Ατομική Διπλωματική Εργασία

ΜΟΝΤΕΛΟΠΟΙΗΣΗ ΚΑΙ ΕΠΑΛΗΘΕΥΣΗ ΑΛΓΟΡΙΘΜΩΝ ΜΕΤΑΔΟΣΗΣ ΜΗΝΥΜΑΤΟΣ ΜΕ ΑΛΓΕΒΡΑ ΔΙΕΡΓΑΣΙΩΝ ΣΤΗΝ ΠΑΡΟΥΣΙΑ ΣΦΑΛΜΑΤΩΝ ΚΑΤΑΡΡΕΥΣΗΣ

Μιχάλης Κασιουλής

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ιούνιος 2014

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΟΝΤΕΛΟΠΟΙΗΣΗ ΚΑΙ ΕΠΑΛΗΘΕΥΣΗ ΑΛΓΟΡΙΘΜΩΝ ΜΕΤΑΔΟΣΗΣ ΜΗΝΥΜΑΤΟΣ ΜΕ ΑΛΓΕΒΡΑ ΔΙΕΡΓΑΣΙΩΝ ΣΤΗΝ ΠΑΡΟΥΣΙΑ ΣΦΑΛΜΑΤΩΝ ΚΑΤΑΡΡΕΥΣΗΣ

Μιχάλης Κασιουλής

Επιβλέπουσα Καθηγήτρια
Άννα Φιλίππου

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

Ιούνιος 2014

Ευχαριστίες

Ιδιαίτερες ευχαριστίες θα ήθελα να εκφράσω στην επιβλέπουσα καθηγήτρια μου Αναπληρώτρια Καθηγήτρια Άννα Φιλίππου για την βοήθεια και την στήριξη που μου πρόσφερε όλο αυτό το διάστημα στα πλαίσια της εκπόνησης της διπλωματικής μου εργασίας όπως επίσης και για τις πολύτιμες συμβουλές της και το αμείωτο ενδιαφέρον της.

Περίληψη

Οι κοινές αλγοριθμικές λύσεις σε κάποια αλγοριθμικά προβλήματα είναι άτυπα διατυπωμένες σε φυσική γλώσσα που κάποιες φορές μπορεί να είναι ελλιπής και να περιέχουν ασάφεια. Για πιο τυπική ανάλυση κατανεμημένων αλγορίθμων απαραίτητη είναι η χρήση τυπικών περιγραφών. Παρ' όλα αυτά η τυποποίηση τέτοιων αλγορίθμων τείνει στο να μας οδηγεί σε πολύπλοκες περιγραφές και αποδείξεις.

Σε αυτή την διπλωματική εργασία παρουσιάζουμε κάποιες μεθοδολογίες οι οποίες μας βοηθούν στην απλοποίηση της απόδειξης ορθότητας κατανεμημένων αλγορίθμων στην παρουσία σφαλμάτων. Στο πλαίσιο της εργασίας μας υιοθετήσαμε μια παραλλαγή της άλγεβρας Dpi-calculus των Francalanza και Hennessy την οποία ονομάσαμε Dπf γύρω από την οποία αναπτύξαμε την έννοια της συρροής (confluence) για την σχετική άλγεβρα μαζί με την τεχνική για απόδειξη ορθότητας. Σύμφωνα με την μεθοδολογία μας ένας κατανεμημένος αλγόριθμος μπορεί να μοντελοποιηθεί σαν μια Dπf διεργασία. Η επιθυμητή συμπεριφορά αυτής της διεργασίας μπορεί επίσης να μοντελοποιηθεί σαν μια Dπf διεργασία. Επομένως το κριτήριο ορθότητας εκφράζεται ως η απαίτηση ο αλγόριθμος να είναι ισοδύναμος με την προδιαγραφή την οποία ορίσαμε. Για να αποδείξουμε αυτή την ισοδυναμία ορίζουμε την έννοια της διεργασίας περιβλήματος (wrapper) όπως επίσης χρησιμοποιήθηκε από τους Francalanza και Hennessy και την έννοια της συρροής όπως αναπτύχθηκε από αυτούς.

Χρησιμοποιούμε την μεθοδολογία αυτή για να αποδείξουμε ορθότητα για δύο αλγορίθμους μετάδοσης μηνύματος τον LRB(Lazy Reliable Broadcast) και τον URB(Uniform Reliable Broadcast)

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός Διπλωματικής

Ο απώτερος σκοπός της διπλωματικής μου εργασίας ήταν η ανάπτυξη μεθοδολογίας για την τυπική ανάλυση κατανεμημένων αλγορίθμων στην παρουσία σφαλμάτων. Πιο συγκεκριμένα στην δική μου εργασία ήταν η μοντελοποίηση σε άλγεβρα διεργασιών και η απόδειξη ορθότητας δύο αλγορίθμων μετάδοσης μηνύματος οι οποίοι μπορούν να χρησιμοποιηθούν σε περιβάλλον που υπάρχουν σφάλματα κατάρρευσης των επεξεργαστών. Η σημαντικότητα της διπλωματικής αυτής εργασίας έγκειται στο γεγονός ότι τέτοιου είδους αποδείξεις δεν είναι εφικτό στο να γίνουν με φυσική γλώσσα γιατί μπορεί να υπάρξουν ασάφειες και έτσι απαραίτητη είναι η ανάπτυξη μεθοδολογίας μέσω της οποίας θα αποδείξουμε την ορθότητα για δύο κατανεμημένους αλγορίθμους στην παρουσία ενός τυχαίου αριθμού σφαλμάτων. Σαν κίνητρο αυτής της διπλωματικής αποτέλεσε η ανάπτυξη της μεθοδολογίας αυτής.

1.2 Μεθοδολογία

Η μεθοδολογία η οποία ακολουθήθηκε για την επίτευξη του πιο πάνω στόχου ήταν αρχικά η κατανόηση των βασικών εννοιών της άλγεβρας και πιο συγκεκριμένα της σύνταξης, σημασιολογίας και κανόνων ισοδυναμίας που την απαρτίζουν. Σε αυτό το

σημείο αξίζει να σημειωθεί ότι σημαντική ήταν η συνεισφορά πιο παλιών διπλωματικών για την επίτευξη του πιο πάνω στόχου στις οποίες χρησιμοποιήθηκε ως υπόβαθρο η CCS και τις οποίες θα αναφέρουμε στο υποκεφάλαιο 1.4. Ακολούθως κατανοώντας τις ελλείψεις της συγκεκριμένης γλώσσας σε προβλήματα όπου υπάρχουν σφάλματα ορίσαμε την δική μας άλγεβρα στην οποία προσθέσαμε καινούριους τελεστές για να μπορούμε να χειριστούμε τα σφάλματα. Μετέπειτα μελετήσαμε τους δύο αλγορίθμους μετάδοσης μηνύματος τον LRB(Lazy Reliable Broadcast) και τον URB (Uniform Reliable Broadcast) και αφού κατανοήσαμε τον τρόπο λειτουργίας των δύο αλγορίθμων ορίσαμε την αντίστοιχη σύνταξη, σημασιολογία και κανόνες ισοδυναμίας στην γλώσσα Dπf και χρησιμοποιήσαμε τα δύο πρώτα για την μοντελοποίηση των αλγορίθμων. Στη συνέχεια χρησιμοποιήσαμε τους κανόνες ισοδυναμίας για την απόδειξη της ορθότητας τους και πιο συγκεκριμένα δείχνοντας ότι το σύστημα έχει την αναμενόμενη συμπεριφορά σύμφωνα με την προδιαγραφή του. Καθοριστική ήταν έννοια της συρροής σε αυτό το σημείο μιας μορφής ντετερμινισμού μέσω της οποίας μπορούμε να εγγυηθούμε ότι κάθε πιθανή εκτέλεση του αλγορίθμου μας μπορεί να μας δώσει το επιθυμητό αποτέλεσμα.

1.3 Δομή Διπλωματικής

Στο Κεφάλαιο 2 που ακολουθεί γίνεται μια σύντομη περιγραφή των κατανεμημένων αλγορίθμων και ακολούθως γίνεται μια αναφορά στην άλγεβρα διεργασιών CCS περιγράφοντας την σύνταξη, κανόνες της σημασιολογίας της και τις σχέσεις ισοδυναμίας που υπάρχουν.

Στο Κεφάλαιο 3 γίνεται αναφορά στην άλγεβρα Dπf όπου γίνεται και πάλι περιγραφή των βασικών χαρακτηριστικών της τα οποία χρησιμοποιούμε μετέπειτα για μοντελοποίηση των αλγορίθμων μετάδοσης μηνύματος. Επίσης γίνεται αναφορά στις τεχνικές απόδειξης που χρησιμοποιούμε για να αποδείξουμε την ορθότητα των αλγορίθμων.

Στο Κεφάλαιο 4 γίνεται περιγραφή του προβλήματος μετάδοσης μηνύματος σε ένα κατανεμημένο σύστημα και ακολούθως περιγράφονται 3 αλγόριθμοι οι οποίοι λύνουν το πιο πάνω πρόβλημα.

Στο Κεφάλαιο 5 παρουσιάζεται η μοντελοποίηση του αλγόριθμου LRB σύμφωνα με την άλγεβρα Dπf και η απόδειξη της ορθότητας του με βάση της τεχνικές απόδειξης που αναφέραμε πιο πάνω.

Στο Κεφάλαιο 6 παρουσιάζεται η μοντελοποίηση του αλγόριθμου URB σύμφωνα με την άλγεβρα Dπf και η απόδειξη της ορθότητας του με βάση της τεχνικές απόδειξης που αναφέραμε πιο πάνω.

Στο Κεφάλαιο 7 παρουσιάζονται τα συμπεράσματα που προέκυψαν από όλη την διπλωματική εργασία.

1.4 Συνεισφορά

Σημαντική ήταν η βοήθεια που πήρα από παλαιότερες εργασίες στις οποίες έγινε χρήση παραλλαγών της ccs για μοντελοποίηση αλγορίθμων και πιο συγκεκριμένα στην διπλωματική εργασία του Γιώργου Μιχαήλ η οποία εκπονήθηκε τον Ιούνιο του 2006 και αφορούσε την μοντελοποίηση αλγορίθμων εκλογής προέδρου σε τοπολογία δακτυλίου σε μια παραλλαγή της γλώσσας CCS. Σε μια δεύτερη διπλωματική της Μαρίνας Γελαστού επίσης χρησιμοποιήθηκε μια παραλλαγή της CCS για την εφαρμογή τυπικών μεθόδων για ανάλυση και επαλήθευση κατανεμημένων αλγορίθμων. Στην πρώτη περίπτωση μελετήθηκε το άρθρο το οποίο γράφτηκε από την Αναπληρώτρια καθηγήτρια κα Άννα Φιλίππου βασισμένο στην διπλωματική του Γιώργου Μιχαήλ και στην δεύτερη περίπτωση μελετήθηκε το άρθρο το οποίο γράφτηκε από τον Επίκουρο καθηγητή κο Χρύση Γεωργίου μαζί με την Αναπληρώτρια καθηγήτρια κα Άννα Φιλίππου βασισμένο στην διπλωματική της Μαρίνας Γελαστού.

