleTimer(5)
machOff[4]
s3
StepEnd
c3
iTimerOn[4]
Idle
machOff[5]
s5
c5
iTimerOn[5]
machOn[5]
t3
iTimerOff[5]
Off
Busy
machOn[4]
t5
iTimerOff[4]
machOff[0]
s4
c4
iTimerOn[0]
machOn[1]
t4
iTimerOff[1]
machOff[1]
s5
c5
iTimerOn[1]
machOff[4]
iTimerOn[4]
machOn[4]
t5
iTimerOff[4]


machine0
machine1
machine2
machine3
machine4
machine5
job0
job1
job2
CostTimer
IdleTimer(0)
IdleTimer(1)
IdleTimer(2)
IdleTimer(3)
IdleTimer(4)
IdleTimer(5)
end
machOff[5]
s4
StepEnd
c4
iTimerOn[5]
Idle
machOn[0]
t4
iTimerOff[0]
Off
Busy
machOff[4]
iTimerOn[4]
Idle
end
machOff[0]
s5
StepEnd
c5
iTimerOn[0]
Idle
machOn[3]
t5
iTimerOff[3]
Off
Busy
machOff[3]
iTimerOn[3]
Idle
end

3.3.4 Επαλήθευση

Όπως φάνηκε στην προηγούμενη ενότητα, χρησιμοποιώντας την ενσωματωμένη λειτουργία random, το πρόγραμμα κατάφερε να βρει υλοποίηση ώστε όλες οι εργασίες να τελειώνουν. Αυτό ενώ φαίνεται προφανές, στην μοντελοποίηση υπάρχουν συνήθως διάφορες δυσκολίες ή λάθη τα οποία μπορεί να προκαλέσουν στο πρόγραμμα να μείνει κολλημένο σε κάποια κατάσταση ή να σταματήσει να η ροή.

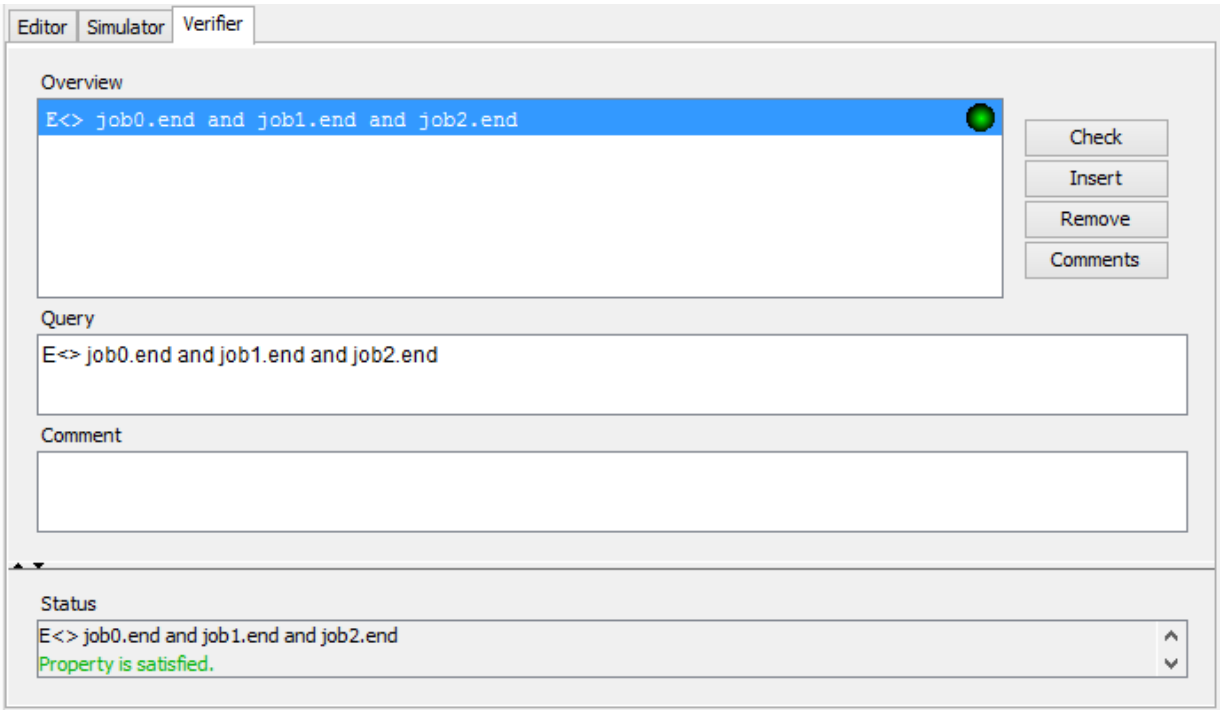
Το Uppaal διαθέτει τη μονάδα επαλήθευσης, όπου με κατάλληλα ερωτήματα γίνεται δυνατό να αντληθούν πληροφορίες σχετικά με την ύπαρξη ή όχι λύσης, τη μοναδικότητα και την καθολικότητα. Μπορούν να γίνουν συνδυασμοί των τελεστών A[], A<>, E<>, E[], -->, όπως περιγράφεται στην ενότητα 3.2.

Στο συγκεκριμένο πρόβλημα θα χρησιμοποιήσουμε μόνο το ερώτημα:

E<> job0.end and job1.end and job2.end

Έτσι θα διασφαλιστεί ότι το πρόβλημα έχει τουλάχιστον μια λύση για την οποία όλες οι εργασίες φτάνουν στο τέλος. Στο επόμενο κεφάλαιο θα επιλέξουμε από όλες τις λύσεις εκείνη που έχει το μικρότερο κόστος.

Όπως φαίνεται στην εικόνα 3.11 επαληθεύεται η ύπαρξη λύσης ώστε όλες οι εργασίες να φτάνουν στο τέλος.



Σχήμα 3.11 - Επαλήθευση

Κεφάλαιο4

Εκτέλεση πειραματικών δοκιμών

4.1 Εισαγωγή

Στο κεφάλαιο αυτό θα εξετάσουμε εάν η εξομοίωση του προγράμματος που αναφέρεται στο κεφάλαιο 3 παράγει ιδανικότερες λύσεις/μονοπάτια, βελτιώνοντας το συνολικό κόστος, καθώς και το ποσοστό βελτίωσης.

Για την επαλήθευση θα χρησιμοποιήσουμε την εφαρμογή verifyta.exe του Uppaal. Συγκεκριμένα θα χρησιμοποιήσουμε την εντολή *verifyta -t3* η οποία βρίσκει το βέλτιστο μονοπάτι μειώνοντας κατά το δυνατό την ενσωματωμένη μεταβλητή κόστους του Uppaal Cora, *cost*. Με την παράμετρο t3, το Uppaal Cora συνεχίζει να ψάχνει μέχρι να βρει μια ιδανική κατάσταση με τη μικρότερη τιμή για την μεταβλητή *cost*. [http://people.cs.aau.dk/~adavid/cora/options.html]

Θα ασχοληθούμε με δύο πειράματα για την επαλήθευση του μοντέλου.

- Στο πρώτο πείραμα χρησιμοποιούνται τυχαία δεδομένα για τους χρόνους/κόστη μηχανών, χρονικούς στόχους εργασιών, κόστη αποθήκευσης και πιθανή ζημία λόγω καθυστέρησης.
- Το δεύτερο πείραμα χρησιμοποιεί πραγματικά δεδομένα από το εργοστάσιο κατασκευής στρωμάτων της Candia Strom στην Παλλήνη, για 4 διαφορετικές ποιότητες προϊόντων (Classic, Onar, Interactive Velvet, Body-fix Heritage).

Με τη δομή του μοντέλου που περιγράφεται στο κεφάλαιο 3, μπορούμε να κάνουμε εξομοίωση για εργοστάσιο έως 6 μηχανών και απεριόριστων προϊόντων και εργασιών.

Για να υπολογίσουμε τη διαφορά ανάμεσα στη βελτιωμένη λύση job shop και στην απλή, θα χρησιμοποιήσουμε μια μεταβλητή (*int extra=0;*) στην οποία θα αποθηκεύεται το κόστος αποθήκευσης/ ζημία καθυστέρησης, αντί της ενσωματωμένης μεταβλητής κόστους *cost*, ώστε να μη γίνεται βελτιστοποίηση βάσει αυτών των παραμέτρων. Στην περίπτωση του Idle Timer, υπάρχει αυτόνομος μετρητής, οπότε το κόστος μπορεί να ρυθμίζεται και να διαχωρίζεται στις δύο περιπτώσεις, χωρίς την ανάγκη ύπαρξης νέας μεταβλητής.

4.2 Τυχαίο Πείραμα

Στο πείραμα αυτό χρησιμοποιούμε τυχαία δεδομένα για τη σειρά των μηχανών που πρέπει να απασχολήσει κάθε εργασία, όπως επίσης και για τους χρόνους. Τυχαίες είναι επίσης οι τιμές για κόστη αποθήκευσης, ζημία καθυστέρησης, idling.

Για τις 6 μηχανές χρησιμοποιούμε τα εξής στοιχεία:

- machine0 = Machine(0,20);
- machine1 = Machine(1,30);
- machine2 = Machine(2,15);
- machine3 = Machine(3,22);
- machine4 = Machine(4,35);
- machine5 = Machine(5,9);

όπου η πρώτη στήλη δείχνει τον αριθμό της μηχανής και η δεύτερη το κόστος έναρξης σε χρονικές μονάδες (μετά από πόσο χρόνο συμφέρει να κλείσει η μηχανή αν παραμένει αχρησιμοποίητη). Για τις 4 εργασίες έχουμε:

- job0=Job(1,4, 3,5, 0,8, 2,6, 4,4, 5,8, 70,1,20);
- job1=Job(3,6, 1,2, 5,9, 0,11, 2,7, 4,6, 20,1,25);
- job2=Job(1,4, 4,7, 3,4, 2,7, 0,2, 5,5, 36,1,21);
- job3=Job(5,8, 2,5, 3,6, 1,5, 4,9, 0,4, 11,2,15);

Τα 5 πρώτα ζευγάρια για κάθε εργασία δείχνουν τον αριθμό της μηχανής που πρέπει να απασχολήσουν και τον συνολικό χρόνο που απαιτείται σε κάθε μηχανή. Η σειρά κατά την οποία αναφέρονται τα ζεύγη τηρείται υποχρεωτικά κατά την εκτέλεση του προγράμματος.

4.2.1 Εκτέλεση βάσει βελτιστοποίησης χρόνου

Εκτελώντας την υλοποίηση για βελτιστοποίηση μόνο βάσει χρόνου, το συνολικό κόστος ανέρχεται στις **1009** μονάδες κόστους, ενώ με κατάλληλη επεξεργασία του log που παράγεται από το command line (διαθέσιμο στο παράρτημα) παρουσιάζεται η σειρά μεταβάσεων που προτείνεται.

job1 --> machine3
job0 --> machine1
job3 --> machine5
job2 --> machine1
job0 --> machine3
job2 --> machine4
job1 --> machine1
job0 --> machine0
job3 --> machine2

job1 --> machine5
job2 --> machine3
job0 --> machine2
job3 --> machine3
job1 --> machine0
job0 --> machine4
job3 --> machine1
job2 --> machine2
job3 --> machine4
job0 --> machine5
job2 --> machine0
job1 --> machine2
job1 --> machine4
job2 --> machine5
job3 --> machine0

4.2.2 Εκτέλεση βάσει συνδυασμένης βελτιστοποίησης χρόνου/κόστους

Στη συγκεκριμένη περίπτωση αντικαθιστούμε τη μεταβλητή extra με την ενσωματωμένη cost, ώστε να ληφθούν υπόψη όλα τα κόστη για την εξαγωγή της βέλτιστης λύσης (πλήρες log μεταβάσεων στο παράρτημα). Με την εκτέλεση του προγράμματος, το συνολικό κόστος ανέρχεται στις **961** μονάδες.

Λίστα μεταβάσεων:

job1 --> machine3
job0 --> machine1
job3 --> machine5
job2 --> machine1
job0 --> machine3
job2 --> machine4
job1 --> machine1
job0 --> machine0
job3 --> machine2
job1 --> machine5
job2 --> machine3
job0 --> machine2
job3 --> machine3
job1 --> machine0
job0 --> machine4
job2 --> machine2
job3 --> machine1
job3 --> machine4
job0 --> machine5

job2 --> machine0
job1 --> machine2
job1 --> machine4
job2 --> machine5
job3 --> machine0

4.2.3 Παρατηρήσεις

Στη δεύτερη περίπτωση, όπου λαμβάνουμε υπόψη τη βελτιστοποίηση κόστους μέσω περισσότερων παραγόντων, παρατηρούμε ότι υπάρχει μια εξοικονόμηση της τάξης του 5% ($\frac{1009 - 961}{1009}$.*100)

Παρατηρούμε επίσης ότι υπάρχει μια μικρή αλλαγή στη σειρά εκτέλεσης των εργασιών, το οποίο είναι επίσης αναμενόμενο.

4.3 Πείραμα Candia Strom

Στο πείραμα αυτό, θα ασχοληθούμε με την παραγωγική διαδικασία 4 ποιοτήτων στρωμάτων στο εργοστάσιο της Candia Strom στα Σπάτα και τις μηχανές που τα επεξεργάζονται μέχρι να καταλήξουμε στο τελικό προϊόν.

Οι τέσσερις ποιότητες είναι:

- Classic
- Onar
- Interactive Velvet
- Body-fix Heritage

Θα τις ονομάσουμε job0 έως job3 αντίστοιχα.

Στο εργοστάσιο χρησιμοποιούν 4 έως 5 μηχανές για την κατασκευή, επεξεργασία και συσκευασία των στρωμάτων, οπότε θα πρέπει να παραμετροποιήσουμε ελαφρώς το μοντέλο. Για να το πετύχουμε αυτό, δε χρειάζεται απαραίτητα να αλλάξουμε τα δομικά στοιχεία του Job template, αλλά θα τοποθετήσουμε ότι κάθε εργασία θα πρέπει να περάσει από την $6^{η}$ μηχανή (machine5) για 0 χρονικές μονάδες.

Οι πέντε μηχανές είναι οι εξής:

- Καπιτονιέρα (η μηχανή που φτιάχνει τα 2 καπάκια – το πάνω και κάτω μέρος του υφάσματος στο στρώμα)
- Δημιουργία τελάρου (ενώνει τις σούστες για τη δημιουργία του τελάρου)
- Δημιουργία φάσας (το πλαϊνό ύφασμα του στρώματος)
- Ρέλιασμα (Το ράψιμο του στρώματος – ένωση του καπακιού με τη φάσα)

- Συσκευαστική (συσκευασία των προϊόντων)

Οι εργασίες job1, job2 (Onar, Interactive Velvet) δεν περνάνε από τη $2^η$ μηχανή (machine1), επειδή τα τελάρα των στρωμάτων εισάγονται έτοιμα από εξωτερικό προμηθευτή.

Οι τιμές αναφέρονται σε παραγωγή στρωμάτων ποσότητας 100 έως 400 τεμαχίων.

- job0=Job(0,1, 1,8, 2,3, 3,16, 4,2, 5,0, 32,0,20);
- job1=Job(0,1, 1,15, 2,5, 3,23, 4,2, 5,0, 50,0,25);
- job2=Job(0,2, 1,11, 2,4, 3,18, 4,3, 5,0, 41,0,25);
- job3=Job(0,2, 1,23, 2,5, 3,29, 4,2, 5,0, 65,0,30);

Σημείωση: Στη στήλη του κόστους αποθήκευσης έχουμε συμπληρώσει 0, αφού το εργοστάσιο έχει ιδιόκτητη αποθήκη και το κόστος θεωρείται αμελητέο.

4.3.1 Εκτέλεση βάσει βελτιστοποίησης χρόνου

Το συνολικό κόστος περάτωσης των εργασιών, όταν βελτιστοποιείται βάσει χρόνου, ανέρχεται στις **2050** μονάδες κόστους.

Παρακάτω φαίνεται η σειρά μεταβάσεων που εξήγαγε ο αλγόριθμος. Αναλυτικά στο παράρτημα υπάρχουν όλες οι μεταβάσεις που πραγματοποιήθηκαν από το ανεπεξέργαστο log του Command line.

job0 --> machine0
job1 --> machine0
job0 --> machine1
job3 --> machine0
job2 --> machine0
job2 --> machine1
job0 --> machine2
job0 --> machine3
job2 --> machine2
job1 --> machine1
job2 --> machine3
job0 --> machine4
job3 --> machine1
job1 --> machine2
job1 --> machine3
job2 --> machine4
job3 --> machine2
job3 --> machine3

job1 --> machine4
job3 --> machine4
job0 --> machine5
job2 --> machine5
job1 --> machine5
job3 --> machine5

4.3.2 Εκτέλεση βάσει συνδυασμένης βελτιστοποίησης χρόνου/κόστους

Όπως και στο προηγούμενο πείραμα, αντικαθιστούμε τη μεταβλητή extra με την ενσωματωμένη μεταβλητή κόστους cost, ώστε η βελτιστοποίηση να λαμβάνει υπόψη και τα έξτρα κόστη. Έτσι προκύπτει ένα συνολικό κόστος **1775** μονάδων

Παρακάτω φαίνονται οι μεταβάσεις (Αναλυτικά στο παράρτημα).

job0 --> machine0
job0 --> machine1
job1 --> machine0
job2 --> machine0
job3 --> machine0
job2 --> machine1
job0 --> machine2
job0 --> machine3
job1 --> machine1
job2 --> machine2
job2 --> machine3
job0 --> machine4
job1 --> machine2
job3 --> machine1
job2 --> machine4
job1 --> machine3
job3 --> machine2
job3 --> machine3
job1 --> machine4
job3 --> machine4
job0 --> machine5
job2 --> machine5
job1 --> machine5
job3 --> machine5

4.3.3 Παρατηρήσεις

Όπως και στο προηγούμενο πείραμα παρατηρείται μείωση του κόστους, έπειτα από την εφαρμογή των βελτιστοποιήσεων, της τάξης του 13% ($\frac{2050 - 1775}{2050}. * 100$).

Επίσης παρατηρούνται μεγαλύτερες διαφορές στη σειρά εκτέλεσης των μεταβάσεων.

Κεφάλαιο 5

Συμπεράσματα & Προοπτικές

Στην παρούσα διπλωματική εργασία ασχοληθήκαμε με το πρόβλημα του χρονοπρογραμματισμού παραγωγής, τόσο θεωρητικά, όσο και πρακτικά. Η θεωρητική προσέγγιση αφορά στην ανάλυση του προβλήματος και την εξερεύνηση διαφόρων λύσεων που έχουν δοθεί στο πρόβλημα, ενώ η πρακτική προσέγγιση αφορά στη δημιουργία ενός γενικευμένου μοντέλου με τη χρήση χρονισμένων αυτομάτων και την ενσωμάτωση τεχνικών για μεγαλύτερη αποδοτικότητα.

Με τη χρήση προσομοιώσεων επιβεβαιώσαμε την ορθότητα της μεθοδολογίας και της σχεδίασης και καταλήξαμε στο συμπέρασμα ότι τα χρονισμένα αυτόματα είναι ένα αξιόλογο μοντέλο για χρήση στο πεδίο του χρονοπρογραμματισμού παραγωγής με ευρείες δυνατότητες εξέλιξης. Η προσπάθεια επικεντρώθηκε στην παραμετροποίηση των μοντέλων ώστε να λαμβάνουν υπόψη περισσότερους παράγοντες για την εξαγωγή του συνολικού κόστους, όπως επίσης και στη δημιουργία προσαρμοσμένων μετρητών για τον έλεγχο των αποτελεσμάτων. Στις πειραματικές δοκιμές παρατηρήθηκε μείωση του συνολικού κόστους, ακόμη και όταν ο χρόνος ολοκλήρωσης των εργασιών ήταν μεγαλύτερος.

Παρά τα διάφορα μικρά προβλήματα που προέκυψαν κατά τον σχεδιασμό της δομής του μοντέλου, έγινε ξεκάθαρο ότι τα χρονισμένα αυτόματα είναι ένας από τους ιδανικούς τρόπους να επιλύσουν το πρόβλημα χρονοπρογραμματισμού παραγωγής και είναι πολύ ευέλικτα, ώστε να προσαρμόζονται στις απαιτήσεις της εκάστοτε υλοποίησης. Εξάλου το πρώτο πείραμα αφορούσε περιβάλλον job-shop, ενώ το δεύτερο περιβάλλον flow-shop. Παρά ταύτα δε χρειάστηκε σχεδόν καμιά τροποποίηση ώστε να εκτελεστεί το πρόγραμμα.

Υπάρχουν κάποιες επεκτάσεις που θα μπορούσαν να πραγματοποιηθούν ώστε το μοντέλο να γίνει πιο γενικευμένο ώστε να καλύπτει περισσότερες περιπτώσεις και να αυξηθεί η χρηστικότητά του. Στην υλοποίηση του μοντέλου δεν έχει ληφθεί υπόψη η περίπτωση ύπαρξης παράλληλων μηχανών που μπορεί να έχουν τη δυνατότητα να εκτελέσουν την ίδια εργασία, όπως επίσης υπάρχει ο περιορισμός των 6 μηχανών που μπορεί να υποστηρίξει το μοντέλο. Τέλος, η δημιουργία ενός αλγορίθμου εξεύρεσης της βέλτιστης λύσης θα μπορούσε να αντικαταστήσει την ενσωματωμένη λειτουργία του verifier για τον εντοπισμό της οικονομικότερης διαδρομής. Αυτά θα μπορούσαν να πραγματοποιηθούν στο μέλλον για τη βελτίωση των αποτελεσμάτων του προγράμματος, αλλά κυρίως της χρηστικότητας και της δυνατότητας εφαρμογής σε ένα ευρύτερο πεδίο συστημάτων παραγωγής.

Βιβλιογραφία

- Alur, Dill – Automata for modelling Real-Time Systems.
- An optimization-based algorithm for job shop scheduling, JIHUA WANG, PETER B LUH, XING ZHAO and JINLIN WANG, Department of Electrical and Systems Engineering, University of Connecticut, April 1997
- Brooks, F.P. (1975). The Mythical Man-Hour. Reading, MA: Addison Wesley
- Brucker, P., 'Scheduling Algorithms', 3rd Edition, Springer – Verlag, Berlin (2001)
- Combining Neural Networks and CLP for Production Scheduling, Dimitrios Panopoulos, Dimitrios Ptochos, Athina Oikonomou, Dimitrios Askounis, John Psarras, School of Electrical and Computer Engineering National Technical University of Athens
- Emma Hart, Peter Ross, A Heuristic Combination Method for Solving Job-Shop Scheduling Problems, Department of Artificial Intelligence, University of Edinburgh, Edinburgh EH1 2QL, Scotland
- Frits Vaandrager, A First Introduction to Uppaal, Institute for Computing and Information Sciences, Radboud University Nijmegen
- Garrido, M. A. Salido, F. Barber, M. A. López, Heuristic Methods for Solving Job-Shop Scheduling Problems
- Gerd Behrmann, Alexandre David, and Kim G. Larsen, A Tutorial on Uppaal, Department of Computer Science, Aalborg University, Denmark
- Global Optimization Algorithms - Theory and Application
- H. Land and A. G. Doig (1960). "An automatic method of solving discrete programming problems". Econometrica 28 (3). pp. 497–520. doi:10.2307/1910129.
- Hansen, E.R. (1992). Global Optimization using Interval Analysis. New York: Marcel Dekker.
- Haykin, S. (1999) Neural Networks: A Comprehensive Foundation, Prentice Hall, ISBN 0-13-273350-1
- Hendrickson, C., B.N Janson (1984). "A Common Network Flow Formulation for Several Civil Engineering Problems". Civil Engineering Systems. 4 1: 195-203
- Hendrickson, Chris Tung, Au (2008). Advanced_Scheduling_Techniques .Project Management Construction (2.2 ed).Prentice Hall. Ανακτήθηκε 27 Οκτωβρίου 2011
- Jack R. Meredith, Sammel L. Mantel, JR "Project management: A management approach", Fifth edition.
- Jaulin, L.; Kieffer, M.; Didrit, O.; Walter, E. (2001). Applied Interval Analysis. Berlin: Springer. ISBN 1-85233-219-0.
- Jayamohan, M.S., Rajendran, C., 'New dispatching rules for shop scheduling: a step forward', Int J. Prod. Res., 38, (3), pp. 563-586 (2000)
- Jeffrey W. Herrmann, The Legacy of Taylor, Gantt, and Johnson: How to Improve Production Scheduling, The InsTITuTe for sysTems research.Isr TechnIcal rePorT 2007-26
- Joc Cing Tay *, Nhu Binh Ho, Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems, Computers & Industrial Engineering

54 (2008) 453–473,Evolutionary and Complex Systems Program, School of Computer Engineering, Nanyang Technological University

- Johan Bengtsson, Kim Larsen, Fredik Larsson, Paul Pettersson, Wang Yi: UPPAAL – a Tool Suite for Automatic Verification of Real-Time Systems.
- Julián Proenza,The UPPAAL Model Checker,Systems, Robotics and Vision Group. UIB. SPAIN,Oct. 2008
- K.G. Larsen, P. Pettersson, Wang Yi : ‘’UPPAAL in a Nutshell’’
- Lin S-F.; Chen J-Y.Two Parallel Machines Scheduling,: Systems Analysis Modelling Simulation, Volume 42, Number 10, 1 January 2002 , pp. 1429-1437(9)
- Marius Mikucionis, Egle Sasnauskaite – On the fly testing using UPPAAL (Master thesis)
- Moore, R. E. (1966). Interval Analysis. Englewood Cliff, New Jersey: Prentice-Hall. ISBN 0-13-476853-1.
- Paul Pettersson – Modelling and Verification of Real-Time Systems Using Timed-Automata
- Poli, R., Langdon, W. B., McPhee, N. F. (2008), A Field Guide to Genetic Programming, Lulu.com, freely available from the internet ISBN 978-1-4092-0073-4
- Rajendran C., Ziegler, H., ‘Heuristics for scheduling in flowshops and flowline-based manufacturing cells to minimize the sum of weighted flowtime and weighted tardiness of jobs’, Computers & Industrial Engineering, 37, pp. 671-690 (1999)
- S.J. Hu, J. Ko, L. Weyand, H.A. ElMaraghy, T.K. Lien, Y. Koren, H. Bley, G.Chryssolouris, N. Nasr, M. Shpitalni , Assembly system design and operations for product variety, CIRP Annals - Manufacturing Technology Volume 60, Issue 2, 2011, Pages 715–733
- Selin Bilgin Özpeynirci a, Meral Azizog˘lu b,Beam search algorithm for capacity allocation problem in flexible manufacturing systems, Computers & Industrial Engineering 56 (2009)
- Stuart Russel και Peter Norvig, Τεχνητή Νοημοσύνη, μια σύγχρονη προσέγγιση
- Subanatarajan Subbiaha,, Thomas Tometzkia, Sebastian Panekb, Sebastian Engella,Multi-product batch scheduling with intermediate due dates using priced timed automata models, Computers and Chemical Engineering 33 (2009) 1661–1676
- The disjunctive graph machine representation of the job shop scheduling problem,Jacek Blazewicz ,Erwin Pesch, Malgorzata Sterna , European Journal of Operational Research 127 (2000)
- V. Vinoda , R. Sridharan,Simulation modeling and analysis of due-date assignment methods and scheduling decision rules in a dynamic job shop production system,Int. J. Production Economics 129 (2011) 127–14
- Yasmina Abdeddaim, Eugene Asarin, Oded Maler, Scheduling with timed automata,Theoretical Computer Science 354 (2006) 272 – 300
- Βλαχάβας, Κεφαλάς, Βασιλειάδης, Κόκκορας, Τεχνητή Νοημοσύνη - Γ' Έκδοση, Εκδόσεις Γκιούρδας, Σακελλαρίου, 2011
- Γιάννης Α. Φίλης, Μάρτιος 2006, Συστήματα Παραγωγής, Πολυτεχνείο Κρήτης.
- Διαμαντάρας, Κ. (2007) Τεχνητά Νευρωνικά Δίκτυα, Κλειδάριθμος, ISBN 9604610805
- Δίπλας Αλέξανδρος - Τσακίρης Χρήστος, (Αθήνα Φεβρουάριος 2004)

- Εργαστήριο Συστημάτων Αποφάσεων και Διοίκησης, Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών, ΕΜΠ, http://academics.epu.ntua.gr
- Καλοειδάς Εμμανουήλ, (Αθήνα Σεπτέμβριος 2012) Μελέτη χρονοπρογραμματισμού έργων υπό περιορισμένους πόρους και διακριτή σχέση χρόνου-κόστους, Διπλωματική Εργασία για τη Σχολή Μηχανολόγων Μηχανικών, ΕΜΠ
- Κάντζαρη Μαρία, (Πάτρα 2010). Μοντέλα για τον χρονοπρογραμματισμό έργων με περιορισμένους πόρους, Πανεπιστήμιο Πατρών Τμήμα Μαθηματικών-Μηχανικών Η/Υ & Πληροφορικής. Μεταπτυχιακό Πρόγραμμα Σπουδών «Μαθηματικά των Υπολογιστών και των Αποφάσεων»
- Καρβούνης Κ. Σωτήρης, «Οικονομοτεχνικές Μελέτες», Εκδόσεις ΑΘ. Σταμούλη, 2000
- Καρόπουλος Παναγιώτης, (Αθήνα Ιανουάριος 2005) Μελέτη Εργαλείων Υλοποίησης Εμπειρων Συστημάτων Για Εφαρμογή Στον Χρονοπρογραμματισμό Παραγωγής, Διπλωματική Εργασία για τη Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών, ΕΜΠ
- Λεραντζής Παρασκευάς, (Αθήνα Οκτώβριος 2004) Βελτιστοποίηση χρονοπρογραμματισμού παραγωγής με χρήση γενετικών αλγορίθμων
- Λύγουρας Χρήστος, Σκλαβενίτη Μαρία (Αθήνα Μάιος 2004) Προγραμματισμός Παραγωγής με χρήση μηχανών πεπερασμένων καταστάσεων,Διπλωματική Εργασία για το ΔΠΜΣ Τεχνοοικονομικά Συστήματα, Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών, ΕΜΠ
- Ματσατσίνης Ν., Συστήματα Υποστήριξης Αποφάσεων, Εκδόσεις Νέων Τεχνολογιών, 2010
- Μαυράκης Δημήτριος , «Στοιχεία Διοίκησης Επιχειρήσεων», Εκδόσεις Ε.Κ.Π.Α , Αθήνα 2005
- Φερλέμης Νικόλαος (Αθήνα Νοέμβριος 2012), Συγκριτική αξιολόγηση μεθόδων επίλυσης προβλημάτων χρονοπρογραμματισμού εργασιών, Διπλωματική Εργασία για τη Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών, ΕΜΠ

Παράρτημα

Στην ενότητα αυτή υπάρχουν τα αρχεία xml που χρησιμοποιήθηκαν για την μοντελοποίηση των πειραμάτων, όπως επίσης και οι μεταβάσεις από τα επεξεργασμένα log files του Command Line για την εντολή verifyta -t3.

ΠΕΙΡΑΜΑ 4.2.1 – Αρχείο xml

<?xml version="1.0" encoding="utf-8"?> <!DOCTYPE nta PUBLIC '-//Uppaal Team//DTD Flat System 1.1//EN' 'http://www.it.uu.se/research/group/darts/uppaal/flat-1_1.dtd'> <nta> <declaration>//Insert declarations of global clocks, variables, constants and channels.
chan machOn[6]; chan machOff[6];
chan iTimerOn[6]; chan iTimerOff[6];
clock glc;
bool busy[6]={false, false, false, false, false, false};
urgent broadcast chan StepEnd;
chan trigger;
int best=1000;
meta int[-1000,1000] remaining;
int glt=0;
int extra=0;</declaration> <template> <name x="5" y="5">Job</name> <parameter>const int m0, const int d0, const int m1, const int d1, const int m2, const int d2, const int m3, const int d3, const int m4, const int d4, const int m5, const int d5, const int Target, const int Early, const int Late</parameter> <declaration>clock c;
clock k;
int targetDeviation=0;</declaration> <location id="id0" x="-128" y="504"> <name x="-138" y="474">end</name> </location> <location id="id1" x="-40" y="584"> <urgent/> </location> <location id="id2" x="304" y="440"> <name x="294" y="410">t5</name> <label kind="invariant" x="294" y="455">c<=d5</label> </location> <location id="id3" x="400" y="600"> <name x="390" y="570">c5</name> </location> <location id="id4" x="400" y="440"> <name x="390" y="410">s5</name> </location> <location id="id5" x="552" y="440"> <name x="542" y="410">t4</name> <label kind="invariant" x="542" y="455">c<=d4</label> </location> <location id="id6" x="680" y="600"> <name x="670" y="570">c4</name> <urgent/> </location> <location id="id7" x="680" y="440"> <name x="670" y="410">s4</name> </location> <location id="id8" x="-288" y="208"> <name x="-298" y="178">s0</name> </location> <location id="id9" x="-136" y="208"> <name x="-146" y="178">t0</name> <label kind="invariant" x="-168" y="224">c<=d0</label> </location> <location id="id10" x="8" y="208"> <name x="-2" y="178">s1</name> </location> <location id="id11" x="128" y="208"> <name x="118" y="178">t1</name> <label kind="invariant" x="104" y="224">c<=d1</label> </location> <location id="id12" x="272" y="208"> <name x="262" y="178">s2</name> </location> <location id="id13" x="416" y="208"> <name x="406" y="178">t2</name> <label kind="invariant" x="384" y="224">c<=d2</label> </location> <location id="id14" x="560" y="208"> <name x="550" y="178">s3</name> </location> <location id="id15" x="680" y="216"> <name x="670" y="186">t3</name> <label kind="invariant" x="670" y="231">c<=d3</label> </location> <location id="id16" x="160" y="440"> <urgent/> </location> <location id="id17" x="272" y="-16"> <name x="262" y="-46">stop</name> </location> <location id="id18" x="-288" y="368"> <name x="-298" y="338">c0</name> <urgent/> </location> <location id="id19" x="8" y="368"> <name x="-2" y="338">c1</name> <urgent/> </location> <location id="id20" x="272" y="368"> <name x="262" y="338">c2</name> <urgent/> </location> <location id="id21" x="560" y="368"> <name x="550" y="338">c3</name> <urgent/> </location> <init ref="id8"/> <transition> <source