Κεφάλαιο 2

Υπόβαθρο

2.1 Κατανεμημένοι Αλγόριθμοι

Οι κατανεμημένοι αλγόριθμοι είναι οι αλγόριθμοι οι οποίοι είναι σχεδιασμένοι να λειτουργούν σε κατανεμημένα περιβάλλοντα όπου υπάρχουν ένα σύνολο από επεξεργαστές οι οποίοι βρίσκονται σε ξεχωριστά σημεία και οι οποίοι επικοινωνούν μεταξύ τους για την εκτέλεση μιας κοινής εργασίας ή ενός υπολογισμού[8]. Ο τρόπος που επικοινωνούν μεταξύ τους οι οντότητες είναι είτε μέσω κοινόχρηστης μνήμης είτε μέσω ανταλλαγής μηνυμάτων. Κλασικά παραδείγματα κατανεμημένων συστημάτων αποτελούν μηχανές συνδεδεμένες σε ένα τοπικό δίκτυο και το διαδίκτυο. Μπορούμε να διαχωρίσουμε τα κατανεμημένα συστήματα σε δύο κατηγορίες ανάλογα με το μοντέλο χρονισμού τους. Η πρώτη κατηγορία αποτελεί το σύγχρονο μοντέλο όπου οι

επεξεργαστές εκτελούν τους υπολογισμούς τους συγχρονισμένα σε γύρους και όλα τα μηνύματα παραδίδονται με το τέλος του γύρου. Η δεύτερη κατηγορία αποτελεί το ασύγχρονο μοντέλο όπου οι επεξεργαστές λειτουργούν με αυθαίρετες ταχύτητες και υπάρχουν αυθαίρετες καθυστερήσεις στις παραλαβές των μηνυμάτων όπως και αυθαίρετη καθυστέρηση στην πρόσβαση της κοινόχρηστης μνήμης αν χρησιμοποιείται κοινόχρηστη μνήμη. Σε ένα κατανεμημένο περιβάλλον μπορούν να προκύψουν διαφόρων ειδών σφάλματα των οντοτήτων που αποτελούν το σύστημα και μπορούμε να χωρίσουμε τα σφάλματα σε δύο κύριες κατηγορίες όπου η πρώτη είναι τα σφάλματα κατάρρευσης όταν ένας επεξεργαστής καταρρέει εντελώς και σταματά να παίρνει μέρος στον υπολογισμό και τα βυζαντινά σφάλματα όπου υπάρχει αυθαίρετη συμπεριφορά των επεξεργαστών[8]. Μια υποκατηγορία αυτών αποτελούν τα σφάλματα επανεκκίνησης. Στα σφάλματα κατάρρευσης οι επεξεργαστές δεν μπορούν να ξαναεπανέλθουν πίσω στον υπολογισμό ενώ στα βυζαντινά σφάλματα μπορεί να συμβεί αυτό καθώς επίσης ένας βυζαντινός επεξεργαστής μπορεί να στέλνει λανθασμένες τιμές στους υπόλοιπους έχοντας αυθαίρετη συμπεριφορά. Στην περίπτωση των αλγορίθμων μας μπορούν να υπάρξουν μόνο σφάλματα κατάρρευσης. Ένας κατανεμημένος αλγόριθμος σχεδιάζεται με βάση τον αριθμό σφαλμάτων που μπορεί να ανεχτεί όπως επίσης και από την πολυπλοκότητα μηνυμάτων του και την χρονική του πολυπλοκότητα. Ορίζουμε ως νόμιμη εκτέλεση σε ένα κατανεμημένο αλγόριθμο την εκτέλεση η οποία ικανοποιεί την συνθήκη ασφαλείας που διασφαλίζει ότι η εκτέλεση ακολουθά σωστά τις μεταβάσεις και την συνθήκη ζωτικότητας που διασφαλίζει ότι το σύστημα δεν θα μείνει άπρακτο επ’ άπειρον. Επίσης ορίζουμε ως τερματική κατάσταση για κάποια οντότητα την κατάσταση από την οποία δεν μπορεί να φύγει και μια νόμιμη εκτέλεση τερματίζει αν όλοι οι επιμέρους επεξεργαστές μπαίνουν σε τερματικές καταστάσεις και δεν υπάρχουν μεταβιβαζόμενα μηνύματα. Οι δύο αλγόριθμοι που θα μοντελοποιήσουμε ο LRB και ο URB δουλεύουν σε κατανεμημένο περιβάλλον όπου υπάρχουν σφάλματα κατάρρευσης των επεξεργαστών. Πιο συγκεκριμένα ο κάθε επεξεργαστής μπορεί να σταματήσει να παίρνει μέρος στον υπολογισμό χωρίς να μπορεί να επανέλθει στη συνέχεια. Επίσης στο σύστημα μπορούν να προκύψουν μέχρι και n σφάλματα κατάρρευσης όπου n ο αριθμός των οντοτήτων που απαρτίζουν το σύστημα. Για την ανίχνευση των σφαλμάτων που υπάρχουν στο σύστημα χρησιμοποιούμε τον ανιχνευτή σφαλμάτων perfect failure detector ο οποίος τρέχει παράλληλα με το σύστημα και αν συμβεί κάποιο σφάλμα σε κάποιο επεξεργαστή θα το

ανιχνεύσει σίγουρα μειώνοντας τον αριθμό των επιτρεπτών σφαλμάτων κατά ένα και βγάζοντας από τον υπολογισμό τον επεξεργαστή οποίος υπόπεσε σε σφάλμα. Αν ο ανιχνευτής μας πει ότι ένας επεξεργαστής κατάρρευσε τότε σίγουρα ο επεξεργαστής αυτός σταμάτησε να παίρνει μέρος στον υπολογισμό.

2.2 Εισαγωγή στην Άλγεβρα Διεργασιών CCS

Η γλώσσα CCS[1] (Calculus of Communicating Systems) αποτελεί μια γλώσσα μοντελοποίησης συστημάτων σε άλγεβρα διεργασιών που εμφανίστηκε την δεκαετία του 1980 από τον Robin Milner. Μέσω της γλώσσας μπορούμε να περιγράψουμε συστήματα που αποτελούνται από ένα σύνολο από συντρέχουσες οντότητες/διεργασίες οι οποίες μπορούν να επικοινωνούν μεταξύ τους μέσω καναλιών. Η CCS περιλαμβάνει ένα σύνολο από τελεστές για διατύπωση των πράξεων των οντοτήτων σε ένα σύστημα.

Η CCS συνοδεύεται από ένα σύνολο κανόνων λειτουργικής σημασιολογίας που δηλώνει όλες τις επιτρεπτές λειτουργίες που μπορούν να γίνουν σε ένα σύστημα. Συνδυάζοντας επιμέρους συστήματα μπορούμε να δημιουργήσουμε μεγαλύτερα συστήματα και μπορούμε να επαληθεύσουμε την ορθότητα κάποιου συστήματος δείχνοντας ότι ικανοποιεί κάποιες ιδιότητες όπως της αποφυγής αδιεξόδου, ότι δηλαδή ένα σύστημα σε κάθε πιθανή εκτέλεση θα τερματίζει. Επίσης υπάρχει η δυνατότητα να δείξουμε ότι ένα σύστημα εκτελεί τις σωστές λειτουργίες του δείχνοντας ότι το σύστημα έχει την ίδια συμπεριφορά με την προδιαγραφή του για όλο το σύνολο πιθανών εκτελέσεων, ότι δηλαδή προσομοιώνει την προδιαγραφή του μέσω της έννοιας της διπροσομοίωσης. Κάθε διεργασία μπορεί να αποτελείται από ένα σύνολο τελεστών που ανήκουν στην γλώσσα και τους οποίους μπορούμε να χρησιμοποιήσουμε για να συνθέσουμε πολλές διεργασίες μαζί ούτως ώστε να δημιουργήσουμε καινούριες διεργασίες όπου η συμπεριφορά τους θα εξαρτάται από τις επιμέρους. Τα κύρια χαρακτηριστικά μιας άλγεβρας διεργασιών είναι η σύνταξη της, η σημασιολογία της και οι σχέσεις ισοδυναμίας που την ορίζουν. Στη συνέχεια θα μελετήσουμε τις έννοιες αυτές όπως έχουν οριστεί στην CCS.

2.3 Σύνταξη και Λειτουργική Σημασιολογία της CCS

2.3.1 Σύνταξη CCS

Ορίζουμε ως ένα ως το σύνολο Λ το σύνολο των καναλιών τα οποία χρησιμοποιεί μια διεργασία για επικοινωνία τόσο εντός του συστήματος όσο και εξωτερικά με το περιβάλλον του. Ως Α ορίζουμε το σύνολο το οποίο περιέχει τα ονόματα των καναλιών τα οποία χρησιμοποιεί μια διεργασία για ενέργειες εισόδου, ως $\overline{A}$ το αντίστοιχο σύνολο το οποίο περιέχει τα κανάλια τα οποία χρησιμοποιούνται για ενέργειες εξόδου από μια διεργασία. Τέλος ορίζουμε το σύνολο Act ως το σύνολο το οποίο περιέχει όλες τις εξωτερικές ενέργειες που μπορεί να γίνουν από ένα σύστημα και οι οποίες ανήκουν στο σύνολο $Act=A \cup \overline{A} \cup \{\tau\}$ όπου τ, η εσωτερική ενέργεια που προκύπτει όταν δύο διεργασίες εκτελέσουν ταυτόχρονα είσοδο και έξοδο στο ίδιο κανάλι.

Αν $a \epsilon Act$, $L \subseteq \Lambda$, $C \in C$ όπου C ένα σύνολο από διεργασίες. Πιο κάτω δίνεται η σύνταξη της γλώσσας

E:=0	τερματισμός
| a.E	διαδοχή
| E_1+E_2	επιλογή
| E_1|E_2	ταυτοχρονισμός
| E\L	περιορισμός
| C	κλήση

Ο τελεστής του τερματισμού δηλώνει ότι μια διεργασία έφτασε στον τερματισμό της και δεν μπορεί να εκτελέσει άλλες ενέργειες, ο τελεστής της διαδοχής δηλώνει ότι η διεργασία αφού εκτελέσει την ενέργεια a πάνω σε κάποιο κανάλι εισόδου τότε μπορεί να εξελιχτεί στην διεργασία Ε. Ο τελεστής της επιλογής δηλώνει ότι μια διεργασία μπορεί είτε να συνεχίσει σύμφωνα με την διεργασία E_1 είτε σύμφωνα με την διεργασία E_2 και αυτό μπορεί να το πράξει μη ντετερμινιστικά. Ο τελεστής του ταυτοχρονισμού δηλώνει ότι δύο διεργασίες μπορούν να εκτελούνται ταυτόχρονα είτε εκτελώντας ξεχωριστές ενέργειες είτε μπορούν να συγχρονιστούν πάνω σε κάποιο κοινό κανάλι για να εκτελέσουν συμπληρωματικές ενέργειες εσωτερικά του συστήματος. Ο τελεστής του περιορισμού δηλώνει ότι μια διεργασία μπορεί να εκτελέσει εσωτερικές ενέργειες μόνο

πάνω στα κανάλια που είναι υπό περιορισμό εντός του συστήματος και τα οποία ανήκουν στο σύνολο L. Ο τελεστής κλήσης χρησιμοποιείται για την κλήση κάποιας διεργασίας που υπάρχει στο σύστημα και επιτρέπει την διατύπωση αναδρομικών διεργασιών.

Ο τελεστής διαδοχή έχει μεγαλύτερη προτεραιότητα από τον τελεστή της επιλογής και ο τελεστής της επιλογής έχει μεγαλύτερη προτεραιότητα από αυτόν του ταυτοχρονισμού.

2.3.2 Σημασιολογία της CCS

Η σημασιολογία ορίζεται ως η σχέση

$\rightarrow \subseteq$ Proc x Act x Proc

Ορίζουμε ως Proc το σύνολο των διεργασιών. Επίσης το <P, a, Q> $\in \rightarrow$ είναι ισοδύναμο με το "η διεργασία P μπορεί να εκτελέσει την ενέργεια α και να εξελιχθεί στην διεργασία Q". Γράφουμε $P \xrightarrow{\alpha} Q$ αντί <P, a, Q> $\in \rightarrow$

Ορίζουμε την σχέση → αναδρομικά ως ένα σύνολο από κανόνες της μορφής

$$\frac{συνθήκες\ μεταβάσεων}{συμπέρασμα}\ \ επιμέρους\ συνθήκη$$

οι οποίοι ερμηνεύονται ως "αν ισχύουν οι συνθήκες μεταβάσεων και η επιμέρους συνθήκη τότε θα πρέπει να ισχύει και το συμπέρασμα".

Οι κανόνες μεταβάσεων της CCS είναι η εξής:

Act $\quad \dfrac{-}{a.E \xrightarrow{a} E}$

Ο κανόνας αυτός δηλώνει πως αν εκτελεστεί η ενέργεια α στο επόμενο βήμα τότε η διεργασία θα συνεχίσει σύμφωνα με την διεργασία Ε. Μπορούμε να πούμε ότι ο πιο πάνω κανόνας αποτελεί αξίωμα αφού δεν περιέχει οποιεσδήποτε προ-συνθήκες.

$$\textbf{Sum}_1 \quad \frac{E \xrightarrow{a} E'}{E + F \xrightarrow{a} E'}$$

Ο κανόνας Sum_1 μας λέει ότι αν μια διεργασία Ε μπορεί να εκτελέσει την ενέργεια α και να εξελιχτεί σε Ε' τότε η επιλογή μεταξύ δύο διεργασιών Ε και F μπορεί πάλι να εκτελέσει την ενέργεια α και η επιλογή να συνεχίσει ως Ε'.

$$\textbf{Sum}_2 \quad \frac{F \xrightarrow{a} F'}{E + F \xrightarrow{a} F'}$$

Ο κανόνας Sum_2 αποτελεί συμπλήρωμα του κανόνα Sum_1 και μας λέει ότι αν στο επόμενο βήμα υπάρχει επιλογή μεταξύ δύο διεργασιών Ε και F και γίνει μια ενέργεια α από την F τότε η διεργασία Ε+F θα συνεχίσει σύμφωνα με την διεργασία F'.

$$\textbf{Com}_1 \quad \frac{E \xrightarrow{a} E'}{E \mid F \xrightarrow{a} E' \mid F}$$

Ο κανόνας Com_1 υποδηλώνει πως όταν μια διεργασία Ε μπορεί να εκτελέσει μια ενέργεια α και να εξελιχτεί στην Ε', αν η ίδια διεργασία τρέχει ταυτόχρονα με μια άλλη F και εκτελεστεί η ενέργεια α στην Ε αυτή η διεργασία θα συνεχίσει να εκτελείται ως Ε' και οι δύο διεργασίες θα συνεχίσουν να εκτελούνται παράλληλα.

$$\textbf{Com}_2 \quad \frac{F \xrightarrow{a} F'}{E \mid F \xrightarrow{a} E \mid F'}$$

Ο κανόνας Com_2 αποτελεί συμπλήρωμα του κανόνα com_1 και σε αυτή την περίπτωση αν γίνει η ενέργεια α στην F τότε θα εξελιχτεί σε F' και θα συνεχίσει να εκτελείται παράλληλα με την διεργασία E.

$$\textbf{Com}_3 \quad \frac{E \xrightarrow{a} E', F \xrightarrow{\bar{a}} F'}{E \mid F \xrightarrow{\tau} E' \mid F'}$$

Ο κανόνας Com_3 μας λέει ότι δύο διεργασίες μπορούν να συγχρονιστούν πάνω σε κάποιο κοινό κανάλι και η μία να εκτελέσει μια ενέργεια εξόδου και η άλλη ενέργεια εισόδου εκτελώντας μια εσωτερική ενέργεια τ μέσα σε ένα σύστημα. Έτσι οι δύο διεργασίες θα μεταβούν στις καινούριες τους καταστάσεις E' και F' αντίστοιχα και θα συνεχίσουν να εκτελούνται παράλληλα.

$$\textbf{Res} \quad \frac{E \xrightarrow{a} E'}{E \setminus L \xrightarrow{a} E' \setminus L} \quad a, \bar{a} \notin L$$

Ο κανόνας Res μας λέει ότι αν μια διεργασία μπορεί να εκτελέσει μια ενέργεια πάνω στο κανάλι α και να συνεχίσει ως E' τότε σημαίνει ότι και η ίδια διεργασία υπό περιορισμό μπορεί να εκτελέσει την ενέργεια α και να συνεχίσει ως E'.

$$\textbf{Con} \quad \frac{E \xrightarrow{a} E'}{C \xrightarrow{a} E'} \quad C \in \mathrm{C},\ C \overset{def}{=} E$$

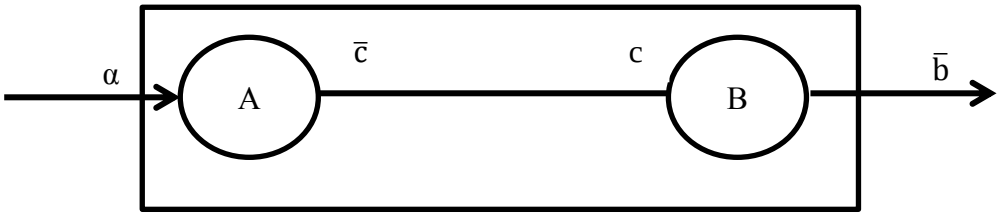
Ο κανόνας Con μας λέει ότι αν μια διεργασία E μπορεί να εκτελέσει μια ενέργεια α και να συνεχίσει ως E' και αν υπάρχει μια σταθερά C η οποία ορίζεται ως η διεργασία E τότε και η C μπορεί να εκτελέσει μια ενέργεια α και να συνεχίσει ως E'.

Παράδειγμα

Υποθέτουμε ότι έχουμε ένα σύστημα το οποίο αποτελείται από δύο οντότητες A και B σαν αποστολέας και παραλήπτης αντίστοιχα. Η διεργασία A παίρνει είσοδο πάνω στο

κανάλι α και μπορεί να εκτελέσει μια εσωτερική ενέργεια με την οντότητα Β πάνω στο κανάλι c εκτελώντας η Α ενέργεια εξόδου και η Β ενέργεια εισόδου. Η διεργασία Β μπορεί να εκτελέσει μια ενέργεια εξόδου στο περιβάλλον της πάνω στο κανάλι b.

Ορίζουμε το σύστημα το οποίο αποτελείται από δύο διεργασίες Α και Β ως εξής
a,b,c ∈ Act, c ∈ L



Οι ενέργειες α και b αποτελούν εξωτερικές ενέργειες για το πιο πάνω σύστημα και πιο συγκεκριμένα η πρώτη αποτελεί ενέργεια εισόδου και η δεύτερη ενέργεια εξόδου. Η ενέργεια c αποτελεί εσωτερική ενέργεια μεταξύ των δύο διεργασιών που αποτελούν το σύστημα.

<u>Μοντελοποίηση συστήματος στην CCS</u>

System=(A|B)\{c}

A=a.A’
A’=$\bar{c}$.A
B=c.B’
B’=$\bar{b}$.B

<u>Μεταβάσεις συστήματος</u>
(A|B)\{c}$\xrightarrow{a}$ (A’|B)\{c}$\xrightarrow{\tau}$ (A|B’)\{c}$\xrightarrow{\bar{b}}$ (A|B)\{c}

2.4 CCS με πέρασμα τιμών

Μπορούμε να επεκτείνουμε την CCS ούτως ώστε να έχουμε την δυνατότητα να στείλουμε τιμές μεταβλητών μεταξύ των οντοτήτων[1].

Στην σύναξη της γλώσσας διαμορφώνουμε τους κανόνες εισόδου και εξόδου ως εξής E= α(x).E | $\overline{\alpha}(e)$.E .

Ο κανόνας εισόδου μας λέει ότι η διεργασία α(x).E μπορεί να πάρει κάποιο δεδομένο έστω v στο κανάλι α και να συνεχίσει ως E αντικαθιστώντας στο E κάθε εμφάνιση της μεταβλητής x με v.

Ο κανόνας εξόδου $\overline{\alpha}(e)$.E ορίζεται ως ο κανόνας στον οποίο η διεργασία $\overline{\alpha}(e)$.E στέλνει στο κανάλι α την τιμή της έκφρασης e και συνεχίζει ως E.

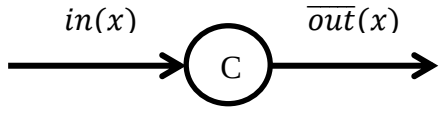
Ορίζουμε επίσης τους σημασιολογικούς κανόνες για τους πιο πάνω τελεστές ως εξής

Έστω Val(e) η τιμή της έκφρασης e

$$ActIn\ \frac{-}{\alpha(x).E \xrightarrow{a(v)} E\{^{v}/_{x}\}}$$

$$ActOut\ \frac{-}{\alpha(e).E \xrightarrow{\bar{a}(v)} E'},\ Val(e) = v$$

Παράδειγμα:



Ορίζουμε την διεργασία C ως την διεργασία η οποία παίρνει στην είσοδο της μια μεταβλητή x πάνω στο κανάλι in και ακολούθως την βγάζει στην έξοδο της στο κανάλι out.

Ορισμός διεργασίας C:

$C = in(x).C'(x)$

$C'(x) = \overline{out}(x).C$

Η διεργασία παίρνει ως είσοδο στο κανάλι in την μεταβλητή x και συνεχίζει σύμφωνα με τον ορισμό της διεργασίας C' η οποία βγάζει στην έξοδο την τιμή αυτή πάνω στο κανάλι out. Η εμβέλεια της μεταβλητής x είναι η διεργασία C.

2.5 Σχέσεις Ισοδυναμίας στην CCS

Για την προδιαγραφή και επαλήθευση ενός συστήματος σε άλγεβρα διεργασιών μπορεί να χρησιμοποιηθεί η ακόλουθη μέθοδος:

1. Ορίζουμε την επιθυμητή συμπεριφορά του συστήματος Spec.
2. Δημιουργούμε μια υποψήφια λύση Imp.
3. Ελέγχουμε κατά πόσο η υλοποίηση ικανοποιεί την προδιαγραφή δείχνοντας ότι οι Imp και Spec έχουν την ίδια συμπεριφορά.

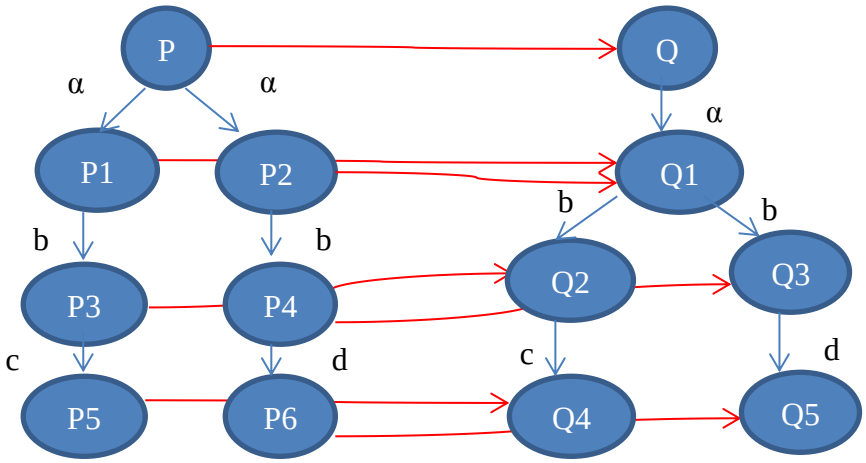
Για να εκφράσουμε "ίδια συμπεριφορά" έχουν οριστεί διάφορες έννοιες ισοδυναμίας. Στη συνέχεια παρουσιάζουμε τις πιο σημαντικές από αυτές στο πλαίσιο της CCS.

2.5.1 Ισοδυναμία Προσομοίωσης

Ορισμός 1: Μια σχέση $S \subseteq P \times P$ ονομάζεται ισοδυναμία προσομοίωσης[1] (simulation equivalence) αν για κάθε $(P,Q) \in S$, τότε για κάθε $\alpha \in Act$

Αν $P \xrightarrow{\alpha} P'$ τότε υπάρχει Q' τέτοιο ώστε $Q \xrightarrow{\alpha} Q'$ και $(P',Q') \in S$.

Παράδειγμα σχέσης ισοδυναμίας όπου η διεργασία Q προσομοιώνει την διεργασία P.



Σχήμα 2.1

S={(P,Q),(P1,Q1),(P2,Q1),(P3,Q2),(P4,Q3),(P5,Q4),(P6,Q5)}

Το αντίθετο δεν ισχύει για τον λόγο ότι στην Q υπάρχει μη-ντετερμινισμός στο σημείο όπου υπάρχει επιλογή ανάμεσα στο Q2 και Q3.

Η πιο πάνω σχέση δεν είναι κατάλληλη για απόδειξη ορθότητας στην περίπτωση των αλγορίθμων μας επειδή η πιο πάνω σχέση είναι μονής κατεύθυνσης ενώ στην περίπτωση μας χρειαζόμαστε μια πιο ισχυρή σχέση διπλής κατεύθυνσης. Πιο συγκεκριμένα το σύστημα μας πρέπει να είναι ισοδύναμο με την προδιαγραφή του και η προδιαγραφή να είναι ισοδύναμη με το σύστημα. Έτσι αυτό μας οδηγεί στο να ορίσουμε μια καινούρια σχέση πιο κάτω.

2.5.2 Ισχυρή διπροσομοίωση

Ορισμός 2: Μια σχέση S⊆PxP ονομάζεται ισχυρή διπροσομοίωση[1] αν για κάθε *(P,Q)*∈*S*, τότε για κάθε *α*∈*Act*

1. Αν $P \xrightarrow{\alpha} P'$ τότε υπάρχει Q' τέτοιο ώστε $Q \xrightarrow{\alpha} Q'$ και $(P',Q') \in S$
2. Αν $Q \xrightarrow{\alpha} Q'$ τότε υπάρχει P' τέτοιο ώστε $P \xrightarrow{\alpha} P'$ και $(P',Q') \in S$

Αν δύο διεργασίες P και Q σχετίζονται από μια σχέση ισχυρής διπροσομοίωσης τότε γράφουμε P~Q.

Ορίζουμε ως α-παράγωγο μιας διεργασίας οποιαδήποτε κατάσταση μπορεί να μεταβεί η διεργασία P όταν εκτελέσει την ενέργεια α∈Act.