ref="id1"/> <target ref="id0"/> <nail x="-128" y="584"/> </transition> <transition> <source ref="id16"/> <target ref="id1"/> <label kind="guard" x="8" y="560">glt>Target</label> <label kind="assignment" x="-128" y="592">targetDeviation=Late*(glt-Target), extra=targetDeviation</label> <nail x="160" y="584"/> </transition> <transition> <source ref="id16"/> <target ref="id1"/> <label kind="guard" x="-24" y="448">glt<=Target</label> <label kind="assignment" x="-144" y="408">targetDeviation=Early*(Target-glt), extra=targetDeviation</label> <nail x="-40" y="440"/> </transition> <transition> <source ref="id2"/> <target ref="id17"/> <label kind="guard" x="228" y="182">(glc>best)</label> <nail x="304" y="256"/> </transition> <transition> <source ref="id5"/> <target ref="id17"/> <label kind="guard" x="352" y="182">(glc>best)</label> <nail x="552" y="392"/> <nail x="336" y="392"/> </transition> <transition> <source ref="id3"/> <target ref="id4"/> <label kind="guard" x="416" y="504">busy[m5]</label> <nail x="440" y="504"/> </transition> <transition> <source ref="id3"/> <target ref="id2"/> <label kind="synchronisation" x="272" y="512">machOn[m5]!</label> <label kind="assignment" x="280" y="528">c=0</label> <nail x="304" y="600"/> </transition> <transition> <source ref="id4"/> <target ref="id3"/> <label kind="synchronisation" x="352" y="488">StepEnd?</label> <nail x="376" y="496"/> </transition> <transition> <source ref="id5"/> <target ref="id4"/> <label kind="guard" x="448" y="408">(c>=d4) and (glc<=best)</label> <label kind="synchronisation" x="448" y="440">machOff[m4]!</label> </transition> <transition> <source ref="id6"/> <target ref="id5"/> <label kind="synchronisation" x="528" y="520">machOn[m4]!</label> <label kind="assignment" x="528" y="536">c=0</label> <nail x="552" y="600"/> </transition> <transition> <source ref="id6"/> <target ref="id7"/> <label kind="guard" x="712" y="480">busy[m4]</label> <nail x="712" y="488"/> </transition> <transition> <source ref="id7"/> <target ref="id6"/> <label kind="synchronisation" x="620" y="465">StepEnd?</label> <nail x="656" y="480"/> </transition> <transition> <source ref="id15"/> <target ref="id7"/> <label kind="guard" x="752" y="288">(c>=d3) and (glc<=best)</label> <label kind="synchronisation" x="744" y="328">machOff[m3]!</label> <nail x="744" y="216"/> <nail x="744" y="440"/> </transition> <transition> <source ref="id2"/> <target ref="id16"/> <label kind="guard" x="200" y="448">(c>=d5) and (glc<=best)</label> <label kind="synchronisation" x="192" y="416">machOff[m5]!</label> </transition> <transition> <source ref="id8"/> <target ref="id9"/> <label kind="guard" x="-232" y="184">glc==0</label> <label kind="synchronisation" x="-248" y="168">machOn[m0]!</label> <label kind="assignment" x="-224" y="128">c=0</label> <nail x="-256" y="160"/> <nail x="-176" y="160"/> </transition> <transition> <source ref="id9"/> <target ref="id10"/> <label kind="guard" x="-88" y="168">(c>=d0) and (glc<=best)</label> <label kind="synchronisation" x="-104" y="224">machOff[m0]!</label> </transition> <transition> <source ref="id11"/> <target ref="id12"/> <label kind="guard" x="176" y="176">(c>=d1) and (glc<=best)</label> <label kind="synchronisation" x="160" y="216">machOff[m1]!</label> </transition> <transition> <source ref="id13"/> <target ref="id14"/> <label kind="guard" x="456" y="176">(c>=d2) and (glc<=best)</label> <label kind="synchronisation" x="456" y="216">machOff[m2]!</label> </transition> <transition> <source ref="id9"/> <target ref="id17"/> <label kind="guard" x="-176" y="8">(glc>best)</label> <nail x="-136" y="-16"/> </transition> <transition> <source

ref="id11"/> <target ref="id17"/> <label kind="guard" x="140" y="66">(glc>best)</label> </transition> <transition> <source ref="id13"/> <target ref="id17"/> <label kind="guard" x="284" y="66">(glc>best)</label> </transition> <transition> <source ref="id15"/> <target ref="id17"/> <label kind="guard" x="640" y="48">(glc>best)</label> <nail x="680" y="-16"/> </transition> <transition> <source ref="id18"/> <target ref="id9"/> <label kind="synchronisation" x="-184" y="320">machOn[m0]!</label> <label kind="assignment" x="-152" y="336">c=0</label> <nail x="-136" y="368"/> </transition> <transition> <source ref="id19"/> <target ref="id11"/> <label kind="synchronisation" x="96" y="312">machOn[m1]!</label> <label kind="assignment" x="120" y="336">c=0</label> <nail x="128" y="368"/> </transition> <transition> <source ref="id21"/> <target ref="id15"/> <label kind="synchronisation" x="632" y="320">machOn[m3]!</label> <label kind="assignment" x="664" y="344">c=0</label> <nail x="680" y="368"/> </transition> <transition> <source ref="id20"/> <target ref="id13"/> <label kind="synchronisation" x="376" y="312">machOn[m2]!</label> <label kind="assignment" x="400" y="336">c=0</label> <nail x="416" y="368"/> </transition> <transition> <source ref="id10"/> <target ref="id19"/> <label kind="synchronisation" x="-52" y="281">StepEnd?</label> <nail x="-32" y="344"/> </transition> <transition> <source ref="id12"/> <target ref="id20"/> <label kind="synchronisation" x="212" y="281">StepEnd?</label> <nail x="240" y="344"/> </transition> <transition> <source ref="id14"/> <target ref="id21"/> <label kind="synchronisation" x="500" y="285">StepEnd?</label> <nail x="528" y="336"/> </transition> <transition> <source ref="id8"/> <target ref="id18"/> <label kind="synchronisation" x="-348" y="281">StepEnd?</label> <nail x="-320" y="344"/> </transition> <transition> <source ref="id18"/> <target ref="id8"/> <label kind="guard" x="-288" y="320">busy[m0]</label> <nail x="-256" y="352"/> </transition> <transition> <source ref="id19"/> <target ref="id10"/> <label kind="guard" x="16" y="320">busy[m1]</label> <nail x="40" y="344"/> </transition> <transition> <source ref="id20"/> <target ref="id12"/> <label kind="guard" x="272" y="320">busy[m2]</label> <nail x="296" y="352"/> </transition> <transition> <source ref="id21"/> <target ref="id14"/> <label kind="guard" x="544" y="320">busy[m3]</label> <nail x="584" y="336"/> </transition> </template> <template> <name x="5" y="5">Machine</name> <parameter>const int no, const int startupCost</parameter> <declaration>//Insert local declarations of clocks, variables and constants.
clock cIdle;</declaration> <location id="id22" x="560" y="184"> <committed/> </location> <location id="id23" x="72" y="96"> <name x="40" y="64">ColdStart</name> </location> <location id="id24" x="416" y="-40"> <committed/> </location> <location id="id25" x="256" y="-40"> <name x="246" y="-70">Off</name> <committed/> </location> <location id="id26" x="256" y="96"> <name x="246" y="66">Busy</name> </location> <location id="id27" x="560" y="-40"> <name x="550" y="-70">Idle</name> </location> <location id="id28" x="416" y="184"> <committed/> </location> <init ref="id23"/> <transition> <source ref="id22"/> <target ref="id27"/> <label kind="synchronisation" x="512" y="56">iTimerOn[no]!</label> </transition> <transition> <source ref="id28"/> <target ref="id22"/> <label kind="synchronisation" x="456" y="160">StepEnd!</label> </transition> <transition> <source ref="id23"/> <target ref="id26"/> <label kind="synchronisation" x="136" y="72">machOn[no]?</label> <label kind="assignment" x="136" y="96">busy[no]=true</label> </transition> <transition> <source ref="id25"/> <target ref="id26"/> </transition> <transition> <source ref="id24"/> <target ref="id26"/> <label kind="guard" x="288" y="72">cIdle<=startupCost</label> <label kind="synchronisation"

x="304" y="96">iTimerOff[no]!</label> <nail x="416" y="96"/> </transition> <transition> <source ref="id24"/> <target ref="id25"/> <label kind="guard" x="280" y="-64">cIdle>startupCost</label> <label kind="synchronisation" x="296" y="-40">iTimerOff[no]!</label> </transition> <transition> <source ref="id27"/> <target ref="id24"/> <label kind="synchronisation" x="448" y="-64">machOn[no]?</label> <label kind="assignment" x="448" y="-40">busy[no]=true</label> </transition> <transition> <source ref="id26"/> <target ref="id28"/> <label kind="synchronisation" x="288" y="160">machOff[no]?</label> <label kind="assignment" x="280" y="184">busy[no]=false</label> <nail x="256" y="184"/> </transition> </template> <template> <name>CostTimer</name> <location id="id29" x="-280" y="-136"> <name x="-290" y="-166">idle</name> <label kind="invariant" x="-290" y="-121">cost'==15</label> </location> <init ref="id29"/> </template> <template> <name>Timer</name> <declaration>clock c;</declaration> <location id="id30" x="-328" y="-112"> <label kind="invariant" x="-336" y="-144">c<=2</label> </location> <location id="id31" x="-480" y="-40"> <label kind="invariant" x="-490" y="-25">c<=1</label> </location> <init ref="id31"/> <transition> <source ref="id30"/> <target ref="id31"/> <label kind="guard" x="-376" y="-64">c>1</label> <label kind="assignment" x="-408" y="-40">c=0 & glt++</label> <nail x="-328" y="-40"/> </transition> <transition> <source ref="id31"/> <target ref="id30"/> <label kind="guard" x="-408" y="-136">c>0</label> <label kind="assignment" x="-456" y="-136">glt++</label> <nail x="-480" y="-112"/> </transition> </template> <template> <name>IdleTimer</name> <parameter>const int[0,5] no</parameter> <declaration>clock c;</declaration> <location id="id32" x="-704" y="96"> <label kind="invariant" x="-714" y="111">cost'==2</label> </location> <location id="id33" x="-880" y="96"> </location> <init ref="id33"/> <transition> <source ref="id32"/> <target ref="id33"/> <label kind="synchronisation" x="-848" y="-8">iTimerOff[no]?</label> <nail x="-704" y="16"/> <nail x="-880" y="16"/> </transition> <transition> <source ref="id33"/> <target ref="id32"/> <label kind="synchronisation" x="-840" y="72">iTimerOn[no]?</label> </transition> </template> <system>//Insert process assignments.
machine0 = Machine(0,20);
machine1 = Machine(1,30);
machine2 = Machine(2,15);
machine3 = Machine(3,22);
machine4 = Machine(4,35);
machine5 = Machine(5,9);
job0=Job(1,4, 3,5, 0,8, 2,6, 4,4, 5,8, 70,1,20);
job1=Job(3,6, 1,2, 5,9, 0,11, 2,7, 4,6, 20,1,25);
job2=Job(1,4, 4,7, 3,4, 2,7, 0,2, 5,5, 36,1,21);
job3=Job(5,8, 2,5, 3,6, 1,5, 4,9, 0,4, 11,2,15);
//Edit system definition.
system machine0, machine1, machine2, machine3, machine4, machine5, job0, job1, job2, job3, CostTimer, IdleTimer;
</system> </nta>

ΠΕΙΡΑΜΑ 4.2.1 – Μεταβάσεις

job1.s0->job1.t0 { glc == 0, machOn[m0]!, c := 0 } machine3.ColdStart->machine3.Busy { 1, machOn[no]?, busy[no] := 1 }
job0.s0->job0.t0 { glc == 0, machOn[m0]!, c := 0 } machine1.ColdStart->machine1.Busy { 1, machOn[no]?, busy[no] := 1 }

job0.t0->job0.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job0.s1->job0.c1 { 1, StepEnd?, 1 }
job2.s0->job2.c0 { 1, StepEnd?, 1 } job3.s0->job3.c0 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job3.c0->job3.t0 { 1, machOn[m0]!, c := 0 } machine5.ColdStart->machine5.Busy { 1, machOn[no]?, busy[no] := 1 }
job2.c0->job2.t0 { 1, machOn[m0]!, c := 0 } machine1.Idle->machine1._id24 { 1, machOn[no]?, busy[no] := 1 }
machine1._id24->machine1.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(1)._id32->IdleTimer(1)._id33 { 1, iTimerOff[no]?, 1 }
job0.c1->job0.s1 { busy[m1], tau, 1 }

job1.t0->job1.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job0.s1->job0.c1 { 1, StepEnd?, 1 }
job1.s1->job1.c1 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job1.c1->job1.s1 { busy[m1], tau, 1 }
job0.c1->job0.t1 { 1, machOn[m1]!, c := 0 } machine3.Idle->machine3._id24 { 1, machOn[no]?, busy[no] := 1 }
machine3._id24->machine3.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(3)._id32->IdleTimer(3)._id33 { 1, iTimerOff[no]?, 1 }

job2.t0->job2.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job1.s1->job1.c1 { 1, StepEnd?, 1 }
job2.s1->job2.c1 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job2.c1->job2.t1 { 1, machOn[m1]!, c := 0 } machine4.ColdStart->machine4.Busy { 1, machOn[no]?, busy[no] := 1 }
job1.c1->job1.t1 { 1, machOn[m1]!, c := 0 } machine1.Idle->machine1._id24 { 1, machOn[no]?, busy[no] := 1 }
machine1._id24->machine1.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(1)._id32->IdleTimer(1)._id33 { 1, iTimerOff[no]?, 1 }

job1.t1->job1.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job1.s2->job1.c2 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job1.c2->job1.s2 { busy[m2], tau, 1 }

job0.t1->job0.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job0.s2->job0.c2 { 1, StepEnd?, 1 }
job1.s2->job1.c2 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job1.c2->job1.s2 { busy[m2], tau, 1 }
job0.c2->job0.t2 { 1, machOn[m2]!, c := 0 } machine0.ColdStart->machine0.Busy { 1, machOn[no]?, busy[no] := 1 }

job3.t0->job3.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }
machine5._id28->machine5._id22 { 1, StepEnd!, 1 } job1.s2->job1.c2 { 1, StepEnd?, 1 }
job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }
job3.c1->job3.t1 { 1, machOn[m1]!, c := 0 } machine2.ColdStart->machine2.Busy { 1, machOn[no]?, busy[no] := 1 }
job1.c2->job1.t2 { 1, machOn[m2]!, c := 0 } machine5.Idle->machine5._id24 { 1, machOn[no]?, busy[no] := 1 }
machine5._id24->machine5.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(5)._id32->IdleTimer(5)._id33 { 1, iTimerOff[no]?, 1 }
machine5.Off->machine5.Busy { 1, tau, 1 }

job2.t1->job2.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 } job2.s2->job2.c2 { 1, StepEnd?, 1 }
machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32 { 1, iTimerOn[no]?, 1 }
job2.c2->job2.t2 { 1, machOn[m2]!, c := 0 } machine3.Idle->machine3._id24 { 1, machOn[no]?, busy[no] := 1 }
machine3._id24->machine3.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(3)._id32->IdleTimer(3)._id33 { 1, iTimerOff[no]?, 1 }

job3.t1->job3.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job3.s2->job3.c2 { 1, StepEnd?, 1 }
machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32

{ 1, iTimerOn[no]?, 1 }
job3.c2->job3.s2 { busy[m2], tau, 1 }

job0.t2->job0.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 } job0.s3->job0.c3 { 1, StepEnd?, 1 }
job3.s2->job3.c2 { 1, StepEnd?, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job3.c2->job3.s2 { busy[m2], tau, 1 }
job2.t2->job2.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job2.s3->job2.c3 { 1, StepEnd?, 1 }
job3.s2->job3.c2 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job0.c3->job0.t3 { 1, machOn[m3]!, c := 0 } machine2.Idle->machine2._id24 { 1, machOn[no]?, busy[no] := 1 }
machine2._id24->machine2.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(2)._id32->IdleTimer(2)._id33 { 1, iTimerOff[no]?, 1 }
machine2.Off->machine2.Busy { 1, tau, 1 }
job3.c2->job3.t2 { 1, machOn[m2]!, c := 0 } machine3.Idle->machine3._id24 { 1, machOn[no]?, busy[no] := 1 }
machine3._id24->machine3.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(3)._id32->IdleTimer(3)._id33 { 1, iTimerOff[no]?, 1 }
job2.c3->job2.s3 { busy[m3], tau, 1 }

job1.t2->job1.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }
machine5._id28->machine5._id22 { 1, StepEnd!, 1 } job1.s3->job1.c3 { 1, StepEnd?, 1 }
job2.s3->job2.c3 { 1, StepEnd?, 1 }
machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }
job1.c3->job1.t3 { 1, machOn[m3]!, c := 0 } machine0.Idle->machine0._id24 { 1, machOn[no]?, busy[no] := 1 }
machine0._id24->machine0.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(0)._id32->IdleTimer(0)._id33 { 1, iTimerOff[no]?, 1 }
machine0.Off->machine0.Busy { 1, tau, 1 }
job2.c3->job2.s3 { busy[m3], tau, 1 }

job0.t3->job0.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job0.s4->job0.c4 { 1, StepEnd?, 1 }
job2.s3->job2.c3 { 1, StepEnd?, 1 }
machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32