Γενικά για να μπορούμε να πούμε ότι δύο διεργασίες συνδέονται με σχέση ισχυρής διπροσομοίωσης τότε θα πρέπει να ισχύει ότι για κάθε α-παράγωγο του P υπάρχει ισοδύναμο α-παράγωγο του Q και αντίστροφα. Άρα σε κάθε πιθανή μετάβαση στο ένα σύστημα θα πρέπει να μεταβαίνουμε στην αντίστοιχη κατάσταση στο άλλο και το αντίθετο ακόμα και αν εκτελούνται εσωτερικές ενέργειες στο σύστημα οι οποίες πρέπει να έχουν αντιστοιχία μια προς μία μεταξύ των δύο συστημάτων όπως και οι υπόλοιπες ενέργειες. Η ισοδυναμία αυτή επίσης δεν είναι κατάλληλη για την απόδειξη ορθότητας στην περίπτωση των αλγορίθμων μας γιατί είναι πολύ περιοριστική όσο αφορά τις εσωτερικές ενέργειες. Για να πούμε ότι ένα σύστημα είναι ισοδύναμο με την προδιαγραφή του τότε πρέπει να έχουν την ίδια εξωτερική συμπεριφορά με το περιβάλλον τους ανεξαρτήτως οποιοδήποτε εσωτερικών ενεργειών μπορούν να γίνουν εσωτερικά του συστήματος. Έτσι αυτό μας οδηγεί στην ανάγκη να ορίσουμε μια πιο χαλαρή μορφή ισοδυναμίας όσο αφορά τις εσωτερικές ενέργειες.

2.5.3 Ασθενής Διπροσομοίωση

Ορισμός 3: Μια σχέση S⊆PxP ονομάζεται ασθενής διπροσομοίωση[1] αν για κάθε *(P,Q)*∈*S*, τότε για κάθε *α*∈*Act*

1. Αν $P \xrightarrow{\alpha} P'$ τότε υπάρχει *Q'* τέτοιο ώστε $Q \overset{\hat{\alpha}}{\Rightarrow} Q'$ και *(P',Q')* ∈*S*
2. Αν $Q \xrightarrow{\alpha} Q'$ τότε υπάρχει *P'* τέτοιο ώστε $P \overset{\hat{\alpha}}{\Rightarrow} P'$ και *(P',Q')* ∈*S*

Ορίζουμε το $\hat{\alpha}$ ως μια ακολουθία ενεργειών οι οποίες περιέχουν 0 ή περισσότερες εμφανίσεις της εσωτερικής ενέργειας τ και μπορεί να περιέχει και την ενέργεια α. Δηλαδή μπορούμε να πούμε ότι $\hat{\alpha} = \varepsilon$ εάν $\alpha=\tau$ διαφορετικά $\hat{\alpha} = \alpha$.

Για να απορροφήσουμε τις εσωτερικές ενέργειες ορίζουμε την πιο κάτω σχέση

$P \overset{\varepsilon}{\Rightarrow} Q \quad iff \quad P \overset{\tau}{\rightarrow} \ldots \overset{\tau}{\rightarrow} Q, \geq 0$

$P \overset{a}{\Rightarrow} Q \quad iff \quad P \overset{\varepsilon}{\Rightarrow} P' \overset{\alpha}{\rightarrow} Q' \overset{\varepsilon}{\Rightarrow} Q$

$\hat{\alpha} = \varepsilon \; εαν \; \alpha = \tau \; διαφορετικά \; \hat{\alpha} = \alpha$

Αν δύο διεργασίες P και Q σχετίζονται από μια σχέση ασθενής διπροσομοίωσης τότε γράφουμε P≈Q.

Στην περίπτωση της ασθενούς διπροσομοίωσης δεν μας ενδιαφέρουν οι εσωτερικές ενέργειες που εκτελούνται σε ένα σύστημα φτάνει δύο συστήματα να έχουν την ίδια εξωτερική συμπεριφορά με το περιβάλλον τους και πιο συγκεκριμένα μια εσωτερική ενέργεια στο σύστημα P μπορεί να ταυτιστεί με καμιά ή με πολλές εσωτερικές ενέργειες στο σύστημα Q και το αντίστροφο. Έτσι κάθε εξωτερική ενέργεια μπορεί να προηγηθεί και να ακολουθηθεί από 0 ή περισσότερες εσωτερικές ενέργειες.

2.6 Συρροή

Η συρροή[4] αποτελεί μια μορφή ντετερμινισμού και η οποία ορίζεται ως εξής:
Μια διεργασία P μπορούμε να πούμε ότι είναι ντετερμινιστική αν κάθε φορά που εκτελούμε το ίδιο σύνολο ενεργειών σε αυτή τότε οδηγούμαστε σε ισοδύναμες καταστάσεις.

Μια διεργασία P στη CCS είναι συρρέουσα εάν η διεργασία είναι ντετερμινιστική και για κάθε παραγόμενη διεργασία Q και μοναδικές ενέργειες α,b οι οποίες δεν ανήκουν στο ίδιο κανάλι τότε αν $Q \overset{a}{\rightarrow} Q_1$ και $Q \overset{b}{\rightarrow} Q_2$ τότε θα πρέπει να υπάρχουν διεργασίες Q_1' και Q_2' τέτοιες ώστε $Q_2 \overset{\hat{\alpha}}{\Rightarrow} Q_2$', $Q_1 \overset{\hat{b}}{\Rightarrow} Q_1$' και $Q_1' \approx Q_2'$.

Σύμφωνα με τον πιο πάνω ορισμό αν η διεργασία είναι σε θέση να εκτελέσει μια ενέργεια α τότε ακολούθως θα μπορεί να εκτελέσει την ενέργεια b συνοδευόμενη από ένα σύνολο εσωτερικών ενεργειών ή και το αντίθετο και πάντα θα μεταβαίνει σε ισοδύναμες καταστάσεις όπου θα μπορεί να εκτελεί το ίδιο σύνολο ενεργειών. Έτσι δεν αποκλείονται οποιεσδήποτε ενέργειες από το να εκτελεστούν.

2.7 Συμπεράσματα

Η ανάγκη για μοντελοποίηση των αλγορίθμων μας στην παρουσία σφαλμάτων μας οδήγησε στην επέκταση της άλγεβρας CCS για τον χειρισμό των σφαλμάτων. Επιπρόσθετα για την απόδειξη ορθότητας των αλγορίθμων ορίσαμε 3 ειδών σχέσεις ισοδυναμίας και λόγω κάποιων περιορισμών στις δύο πρώτες επιλέξαμε την σχέση ασθενής διπροσομοίωσης για να δείξουμε ότι το σύστημα είναι ισοδύναμο με την προδιαγραφή του. Καταλυτική θα είναι και η έννοια της συρροής για τις σχετικές αποδείξεις ορθότητας των δύο αλγορίθμων.

Κεφάλαιο 3

Άλγεβρα Διεργασιών Dπf

3.1 Εισαγωγή στην Dπf

Έχοντας ως στόχο την μοντελοποίηση κατανεμημένων αλγορίθμων στην παρουσία σφαλμάτων παρατηρήσαμε ότι η άλγεβρα CCS δεν μας παρέχει όλα τα απαραίτητα εφόδια γι' αυτό και έτσι καταλήξαμε στο συμπέρασμα ότι υπάρχει η ανάγκη για δημιουργία ενός νέου πρότυπου το οποίο ονομάσαμε Dπf-calculus και το οποίο αποτελεί επέκταση της γλώσσας Dπi-calculus (partial failure calculus of F.H.)[3] η οποία δημιουργήθηκε από τους Adrian Francalanza και Matthew Hennessy στην οποία προσθέσαμε κάποια επιπλέον χαρακτηριστικά. Στην Dπi-calculus υπάρχουν γενικές μεταβλητές σαν παράμετροι του συστήματος και επίσης υπάρχουν κανόνες για τον εντοπισμό των σφαλμάτων που προκύπτουν στο σύστημα. Κάθε διεργασία ανήκει σε μια μοναδική τοποθεσία και κάθε φορά που θέλουμε να ελέγξουμε την κατάσταση ενός επεξεργαστή ελέγχουμε την τοποθεσία του.

Στην Dπf μπορούμε να έχουμε τοπικές μεταβλητές σε κάθε διεργασία οι οποίες μπορούν να ανανεώνονται κάθε φορά που η διεργασία επιστρέφει στην αρχική της κατάσταση. Οι μεταβλητές αυτές μπορεί να είναι ή απλές μεταβλητές ή λίστες. Επίσης υπάρχει η δυνατότητα να υπάρχουν και γενικές μεταβλητές για κάποιο σύστημα στις οποίες μπορούν να έχουν πρόσβαση όλες οι επιμέρους οντότητες που απαρτίζουν το

σύστημα. Επιπρόσθετα μπορούμε να επεκτείνουμε την γλώσσα ccs έτσι ώστε τα βασικά κανάλια επικοινωνίας μεταξύ των διεργασιών να υποστηρίζουν την αποστολή τιμών μεταξύ των διεργασιών που στην περίπτωση των αλγορίθμων που μοντελοποιήσαμε οι διεργασίες μεταξύ τους θα μπορούν να στέλλουν μηνύματα μεταξύ τους. Σε αυτή την άλγεβρα έχουμε την δυνατότητα να μοντελοποιήσουμε την συνθήκη if-else if μεταξύ ενός συνόλου συνθηκών. Και εδώ υποθέτουμε ότι κάθε διεργασία βρίσκεται σε μια μοναδική τοποθεσία l.

3.2 Σύνταξη και λειτουργική σημασιολογία της Dπf

Όπως έχουμε ήδη αναφέρει η άλγεβρα διεργασιών Dπf αποτελεί επέκταση του partial-failure calculus όπου ορίζουμε το σύνολο Act ως το σύνολο που περιέχει όλες τις δράσεις οι οποίες μπορούν να γίνουν στο σύστημα. Για κάθε ενέργεια α που ανήκει στο πιο πάνω σύνολο έχουμε την συμπληρωματική της $\overline{\alpha}$. Συμβολίζουμε τις εσωτερικές ενέργειες που μπορούν να προκύψουν εσωτερικά του συστήματος μεταξύ δύο συμπληρωματικών ενεργειών με τ. Επίσης ορίζουμε το σύνολο Locs ως το σύνολο το οποίο περιέχει τις τοποθεσίες όλων των επεξεργαστών και το οποίο περιέχει την αθάνατη τοποθεσία ★ στην οποία ανήκουν οι οντότητες οι οποίες δεν μπορούν να υποθέσουν σε σφάλμα.

Σύνταξη Dπf

Η σύνταξη της Dπf δίνεται σε δύο επίπεδα, το επίπεδο των διεργασιών και το επίπεδο των συστημάτων ως εξής

Διεργασίες

$$\mathrm{P}, \mathrm{Q} := 0 \mid a(\tilde{x}).P \mid \bar{a}(\tilde{x}).P \mid P + Q \mid P\backslash L \mid P|Q \mid susp\ k\colon (P;Q) \mid$$
$$cond(e_1 \rhd P_1, \dots, e_n \rhd P_n) \mid \mathrm{C}\langle \tilde{v} \rangle$$

Συστήματα

$$M, N := l[P] \mid N|M \mid N\backslash L$$

Ακολουθεί εξήγηση των δυνατών τύπων διεργασιών στο επίπεδο των διεργασιών:

Η διεργασία 0 δηλώνει την τερματική διεργασία στην οποία φτάνει κάθε διεργασία με το πέρας της. Το σύνολο Act περιέχει το σύνολο των ενεργειών εισόδου της μορφής $\alpha(\tilde{v})$. Περιέχει επίσης τις αντίστοιχες πράξεις εξόδου του τύπου $\bar{a}(\tilde{v})$ και τέλος όλες τις εσωτερικές ενέργειες που μπορούν να προκύψουν όταν δύο ενέργειες εισόδου και εξόδου εκτελεστούν συγχρονισμένα πάνω στο κοινό τους κανάλι. Η διεργασία $a(\tilde{x}).P$ δηλώνει ότι αν στο επόμενο βήμα εκτελεστεί μια ενέργεια εισόδου a με είσοδο $\tilde{x}$ τότε η διεργασία θα συνεχίσει να εκτελείται σύμφωνα με τον ορισμό της διεργασίας P. Η πλειάδα $\tilde{x}$ ορίζεται ως ένα σύνολο από μεταβλητές που στέλλονται σαν είσοδο στο κανάλι α. Αντίστοιχα μπορεί να εκτελεστεί μια ενέργεια εξόδου $\bar{a}$ με έξοδο $\tilde{x}$ και η διεργασία να συνεχίσει ως P . Η διεργασία $P + Q$ δηλώνει ότι υπάρχει μη ντετερμινιστική επιλογή μεταξύ δύο διεργασιών P, Q . Η διεργασία $P\backslash L$ ορίζει το σύνολο των καναλιών L τα οποία είναι δεσμευμένα για εκτέλεση ενεργειών εντός του συστήματος P μεταξύ των διεργασιών και τα οποία έχουν εμβέλεια την P. Τα κανάλια που είναι ελεύθερα μπορούν να επικοινωνούν με το εξωτερικό περιβάλλον της διεργασίας P. Η διεργασία $P|Q$ ορίζεται ως η διεργασία στην οποία οι διεργασίες P και Q εκτελούνται παράλληλα και μπορούν να εκτελέσουν ανεξάρτητες ενέργειες και να συνεχίσουν να εκτελούνται παράλληλα ή να συγχρονιστούν πάνω σε κάποιο κοινό κανάλι και να εκτελέσουν μια κοινή ενέργεια. Η διεργασία $susp\ k.P$ είναι η διεργασία στην οποία γίνεται έλεγχος για τυχόν σφάλμα μιας διεργασίας ελέγχοντας τη τοποθεσία της διεργασίας αυτής (k) και αν εντοπίσουμε σφάλμα τότε συνεχίζουμε αναλόγως της διεργασίας P. Η διεργασία $cond(e_1 \rhd P_1, \dots, e_n \rhd P_n)$ μας λέει ότι υπάρχει μια υπό συνθήκη επιλογή για το πώς θα συνεχίσει μια διεργασία και η επιλογή γίνεται επιλέγοντας την διεργασία P_ι η οποία αντιστοιχεί στην πρώτη συνθήκη e_ι η οποία είναι αληθής από αριστερά προς τα δεξιά. Μια συνθήκη είναι μια πρόταση του προτασιακού λογισμού όπου οι ατομικές προτάσεις αναφέρονται στις παραμέτρους του συστήματος που εμφανίζονται ως $\tilde{v}$ στις σταθερές διεργασιών $C\langle\tilde{v}\rangle$. Η διεργασία $C\langle\tilde{v}\rangle$ ορίζεται ως η διεργασία στην οποία μπορούμε να έχουμε μια πλειάδα από παραμέτρους.

Ως προς το επίπεδο των συστημάτων:

Ως $l[P]$ ορίζουμε το σύστημα το οποίο περιέχει την διεργασία P η οποία ανήκει στην τοποθεσία l. Επίσης ως $N|M$ ορίζουμε το σύστημα το σύστημα το οποίο αποτελείται από δύο επιμέρους συστήματα τα οποία τρέχουν παράλληλα μεταξύ τους. Ως $N\backslash L$ ορίζουμε το σύστημα στο οποίο τα κανάλια που ανήκουν στο σύνολο L είναι υπό περιορισμό εσωτερικά του συστήματος.

Μπορούμε να δηλώσουμε ένα σύνολο από επιλογές μεταξύ των διεργασιών ως $P_1,P_2,P_3,\ldots,P_n$ ως $\sum_{\iota\epsilon I} P\iota$ για I=[1,..,n] και ένα σύνολο από διεργασίες $P_1,P_2,P_3,\ldots,P_n$ οι οποίες εκτελούνται παράλληλα ως $\prod_{\iota\epsilon I} P\iota$ για I=[1,..,n].

Ορίζουμε ως παραμέτρους του συστήματος μας τα L και n όπου L το σύνολο των ζωντανών τοποθεσιών ($l_1,\ldots, l_n$) τα οποία υπάρχουν στο σύστημα. Ως n ορίζουμε το μέγιστο αριθμό σφαλμάτων στα οποία μπορεί να υποπέσει το σύστημα μας. Οι δύο αυτές παράμετροι αποτελούν γενικές μεταβλητές στο σύστημα και όλες οι οντότητες μπορούν να έχουν πρόσβαση σε αυτές. Οι αλλαγές στις παραμέτρους αυτές γίνονται δυναμικά με την παρουσία κάποιου σφάλματος κάποιας οντότητας. Επίσης ορίζουμε ως φάση $\langle L, n\rangle \triangleright$M το σύστημα M στο οποίο οι ζωντανοί επεξεργαστές που το αποτελούν βρίσκονται στο σύνολο L και μπορούν να προκύψουν το πολύ n σφάλματα εσωτερικά του συστήματος.

<u>Σημασιολογικοί κανόνες Dπf</u>

Υποθέτουμε στους πιο κάτω κανόνες ότι $l\epsilon L$ και $n \geq 0$. Σε περίπτωση που υπάρχουν προ-συνθήκες ή επιμέρους συνθήκες σε κάποιο συμπέρασμα τότε αυτές είναι απαραίτητες να ισχύουν για να ισχύει και το συμπέρασμα ενός κανόνα.

(In)

$$\frac{}{\langle L, n\rangle \triangleright l[a(\tilde{x}).P] \xrightarrow{a(\tilde{v})} l[P]\left\{{}^{\tilde{v}}/_{\tilde{x}}\right\}}$$

(Out)

$$\frac{}{\langle L, n\rangle \rhd l[\bar{a}(\tilde{x}).P] \xrightarrow{\bar{a}(\tilde{v})} l[P]}$$

(Susp)

$$\frac{}{\langle L, n\rangle \rhd l[susp\ k\colon (P;Q)] \xrightarrow{\tau} l[P], k \notin L}$$
$$\langle L, n\rangle \rhd l[susp\ k\colon (P;Q)] \xrightarrow{\tau} l[Q], k \in L$$

(Halt)

$$\frac{}{\langle L, n+1\rangle \rhd M \xrightarrow{\tau} \langle L - l, n\rangle \rhd M}$$

(Sum)

$$\frac{\langle L, n\rangle \rhd l[P_i] \xrightarrow{a} l[P]}{\langle L, n\rangle \rhd l[\sum_{i\epsilon I} P_i] \xrightarrow{a} l[P]}$$

(Rest)

$$\frac{\langle L, n\rangle \rhd M \xrightarrow{a} \langle L', n'\rangle \rhd M'}{\langle L, n\rangle \rhd M\backslash L \xrightarrow{a} \langle L', n'\rangle \rhd M'\backslash L} \quad a \notin L$$

(Par)

$$\frac{\langle L, n\rangle \rhd M \xrightarrow{a} \langle L', n'\rangle \rhd M'}{\langle L, n\rangle \rhd M|N \xrightarrow{a} \langle L', n'\rangle \rhd M'|N}$$
$$\langle L, n\rangle \rhd N|M \xrightarrow{a} \langle L', n'\rangle \rhd N|M'$$

(Com)

$$\frac{\langle L,n\rangle \rhd M \xrightarrow{a} M' \qquad \langle L,n\rangle \rhd N \xrightarrow{\bar{a}} N'}{\begin{array}{c}\langle L,n\rangle \rhd M|N \xrightarrow{\tau} M'|N' \\ \langle L,n\rangle \rhd N|M \xrightarrow{\tau} N'|M'\end{array}}$$

(Fork)

$$\frac{}{\langle L,n\rangle \rhd l[P|Q] \xrightarrow{\tau} l[P]|\, l[Q]}$$

(New)

$$\frac{}{\langle L,n\rangle \rhd l[P\backslash L] \xrightarrow{\tau} l[P]\backslash L}$$

(Cond)

$$\frac{\langle L,n\rangle \rhd l[P_i] \xrightarrow{a} \langle L',n'\rangle \rhd l[P_i{}']}{\langle L,n\rangle \rhd l[cond(e_1 \Rightarrow l_1[P_1], e_2 \Rightarrow l_2[P_2], \ldots, e_n \Rightarrow l_n[P_n])] \xrightarrow{a} \langle L',n'\rangle \rhd l_i[P_i{}']} \quad e_i = true, \forall j < i, e_j = false$$

(Const)

$$\frac{\langle L,n\rangle \rhd l[P\{\tilde{v}/\tilde{x}\}] \xrightarrow{a} \langle L',n'\rangle \rhd l[P']}{\langle L,n\rangle \rhd l[C(\tilde{v})] \xrightarrow{a} \langle L',n'\rangle \rhd l[P']} \quad C(\tilde{x}) \stackrel{\text{def}}{=} P$$

Ο κανόνας In μας λέει ότι αν εκτελεστεί μια ενέργεια εισόδου σε ένα κανάλι α με είσοδο την πλειάδα $\tilde{v}$ τότε το σύστημα θα συνεχίσει να εκτελείται σύμφωνα με την διεργασία P με όλες τις εμφανίσεις των μεταβλητών που ανήκουν στην πλειάδα $\tilde{x}$ να αντικαθιστούνται με αντίστοιχες σταθερές από την πλειάδα $\tilde{v}$. Ο κανόνας Out δηλώνει ότι η διεργασία $l[\bar{a}(\tilde{x}).P]$ μπορεί να στείλει πάνω στο κανάλι εξόδου την πλειάδα $\tilde{v}$ και να συνεχίσει σύμφωνα με την διεργασία P. Ο κανόνας Susp αν εντοπίσει σφάλμα της διεργασίας που βρίσκεται στη τοποθεσία k τότε θα συνεχίσει να εκτελείται ως η διεργασία P διαφορετικά θα συνεχίσει να εκτελείται σύμφωνα με το Q. Ο κανόνας Halt

είναι ο κανόνας ο οποίος βγάζει από το σύνολο *L* τη τοποθεσία της διεργασίας η οποία υπόπεσε σε σφάλμα και μειώνει τον αριθμό των επιτρεπτών σφαλμάτων κατά ένα. Ο κανόνας Sum μας λέει ότι αν σε ένα σύστημα υπάρχει επιλογή μεταξύ ενός συνόλου διεργασιών για το πώς θα συνεχίσει η λειτουργία του συστήματος μας τότε αυτό θα γίνει μη-ντετερμινιστικά. Ο κανόνας Rest μας λέει ότι αν ένα σύστημα M μπορεί να εκτελέσει μια ενέργεια α πάνω σε κάποιο κανάλι και να συνεχίσει ως M' τότε το ίδιο σύστημα υπό περιορισμό μπορεί να εκτελέσει την ενέργεια α και να συνεχίσει ως M'. Ο κανόνας Par είναι ο κανόνας που εκφράζει ότι αν ένα σύστημα M μπορεί να εκτελέσει μια ενέργεια α και να συνεχίσει ως M' και το ίδιο σύστημα τρέχει παράλληλα με ένα άλλο N τότε μπορεί να εκτελέσει την ενέργεια α και να συνεχίσει να εκτελείται παράλληλα με το σύστημα N. Ακολούθως ο κανόνας Com δηλώνει ότι δύο συστήματα που τρέχουν παράλληλα μπορούν να συγχρονιστούν εκτελώντας το ένα ενέργεια εισόδου και το άλλο ενέργεια εξόδου μεταξύ τους πάνω σε κάποιο κοινό τους κανάλι. Ο κανόνας Fork μας εγγυάται ότι δύο διαφορετικές διεργασίες ενός συστήματος μπορούν να τρέχουν ταυτόχρονα και να ανήκουν στην ίδια τοποθεσία. Ο κανόνας Cond είναι ο κανόνας που μας λέει ότι αν υπάρχει υπό συνθήκη επιλογή μεταξύ ενός συνόλου διεργασιών τότε επιλέγεται η διεργασία η οποία αντιστοιχεί στην πρώτη συνθήκη που είναι αληθής από αριστερά προς τα δεξιά. Ο κανόνας New ορίζεται ως ο κανόνας όπου μπορούμε να μεταφέρουμε τον περιορισμό από το επίπεδο της διεργασίας στο επίπεδο του συστήματος. Τέλος ο κανόνας Const μας λέει ότι αν η $C(\tilde{x})$ ορίζεται ως P τότε η $C(\tilde{v})$ μπορεί να κάνει ότι και η $P\{^{\tilde{v}}/_{\tilde{x}}\}$.

3.3 Σχέσεις Ισοδυναμίας στην Dπf

Ορίζουμε ως σχέση ισοδυναμίας μεταξύ δύο διεργασιών την σχέση η οποία περιέχει όλες τις αντίστοιχες δυάδες καταστάσεων που παράγονται από τις δύο διεργασίες σε κάθε πιθανή μετάβαση. Η σχέση αυτή μπορεί να είναι μονομερής ή διμερής αναλόγως της σχέσης ισοδυναμίας που συνδέει δύο διεργασίες μεταξύ τους. Αν δύο διεργασίες συνδέονται με σχέση ισοδυναμίας μπορούμε να πούμε ότι έχουν την ίδια εξωτερική συμπεριφορά σε κάθε πιθανή εκτέλεση. Χρησιμοποιώντας τις σχέσεις ισοδυναμίας μπορούμε να δείξουμε ότι ένα σύστημα έχει την επιθυμητή συμπεριφορά δείχνοντας ότι είναι ισοδύναμο με την προδιαγραφή του και το αντίθετο.