{ 1, iTimerOn[no]?, 1 }
job0.c4->job0.t4 { 1, machOn[m4]!, c := 0 } machine4.Idle->machine4._id24 { 1, machOn[no]?, busy[no] := 1 }
machine4._id24->machine4.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(4)._id32->IdleTimer(4)._id33 { 1, iTimerOff[no]?, 1 }
job3.t2->job3.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job3.s3->job3.c3 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job3.c3->job3.t3 { 1, machOn[m3]!, c := 0 } machine1.Idle->machine1._id24 { 1, machOn[no]?, busy[no] := 1 }
machine1._id24->machine1.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(1)._id32->IdleTimer(1)._id33 { 1, iTimerOff[no]?, 1 }
job2.c3->job2.t3 { 1, machOn[m3]!, c := 0 } machine2.Idle->machine2._id24 { 1, machOn[no]?, busy[no] := 1 }
machine2._id24->machine2.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(2)._id32->IdleTimer(2)._id33 { 1, iTimerOff[no]?, 1 }
machine2.Off->machine2.Busy { 1, tau, 1 }

job0.t4->job0.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 } job0.s5->job0.c5 { 1, StepEnd?, 1 }
machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32 { 1, iTimerOn[no]?, 1 }

job3.t3->job3.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job3.s4->job3.c4 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job3.c4->job3.t4 { 1, machOn[m4]!, c := 0 } machine4.Idle->machine4._id24 { 1, machOn[no]?, busy[no] := 1 }
machine4._id24->machine4.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(4)._id32->IdleTimer(4)._id33 { 1, iTimerOff[no]?, 1 }

job0.c5->job0.t5 { 1, machOn[m5]!, c := 0 } machine5.Idle->machine5._id24 { 1, machOn[no]?, busy[no] := 1 }
machine5._id24->machine5.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(5)._id32->IdleTimer(5)._id33 { 1, iTimerOff[no]?, 1 }
machine5.Off->machine5.Busy { 1, tau, 1 }

job2.t3->job2.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job2.s4->job2.c4 { 1, StepEnd?, 1 }

machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32 { 1, iTimerOn[no]?, 1 }
job1.t3->job1.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 } job1.s4->job1.c4 { 1, StepEnd?, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job2.c4->job2.t4 { 1, machOn[m4]!, c := 0 } machine0.Idle->machine0._id24 { 1, machOn[no]?, busy[no] := 1 }
machine0._id24->machine0.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(0)._id32->IdleTimer(0)._id33 { 1, iTimerOff[no]?, 1 }
machine0.Off->machine0.Busy { 1, tau, 1 }
job1.c4->job1.t4 { 1, machOn[m4]!, c := 0 } machine2.Idle->machine2._id24 { 1, machOn[no]?, busy[no] := 1 }
machine2._id24->machine2.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(2)._id32->IdleTimer(2)._id33 { 1, iTimerOff[no]?, 1 }
machine2.Off->machine2.Busy { 1, tau, 1 }

job2.t4->job2.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 } job2.s5->job2.c5 { 1, StepEnd?, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job2.c5->job2.s5 { busy[m5], tau, 1 }

job0.t5->job0._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }
machine5._id28->machine5._id22 { 1, StepEnd!, 1 } job2.s5->job2.c5 { 1, StepEnd?, 1 }
machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }
job1.t4->job1.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job1.s5->job1.c5 { 1, StepEnd?, 1 }
machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32 { 1, iTimerOn[no]?, 1 }
job0._id16->job0._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), extra := targetDeviation }
job1.c5->job1.s5 { busy[m5], tau, 1 }
job0._id1->job0.end { 1, tau, 1 }
job3.t4->job3.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 } job1.s5->job1.c5 { 1, StepEnd?, 1 }
job3.s5->job3.c5 { 1, StepEnd?, 1 }
machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32 { 1, iTimerOn[no]?, 1 }

job1.c5->job1.t5 { 1, machOn[m5]!, c := 0 } machine4.Idle->machine4._id24 { 1, machOn[no]?, busy[no] := 1 }
machine4._id24->machine4.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(4)._id32->IdleTimer(4)._id33 { 1, iTimerOff[no]?, 1 }
machine4.Off->machine4.Busy { 1, tau, 1 }

job2.c5->job2.t5 { 1, machOn[m5]!, c := 0 } machine5.Idle->machine5._id24 { 1, machOn[no]?, busy[no] := 1 }
machine5._id24->machine5.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(5)._id32->IdleTimer(5)._id33 { 1, iTimerOff[no]?, 1 }
machine5.Off->machine5.Busy { 1, tau, 1 }

job3.c5->job3.t5 { 1, machOn[m5]!, c := 0 } machine0.Idle->machine0._id24 { 1, machOn[no]?, busy[no] := 1 }
machine0._id24->machine0.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(0)._id32->IdleTimer(0)._id33 { 1, iTimerOff[no]?, 1 }
machine0.Off->machine0.Busy { 1, tau, 1 }

job3.t5->job3._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job3._id16->job3._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), extra := targetDeviation }
job2.t5->job2._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }

machine5._id28->machine5._id22 { 1, StepEnd!, 1 }

machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }
job2._id16->job2._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), extra := targetDeviation }
job1.t5->job1._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 }
machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32 { 1, iTimerOn[no]?, 1 }
job3._id1->job3.end { 1, tau, 1 }

job1._id16->job1._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), extra := targetDeviation }
job2._id1->job2.end { 1, tau, 1 }
job1._id1->job1.end { 1, tau, 1 }

ΠΕΙΡΑΜΑ 4.2.2 – Αρχείο xml

<?xml version="1.0" encoding="utf-8"?> <!DOCTYPE nta PUBLIC '-//Uppaal Team//DTD Flat System 1.1//EN' 'http://www.it.uu.se/research/group/darts/uppaal/flat-1_1.dtd'> <nta> <declaration>//Insert declarations of global clocks, variables, constants and channels.
chan machOn[6];
chan machOff[6];
chan iTimerOn[6];
chan iTimerOff[6];
clock glc;
bool busy[6]={false, false, false, false, false, false};
urgent broadcast chan StepEnd;
chan trigger;
int best=1000;
meta int[-1000,1000] remaining;
int glt=0;
int extra=0;</declaration> <template> <name x="5" y="5">Job</name> <parameter>const int m0, const int d0, const int m1, const int d1, const int m2, const int d2, const int m3, const int d3, const int m4, const int d4, const int m5, const int d5, const int Target, const int Early, const int Late</parameter> <declaration>clock c;
clock k;
int targetDeviation=0;</declaration> <location id="id0" x="-128" y="504"> <name x="-138" y="474">end</name> </location> <location id="id1" x="-40" y="584"> <urgent/> </location> <location id="id2" x="304" y="440"> <name x="294" y="410">t5</name> <label kind="invariant" x="294" y="455">c<=d5</label> </location> <location id="id3" x="400" y="600"> <name x="390" y="570">c5</name> </location> <location id="id4" x="400" y="440"> <name x="390" y="410">s5</name> </location> <location id="id5" x="552" y="440"> <name x="542" y="410">t4</name> <label kind="invariant" x="542" y="455">c<=d4</label> </location> <location id="id6" x="680" y="600"> <name x="670" y="570">c4</name> <urgent/> </location> <location id="id7" x="680" y="440"> <name x="670" y="410">s4</name> </location> <location id="id8" x="-288" y="208"> <name x="-298" y="178">s0</name> </location> <location id="id9" x="-136" y="208"> <name x="-146" y="178">t0</name> <label kind="invariant" x="-168" y="224">c<=d0</label> </location> <location id="id10" x="8" y="208"> <name x="-2" y="178">s1</name> </location> <location id="id11" x="128" y="208"> <name x="118" y="178">t1</name> <label kind="invariant" x="104" y="224">c<=d1</label> </location> <location id="id12" x="272" y="208"> <name x="262" y="178">s2</name> </location> <location id="id13" x="416" y="208"> <name x="406" y="178">t2</name> <label kind="invariant" x="384" y="224">c<=d2</label> </location> <location id="id14" x="560" y="208"> <name x="550" y="178">s3</name> </location> <location id="id15" x="680" y="216"> <name x="670" y="186">t3</name> <label kind="invariant" x="670" y="231">c<=d3</label> </location> <location id="id16" x="160" y="440"> <urgent/> </location> <location id="id17" x="272" y="-16"> <name x="262" y="-46">stop</name> </location> <location id="id18" x="-288" y="368"> <name x="-298" y="338">c0</name> <urgent/> </location> <location id="id19" x="8" y="368"> <name x="-2" y="338">c1</name> <urgent/> </location> <location id="id20" x="272" y="368"> <name

x="262" y="338">c2</name> <urgent/> </location> <location id="id21" x="560" y="368"> <name x="550" y="338">c3</name> <urgent/> </location> <init ref="id8"/> <transition> <source ref="id1"/> <target ref="id0"/> <nail x="-128" y="584"/> </transition> <transition> <source ref="id16"/> <target ref="id1"/> <label kind="guard" x="8" y="560">glt>Target</label> <label kind="assignment" x="-128" y="592">targetDeviation=Late*(glt-Target), cost+=targetDeviation</label> <nail x="160" y="584"/> </transition> <transition> <source ref="id16"/> <target ref="id1"/> <label kind="guard" x="-24" y="448">glt<=Target</label> <label kind="assignment" x="-144" y="408">targetDeviation=Early*(Target-glt), cost+=targetDeviation</label> <nail x="-40" y="440"/> </transition> <transition> <source ref="id2"/> <target ref="id17"/> <label kind="guard" x="228" y="182">(glc>best)</label> <nail x="304" y="256"/> </transition> <transition> <source ref="id5"/> <target ref="id17"/> <label kind="guard" x="352" y="182">(glc>best)</label> <nail x="552" y="392"/> <nail x="336" y="392"/> </transition> <transition> <source ref="id3"/> <target ref="id4"/> <label kind="guard" x="416" y="504">busy[m5]</label> <nail x="440" y="504"/> </transition> <transition> <source ref="id3"/> <target ref="id2"/> <label kind="synchronisation" x="272" y="512">machOn[m5]!</label> <label kind="assignment" x="280" y="528">c=0</label> <nail x="304" y="600"/> </transition> <transition> <source ref="id4"/> <target ref="id3"/> <label kind="synchronisation" x="352" y="488">StepEnd?</label> <nail x="376" y="496"/> </transition> <transition> <source ref="id5"/> <target ref="id4"/> <label kind="guard" x="448" y="408">(c>=d4) and (glc<=best)</label> <label kind="synchronisation" x="448" y="440">machOff[m4]!</label> </transition> <transition> <source ref="id6"/> <target ref="id5"/> <label kind="synchronisation" x="528" y="520">machOn[m4]!</label> <label kind="assignment" x="528" y="536">c=0</label> <nail x="552" y="600"/> </transition> <transition> <source ref="id6"/> <target ref="id7"/> <label kind="guard" x="712" y="480">busy[m4]</label> <nail x="712" y="488"/> </transition> <transition> <source ref="id7"/> <target ref="id6"/> <label kind="synchronisation" x="620" y="465">StepEnd?</label> <nail x="656" y="480"/> </transition> <transition> <source ref="id15"/> <target ref="id7"/> <label kind="guard" x="752" y="288">(c>=d3) and (glc<=best)</label> <label kind="synchronisation" x="744" y="328">machOff[m3]!</label> <nail x="744" y="216"/> <nail x="744" y="440"/> </transition> <transition> <source ref="id2"/> <target ref="id16"/> <label kind="guard" x="200" y="448">(c>=d5) and (glc<=best)</label> <label kind="synchronisation" x="192" y="416">machOff[m5]!</label> </transition> <transition> <source ref="id8"/> <target ref="id9"/> <label kind="guard" x="-232" y="184">glc==0</label> <label kind="synchronisation" x="-248" y="168">machOn[m0]!</label> <label kind="assignment" x="-224" y="128">c=0</label> <nail x="-256" y="160"/> <nail x="-176" y="160"/> </transition> <transition> <source ref="id9"/> <target ref="id10"/> <label kind="guard" x="-88" y="168">(c>=d0) and (glc<=best)</label> <label kind="synchronisation" x="-104" y="224">machOff[m0]!</label> </transition> <transition> <source ref="id11"/> <target ref="id12"/> <label kind="guard" x="176" y="176">(c>=d1) and (glc<=best)</label> <label kind="synchronisation" x="160" y="216">machOff[m1]!</label> </transition> <transition> <source ref="id13"/> <target ref="id14"/> <label kind="guard" x="456" y="176">(c>=d2) and (glc<=best)</label> <label kind="synchronisation" x="456" y="216">machOff[m2]!</label>

</transition> <transition> <source ref="id9"/> <target ref="id17"/> <label kind="guard" x="-176" y="8">(glc>best)</label> <nail x="-136" y="-16"/> </transition> <transition> <source ref="id11"/> <target ref="id17"/> <label kind="guard" x="140" y="66">(glc>best)</label> </transition> <transition> <source ref="id13"/> <target ref="id17"/> <label kind="guard" x="284" y="66">(glc>best)</label> </transition> <transition> <source ref="id15"/> <target ref="id17"/> <label kind="guard" x="640" y="48">(glc>best)</label> <nail x="680" y="-16"/> </transition> <transition> <source ref="id18"/> <target ref="id9"/> <label kind="synchronisation" x="-184" y="320">machOn[m0]!</label> <label kind="assignment" x="-152" y="336">c=0</label> <nail x="-136" y="368"/> </transition> <transition> <source ref="id19"/> <target ref="id11"/> <label kind="synchronisation" x="96" y="312">machOn[m1]!</label> <label kind="assignment" x="120" y="336">c=0</label> <nail x="128" y="368"/> </transition> <transition> <source ref="id21"/> <target ref="id15"/> <label kind="synchronisation" x="632" y="320">machOn[m3]!</label> <label kind="assignment" x="664" y="344">c=0</label> <nail x="680" y="368"/> </transition> <transition> <source ref="id20"/> <target ref="id13"/> <label kind="synchronisation" x="376" y="312">machOn[m2]!</label> <label kind="assignment" x="400" y="336">c=0</label> <nail x="416" y="368"/> </transition> <transition> <source ref="id10"/> <target ref="id19"/> <label kind="synchronisation" x="-52" y="281">StepEnd?</label> <nail x="-32" y="344"/> </transition> <transition> <source ref="id12"/> <target ref="id20"/> <label kind="synchronisation" x="212" y="281">StepEnd?</label> <nail x="240" y="344"/> </transition> <transition> <source ref="id14"/> <target ref="id21"/> <label kind="synchronisation" x="500" y="285">StepEnd?</label> <nail x="528" y="336"/> </transition> <transition> <source ref="id8"/> <target ref="id18"/> <label kind="synchronisation" x="-348" y="281">StepEnd?</label> <nail x="-320" y="344"/> </transition> <transition> <source ref="id18"/> <target ref="id8"/> <label kind="guard" x="-288" y="320">busy[m0]</label> <nail x="-256" y="352"/> </transition> <transition> <source ref="id19"/> <target ref="id10"/> <label kind="guard" x="16" y="320">busy[m1]</label> <nail x="40" y="344"/> </transition> <transition> <source ref="id20"/> <target ref="id12"/> <label kind="guard" x="272" y="320">busy[m2]</label> <nail x="296" y="352"/> </transition> <transition> <source ref="id21"/> <target ref="id14"/> <label kind="guard" x="544" y="320">busy[m3]</label> <nail x="584" y="336"/> </transition> </template> <template> <name x="5" y="5">Machine</name> <parameter>const int no, const int startupCost</parameter> <declaration>//Insert local declarations of clocks, variables and constants.
clock cIdle;</declaration> <location id="id22" x="560" y="184"> <committed/> </location> <location id="id23" x="72" y="96"> <name x="40" y="64">ColdStart</name> </location> <location id="id24" x="416" y="-40"> <committed/> </location> <location id="id25" x="256" y="-40"> <name x="246" y="-70">Off</name> <committed/> </location> <location id="id26" x="256" y="96"> <name x="246" y="66">Busy</name> </location> <location id="id27" x="560" y="-40"> <name x="550" y="-70">Idle</name> </location> <location id="id28" x="416" y="184"> <committed/> </location> <init ref="id23"/> <transition> <source ref="id22"/> <target ref="id27"/> <label kind="synchronisation" x="512" y="56">iTimerOn[no]!</label> </transition> <transition> <source ref="id28"/> <target ref="id22"/> <label kind="synchronisation" x="456" y="160">StepEnd!</label> </transition> <transition> <source ref="id23"/> <target ref="id26"/> <label kind="synchronisation" x="136" y="72">machOn[no]?</label> <label kind="assignment" x="136" y="96">busy[no]=true</label> </transition> <transition> <source ref="id25"/> <target

ref="id26"/> </transition> <transition> <source ref="id24"/> <target ref="id26"/> <label kind="guard" x="288" y="72">cIdle<=startupCost</label> <label kind="synchronisation" x="304" y="96">iTimerOff[no]!</label> <nail x="416" y="96"/> </transition> <transition> <source ref="id24"/> <target ref="id25"/> <label kind="guard" x="280" y="-64">cIdle>startupCost</label> <label kind="synchronisation" x="296" y="-40">iTimerOff[no]!</label> </transition> <transition> <source ref="id27"/> <target ref="id24"/> <label kind="synchronisation" x="448" y="-64">machOn[no]?</label> <label kind="assignment" x="448" y="-40">busy[no]=true</label> </transition> <transition> <source ref="id26"/> <target ref="id28"/> <label kind="synchronisation" x="288" y="160">machOff[no]?</label> <label kind="assignment" x="280" y="184">busy[no]=false</label> <nail x="256" y="184"/> </transition> </template> <template> <name>CostTimer</name> <location id="id29" x="-280" y="-136"> <name x="-290" y="-166">idle</name> <label kind="invariant" x="-290" y="-121">cost'==15</label> </location> <init ref="id29"/> </template> <template> <name>Timer</name> <declaration>clock c;</declaration> <location id="id30" x="-328" y="-112"> <label kind="invariant" x="-336" y="-144">c<=2</label> </location> <location id="id31" x="-480" y="-40"> <label kind="invariant" x="-490" y="-25">c<=1</label> </location> <init ref="id31"/> <transition> <source ref="id30"/> <target ref="id31"/> <label kind="guard" x="-376" y="-64">c>1</label> <label kind="assignment" x="-408" y="-40">c=0 & glt++</label> <nail x="-328" y="-40"/> </transition> <transition> <source ref="id31"/> <target ref="id30"/> <label kind="guard" x="-408" y="-136">c>0</label> <label kind="assignment" x="-456" y="-136">glt++</label> <nail x="-480" y="-112"/> </transition> </template> <template> <name>IdleTimer</name> <parameter>const int[0,5] no</parameter> <declaration>clock c;</declaration> <location id="id32" x="-704" y="96"> <label kind="invariant" x="-714" y="111">cost'==1</label> </location> <location id="id33" x="-880" y="96"> </location> <init ref="id33"/> <transition> <source ref="id32"/> <target ref="id33"/> <label kind="synchronisation" x="-848" y="-8">iTimerOff[no]?</label> <nail x="-704" y="16"/> <nail x="-880" y="16"/> </transition> <transition> <source ref="id33"/> <target ref="id32"/> <label kind="synchronisation" x="-840" y="72">iTimerOn[no]?</label> </transition> </template> <system>//Insert process assignments.
machine0 = Machine(0,20);
machine1 = Machine(1,30);
machine2 = Machine(2,15);
machine3 = Machine(3,22);
machine4 = Machine(4,35);
machine5 = Machine(5,9);
job0=Job(1,4, 3,5, 0,8, 2,6, 4,4, 5,8, 70,1,20);
job1=Job(3,6, 1,2, 5,9, 0,11, 2,7, 4,6, 20,1,25);
job2=Job(1,4, 4,7, 3,4, 2,7, 0,2, 5,5, 36,1,21);
job3=Job(5,8, 2,5, 3,6, 1,5, 4,9, 0,4, 11,2,15);
//Edit system definition.
system machine0, machine1, machine2, machine3, machine4, machine5, job0, job1, job2, job3, CostTimer, IdleTimer;
</system> </nta>