Ορισμός 4: Ορίζουμε στην Dπf την σχέση ισχυρής διπροσομοίωσης[3] μεταξύ δύο συστημάτων $\langle L_1, n_1\rangle \triangleright P$ και $\langle L_2, n_2\rangle \triangleright Q$ ως εξής: Αν $\langle L_1, n_1\rangle \triangleright P \sim \langle L_2, n_2\rangle \triangleright Q$ και

(1) $\langle L_1, n_1\rangle \triangleright P \xrightarrow{a} \langle L_1', n_1'\rangle \triangleright P'$ τότε θα πρέπει να ισχύει ότι

$\langle L_2, n_2\rangle \triangleright Q \xrightarrow{\alpha} \langle L_2', n_2'\rangle \triangleright Q'$ έτσι ώστε $\langle L_1', n_1'\rangle \triangleright P' \sim \langle L_2', n_2'\rangle \triangleright Q'$

και

(2) αν $\langle L_2, n_2\rangle \triangleright Q \xrightarrow{\alpha} \langle L_2', n_2'\rangle \triangleright Q'$ τότε θα πρέπει να ισχύει ότι

$\langle L_1, n_1\rangle \triangleright P \xrightarrow{\alpha} \langle L_1', n_1'\rangle \triangleright P'$ έτσι ώστε $\langle L_1', n_1'\rangle \triangleright P' \sim \langle L_2', n_2'\rangle \triangleright Q'$.

Ορισμός 5: Ορίζουμε στην Dπf την σχέση ασθενής διπροσομοίωσης[3] μεταξύ δύο συστημάτων $\langle L_1, n_1\rangle \triangleright P$ και $\langle L_2, n_2\rangle \triangleright Q$ ως εξής: Αν $\langle L_1, n_1\rangle \triangleright P \approx \langle L_2, n_2\rangle \triangleright Q$

και (1)$\langle L_1, n_1\rangle \triangleright P \xrightarrow{a} \langle L_1', n_1'\rangle \triangleright P'$ τότε θα πρέπει να ισχύει ότι

$\langle L_2, n_2\rangle \triangleright Q \overset{\hat{\alpha}}{\Rightarrow} \langle L_2', n_2'\rangle \triangleright Q'$ έτσι ώστε $\langle L_1', n_1'\rangle \triangleright P' \approx \langle L_2', n_2'\rangle \triangleright Q'$

και

(2) αν $\langle L_2, n_2\rangle \triangleright Q \xrightarrow{\alpha} \langle L_2', n_2'\rangle \triangleright Q'$ τότε θα πρέπει να ισχύει ότι

$\langle L_1, n_1\rangle \triangleright P \overset{\hat{\alpha}}{\Rightarrow} \langle L_1', n_1'\rangle \triangleright P'$ έτσι ώστε $\langle L_1', n_1'\rangle \triangleright P' \approx \langle L_2', n_2'\rangle \triangleright Q'$.

Οι δύο πιο πάνω σχέσεις ισχυρής και ασθενούς διπροσομοίωσης μπορούμε να πούμε ότι είναι οι μεγαλύτερες σχέσεις μεταξύ των δύο συστημάτων $\langle L_1, n_1\rangle \triangleright P$ και $\langle L_2, n_2\rangle \triangleright Q$. Δεν υπάρχει οποιαδήποτε σχέση η οποία να είναι υπερσύνολο αυτών.

Ορισμός 6: Μπορούμε να πούμε ότι ένα σύστημα είναι ανεκτικό σε n σφάλματα αν ισχύει η πιο κάτω συνθήκη

$\langle L, n\rangle \triangleright P \approx \langle L, 0\rangle \triangleright P$

3.4 Ντετερμινιστικές διεργασίες

Σε αυτό εδώ το κεφάλαιο ορίζουμε την έννοια της ντετερμινιστικής διεργασίας [1] ούτως ώστε να αποδείξουμε ότι κάθε διεργασία που χρησιμοποιούμε στο σύστημα μας πληροί τον πιο πάνω ορισμό για να μπορούμε να πούμε ότι το σύστημα μας έχει ντετερμινιστική συμπεριφορά. Εδώ αξίζει να σημειωθεί ότι στην παρουσία σφαλμάτων εσωτερικά του συστήματος δεν μπορούμε να πούμε ότι το σύστημα έχει ντετερμινιστική συμπεριφορά γι' αυτό χρησιμοποιούμε τον πιο κάτω ορισμό νοουμένου ότι στο σύστημα δεν μπορούν να προκύψουν άλλα σφάλματα.

Ορισμός 7: Μια διεργασία $\langle L, n\rangle \rhd l[P]$ είναι ντετερμινιστική αν κάθε διεργασία $\langle L', n'\rangle \rhd l[Q]$ που παράγεται από την διεργασία $\langle L, n\rangle \rhd l[P]$ και για όλες τις ενέργειες $\alpha \in Act$ κάθε φορά που $\langle L', n'\rangle \rhd l[Q] \xrightarrow{a} \langle L'', n''\rangle \rhd l[Q'']$ και $\langle L', n'\rangle \rhd l[Q] \stackrel{\widehat{\alpha}}{\Rightarrow} \langle L'', n''\rangle \rhd l[Q'']$ τότε θα πρέπει να ισχύει ότι $\langle L'', n''\rangle \rhd l[Q'] \approx \langle L'', n''\rangle \rhd l[Q'']$.

Σύμφωνα με τον πιο πάνω ορισμό μια διεργασία είναι ντετερμινιστική μόνο αν κάθε φορά που εκτελούμε την ίδια ενέργεια, ισχυρά ή ασθενώς σε μια διεργασία P πρέπει να μας οδηγεί πάντα σε ισοδύναμες καταστάσεις.

Πρόταση 1: Εάν η $\langle L, n\rangle \rhd l[P]$ είναι ντετερμινιστική και $\langle L, n\rangle \rhd P \xrightarrow{\tau} \langle L, n\rangle \rhd l[P']$ τότε και η $\langle L, n\rangle \rhd l[P']$ είναι ντετερμινιστική.[1]

Πρόταση 2: Εάν η $\langle L, n\rangle \rhd l[P]$ είναι ντετερμινιστική και $\langle L, n\rangle \rhd l[P] \approx \langle L, n\rangle \rhd l[Q]$ τότε και η διεργασία $\langle L, n\rangle \rhd l[Q]$ είναι ντετερμινιστική.[1]

Ορίζουμε ως $L(P)$ όλα τα κανάλια που αποτελούν ως κανάλια εισόδου της διεργασίας P και ως $\bar{L}(P)$ όλα τα κανάλια που αποτελούν κανάλια εξόδου στην διεργασία P.
Θεωρούμε ότι $L(P_1) \cap L(P_2) = \emptyset$, $\bar{L}(P_1) \cap \bar{L}(P_2) = \emptyset$ και $L(P_1) \cap \bar{L}(P_2) = \emptyset$

Πρόταση 3

1. Η διεργασία 0 είναι ντετερμινιστική.
2. Εάν η διεργασία P είναι ντετερμινιστική τότε και οι διεργασίες $l[a.P]$ και $l[\overline{\alpha}.P]$ είναι ντετερμινιστικές.
3. Εάν η διεργασία P είναι ντετερμινιστική τότε και η διεργασία $l[P]\backslash F$ θα είναι.
4. Εάν για κάθε $i \in I$ κάθε διεργασία $l[P_i]$ είναι ντετερμινιστική και κάθε α_ι είναι μοναδικό κανάλι τότε και η διεργασία $l[\sum_{\iota\in I} \alpha_\iota(\widetilde{x_\iota}).P_i]$.
5. Εάν για κάθε $i \in I$ κάθε διεργασία $l_i[P_i]$ είναι ντετερμινιστική τότε και η διεργασία $\mathrm{cond}(e_1 \triangleright l_1[P_1], \ldots, e_n \triangleright l_n[P_n])$ θα είναι επίσης.
6. Εάν η διεργασίες l[P] και l[Q] είναι ντετερμινιστικές τότε και η διεργασία l[susp k:(P;Q)] θα είναι ντετερμινιστική.
7. Εάν η διεργασία l[P] είναι ντετερμινιστική και $l[C(\tilde{v})]$=l[P] τότε και η διεργασία $l[C(\tilde{v})]$ θα είναι ντετερμινιστική.
8. Εάν οι διεργασίες $l[P_1]$ και $l[P_2]$ είναι ντετερμινιστικές και $L(P_1) \cap L(P_2) = \emptyset$, $\bar{L}(P_1) \cap \bar{L}(P_2) = \emptyset$ και $L(P_1) \cap \bar{L}(P_2) = \emptyset$ τότε και η διεργασία $l[P_1||P_2]\backslash L$ θα είναι ντετερμινιστική.

Απόδειξη:

1. Η διεργασία 0 είναι ντετερμινιστική διεργασία ικανοποιώντας τον πιο πάνω ορισμό αφού δεν μπορεί να εκτελέσει οποιεσδήποτε ενέργειες.

2. Εάν η διεργασία $l[P]$ είναι ντετερμινιστική τότε και οι διεργασίες $l[a.P]$ και $l[\overline{\alpha}.P]$θα είναι ντετερμινιστικές αφού ικανοποιούν το πιο πάνω περιορισμό. Εδώ δεν μπορούν να υπάρξουν εσωτερικές ενέργειες παρά μόνο μια ενέργεια η α.

$\langle L, n \rangle \triangleright l[a.P] \xrightarrow{a} \langle L', n' \rangle \triangleright l[P]$
$\langle L, n \rangle \triangleright l[\overline{\alpha}.P] \xrightarrow{\bar{a}} \langle L', n' \rangle \triangleright l[P]$

3. Εάν η διεργασία $l[P]$ είναι ντετερμινιστική τότε και η διεργασία $l[P]\backslash F$ είναι ντετερμινιστική

Αν το α αποτελεί ενέργεια εισόδου ή εξόδου τότε

$\langle L, n \rangle \triangleright l[P]\backslash F \xrightarrow{a} \langle L', n' \rangle \triangleright l[P']\backslash F \quad \langle L, n \rangle \triangleright l[P]\backslash F \overset{\widehat{\alpha}}{\Rightarrow} \langle L', n' \rangle \triangleright l[P'']\backslash F$

$\langle L', n' \rangle \rhd l[P'] \approx \langle L', n' \rangle l[P'']$, $\alpha \notin F$

Αν το α αποτελεί εσωτερική ενέργεια

$\langle L, n \rangle \rhd l[P] \backslash F \xrightarrow{a} \langle L, n \rangle \rhd l[P'] \backslash F \quad \langle L, n \rangle \rhd l[P] \backslash F \overset{\hat{\alpha}}{\Rightarrow} \langle L, n \rangle \rhd l[P''] \backslash F$
$\langle L, n \rangle \rhd l[P'] \approx \langle L, n \rangle \rhd l[P'']$, $\alpha \in F$

4. Εάν $\forall i \in I$ η διεργασία $l[P_i]$ είναι ντετερμινιστική και όλα τα α_i είναι μοναδικά κανάλια που αντιστοιχούν ως ενέργειες εισόδου στην κάθε διεργασία $l[P_i]$ τότε και η διεργασία $l[\sum_{\iota \in I} \alpha_\iota(\tilde{x_\iota}).P_i]$ θα είναι επίσης ντετερμινιστική καθώς συμφωνά με την ενέργεια εισόδου α_ι θα συνεχίσουμε σύμφωνα με την αντίστοιχη διεργασία P_i και η μετάβαση αυτή ικανοποιεί τον πιο πάνω ορισμό για ασθενή διπροσομοίωση. Δεν υπάρχουν εσωτερικές ενέργειες εδώ. Μόνο η ενέργεια α_ι μπορεί να εκτελεστεί.

$$\langle L, n \rangle \rhd l[\sum_{\iota \in I} \alpha_\iota(\tilde{x_\iota}).P_i] \xrightarrow{\alpha_\iota(\tilde{x_\iota})} l[P_i]$$

Ο λόγος για τον οποίο λέμε ότι τα κανάλια πρέπει να είναι μοναδικά είναι ότι αν ένα κανάλι α αποτελεί κανάλι εισόδου σε περισσότερες από μία διεργασίες π.χ. στις διεργασίες $l_1[P]$ και $l_2[Q]$ τότε παρόλο που και οι δύο διεργασίες μπορεί να είναι ντετερμινιστικές το άθροισμα $l[\alpha.P + \alpha.Q]$ δεν θα είναι γιατί σε περίπτωση που γίνει ενέργεια εισόδου στο κανάλι α τότε δεν γνωρίζουμε σε ποια από τις δύο διεργασίες θα συνεχίσει η εκτέλεση και αυτό θα γίνει μη-ντετερμινιστικά.

5. Εάν $\forall i \in I$ η διεργασία $l_i[P_i]$ είναι ντετερμινιστική τότε και η διεργασία $cond(e_1 \rhd l_1[P_1], \ldots, e_n \rhd l_n[P_n])$ θα είναι αφού εξ' ορισμού η διεργασία cond θα εκτελεί την πρώτη διεργασία στην σειρά της οποίας η συνθήκη είναι αληθή. Άρα δεν έχουμε οποιεσδήποτε εσωτερικές ή εξωτερικές ενέργειες έτσι μπορούμε να πούμε ότι η συμπεριφορά είναι ντετερμινιστική.

$\langle L, n \rangle \rhd cond(e_1 \rhd l_1[P_1], \ldots, e_n \rhd l_n[P_n]) \rightarrow l[P_i]$, $e_i = true$, $\forall j < i \; e_j = false$

6. Εάν οι διεργασίες l[P] και l[Q] είναι ντετερμινιστικές τότε και η διεργασία
l[susp k:(P;Q)] θα είναι ντετερμινιστική αφού ικανοποιεί τον ορισμό. Μια ενέργεια μπορεί να εκτελεστεί και είναι η susp k και ακολούθως η διεργασία θα συνεχίσει είτε με

την διεργασία l[P] είτε με την διεργασία l[Q] ανάλογα με την κατάσταση της οντότητας στην τοποθεσία k.

7. Εάν η διεργασία l[P] είναι ντετερμινιστική και l[C($\tilde{v}$)]=l[P] τότε και η διεργασία τότε και η διεργασία l[C($\tilde{v}$)] θα είναι ντετερμινιστική.

8. Εάν δύο διεργασίες l[P_1] και l[P_2] είναι ντετερμινιστικές και δεν έχουν κοινά κανάλια μεταξύ τους και ούτε μπορούν να εκτελέσουν αντίθετες ενέργειες τότε και η διεργασία l[P_1||P_2]\L θα είναι ντετερμινιστική καθώς αν οι δύο διεργασίες εκτελούν διαφορετικές ενέργειες εισόδου εξόδου δεν μας ενδιαφέρει η σειρά με την οποία θα εκτελεστούν οι ενέργειες αυτές καθώς το τελικό αποτέλεσμα θα είναι το ίδιο. Αν όμως υπάρχουν κοινά κανάλια μεταξύ τους τότε μπορεί να υπάρχουν εκτελέσεις οι οποίες να είναι μη ντετερμινιστικές αποκλείοντας άλλες εκτελέσεις.

Αν οι δύο διεργασίες έχουν κοινά κανάλια εισόδου τότε μπορεί να προκύψει η πιο κάτω εκτέλεση

$\langle L , n \rangle \rhd l[P_1 \,||\, P_2] \backslash L \xrightarrow{a} \langle L' , n' \rangle \rhd l[P_1' \,||\, P_2] \backslash L$
$\langle L , n \rangle \rhd l[P_1 \,||\, P_2] \backslash L \xrightarrow{\alpha} \langle L' , n' \rangle \rhd l[P_1 \,||\, P_2'] \backslash L$

$\langle L' , n' \rangle \rhd l[P_1' \,||\, P_2] \backslash L \nsim \langle L' , n' \rangle \rhd l[P_1 \,||\, P_2'] \backslash L$

Το ίδιο ισχύει και όταν δύο διεργασίες έχουν κοινά κανάλια εξόδου όπου μπορεί να προκύψει

$\langle L , n \rangle \rhd l[P_1 \,||\, P_2] \backslash L \xrightarrow{\bar{a}} \langle L' , n' \rangle \rhd l[P_1' \,||\, P_2] \backslash L$
$\langle L , n \rangle \rhd l[P_1 \,||\, P_2] \backslash L \xrightarrow{\bar{a}} \langle L' , n' \rangle \rhd l[P_1 \,||\, P_2'] \backslash L$

$\langle L' , n' \rangle \rhd l[P_1' \,||\, P_2] \backslash L \nsim \langle L' , n' \rangle \rhd l[P_1 \,||\, P_2'] \backslash L$

Αν δύο διεργασίες μπορούν να εκτελέσουν αντίθετες ενέργειες τότε μπορεί να υπάρξει η πιο κάτω εκτέλεση

$\langle L , n \rangle \rhd l[P_1 \,||\, P_2] \backslash L \xrightarrow{\tau} \langle L , n \rangle \rhd l[P_1' \,||\, P_2'] \backslash L$
και $\langle L , n \rangle \rhd l[P_1 \,||\, P_2] \backslash L \not\approx \langle L , n \rangle \rhd l[P_1' \,||\, P_2'] \backslash L$

Αν δεν ισχύει κανένα από τα πιο πάνω τότε

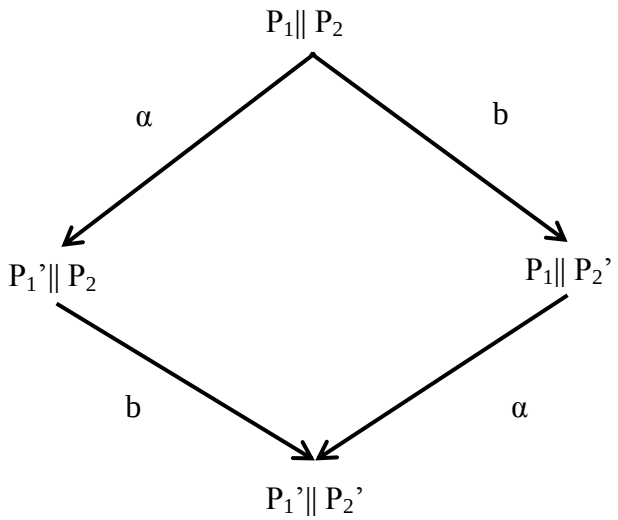
$\langle L , n \rangle \rhd l[P_1 \,||\, P_2] \backslash L \xrightarrow{\alpha} \langle L' , n' \rangle \rhd l[P_1' \,||\, P_2] \backslash L$
$\langle L , n \rangle \rhd l[P_1 \,||\, P_2] \backslash L \xrightarrow{\beta} \langle L' , n' \rangle \rhd l[P_1 \,||\, P_2'] \backslash L$

Αφού οι διεργασίες $l[P_1]$ και $l[P_2]$ είναι συρρέουσες τότε οι ακόλουθες μεταβάσεις μπορούν να συμβούν

$\langle L', n' \rangle \rhd l[P_1 \,||\, P_2'] \backslash L \xrightarrow{\alpha} \langle L'', n'' \rangle \rhd l[P_1' \,||\, P_2'] \backslash L$

$\langle L', n' \rangle \rhd l[P_1' \,||\, P_2] \backslash L \xrightarrow{\beta} \langle L'', n'' \rangle \rhd l[P_1' \,||\, P_2'] \backslash L \qquad , \alpha,\beta \notin L$

Περίπτωση όπου οι δύο διεργασίες εκτελούν ενέργειες ανεξάρτητα



Σχήμα 2.2

Με βάση τον περιορισμό που θέσαμε δεν μπορούν να προκύψουν οποιεσδήποτε εσωτερικές ενέργειες μεταξύ των διεργασιών και έτσι μπορούμε να πούμε ισχύει ο κανόνας που θέσαμε για ντετερμινιστικές διεργασίες. Η σειρά με την οποία θα εκτελεστούν οι εξωτερικές ενέργειες δεν μας ενδιαφέρει αφού το τελικό αποτέλεσμα θα είναι το ίδιο. Επίσης εδώ υποθέτουμε ότι οι διεργασίες P_1 και P_2 είναι ντετερμινιστικές.

Μπορούμε να πούμε ότι κάθε σύστημα που αποτελεί σύνθεση ντετερμινιστικών διεργασιών με τους πιο πάνω τελεστές έχει ντετερμινιστική συμπεριφορά για κάθε πιθανή εκτέλεση.

Ο πιο πάνω ορισμός μπορούμε να πούμε ότι είναι αρκετά αυστηρός καθώς δεν επιτρέπει την επικοινωνία μεταξύ διεργασιών που τρέχουν παράλληλα εσωτερικά του συστήματος γι' αυτό ορίζουμε μια καινούρια μορφή ντετερμινισμού πιο κάτω.

3.5 Συρροή

Θέλοντας να δυναμώσουμε ακόμη περισσότερο την έννοια του ντετερμινιστικού συστήματος εισάγουμε την έννοια της συρροής (confluence)[1] ούτως ώστε να διατηρήσουμε την έννοια του ντετερμινισμού και για συστήματα που τρέχουν παράλληλα κάτω από περιορισμό.

Με τον όρο συρροή εννοούμε μια μορφή ντετερμινισμού η οποία επιτρέπει στις διεργασίες μεταξύ τους να επικοινωνούν νοουμένου ότι τα κοινά τους κανάλια είναι υπό περιορισμό. Με αυτή την μορφή ντετερμινισμού εγγυόμαστε ότι σε ένα σύστημα δεν υπάρχουν ενέργειες που αποκλείονται από το να εκτελεστούν.

Ορισμός 8: Μια διεργασία $\langle \mathrm{L}, \mathrm{n} \rangle \rhd l[\mathrm{P}]$ είναι συρρέουσα αν είναι ντετερμινιστική και για κάθε διεργασία $\langle \mathrm{L}', \mathrm{n}' \rangle \rhd l[\mathrm{Q}]$ που παράγεται από την διεργασία $l[\mathrm{P}]$ και ενέργειες α,β σε διαφορετικά κανάλια αν $\langle \mathrm{L}', \mathrm{n}' \rangle \rhd l[\mathrm{Q}] \xrightarrow{a} \langle L'', \mathrm{n}'' \rangle \rhd l[\mathrm{Q}_1]$ και $\langle \mathrm{L}', \mathrm{n}' \rangle \rhd l[\mathrm{Q}] \xrightarrow{\beta} \langle L'', \mathrm{n}'' \rangle \rhd l[\mathrm{Q}_2]$ τότε πρέπει να υπάρχουν Q_1' και Q_2' τέτοιες ώστε $\langle L'', \mathrm{n}'' \rangle \rhd l[\mathrm{Q}_2] \overset{\widehat{\alpha}}{\Rightarrow} \langle \mathrm{L}''', \mathrm{n}''' \rangle \rhd l[\mathrm{Q}_2']$, $\langle \mathrm{L}'', \mathrm{n}'' \rangle \rhd l[\mathrm{Q}_1] \overset{\widehat{\beta}}{\Rightarrow} \langle L''', \mathrm{n}''' \rangle \rhd l[\mathrm{Q}_1']$ τέτοια ώστε $l[\mathrm{Q1}'] \approx l[\mathrm{Q2}']$.

Πρόταση 4:

1. Η διεργασία 0 είναι συρρέουσα.
2. Εάν η διεργασία P είναι συρρέουσα τότε και οι διεργασίες $l[a.P]$ και $l[\bar{\alpha}.P]$ είναι.
3. Εάν η διεργασία P είναι συρρέουσα τότε και η διεργασία $l[P]\backslash F$ θα είναι.
4. Εάν για κάθε $i \in I$ κάθε διεργασία $l_\iota[P_\iota]$ είναι συρρέουσα τότε και η διεργασία $\mathrm{cond}(\mathrm{e}_1 \rhd \mathrm{l}_1[\mathrm{P}_1], \ldots, \mathrm{e}_n \rhd \mathrm{l}_n[\mathrm{P}_n])$ θα είναι επίσης.
5. Εάν η διεργασίες l[P] και l[Q] είναι συρρέουσες τότε και η διεργασία l[susp k:(P;Q)] θα είναι συρρέουσα.
6. Εάν η διεργασία l[P] είναι συρρέουσα και $\mathrm{l}[\mathrm{C}(\tilde{v})]$=l[P] τότε και η διεργασία $\mathrm{l}[\mathrm{C}(\tilde{v})]$ θα είναι συρρέουσα.
7. Εάν οι διεργασίες $\mathrm{l}[\mathrm{P}_1]$ και $\mathrm{l}[\mathrm{P}_2]$ είναι συρρέουσες και $L(P_1) \cap L(P_2) = \emptyset$, $\bar{L}(P_1) \cap \bar{L}(P_2) = \emptyset$ και τα κοινά τους κανάλια βρίσκονται υπό περιορισμό τότε και η διεργασία $l(P_1 || P_2)\backslash \mathrm{L}$ θα είναι συρρέουσα.

Αποδειξη:

1. Η διεργασία 0 είναι συρρέουσα αφού είναι ντετερμινιστική και ικανοποιεί τον πιο πάνω περιορισμό. Η διεργασία 0 δεν μπορεί να εκτελέσει καμιά ενέργεια.