ΠΕΙΡΑΜΑ 4.2.2 – Μεταβάσεις

job1.s0->job1.t0 { glc == 0, machOn[m0]!, c := 0 } machine3.ColdStart->machine3.Busy { 1, machOn[no]?, busy[no] := 1 }
job0.s0->job0.t0 { glc == 0, machOn[m0]!, c := 0 } machine1.ColdStart->machine1.Busy { 1, machOn[no]?, busy[no] := 1 }

job0.t0->job0.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job0.s1->job0.c1 { 1, StepEnd?, 1 }
job2.s0->job2.c0 { 1, StepEnd?, 1 } job3.s0->job3.c0 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job3.c0->job3.t0 { 1, machOn[m0]!, c := 0 } machine5.ColdStart->machine5.Busy { 1, machOn[no]?, busy[no] := 1 }
job2.c0->job2.t0 { 1, machOn[m0]!, c := 0 } machine1.Idle->machine1._id24 { 1, machOn[no]?, busy[no] := 1 }
machine1._id24->machine1.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(1)._id32->IdleTimer(1)._id33 { 1, iTimerOff[no]?, 1 }
job0.c1->job0.s1 { busy[m1], tau, 1 }

job1.t0->job1.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job0.s1->job0.c1 { 1, StepEnd?, 1 }
job1.s1->job1.c1 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job0.c1->job0.t1 { 1, machOn[m1]!, c := 0 } machine3.Idle->machine3._id24 { 1, machOn[no]?, busy[no] := 1 }
machine3._id24->machine3.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(3)._id32->IdleTimer(3)._id33 { 1, iTimerOff[no]?, 1 }
job1.c1->job1.s1 { busy[m1], tau, 1 }

job2.t0->job2.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job1.s1->job1.c1 { 1, StepEnd?, 1 }
job2.s1->job2.c1 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job2.c1->job2.t1 { 1, machOn[m1]!, c := 0 } machine4.ColdStart->machine4.Busy { 1, machOn[no]?, busy[no] := 1 }
job1.c1->job1.t1 { 1, machOn[m1]!, c := 0 } machine1.Idle->machine1._id24 { 1, machOn[no]?, busy[no] := 1 }
machine1._id24->machine1.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(1)._id32->IdleTimer(1)._id33 { 1, iTimerOff[no]?, 1 }

job1.t1->job1.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job1.s2->job1.c2 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job1.c2->job1.s2 { busy[m2], tau, 1 }

job0.t1->job0.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job0.s2->job0.c2 { 1, StepEnd?, 1 }
job1.s2->job1.c2 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job1.c2->job1.s2 { busy[m2], tau, 1 }
job0.c2->job0.t2 { 1, machOn[m2]!, c := 0 } machine0.ColdStart->machine0.Busy { 1, machOn[no]?, busy[no] := 1 }

job3.t0->job3.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }
machine5._id28->machine5._id22 { 1, StepEnd!, 1 } job1.s2->job1.c2 { 1, StepEnd?, 1 }
job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }
job3.c1->job3.t1 { 1, machOn[m1]!, c := 0 } machine2.ColdStart->machine2.Busy { 1, machOn[no]?, busy[no] := 1 }
job1.c2->job1.t2 { 1, machOn[m2]!, c := 0 } machine5.Idle->machine5._id24 { 1, machOn[no]?, busy[no] := 1 }
machine5._id24->machine5.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(5)._id32->IdleTimer(5)._id33 { 1, iTimerOff[no]?, 1 }
machine5.Off->machine5.Busy { 1, tau, 1 }

job2.t1->job2.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 } job2.s2->job2.c2 { 1, StepEnd?, 1 }
machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32 { 1, iTimerOn[no]?, 1 }
job2.c2->job2.t2 { 1, machOn[m2]!, c := 0 } machine3.Idle->machine3._id24 { 1, machOn[no]?, busy[no] := 1 }
machine3._id24->machine3.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(3)._id32->IdleTimer(3)._id33 { 1, iTimerOff[no]?, 1 }

job3.t1->job3.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job3.s2->job3.c2 { 1, StepEnd?, 1 }
machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32

{ 1, iTimerOn[no]?, 1 }
job3.c2->job3.s2 { busy[m2], tau, 1 }

job0.t2->job0.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 } job0.s3->job0.c3 { 1, StepEnd?, 1 }
job3.s2->job3.c2 { 1, StepEnd?, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job3.c2->job3.s2 { busy[m2], tau, 1 }
job0.c3->job0.t3 { 1, machOn[m3]!, c := 0 } machine2.Idle->machine2._id24 { 1, machOn[no]?, busy[no] := 1 }
machine2._id24->machine2.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(2)._id32->IdleTimer(2)._id33 { 1, iTimerOff[no]?, 1 }
machine2.Off->machine2.Busy { 1, tau, 1 }
job2.t2->job2.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job2.s3->job2.c3 { 1, StepEnd?, 1 }
job3.s2->job3.c2 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job2.c3->job2.s3 { busy[m3], tau, 1 }
job3.c2->job3.t2 { 1, machOn[m2]!, c := 0 } machine3.Idle->machine3._id24 { 1, machOn[no]?, busy[no] := 1 }
machine3._id24->machine3.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(3)._id32->IdleTimer(3)._id33 { 1, iTimerOff[no]?, 1 }

job1.t2->job1.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }
machine5._id28->machine5._id22 { 1, StepEnd!, 1 } job1.s3->job1.c3 { 1, StepEnd?, 1 }
job2.s3->job2.c3 { 1, StepEnd?, 1 }
machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }
job1.c3->job1.t3 { 1, machOn[m3]!, c := 0 } machine0.Idle->machine0._id24 { 1, machOn[no]?, busy[no] := 1 }
machine0._id24->machine0.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(0)._id32->IdleTimer(0)._id33 { 1, iTimerOff[no]?, 1 }
machine0.Off->machine0.Busy { 1, tau, 1 }
job2.c3->job2.s3 { busy[m3], tau, 1 }

job0.t3->job0.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job0.s4->job0.c4 { 1, StepEnd?, 1 }
job2.s3->job2.c3 { 1, StepEnd?, 1 }
machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32

{ 1, iTimerOn[no]?, 1 }
job0.c4->job0.t4 { 1, machOn[m4]!, c := 0 } machine4.Idle->machine4._id24 { 1, machOn[no]?, busy[no] := 1 }
machine4._id24->machine4.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(4)._id32->IdleTimer(4)._id33 { 1, iTimerOff[no]?, 1 }
job2.c3->job2.t3 { 1, machOn[m3]!, c := 0 } machine2.Idle->machine2._id24 { 1, machOn[no]?, busy[no] := 1 }
machine2._id24->machine2.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(2)._id32->IdleTimer(2)._id33 { 1, iTimerOff[no]?, 1 }
machine2.Off->machine2.Busy { 1, tau, 1 }
job3.t2->job3.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job3.s3->job3.c3 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job3.c3->job3.t3 { 1, machOn[m3]!, c := 0 } machine1.Idle->machine1._id24 { 1, machOn[no]?, busy[no] := 1 }
machine1._id24->machine1.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(1)._id32->IdleTimer(1)._id33 { 1, iTimerOff[no]?, 1 }

job0.t4->job0.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 } job0.s5->job0.c5 { 1, StepEnd?, 1 }
machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32 { 1, iTimerOn[no]?, 1 }

job3.t3->job3.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job3.s4->job3.c4 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job3.c4->job3.t4 { 1, machOn[m4]!, c := 0 } machine4.Idle->machine4._id24 { 1, machOn[no]?, busy[no] := 1 }
machine4._id24->machine4.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(4)._id32->IdleTimer(4)._id33 { 1, iTimerOff[no]?, 1 }

job0.c5->job0.t5 { 1, machOn[m5]!, c := 0 } machine5.Idle->machine5._id24 { 1, machOn[no]?, busy[no] := 1 }
machine5._id24->machine5.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(5)._id32->IdleTimer(5)._id33 { 1, iTimerOff[no]?, 1 }
machine5.Off->machine5.Busy { 1, tau, 1 }

job2.t3->job2.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job2.s4->job2.c4 { 1, StepEnd?, 1 }

machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32 { 1, iTimerOn[no]?, 1 }
job1.t3->job1.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 } job1.s4->job1.c4 { 1, StepEnd?, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job2.c4->job2.t4 { 1, machOn[m4]!, c := 0 } machine0.Idle->machine0._id24 { 1, machOn[no]?, busy[no] := 1 }
machine0._id24->machine0.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(0)._id32->IdleTimer(0)._id33 { 1, iTimerOff[no]?, 1 }
machine0.Off->machine0.Busy { 1, tau, 1 }
job1.c4->job1.t4 { 1, machOn[m4]!, c := 0 } machine2.Idle->machine2._id24 { 1, machOn[no]?, busy[no] := 1 }
machine2._id24->machine2.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(2)._id32->IdleTimer(2)._id33 { 1, iTimerOff[no]?, 1 }
machine2.Off->machine2.Busy { 1, tau, 1 }

job2.t4->job2.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 } job2.s5->job2.c5 { 1, StepEnd?, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job2.c5->job2.s5 { busy[m5], tau, 1 }

job1.t4->job1.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job1.s5->job1.c5 { 1, StepEnd?, 1 }
job2.s5->job2.c5 { 1, StepEnd?, 1 }
machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32 { 1, iTimerOn[no]?, 1 }
job3.t4->job3.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 } job3.s5->job3.c5 { 1, StepEnd?, 1 }
machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32 { 1, iTimerOn[no]?, 1 }
job0.t5->job0._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }
machine5._id28->machine5._id22 { 1, StepEnd!, 1 }
machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }
job0._id16->job0._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), cost += targetDeviation }
job0._id1->job0.end { 1, tau, 1 }
job1.c5->job1.t5 { 1, machOn[m5]!, c := 0 } machine4.Idle->machine4._id24 { 1, machOn[no]?,

busy[no] := 1 }
machine4._id24->machine4.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(4)._id32->IdleTimer(4)._id33 { 1, iTimerOff[no]?, 1 }
machine4.Off->machine4.Busy { 1, tau, 1 }

job2.c5->job2.t5 { 1, machOn[m5]!, c := 0 } machine5.Idle->machine5._id24 { 1, machOn[no]?, busy[no] := 1 }
machine5._id24->machine5.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(5)._id32->IdleTimer(5)._id33 { 1, iTimerOff[no]?, 1 }
machine5.Off->machine5.Busy { 1, tau, 1 }

job3.c5->job3.t5 { 1, machOn[m5]!, c := 0 } machine0.Idle->machine0._id24 { 1, machOn[no]?, busy[no] := 1 }
machine0._id24->machine0.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(0)._id32->IdleTimer(0)._id33 { 1, iTimerOff[no]?, 1 }
machine0.Off->machine0.Busy { 1, tau, 1 }

job3.t5->job3._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job1.t5->job1._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 }
machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32 { 1, iTimerOn[no]?, 1 }

job1._id16->job1._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), cost += targetDeviation }
job1._id1->job1.end { 1, tau, 1 }
job2.t5->job2._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }
machine5._id28->machine5._id22 { 1, StepEnd!, 1 }
machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }
job3._id16->job3._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), cost += targetDeviation }

job3._id1->job3.end { 1, tau, 1 }
job2._id16->job2._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), cost += targetDeviation }
job2._id1->job2.end { 1, tau, 1 }

ΠΕΙΡΑΜΑ 4.3.1 – Αρχείο xml

<?xml version="1.0" encoding="utf-8"?> <!DOCTYPE nta PUBLIC '-//Uppaal Team//DTD Flat System 1.1//EN' 'http://www.it.uu.se/research/group/darts/uppaal/flat-1_1.dtd'> <nta> <declaration>//Insert declarations of global clocks, variables, constants and channels.
chan machOn[6];
chan machOff[6];
chan iTimerOn[6];
chan iTimerOff[6];
clock glc;
bool busy[6]={false, false, false, false, false, false};
urgent broadcast chan StepEnd;
chan trigger;
int best=1000;
meta int[-1000,1000] remaining;
int glt=0;
int extra=0;</declaration> <template> <name x="5" y="5">Job</name> <parameter>const int m0, const int d0, const int m1, const int d1, const int m2, const int d2, const int m3, const int d3, const int m4, const int d4, const int m5, const int d5, const int Target, const int Early, const int Late</parameter> <declaration>clock c;
clock k;
int targetDeviation=0;</declaration> <location id="id0" x="-128" y="504"> <name x="-138" y="474">end</name> </location> <location id="id1" x="-40" y="584"> <urgent/> </location> <location id="id2" x="304" y="440"> <name x="294" y="410">t5</name> <label kind="invariant" x="294" y="455">c<=d5</label> </location> <location id="id3" x="400" y="600"> <name x="390" y="570">c5</name> </location> <location id="id4" x="400" y="440"> <name x="390" y="410">s5</name> </location> <location id="id5" x="552" y="440"> <name x="542" y="410">t4</name> <label kind="invariant" x="542" y="455">c<=d4</label> </location> <location id="id6" x="680" y="600"> <name x="670" y="570">c4</name> <urgent/> </location> <location id="id7" x="680" y="440"> <name x="670" y="410">s4</name> </location> <location id="id8" x="-288" y="208"> <name x="-298" y="178">s0</name> </location> <location id="id9" x="-136" y="208"> <name x="-146" y="178">t0</name> <label kind="invariant" x="-168" y="224">c<=d0</label> </location> <location id="id10" x="8" y="208"> <name x="-2" y="178">s1</name> </location> <location id="id11" x="128" y="208"> <name x="118" y="178">t1</name> <label kind="invariant" x="104" y="224">c<=d1</label> </location> <location id="id12" x="272" y="208"> <name x="262" y="178">s2</name> </location> <location id="id13" x="416" y="208"> <name x="406" y="178">t2</name> <label kind="invariant" x="384" y="224">c<=d2</label> </location> <location id="id14" x="560" y="208"> <name x="550" y="178">s3</name> </location> <location id="id15" x="680" y="216"> <name x="670" y="186">t3</name> <label kind="invariant" x="670" y="231">c<=d3</label> </location> <location id="id16" x="160" y="440"> <urgent/> </location> <location id="id17" x="272" y="-16"> <name x="262" y="-46">stop</name> </location> <location id="id18" x="-288" y="368"> <name x="-298" y="338">c0</name> <urgent/> </location> <location id="id19" x="8" y="368"> <name x="-2" y="338">c1</name> <urgent/> </location> <location id="id20" x="272" y="368"> <name

x="262" y="338">c2</name> <urgent/> </location> <location id="id21" x="560" y="368"> <name x="550" y="338">c3</name> <urgent/> </location> <init ref="id8"/> <transition> <source ref="id1"/> <target ref="id0"/> <nail x="-128" y="584"/> </transition> <transition> <source ref="id16"/> <target ref="id1"/> <label kind="guard" x="8" y="560">glt>Target</label> <label kind="assignment" x="-128" y="592">targetDeviation=Late*(glt-Target), extra+=targetDeviation</label> <nail x="160" y="584"/> </transition> <transition> <source ref="id16"/> <target ref="id1"/> <label kind="guard" x="-24" y="448">glt<=Target</label> <label kind="assignment" x="-144" y="408">targetDeviation=Early*(Target-glt), extra+=targetDeviation</label> <nail x="-40" y="440"/> </transition> <transition> <source ref="id2"/> <target ref="id17"/> <label kind="guard" x="228" y="182">(glc>best)</label> <nail x="304" y="256"/> </transition> <transition> <source ref="id5"/> <target ref="id17"/> <label kind="guard" x="352" y="182">(glc>best)</label> <nail x="552" y="392"/> <nail x="336" y="392"/> </transition> <transition> <source ref="id3"/> <target ref="id4"/> <label kind="guard" x="416" y="504">busy[m5]</label> <nail x="440" y="504"/> </transition> <transition> <source ref="id3"/> <target ref="id2"/> <label kind="synchronisation" x="272" y="512">machOn[m5]!</label> <label kind="assignment" x="280" y="528">c=0</label> <nail x="304" y="600"/> </transition> <transition> <source ref="id4"/> <target ref="id3"/> <label kind="synchronisation" x="352" y="488">StepEnd?</label> <nail x="376" y="496"/> </transition> <transition> <source ref="id5"/> <target ref="id4"/> <label kind="guard" x="448" y="408">(c>=d4) and (glc<=best)</label> <label kind="synchronisation" x="448" y="440">machOff[m4]!</label> </transition> <transition> <source ref="id6"/> <target ref="id5"/> <label kind="synchronisation" x="528" y="520">machOn[m4]!</label> <label kind="assignment" x="528" y="536">c=0</label> <nail x="552" y="600"/> </transition> <transition> <source ref="id6"/> <target ref="id7"/> <label kind="guard" x="712" y="480">busy[m4]</label> <nail x="712" y="488"/> </transition> <transition> <source ref="id7"/> <target ref="id6"/> <label kind="synchronisation" x="620" y="465">StepEnd?</label> <nail x="656" y="480"/> </transition> <transition> <source ref="id15"/> <target ref="id7"/> <label kind="guard" x="752" y="288">(c>=d3) and (glc<=best)</label> <label kind="synchronisation" x="744" y="328">machOff[m3]!</label> <nail x="744" y="216"/> <nail x="744" y="440"/> </transition> <transition> <source ref="id2"/> <target ref="id16"/> <label kind="guard" x="200" y="448">(c>=d5) and (glc<=best)</label> <label kind="synchronisation" x="192" y="416">machOff[m5]!</label> </transition> <transition> <source ref="id8"/> <target ref="id9"/> <label kind="guard" x="-232" y="184">glc==0</label> <label kind="synchronisation" x="-248" y="168">machOn[m0]!</label> <label kind="assignment" x="-224" y="128">c=0</label> <nail x="-256" y="160"/> <nail x="-176" y="160"/> </transition> <transition> <source ref="id9"/> <target ref="id10"/> <label kind="guard" x="-88" y="168">(c>=d0) and (glc<=best)</label> <label kind="synchronisation" x="-104" y="224">machOff[m0]!</label> </transition> <transition> <source ref="id11"/> <target ref="id12"/> <label kind="guard" x="176" y="176">(c>=d1) and (glc<=best)</label> <label kind="synchronisation" x="160" y="216">machOff[m1]!</label> </transition> <transition> <source ref="id13"/> <target ref="id14"/> <label kind="guard" x="456" y="176">(c>=d2) and (glc<=best)</label> <label kind="synchronisation" x="456" y="216">machOff[m2]!</label>

</transition> <transition> <source ref="id9"/> <target ref="id17"/> <label kind="guard" x="-176" y="8">(glc>best)</label> <nail x="-136" y="-16"/> </transition> <transition> <source ref="id11"/> <target ref="id17"/> <label kind="guard" x="140" y="66">(glc>best)</label> </transition> <transition> <source ref="id13"/> <target ref="id17"/> <label kind="guard" x="284" y="66">(glc>best)</label> </transition> <transition> <source ref="id15"/> <target ref="id17"/> <label kind="guard" x="640" y="48">(glc>best)</label> <nail x="680" y="-16"/> </transition> <transition> <source ref="id18"/> <target ref="id9"/> <label kind="synchronisation" x="-184" y="320">machOn[m0]!</label> <label kind="assignment" x="-152" y="336">c=0</label> <nail x="-136" y="368"/> </transition> <transition> <source ref="id19"/> <target ref="id11"/> <label kind="synchronisation" x="96" y="312">machOn[m1]!</label> <label kind="assignment" x="120" y="336">c=0</label> <nail x="128" y="368"/> </transition> <transition> <source ref="id21"/> <target ref="id15"/> <label kind="synchronisation" x="632" y="320">machOn[m3]!</label> <label kind="assignment" x="664" y="344">c=0</label> <nail x="680" y="368"/> </transition> <transition> <source ref="id20"/> <target ref="id13"/> <label kind="synchronisation" x="376" y="312">machOn[m2]!</label> <label kind="assignment" x="400" y="336">c=0</label> <nail x="416" y="368"/> </transition> <transition> <source ref="id10"/> <target ref="id19"/> <label kind="synchronisation" x="-52" y="281">StepEnd?</label> <nail x="-32" y="344"/> </transition> <transition> <source ref="id12"/> <target ref="id20"/> <label kind="synchronisation" x="212" y="281">StepEnd?</label> <nail x="240" y="344"/> </transition> <transition> <source ref="id14"/> <target ref="id21"/> <label kind="synchronisation" x="500" y="285">StepEnd?</label> <nail x="528" y="336"/> </transition> <transition> <source ref="id8"/> <target ref="id18"/> <label kind="synchronisation" x="-348" y="281">StepEnd?</label> <nail x="-320" y="344"/> </transition> <transition> <source ref="id18"/> <target ref="id8"/> <label kind="guard" x="-288" y="320">busy[m0]</label> <nail x="-256" y="352"/> </transition> <transition> <source ref="id19"/> <target ref="id10"/> <label kind="guard" x="16" y="320">busy[m1]</label> <nail x="40" y="344"/> </transition> <transition> <source ref="id20"/> <target ref="id12"/> <label kind="guard" x="272" y="320">busy[m2]</label> <nail x="296" y="352"/> </transition> <transition> <source ref="id21"/> <target ref="id14"/> <label kind="guard" x="544" y="320">busy[m3]</label> <nail x="584" y="336"/> </transition> </template> <template> <name x="5" y="5">Machine</name> <parameter>const int no, const int startupCost</parameter> <declaration>//Insert local declarations of clocks, variables and constants.
clock cIdle;</declaration> <location id="id22" x="560" y="184"> <committed/> </location> <location id="id23" x="72" y="96"> <name x="40" y="64">ColdStart</name> </location> <location id="id24" x="416" y="-40"> <committed/> </location> <location id="id25" x="256" y="-40"> <name x="246" y="-70">Off</name> <committed/> </location> <location id="id26" x="256" y="96"> <name x="246" y="66">Busy</name> </location> <location id="id27" x="560" y="-40"> <name x="550" y="-70">Idle</name> </location> <location id="id28" x="416" y="184"> <committed/> </location> <init ref="id23"/> <transition> <source ref="id22"/> <target ref="id27"/> <label kind="synchronisation" x="512" y="56">iTimerOn[no]!</label> </transition> <transition> <source ref="id28"/> <target ref="id22"/> <label kind="synchronisation" x="456" y="160">StepEnd!</label> </transition> <transition> <source ref="id23"/> <target ref="id26"/> <label kind="synchronisation" x="136" y="72">machOn[no]?</label> <label kind="assignment" x="136" y="96">busy[no]=true</label> </transition> <transition> <source ref="id25"/> <target

ref="id26"/> </transition> <transition> <source ref="id24"/> <target ref="id26"/> <label kind="guard" x="288" y="72">cIdle<=startupCost</label> <label kind="synchronisation" x="304" y="96">iTimerOff[no]!</label> <nail x="416" y="96"/> </transition> <transition> <source ref="id24"/> <target ref="id25"/> <label kind="guard" x="280" y="-64">cIdle>startupCost</label> <label kind="synchronisation" x="296" y="-40">iTimerOff[no]!</label> </transition> <transition> <source ref="id27"/> <target ref="id24"/> <label kind="synchronisation" x="448" y="-64">machOn[no]?</label> <label kind="assignment" x="448" y="-40">busy[no]=true</label> </transition> <transition> <source ref="id26"/> <target ref="id28"/> <label kind="synchronisation" x="288" y="160">machOff[no]?</label> <label kind="assignment" x="280" y="184">busy[no]=false</label> <nail x="256" y="184"/> </transition> </template> <template> <name>CostTimer</name> <location id="id29" x="-280" y="-136"> <name x="-290" y="-166">idle</name> <label kind="invariant" x="-290" y="-121">cost'==15</label> </location> <init ref="id29"/> </template> <template> <name>Timer</name> <declaration>clock c;</declaration> <location id="id30" x="-328" y="-112"> <label kind="invariant" x="-336" y="-144">c<=2</label> </location> <location id="id31" x="-480" y="-40"> <label kind="invariant" x="-490" y="-25">c<=1</label> </location> <init ref="id31"/> <transition> <source ref="id30"/> <target ref="id31"/> <label kind="guard" x="-376" y="-64">c>1</label> <label kind="assignment" x="-408" y="-40">c=0 & glt++</label> <nail x="-328" y="-40"/> </transition> <transition> <source ref="id31"/> <target ref="id30"/> <label kind="guard" x="-408" y="-136">c>0</label> <label kind="assignment" x="-456" y="-136">glt++</label> <nail x="-480" y="-112"/> </transition> </template> <template> <name>IdleTimer</name> <parameter>const int[0,5] no</parameter> <declaration>clock c;</declaration> <location id="id32" x="-704" y="96"> <label kind="invariant" x="-714" y="111">cost'==2</label> </location> <location id="id33" x="-880" y="96"> </location> <init ref="id33"/> <transition> <source ref="id32"/> <target ref="id33"/> <label kind="synchronisation" x="-848" y="-8">iTimerOff[no]?</label> <nail x="-704" y="16"/> <nail x="-880" y="16"/> </transition> <transition> <source ref="id33"/> <target ref="id32"/> <label kind="synchronisation" x="-840" y="72">iTimerOn[no]?</label> </transition> </template> <system>//Insert process assignments.
machine0 = Machine(0,1);
machine1 = Machine(1,8);
machine2 = Machine(2,2);
machine3 = Machine(3,11);
machine4 = Machine(4,2);
machine5 = Machine(5,0);
job0=Job(0,1, 1,8, 2,3, 3,16, 4,2, 5,0, 32,0,20);
job1=Job(0,1, 1,15, 2,5, 3,23, 4,2, 5,0, 50,0,25);
job2=Job(0,2, 1,11, 2,4, 3,18, 4,3, 5,0, 41,0,25);
job3=Job(0,2, 1,23, 2,5, 3,29, 4,2, 5,0, 65,0,30);
//Edit system definition.
system machine0, machine1, machine2, machine3, machine4, machine5, job0, job1, job2, job3, CostTimer, IdleTimer;
</system> </nta>

ΠΕΙΡΑΜΑ 4.3.1 – Μεταβάσεις

job0.s0->job0.t0 { glc == 0, machOn[m0]!, c := 0 } machine0.ColdStart->machine0.Busy { 1, machOn[no]?, busy[no] := 1 }

job0.t0->job0.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 } job0.s1->job0.c1 { 1, StepEnd?, 1 }
job1.s0->job1.c0 { 1, StepEnd?, 1 } job2.s0->job2.c0 { 1, StepEnd?, 1 } job3.s0->job3.c0 { 1, StepEnd?, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job1.c0->job1.t0 { 1, machOn[m0]!, c := 0 } machine0.Idle->machine0._id24 { 1, machOn[no]?, busy[no] := 1 }
machine0._id24->machine0.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(0)._id32->IdleTimer(0)._id33 { 1, iTimerOff[no]?, 1 }
job0.c1->job0.t1 { 1, machOn[m1]!, c := 0 } machine1.ColdStart->machine1.Busy { 1, machOn[no]?, busy[no] := 1 }
job2.c0->job2.s0 { busy[m0], tau, 1 }
job3.c0->job3.s0 { busy[m0], tau, 1 }

job1.t0->job1.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 } job1.s1->job1.c1 { 1, StepEnd?, 1 }
job2.s0->job2.c0 { 1, StepEnd?, 1 } job3.s0->job3.c0 { 1, StepEnd?, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job3.c0->job3.t0 { 1, machOn[m0]!, c := 0 } machine0.Idle->machine0._id24 { 1, machOn[no]?, busy[no] := 1 }
machine0._id24->machine0.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(0)._id32->IdleTimer(0)._id33 { 1, iTimerOff[no]?, 1 }
machine0.Off->machine0.Busy { 1, tau, 1 }
job1.c1->job1.s1 { busy[m1], tau, 1 }
job2.c0->job2.s0 { busy[m0], tau, 1 }

job3.t0->job3.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 } job1.s1->job1.c1 { 1, StepEnd?, 1 }
job2.s0->job2.c0 { 1, StepEnd?, 1 } job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job1.c1->job1.s1 { busy[m1], tau, 1 }
job2.c0->job2.t0 { 1, machOn[m0]!, c := 0 } machine0.Idle->machine0._id24 { 1, machOn[no]?, busy[no] := 1 }
machine0._id24->machine0.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(0)._id32-

>IdleTimer(0)._id33 { 1, iTimerOff[no]?, 1 }
machine0.Off->machine0.Busy { 1, tau, 1 }
job3.c1->job3.s1 { busy[m1], tau, 1 }

job2.t0->job2.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 } job1.s1->job1.c1 { 1, StepEnd?, 1 }
job2.s1->job2.c1 { 1, StepEnd?, 1 } job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job3.c1->job3.s1 { busy[m1], tau, 1 }
job2.c1->job2.s1 { busy[m1], tau, 1 }
job1.c1->job1.s1 { busy[m1], tau, 1 }

job0.t1->job0.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job0.s2->job0.c2 { 1, StepEnd?, 1 }
job1.s1->job1.c1 { 1, StepEnd?, 1 } job2.s1->job2.c1 { 1, StepEnd?, 1 } job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job2.c1->job2.t1 { 1, machOn[m1]!, c := 0 } machine1.Idle->machine1._id24 { 1, machOn[no]?, busy[no] := 1 }
machine1._id24->machine1.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(1)._id32->IdleTimer(1)._id33 { 1, iTimerOff[no]?, 1 }
machine1.Off->machine1.Busy { 1, tau, 1 }
job1.c1->job1.s1 { busy[m1], tau, 1 }
job3.c1->job3.s1 { busy[m1], tau, 1 }
job0.c2->job0.t2 { 1, machOn[m2]!, c := 0 } machine2.ColdStart->machine2.Busy { 1, machOn[no]?, busy[no] := 1 }

job0.t2->job0.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job0.s3->job0.c3 { 1, StepEnd?, 1 }
job1.s1->job1.c1 { 1, StepEnd?, 1 } job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32 { 1, iTimerOn[no]?, 1 }
job3.c1->job3.s1 { busy[m1], tau, 1 }
job0.c3->job0.t3 { 1, machOn[m3]!, c := 0 } machine3.ColdStart->machine3.Busy { 1, machOn[no]?, busy[no] := 1 }
job1.c1->job1.s1 { busy[m1], tau, 1 }

job2.t1->job2.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job1.s1->job1.c1 { 1, StepEnd?, 1 }

job2.s2->job2.c2 { 1, StepEnd?, 1 } job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job2.c2->job2.t2 { 1, machOn[m2]!, c := 0 } machine2.Idle->machine2._id24 { 1, machOn[no]?, busy[no] := 1 }
machine2._id24->machine2.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(2)._id32->IdleTimer(2)._id33 { 1, iTimerOff[no]?, 1 }
machine2.Off->machine2.Busy { 1, tau, 1 }
job1.c1->job1.t1 { 1, machOn[m1]!, c := 0 } machine1.Idle->machine1._id24 { 1, machOn[no]?, busy[no] := 1 }
machine1._id24->machine1.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(1)._id32->IdleTimer(1)._id33 { 1, iTimerOff[no]?, 1 }
machine1.Off->machine1.Busy { 1, tau, 1 }
job3.c1->job3.s1 { busy[m1], tau, 1 }

job2.t2->job2.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job2.s3->job2.c3 { 1, StepEnd?, 1 }
job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32 { 1, iTimerOn[no]?, 1 }
job3.c1->job3.s1 { busy[m1], tau, 1 }
job2.c3->job2.s3 { busy[m3], tau, 1 }

job0.t3->job0.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job0.s4->job0.c4 { 1, StepEnd?, 1 }
job2.s3->job2.c3 { 1, StepEnd?, 1 } job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job2.c3->job2.t3 { 1, machOn[m3]!, c := 0 } machine3.Idle->machine3._id24 { 1, machOn[no]?, busy[no] := 1 }
machine3._id24->machine3.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(3)._id32->IdleTimer(3)._id33 { 1, iTimerOff[no]?, 1 }
machine3.Off->machine3.Busy { 1, tau, 1 }
job0.c4->job0.t4 { 1, machOn[m4]!, c := 0 } machine4.ColdStart->machine4.Busy { 1, machOn[no]?, busy[no] := 1 }
job3.c1->job3.s1 { busy[m1], tau, 1 }

job0.t4->job0.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 } job0.s5->job0.c5 { 1, StepEnd?, 1 }
job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32

{ 1, iTimerOn[no]?, 1 }
job3.c1->job3.s1 { busy[m1], tau, 1 }

job1.t1->job1.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job1.s2->job1.c2 { 1, StepEnd?, 1 }
job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job3.c1->job3.t1 { 1, machOn[m1]!, c := 0 } machine1.Idle->machine1._id24 { 1, machOn[no]?, busy[no] := 1 }
machine1._id24->machine1.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(1)._id32->IdleTimer(1)._id33 { 1, iTimerOff[no]?, 1 }
machine1.Off->machine1.Busy { 1, tau, 1 }
job1.c2->job1.t2 { 1, machOn[m2]!, c := 0 } machine2.Idle->machine2._id24 { 1, machOn[no]?, busy[no] := 1 }
machine2._id24->machine2.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(2)._id32->IdleTimer(2)._id33 { 1, iTimerOff[no]?, 1 }
machine2.Off->machine2.Busy { 1, tau, 1 }

job1.t2->job1.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job1.s3->job1.c3 { 1, StepEnd?, 1 }
machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32 { 1, iTimerOn[no]?, 1 }
job1.c3->job1.s3 { busy[m3], tau, 1 }

job2.t3->job2.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job1.s3->job1.c3 { 1, StepEnd?, 1 }
job2.s4->job2.c4 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job1.c3->job1.t3 { 1, machOn[m3]!, c := 0 } machine3.Idle->machine3._id24 { 1, machOn[no]?, busy[no] := 1 }
machine3._id24->machine3.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(3)._id32->IdleTimer(3)._id33 { 1, iTimerOff[no]?, 1 }
machine3.Off->machine3.Busy { 1, tau, 1 }
job2.c4->job2.t4 { 1, machOn[m4]!, c := 0 } machine4.Idle->machine4._id24 { 1, machOn[no]?, busy[no] := 1 }
machine4._id24->machine4.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(4)._id32->IdleTimer(4)._id33 { 1, iTimerOff[no]?, 1 }
machine4.Off->machine4.Busy { 1, tau, 1 }

job2.t4->job2.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 } job2.s5->job2.c5 { 1, StepEnd?, 1 }
machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32 { 1, iTimerOn[no]?, 1 }

job3.t1->job3.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job3.s2->job3.c2 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job3.c2->job3.t2 { 1, machOn[m2]!, c := 0 } machine2.Idle->machine2._id24 { 1, machOn[no]?, busy[no] := 1 }
machine2._id24->machine2.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(2)._id32->IdleTimer(2)._id33 { 1, iTimerOff[no]?, 1 }
machine2.Off->machine2.Busy { 1, tau, 1 }

job3.t2->job3.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job3.s3->job3.c3 { 1, StepEnd?, 1 }
machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32 { 1, iTimerOn[no]?, 1 }
job3.c3->job3.s3 { busy[m3], tau, 1 }

job1.t3->job1.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job1.s4->job1.c4 { 1, StepEnd?, 1 }
job3.s3->job3.c3 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job3.c3->job3.t3 { 1, machOn[m3]!, c := 0 } machine3.Idle->machine3._id24 { 1, machOn[no]?, busy[no] := 1 }
machine3._id24->machine3.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(3)._id32->IdleTimer(3)._id33 { 1, iTimerOff[no]?, 1 }
machine3.Off->machine3.Busy { 1, tau, 1 }
job1.c4->job1.t4 { 1, machOn[m4]!, c := 0 } machine4.Idle->machine4._id24 { 1, machOn[no]?, busy[no] := 1 }
machine4._id24->machine4.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(4)._id32->IdleTimer(4)._id33 { 1, iTimerOff[no]?, 1 }
machine4.Off->machine4.Busy { 1, tau, 1 }

job1.t4->job1.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 } job1.s5->job1.c5 { 1, StepEnd?, 1 }

machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32 { 1, iTimerOn[no]?, 1 }

job3.t3->job3.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job3.s4->job3.c4 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job3.c4->job3.t4 { 1, machOn[m4]!, c := 0 } machine4.Idle->machine4._id24 { 1, machOn[no]?, busy[no] := 1 }
machine4._id24->machine4.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(4)._id32->IdleTimer(4)._id33 { 1, iTimerOff[no]?, 1 }
machine4.Off->machine4.Busy { 1, tau, 1 }

job0.c5->job0.t5 { 1, machOn[m5]!, c := 0 } machine5.ColdStart->machine5.Busy { 1, machOn[no]?, busy[no] := 1 }

job2.c5->job2.s5 { busy[m5], tau, 1 }

job0.t5->job0._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }

machine5._id28->machine5._id22 { 1, StepEnd!, 1 } job2.s5->job2.c5 { 1, StepEnd?, 1 }

machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }

job0._id16->job0._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), extra += targetDeviation }

job2.c5->job2.t5 { 1, machOn[m5]!, c := 0 } machine5.Idle->machine5._id24 { 1, machOn[no]?, busy[no] := 1 }
machine5._id24->machine5.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(5)._id32->IdleTimer(5)._id33 { 1, iTimerOff[no]?, 1 }
machine5.Off->machine5.Busy { 1, tau, 1 }
job2.t5->job2._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }
machine5._id28->machine5._id22 { 1, StepEnd!, 1 }
machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }
job2._id16->job2._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), extra += targetDeviation }
job1.c5->job1.t5 { 1, machOn[m5]!, c := 0 } machine5.Idle->machine5._id24 { 1, machOn[no]?, busy[no] := 1 }
machine5._id24->machine5.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(5)._id32->IdleTimer(5)._id33 { 1, iTimerOff[no]?, 1 }
machine5.Off->machine5.Busy { 1, tau, 1 }

job3.t4->job3.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 } job3.s5->job3.c5 { 1, StepEnd?, 1 }
machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32 { 1, iTimerOn[no]?, 1 }
job0._id1->job0.end { 1, tau, 1 }
job1.t5->job1._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }

machine5._id28->machine5._id22 { 1, StepEnd!, 1 }

machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }
job2._id1->job2.end { 1, tau, 1 }
job1._id16->job1._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), extra += targetDeviation }
job3.c5->job3.t5 { 1, machOn[m5]!, c := 0 } machine5.Idle->machine5._id24 { 1, machOn[no]?, busy[no] := 1 }
machine5._id24->machine5.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(5)._id32->IdleTimer(5)._id33 { 1, iTimerOff[no]?, 1 }
machine5.Off->machine5.Busy { 1, tau, 1 }
job3.t5->job3._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }
machine5._id28->machine5._id22 { 1, StepEnd!, 1 }
machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }

job3._id16->job3._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), extra += targetDeviation }
job1._id1->job1.end { 1, tau, 1 }
job3._id1->job3.end { 1, tau, 1 }

ΠΕΙΡΑΜΑ 4.3.2 – Αρχείο xml

<?xml version="1.0" encoding="utf-8"?> <!DOCTYPE nta PUBLIC '-//Uppaal Team//DTD Flat System 1.1//EN' 'http://www.it.uu.se/research/group/darts/uppaal/flat-1_1.dtd'> <nta> <declaration>//Insert declarations of global clocks, variables, constants and channels.
chan machOn[6];
chan machOff[6];
chan iTimerOn[6];
chan iTimerOff[6];
clock glc;
bool busy[6]={false, false, false, false, false, false};
urgent broadcast chan StepEnd;
chan trigger;
int best=1000;
meta int[-1000,1000] remaining;
int glt=0;
int extra=0;</declaration> <template> <name x="5" y="5">Job</name> <parameter>const int m0, const int d0, const int m1, const int d1, const int m2, const int d2, const int m3, const int d3, const int m4, const int d4, const int m5, const int d5, const int Target, const int Early, const int Late</parameter> <declaration>clock c;
clock k;
int targetDeviation=0;</declaration> <location id="id0" x="-128" y="504"> <name x="-138" y="474">end</name> </location> <location id="id1" x="-40" y="584"> <urgent/> </location> <location id="id2" x="304" y="440"> <name x="294" y="410">t5</name> <label kind="invariant" x="294" y="455">c<=d5</label> </location> <location id="id3" x="400" y="600"> <name x="390" y="570">c5</name> </location> <location id="id4" x="400" y="440"> <name x="390" y="410">s5</name> </location> <location id="id5" x="552" y="440"> <name x="542" y="410">t4</name> <label kind="invariant" x="542" y="455">c<=d4</label> </location> <location id="id6" x="680" y="600"> <name x="670" y="570">c4</name> <urgent/> </location> <location id="id7" x="680" y="440"> <name x="670" y="410">s4</name> </location> <location id="id8" x="-288" y="208"> <name x="-298" y="178">s0</name> </location> <location id="id9" x="-136" y="208"> <name x="-146" y="178">t0</name> <label kind="invariant" x="-168" y="224">c<=d0</label> </location> <location id="id10" x="8" y="208"> <name x="-2" y="178">s1</name> </location> <location id="id11" x="128" y="208"> <name x="118" y="178">t1</name> <label kind="invariant" x="104" y="224">c<=d1</label> </location> <location id="id12" x="272" y="208"> <name x="262" y="178">s2</name> </location> <location id="id13" x="416" y="208"> <name x="406" y="178">t2</name> <label kind="invariant" x="384" y="224">c<=d2</label> </location> <location id="id14" x="560" y="208"> <name x="550" y="178">s3</name> </location> <location id="id15" x="680" y="216"> <name x="670" y="186">t3</name> <label kind="invariant" x="670" y="231">c<=d3</label> </location> <location id="id16" x="160" y="440"> <urgent/> </location> <location id="id17" x="272" y="-16"> <name x="262" y="-46">stop</name> </location> <location id="id18" x="-288" y="368"> <name x="-298" y="338">c0</name> <urgent/> </location> <location id="id19" x="8" y="368"> <name x="-2" y="338">c1</name> <urgent/> </location> <location id="id20" x="272" y="368"> <name

x="262" y="338">c2</name> <urgent/> </location> <location id="id21" x="560" y="368"> <name x="550" y="338">c3</name> <urgent/> </location> <init ref="id8"/> <transition> <source ref="id1"/> <target ref="id0"/> <nail x="-128" y="584"/> </transition> <transition> <source ref="id16"/> <target ref="id1"/> <label kind="guard" x="8" y="560">glt>Target</label> <label kind="assignment" x="-128" y="592">targetDeviation=Late*(glt-Target), cost+=targetDeviation</label> <nail x="160" y="584"/> </transition> <transition> <source ref="id16"/> <target ref="id1"/> <label kind="guard" x="-24" y="448">glt<=Target</label> <label kind="assignment" x="-144" y="408">targetDeviation=Early*(Target-glt), cost+=targetDeviation</label> <nail x="-40" y="440"/> </transition> <transition> <source ref="id2"/> <target ref="id17"/> <label kind="guard" x="228" y="182">(glc>best)</label> <nail x="304" y="256"/> </transition> <transition> <source ref="id5"/> <target ref="id17"/> <label kind="guard" x="352" y="182">(glc>best)</label> <nail x="552" y="392"/> <nail x="336" y="392"/> </transition> <transition> <source ref="id3"/> <target ref="id4"/> <label kind="guard" x="416" y="504">busy[m5]</label> <nail x="440" y="504"/> </transition> <transition> <source ref="id3"/> <target ref="id2"/> <label kind="synchronisation" x="272" y="512">machOn[m5]!</label> <label kind="assignment" x="280" y="528">c=0</label> <nail x="304" y="600"/> </transition> <transition> <source ref="id4"/> <target ref="id3"/> <label kind="synchronisation" x="352" y="488">StepEnd?</label> <nail x="376" y="496"/> </transition> <transition> <source ref="id5"/> <target ref="id4"/> <label kind="guard" x="448" y="408">(c>=d4) and (glc<=best)</label> <label kind="synchronisation" x="448" y="440">machOff[m4]!</label> </transition> <transition> <source ref="id6"/> <target ref="id5"/> <label kind="synchronisation" x="528" y="520">machOn[m4]!</label> <label kind="assignment" x="528" y="536">c=0</label> <nail x="552" y="600"/> </transition> <transition> <source ref="id6"/> <target ref="id7"/> <label kind="guard" x="712" y="480">busy[m4]</label> <nail x="712" y="488"/> </transition> <transition> <source ref="id7"/> <target ref="id6"/> <label kind="synchronisation" x="620" y="465">StepEnd?</label> <nail x="656" y="480"/> </transition> <transition> <source ref="id15"/> <target ref="id7"/> <label kind="guard" x="752" y="288">(c>=d3) and (glc<=best)</label> <label kind="synchronisation" x="744" y="328">machOff[m3]!</label> <nail x="744" y="216"/> <nail x="744" y="440"/> </transition> <transition> <source ref="id2"/> <target ref="id16"/> <label kind="guard" x="200" y="448">(c>=d5) and (glc<=best)</label> <label kind="synchronisation" x="192" y="416">machOff[m5]!</label> </transition> <transition> <source ref="id8"/> <target ref="id9"/> <label kind="guard" x="-232" y="184">glc==0</label> <label kind="synchronisation" x="-248" y="168">machOn[m0]!</label> <label kind="assignment" x="-224" y="128">c=0</label> <nail x="-256" y="160"/> <nail x="-176" y="160"/> </transition> <transition> <source ref="id9"/> <target ref="id10"/> <label kind="guard" x="-88" y="168">(c>=d0) and (glc<=best)</label> <label kind="synchronisation" x="-104" y="224">machOff[m0]!</label> </transition> <transition> <source ref="id11"/> <target ref="id12"/> <label kind="guard" x="176" y="176">(c>=d1) and (glc<=best)</label> <label kind="synchronisation" x="160" y="216">machOff[m1]!</label> </transition> <transition> <source ref="id13"/> <target ref="id14"/> <label kind="guard" x="456" y="176">(c>=d2) and (glc<=best)</label> <label kind="synchronisation" x="456" y="216">machOff[m2]!</label>

</transition> <transition> <source ref="id9"/> <target ref="id17"/> <label kind="guard" x="-176" y="8">(glc>best)</label> <nail x="-136" y="-16"/> </transition> <transition> <source ref="id11"/> <target ref="id17"/> <label kind="guard" x="140" y="66">(glc>best)</label> </transition> <transition> <source ref="id13"/> <target ref="id17"/> <label kind="guard" x="284" y="66">(glc>best)</label> </transition> <transition> <source ref="id15"/> <target ref="id17"/> <label kind="guard" x="640" y="48">(glc>best)</label> <nail x="680" y="-16"/> </transition> <transition> <source ref="id18"/> <target ref="id9"/> <label kind="synchronisation" x="-184" y="320">machOn[m0]!</label> <label kind="assignment" x="-152" y="336">c=0</label> <nail x="-136" y="368"/> </transition> <transition> <source ref="id19"/> <target ref="id11"/> <label kind="synchronisation" x="96" y="312">machOn[m1]!</label> <label kind="assignment" x="120" y="336">c=0</label> <nail x="128" y="368"/> </transition> <transition> <source ref="id21"/> <target ref="id15"/> <label kind="synchronisation" x="632" y="320">machOn[m3]!</label> <label kind="assignment" x="664" y="344">c=0</label> <nail x="680" y="368"/> </transition> <transition> <source ref="id20"/> <target ref="id13"/> <label kind="synchronisation" x="376" y="312">machOn[m2]!</label> <label kind="assignment" x="400" y="336">c=0</label> <nail x="416" y="368"/> </transition> <transition> <source ref="id10"/> <target ref="id19"/> <label kind="synchronisation" x="-52" y="281">StepEnd?</label> <nail x="-32" y="344"/> </transition> <transition> <source ref="id12"/> <target ref="id20"/> <label kind="synchronisation" x="212" y="281">StepEnd?</label> <nail x="240" y="344"/> </transition> <transition> <source ref="id14"/> <target ref="id21"/> <label kind="synchronisation" x="500" y="285">StepEnd?</label> <nail x="528" y="336"/> </transition> <transition> <source ref="id8"/> <target ref="id18"/> <label kind="synchronisation" x="-348" y="281">StepEnd?</label> <nail x="-320" y="344"/> </transition> <transition> <source ref="id18"/> <target ref="id8"/> <label kind="guard" x="-288" y="320">busy[m0]</label> <nail x="-256" y="352"/> </transition> <transition> <source ref="id19"/> <target ref="id10"/> <label kind="guard" x="16" y="320">busy[m1]</label> <nail x="40" y="344"/> </transition> <transition> <source ref="id20"/> <target ref="id12"/> <label kind="guard" x="272" y="320">busy[m2]</label> <nail x="296" y="352"/> </transition> <transition> <source ref="id21"/> <target ref="id14"/> <label kind="guard" x="544" y="320">busy[m3]</label> <nail x="584" y="336"/> </transition> </template> <template> <name x="5" y="5">Machine</name> <parameter>const int no, const int startupCost</parameter> <declaration>//Insert local declarations of clocks, variables and constants.
clock cIdle;</declaration> <location id="id22" x="560" y="184"> <committed/> </location> <location id="id23" x="72" y="96"> <name x="40" y="64">ColdStart</name> </location> <location id="id24" x="416" y="-40"> <committed/> </location> <location id="id25" x="256" y="-40"> <name x="246" y="-70">Off</name> <committed/> </location> <location id="id26" x="256" y="96"> <name x="246" y="66">Busy</name> </location> <location id="id27" x="560" y="-40"> <name x="550" y="-70">Idle</name> </location> <location id="id28" x="416" y="184"> <committed/> </location> <init ref="id23"/> <transition> <source ref="id22"/> <target ref="id27"/> <label kind="synchronisation" x="512" y="56">iTimerOn[no]!</label> </transition> <transition> <source ref="id28"/> <target ref="id22"/> <label kind="synchronisation" x="456" y="160">StepEnd!</label> </transition> <transition> <source ref="id23"/> <target ref="id26"/> <label kind="synchronisation" x="136" y="72">machOn[no]?</label> <label kind="assignment" x="136" y="96">busy[no]=true</label> </transition> <transition> <source ref="id25"/> <target

ref="id26"/> </transition> <transition> <source ref="id24"/> <target ref="id26"/> <label kind="guard" x="288" y="72">cIdle<=startupCost</label> <label kind="synchronisation" x="304" y="96">iTimerOff[no]!</label> <nail x="416" y="96"/> </transition> <transition> <source ref="id24"/> <target ref="id25"/> <label kind="guard" x="280" y="-64">cIdle>startupCost</label> <label kind="synchronisation" x="296" y="-40">iTimerOff[no]!</label> </transition> <transition> <source ref="id27"/> <target ref="id24"/> <label kind="synchronisation" x="448" y="-64">machOn[no]?</label> <label kind="assignment" x="448" y="-40">busy[no]=true</label> </transition> <transition> <source ref="id26"/> <target ref="id28"/> <label kind="synchronisation" x="288" y="160">machOff[no]?</label> <label kind="assignment" x="280" y="184">busy[no]=false</label> <nail x="256" y="184"/> </transition> </template> <template> <name>CostTimer</name> <location id="id29" x="-280" y="-136"> <name x="-290" y="-166">idle</name> <label kind="invariant" x="-290" y="-121">cost'==15</label> </location> <init ref="id29"/> </template> <template> <name>Timer</name> <declaration>clock c;</declaration> <location id="id30" x="-328" y="-112"> <label kind="invariant" x="-336" y="-144">c<=2</label> </location> <location id="id31" x="-480" y="-40"> <label kind="invariant" x="-490" y="-25">c<=1</label> </location> <init ref="id31"/> <transition> <source ref="id30"/> <target ref="id31"/> <label kind="guard" x="-376" y="-64">c>1</label> <label kind="assignment" x="-408" y="-40">c=0 & glt++</label> <nail x="-328" y="-40"/> </transition> <transition> <source ref="id31"/> <target ref="id30"/> <label kind="guard" x="-408" y="-136">c>0</label> <label kind="assignment" x="-456" y="-136">glt++</label> <nail x="-480" y="-112"/> </transition> </template> <template> <name>IdleTimer</name> <parameter>const int[0,5] no</parameter> <declaration>clock c;</declaration> <location id="id32" x="-704" y="96"> <label kind="invariant" x="-714" y="111">cost'==1</label> </location> <location id="id33" x="-880" y="96"> </location> <init ref="id33"/> <transition> <source ref="id32"/> <target ref="id33"/> <label kind="synchronisation" x="-848" y="-8">iTimerOff[no]?</label> <nail x="-704" y="16"/> <nail x="-880" y="16"/> </transition> <transition> <source ref="id33"/> <target ref="id32"/> <label kind="synchronisation" x="-840" y="72">iTimerOn[no]?</label> </transition> </template> <system>//Insert process assignments.
machine0 = Machine(0,1);
machine1 = Machine(1,8);
machine2 = Machine(2,2);
machine3 = Machine(3,11);
machine4 = Machine(4,2);
machine5 = Machine(5,0);
job0=Job(0,1, 1,8, 2,3, 3,16, 4,2, 5,0, 32,0,20);
job1=Job(0,1, 1,15, 2,5, 3,23, 4,2, 5,0, 50,0,25);
job2=Job(0,2, 1,11, 2,4, 3,18, 4,3, 5,0, 41,0,25);
job3=Job(0,2, 1,23, 2,5, 3,29, 4,2, 5,0, 65,0,30);
//Edit system definition.
system machine0, machine1, machine2, machine3, machine4, machine5, job0, job1, job2, job3, CostTimer, IdleTimer;
</system> </nta>

ΠΕΙΡΑΜΑ 4.3.2 – Μεταβάσεις

job0.s0->job0.t0 { glc == 0, machOn[m0]!, c := 0 } machine0.ColdStart->machine0.Busy { 1, machOn[no]?, busy[no] := 1 }

job0.t0->job0.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 } job0.s1->job0.c1 { 1, StepEnd?, 1 }
job1.s0->job1.c0 { 1, StepEnd?, 1 } job2.s0->job2.c0 { 1, StepEnd?, 1 } job3.s0->job3.c0 { 1, StepEnd?, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job0.c1->job0.t1 { 1, machOn[m1]!, c := 0 } machine1.ColdStart->machine1.Busy { 1, machOn[no]?, busy[no] := 1 }
job1.c0->job1.t0 { 1, machOn[m0]!, c := 0 } machine0.Idle->machine0._id24 { 1, machOn[no]?, busy[no] := 1 }
machine0._id24->machine0.Busy { cIdle <= startupCost, iTimerOff[no]!, 1 } IdleTimer(0)._id32->IdleTimer(0)._id33 { 1, iTimerOff[no]?, 1 }
job3.c0->job3.s0 { busy[m0], tau, 1 }
job2.c0->job2.s0 { busy[m0], tau, 1 }

job1.t0->job1.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 } job1.s1->job1.c1 { 1, StepEnd?, 1 }
job2.s0->job2.c0 { 1, StepEnd?, 1 } job3.s0->job3.c0 { 1, StepEnd?, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job1.c1->job1.s1 { busy[m1], tau, 1 }
job2.c0->job2.t0 { 1, machOn[m0]!, c := 0 } machine0.Idle->machine0._id24 { 1, machOn[no]?, busy[no] := 1 }
machine0._id24->machine0.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(0)._id32->IdleTimer(0)._id33 { 1, iTimerOff[no]?, 1 }
machine0.Off->machine0.Busy { 1, tau, 1 }
job3.c0->job3.s0 { busy[m0], tau, 1 }

job2.t0->job2.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 } job1.s1->job1.c1 { 1, StepEnd?, 1 }
job2.s1->job2.c1 { 1, StepEnd?, 1 } job3.s0->job3.c0 { 1, StepEnd?, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job2.c1->job2.s1 { busy[m1], tau, 1 }
job3.c0->job3.t0 { 1, machOn[m0]!, c := 0 } machine0.Idle->machine0._id24 { 1, machOn[no]?, busy[no] := 1 }
machine0._id24->machine0.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(0)._id32-

>IdleTimer(0)._id33 { 1, iTimerOff[no]?, 1 }
machine0.Off->machine0.Busy { 1, tau, 1 }
job1.c1->job1.s1 { busy[m1], tau, 1 }

job3.t0->job3.s1 { c >= d0 && glc <= best, machOff[m0]!, 1 } machine0.Busy->machine0._id28 { 1, machOff[no]?, busy[no] := 0 }
machine0._id28->machine0._id22 { 1, StepEnd!, 1 } job1.s1->job1.c1 { 1, StepEnd?, 1 }
job2.s1->job2.c1 { 1, StepEnd?, 1 } job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine0._id22->machine0.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(0)._id33->IdleTimer(0)._id32 { 1, iTimerOn[no]?, 1 }
job1.c1->job1.s1 { busy[m1], tau, 1 }
job3.c1->job3.s1 { busy[m1], tau, 1 }
job2.c1->job2.s1 { busy[m1], tau, 1 }

job0.t1->job0.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job0.s2->job0.c2 { 1, StepEnd?, 1 }
job1.s1->job1.c1 { 1, StepEnd?, 1 } job2.s1->job2.c1 { 1, StepEnd?, 1 } job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job2.c1->job2.t1 { 1, machOn[m1]!, c := 0 } machine1.Idle->machine1._id24 { 1, machOn[no]?, busy[no] := 1 }
machine1._id24->machine1.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(1)._id32->IdleTimer(1)._id33 { 1, iTimerOff[no]?, 1 }
machine1.Off->machine1.Busy { 1, tau, 1 }
job0.c2->job0.t2 { 1, machOn[m2]!, c := 0 } machine2.ColdStart->machine2.Busy { 1, machOn[no]?, busy[no] := 1 }
job3.c1->job3.s1 { busy[m1], tau, 1 }
job1.c1->job1.s1 { busy[m1], tau, 1 }

job0.t2->job0.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job0.s3->job0.c3 { 1, StepEnd?, 1 }
job1.s1->job1.c1 { 1, StepEnd?, 1 } job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32 { 1, iTimerOn[no]?, 1 }
job0.c3->job0.t3 { 1, machOn[m3]!, c := 0 } machine3.ColdStart->machine3.Busy { 1, machOn[no]?, busy[no] := 1 }
job3.c1->job3.s1 { busy[m1], tau, 1 }
job1.c1->job1.s1 { busy[m1], tau, 1 }

job2.t1->job2.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job1.s1->job1.c1 { 1, StepEnd?, 1 }

job2.s2->job2.c2 { 1, StepEnd?, 1 } job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job1.c1->job1.t1 { 1, machOn[m1]!, c := 0 } machine1.Idle->machine1._id24 { 1, machOn[no]?, busy[no] := 1 }
machine1._id24->machine1.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(1)._id32->IdleTimer(1)._id33 { 1, iTimerOff[no]?, 1 }
machine1.Off->machine1.Busy { 1, tau, 1 }
job3.c1->job3.s1 { busy[m1], tau, 1 }
job2.c2->job2.t2 { 1, machOn[m2]!, c := 0 } machine2.Idle->machine2._id24 { 1, machOn[no]?, busy[no] := 1 }
machine2._id24->machine2.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(2)._id32->IdleTimer(2)._id33 { 1, iTimerOff[no]?, 1 }
machine2.Off->machine2.Busy { 1, tau, 1 }

job2.t2->job2.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job2.s3->job2.c3 { 1, StepEnd?, 1 }
job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32 { 1, iTimerOn[no]?, 1 }
job3.c1->job3.s1 { busy[m1], tau, 1 }
job2.c3->job2.s3 { busy[m3], tau, 1 }

job0.t3->job0.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job0.s4->job0.c4 { 1, StepEnd?, 1 }
job2.s3->job2.c3 { 1, StepEnd?, 1 } job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job3.c1->job3.s1 { busy[m1], tau, 1 }
job2.c3->job2.t3 { 1, machOn[m3]!, c := 0 } machine3.Idle->machine3._id24 { 1, machOn[no]?, busy[no] := 1 }
machine3._id24->machine3.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(3)._id32->IdleTimer(3)._id33 { 1, iTimerOff[no]?, 1 }
machine3.Off->machine3.Busy { 1, tau, 1 }
job0.c4->job0.t4 { 1, machOn[m4]!, c := 0 } machine4.ColdStart->machine4.Busy { 1, machOn[no]?, busy[no] := 1 }

job0.t4->job0.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 } job0.s5->job0.c5 { 1, StepEnd?, 1 }
job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32

{ 1, iTimerOn[no]?, 1 }
job3.c1->job3.s1 { busy[m1], tau, 1 }

job1.t1->job1.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job1.s2->job1.c2 { 1, StepEnd?, 1 }
job3.s1->job3.c1 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job1.c2->job1.t2 { 1, machOn[m2]!, c := 0 } machine2.Idle->machine2._id24 { 1, machOn[no]?, busy[no] := 1 }
machine2._id24->machine2.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(2)._id32->IdleTimer(2)._id33 { 1, iTimerOff[no]?, 1 }
machine2.Off->machine2.Busy { 1, tau, 1 }
job3.c1->job3.t1 { 1, machOn[m1]!, c := 0 } machine1.Idle->machine1._id24 { 1, machOn[no]?, busy[no] := 1 }
machine1._id24->machine1.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(1)._id32->IdleTimer(1)._id33 { 1, iTimerOff[no]?, 1 }
machine1.Off->machine1.Busy { 1, tau, 1 }

job1.t2->job1.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job1.s3->job1.c3 { 1, StepEnd?, 1 }
machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32 { 1, iTimerOn[no]?, 1 }
job1.c3->job1.s3 { busy[m3], tau, 1 }

job2.t3->job2.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job1.s3->job1.c3 { 1, StepEnd?, 1 }
job2.s4->job2.c4 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job2.c4->job2.t4 { 1, machOn[m4]!, c := 0 } machine4.Idle->machine4._id24 { 1, machOn[no]?, busy[no] := 1 }
machine4._id24->machine4.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(4)._id32->IdleTimer(4)._id33 { 1, iTimerOff[no]?, 1 }
machine4.Off->machine4.Busy { 1, tau, 1 }
job1.c3->job1.t3 { 1, machOn[m3]!, c := 0 } machine3.Idle->machine3._id24 { 1, machOn[no]?, busy[no] := 1 }
machine3._id24->machine3.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(3)._id32->IdleTimer(3)._id33 { 1, iTimerOff[no]?, 1 }
machine3.Off->machine3.Busy { 1, tau, 1 }

job2.t4->job2.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 } job2.s5->job2.c5 { 1, StepEnd?, 1 }
machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32 { 1, iTimerOn[no]?, 1 }

job3.t1->job3.s2 { c >= d1 && glc <= best, machOff[m1]!, 1 } machine1.Busy->machine1._id28 { 1, machOff[no]?, busy[no] := 0 }
machine1._id28->machine1._id22 { 1, StepEnd!, 1 } job3.s2->job3.c2 { 1, StepEnd?, 1 }
machine1._id22->machine1.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(1)._id33->IdleTimer(1)._id32 { 1, iTimerOn[no]?, 1 }
job3.c2->job3.t2 { 1, machOn[m2]!, c := 0 } machine2.Idle->machine2._id24 { 1, machOn[no]?, busy[no] := 1 }
machine2._id24->machine2.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(2)._id32->IdleTimer(2)._id33 { 1, iTimerOff[no]?, 1 }
machine2.Off->machine2.Busy { 1, tau, 1 }

job3.t2->job3.s3 { c >= d2 && glc <= best, machOff[m2]!, 1 } machine2.Busy->machine2._id28 { 1, machOff[no]?, busy[no] := 0 }
machine2._id28->machine2._id22 { 1, StepEnd!, 1 } job3.s3->job3.c3 { 1, StepEnd?, 1 }
machine2._id22->machine2.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(2)._id33->IdleTimer(2)._id32 { 1, iTimerOn[no]?, 1 }
job3.c3->job3.s3 { busy[m3], tau, 1 }

job1.t3->job1.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job1.s4->job1.c4 { 1, StepEnd?, 1 }
job3.s3->job3.c3 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job3.c3->job3.t3 { 1, machOn[m3]!, c := 0 } machine3.Idle->machine3._id24 { 1, machOn[no]?, busy[no] := 1 }
machine3._id24->machine3.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(3)._id32->IdleTimer(3)._id33 { 1, iTimerOff[no]?, 1 }
machine3.Off->machine3.Busy { 1, tau, 1 }
job1.c4->job1.t4 { 1, machOn[m4]!, c := 0 } machine4.Idle->machine4._id24 { 1, machOn[no]?, busy[no] := 1 }
machine4._id24->machine4.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(4)._id32->IdleTimer(4)._id33 { 1, iTimerOff[no]?, 1 }
machine4.Off->machine4.Busy { 1, tau, 1 }

job1.t4->job1.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 } job1.s5->job1.c5 { 1, StepEnd?, 1 }

machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32 { 1, iTimerOn[no]?, 1 }

job3.t3->job3.s4 { c >= d3 && glc <= best, machOff[m3]!, 1 } machine3.Busy->machine3._id28 { 1, machOff[no]?, busy[no] := 0 }
machine3._id28->machine3._id22 { 1, StepEnd!, 1 } job3.s4->job3.c4 { 1, StepEnd?, 1 }
machine3._id22->machine3.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(3)._id33->IdleTimer(3)._id32 { 1, iTimerOn[no]?, 1 }
job3.c4->job3.t4 { 1, machOn[m4]!, c := 0 } machine4.Idle->machine4._id24 { 1, machOn[no]?, busy[no] := 1 }
machine4._id24->machine4.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(4)._id32->IdleTimer(4)._id33 { 1, iTimerOff[no]?, 1 }
machine4.Off->machine4.Busy { 1, tau, 1 }

job0.c5->job0.t5 { 1, machOn[m5]!, c := 0 } machine5.ColdStart->machine5.Busy { 1, machOn[no]?, busy[no] := 1 }

job2.c5->job2.s5 { busy[m5], tau, 1 }

job1.c5->job1.s5 { busy[m5], tau, 1 }

job0.t5->job0._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }

machine5._id28->machine5._id22 { 1, StepEnd!, 1 } job1.s5->job1.c5 { 1, StepEnd?, 1 }
job2.s5->job2.c5 { 1, StepEnd?, 1 }

machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }

job2.c5->job2.t5 { 1, machOn[m5]!, c := 0 } machine5.Idle->machine5._id24 { 1, machOn[no]?, busy[no] := 1 }
machine5._id24->machine5.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(5)._id32->IdleTimer(5)._id33 { 1, iTimerOff[no]?, 1 }
machine5.Off->machine5.Busy { 1, tau, 1 }
job2.t5->job2._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }

machine5._id28->machine5._id22 { 1, StepEnd!, 1 }

machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }
job2._id16->job2._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), cost += targetDeviation }
job1.c5->job1.t5 { 1, machOn[m5]!, c := 0 } machine5.Idle->machine5._id24 { 1, machOn[no]?, busy[no] := 1 }
machine5._id24->machine5.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(5)._id32-

>IdleTimer(5)._id33 { 1, iTimerOff[no]?, 1 }
machine5.Off->machine5.Busy { 1, tau, 1 }
job1.t5->job1._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }
machine5._id28->machine5._id22 { 1, StepEnd!, 1 }
machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }
job1._id16->job1._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), cost += targetDeviation }

job3.t4->job3.s5 { c >= d4 && glc <= best, machOff[m4]!, 1 } machine4.Busy->machine4._id28 { 1, machOff[no]?, busy[no] := 0 }
machine4._id28->machine4._id22 { 1, StepEnd!, 1 } job3.s5->job3.c5 { 1, StepEnd?, 1 }
machine4._id22->machine4.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(4)._id33->IdleTimer(4)._id32 { 1, iTimerOn[no]?, 1 }

job3.c5->job3.t5 { 1, machOn[m5]!, c := 0 } machine5.Idle->machine5._id24 { 1, machOn[no]?, busy[no] := 1 }
machine5._id24->machine5.Off { cIdle > startupCost, iTimerOff[no]!, 1 } IdleTimer(5)._id32->IdleTimer(5)._id33 { 1, iTimerOff[no]?, 1 }
machine5.Off->machine5.Busy { 1, tau, 1 }

job3.t5->job3._id16 { c >= d5 && glc <= best, machOff[m5]!, 1 } machine5.Busy->machine5._id28 { 1, machOff[no]?, busy[no] := 0 }
machine5._id28->machine5._id22 { 1, StepEnd!, 1 }
machine5._id22->machine5.Idle { 1, iTimerOn[no]!, 1 } IdleTimer(5)._id33->IdleTimer(5)._id32 { 1, iTimerOn[no]?, 1 }
job2._id1->job2.end { 1, tau, 1 }
job1._id1->job1.end { 1, tau, 1 }
job3._id16->job3._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), cost += targetDeviation }

job3._id1->job3.end { 1, tau, 1 }
job0._id16->job0._id1 { glt <= Target, tau, targetDeviation := Early * (Target - glt), cost += targetDeviation }
job0._id1->job0.end { 1, tau, 1 }