2. Εάν η διεργασία P είναι συρρέουσα τότε και οι διεργασίες $l[a.P]$ και $l[\bar{\alpha}.P]$ θα είναι αφού πάλι είναι ντετερμινιστικές και ικανοποιούν το πιο πάνω περιορισμό. Μόνο οι ενέργειες a και $\bar{\alpha}$ αντίστοιχα μπορούν να εκτελεστούν.

$\langle L , n \rangle \rhd l[a.P] \xrightarrow{a} \langle L' , n' \rangle \rhd l[P]$
$\langle L , n \rangle \rhd l[\bar{\alpha}.P] \xrightarrow{\bar{\alpha}} \langle L' , n' \rangle \rhd l[P]$

3. Εάν η διεργασία l[P] είναι συρρέουσα τότε και η διεργασία l[P]\F θα είναι αφού είναι ντετερμινιστική και ικανοποιεί τον πιο πάνω περιορισμό.
Εάν η διεργασία l[P] μπορεί να εκτελέσει μια ενέργεια α και να εξελιχτεί στην διεργασία l[P'] όπου α∉ F τότε και η διεργασία l[P]\F μπορεί να εκτελέσει την ενέργεια α και να εξελιχτεί στην διεργασία l[P']\F.

Εάν η διεργασία l[P] μπορεί να εκτελέσει μια ενέργεια b και να εξελιχτεί στην διεργασία l[P''] όπου b∉ F τότε και η διεργασία l[P]\F μπορεί να εκτελέσει την ενέργεια b και να εξελιχτεί στην διεργασία l[P'']\F.

Αφού γνωρίζουμε ότι η διεργασία l[P] είναι συρρέουσα τότε μπορούν να εκτελεστούν οι ακόλουθες μεταβάσεις

$l[P']\backslash F \overset{\hat{b}}{\Rightarrow} l[P_1]\backslash F$
$l[P'']\backslash F \overset{\hat{a}}{\Rightarrow} l[P_2]\backslash F$

όπου $l[P_1]\backslash F \approx l[P_2]\backslash F$

4. Εάν ∀i∈I η διεργασία $l_i[P_i]$ είναι συρρέουσα τότε και η διεργασία $\text{cond}(e_1 \rhd l_1[P_1], \dots, e_n \rhd l_n[P_n])$ θα είναι συρρέουσα αφού είναι ντετερμινιστική και η δεύτερη θα εξελιχθεί σύμφωνα με μια από τις διεργασίες $l_i[P_i]$.

$\langle L , n \rangle \rhd \text{cond}(e_1 \rhd l_1[P_1], \dots, e_n \rhd l_n[P_n]) \rightarrow l_i[P_i]$, $e_i = \text{true}$, $\forall j < i\ e_j = \text{false}$

5. Εάν η διεργασίες l[P] και l[Q] είναι συρρέουσες τότε και η διεργασία l[susp k:(P;Q)] θα είναι συρρέουσα αφού είναι ντετερμινιστική και ικανοποιεί τον ορισμό.

6. Εάν η διεργασία l[P] είναι συρρέουσα και $l[C(\tilde{v})]=l[P]$ τότε και η διεργασία $l[C(\tilde{v})]$ θα είναι συρρέουσα.

7. Εάν οι διεργασίες $l[P_1]$ και $l[P_2]$ είναι συρρέουσες και δεν έχουν κοινά κανάλια εισόδου ή εξόδου τότε και η διεργασία $l[P_1||P_2]\backslash L$ είναι συρρέουσα.

Αν όμως δύο διεργασίες έχουν κοινά κανάλια εισόδου και εξόδου μπορεί να έχουμε την πιο κάτω κατάσταση

$\langle L , n \rangle \rhd (P_1 || P_2)\backslash L \xrightarrow{\alpha} \langle L' , n' \rangle \rhd (P_1' || P_2) \backslash L$

$\langle L , n \rangle \rhd (P_1 || P_2)\backslash L \xrightarrow{\alpha} \langle L' , n' \rangle \rhd (P_1 || P_2') \backslash L$ και

$\langle L' , n' \rangle \rhd (P_1' || P_2) \backslash L \nsim \langle L' , n' \rangle \rhd (P_1 || P_2')\backslash L \ \alpha \notin L$

και αντίστοιχα

$\langle L , n \rangle \rhd (P_1 || P_2)\backslash L \xrightarrow{\bar{a}} \langle L' , n' \rangle \rhd (P_1' || P_2) \backslash L$

$\langle L , n \rangle \rhd (P_1 || P_2)\backslash L \xrightarrow{\bar{a}} \langle L' , n' \rangle \rhd (P_1 || P_2') \backslash L$

και $\langle L' , n' \rangle \rhd (P_1' || P_2) \backslash L \nsim \langle L' , n' \rangle \rhd (P_1 || P_2')\backslash L \ \alpha \notin L$

Για να αποδείξουμε ότι η έννοια της συρροής διατηρείται και όταν δύο διεργασίες τρέχουν παράλληλα $l[P_1||P_2]$ τότε παίρνουμε όλες τις πιθανές μεταβάσεις που μπορούν να γίνουν από τις δύο διεργασίες που τρέχουν παράλληλα και βλέπουμε αν διατηρούν τον πιο πάνω ορισμό που ορίσαμε για συρροή.

Αρχικά παίρνουμε την περίπτωση όπου οι δύο διεργασίες εκτελούν ενέργειες ανεξάρτητα

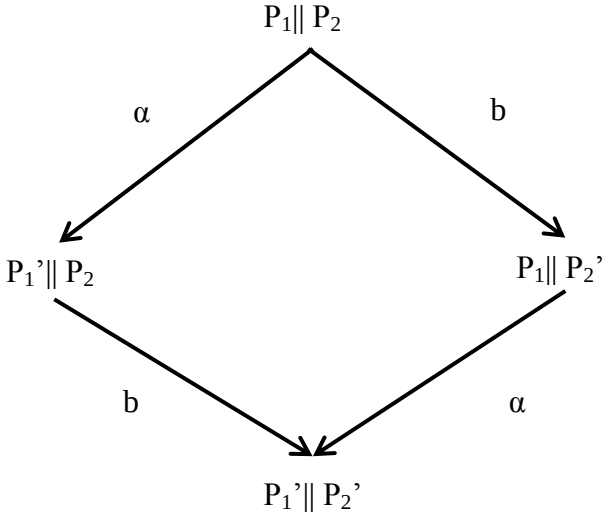
$\langle L , n \rangle \rhd l[P_1 || P_2] \backslash L \xrightarrow{\alpha} \langle L' , n' \rangle \rhd l[P_1' || P_2] \backslash L$

$\langle L , n \rangle \rhd l[P_1 || P_2] \backslash L \xrightarrow{\beta} \langle L' , n' \rangle \rhd l[P_1 || P_2'] \backslash L$

Αφού οι διεργασίες $l[P_1]$ και $l[P_2]$ είναι συρρέουσες τότε οι ακόλουθες μεταβάσεις μπορούν να συμβούν

$\langle L' , n' \rangle \rhd l[P_1' \,||\, P_2] \backslash L \xrightarrow{\beta} \langle L'' , n'' \rangle \rhd l[P_1' \,||\, P_2'] \backslash L$

$\langle L' , n' \rangle \rhd l[P_1 \,||\, P_2'] \backslash L \xrightarrow{\alpha} \langle L'' , n'' \rangle \rhd l[P_1' \,||\, P_2'] \backslash L, \quad \alpha,\beta \notin L$



Η δεύτερη περίπτωση αποτελεί την περίπτωση όπου η μια διεργασία εκτελεί μια εξωτερική ενέργεια και εσωτερική ενέργεια με την διεργασία που τρέχει παράλληλα με αυτή. Εδώ υποθέτουμε ότι η διεργασία $l[P_1]$ εκτελεί την εξωτερική ενέργεια α.

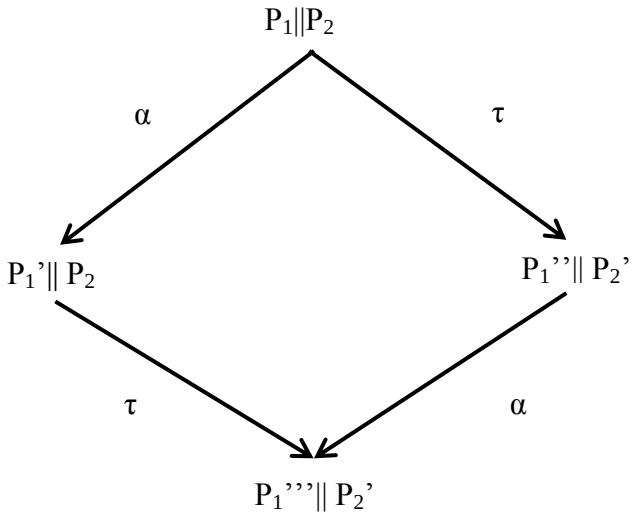
$\langle L , n \rangle \rhd l[P_1 \,||\, P_2] \backslash L \xrightarrow{\alpha} \langle L' , n' \rangle \rhd l[P_1' \,||\, P_2] \backslash L$

$\langle L , n \rangle \rhd l[P_1 \,||\, P_2] \backslash L \xRightarrow{\tau} \langle L , n \rangle \rhd l[P_1'' \,||\, P_2'] \backslash L$

Αφού οι διεργασίες $l[P_1]$ και $l[P_2]$ είναι συρρέουσες τότε οι ακόλουθες μεταβάσεις μπορούν να συμβούν

$\langle L' , n' \rangle \rhd l[P_1' \,||\, P_2] \backslash L \xRightarrow{\tau} \langle L' , n' \rangle \rhd l[P_1''' \,||\, P_2'] \backslash L$

$\langle L , n \rangle \rhd l[P_1'' \,||\, P_2'] \backslash L \xrightarrow{\alpha} \langle L' , n' \rangle \rhd l[P_1'''|| \, P_2'] \backslash L \ , \alpha \notin L$



Σχήμα 2.2

Η τρίτη περίπτωση αποτελεί την περίπτωση όπου οι διεργασίες μεταξύ τους εκτελούν δύο διαφορετικές εσωτερικές ενέργειες

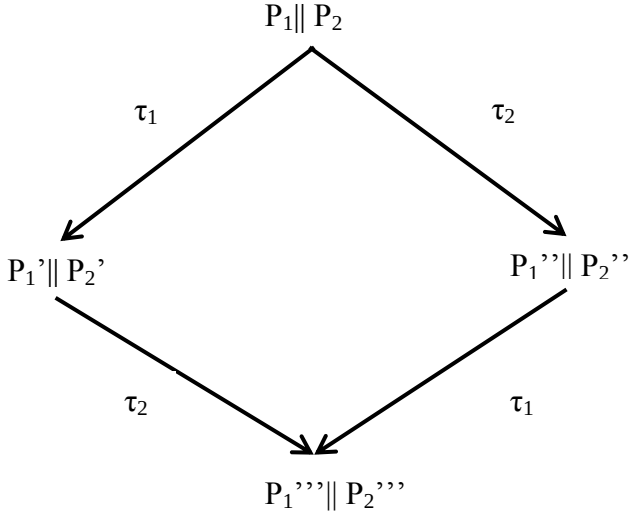
$\langle L , n \rangle \triangleright l[P_1 \,||\, P_2] \backslash L \xrightarrow{\tau_1} \langle L , n \rangle \triangleright l[P_1' \,||\, P_2'] \backslash L$

$\langle L , n \rangle \triangleright l[P_1 \,||\, P_2] \backslash L \overset{\tau_2}{\Rightarrow} \langle L , n \rangle \triangleright l[P_1'' \,||\, P_2''] \backslash L$

Αφού οι διεργασίες $l[P_1]$ και $l[P_2]$ είναι συρρέουσες τότε οι ακόλουθες μεταβάσεις μπορούν να συμβούν

$\langle L , n \rangle \triangleright l[P_1' \,||\, P_2'] \backslash L \overset{\tau_2}{\Rightarrow} \langle L , n \rangle \triangleright l[P_1''' \,||\, P_2'''] \backslash L$

$\langle L , n \rangle \triangleright l[P_1'' \,||\, P_2''] \backslash L \xrightarrow{\tau_1} \langle L , n \rangle \triangleright l[P_1'''|| P_2'''] \backslash L$



Σχήμα 2.3

Από όλα τα πιο πάνω μπορούμε να πούμε ότι το πιο κάτω ισχύει

Αν $\langle L , n \rangle \rhd l[P_1 \,||\, P_2] \backslash L \xrightarrow{\alpha} \langle L' , n' \rangle \rhd l[P_1' \,||\, P_2] \backslash L$ και

$\langle L , n \rangle \rhd l[P_1 \,||\, P_2] \backslash L \xrightarrow{\beta} \langle L' , n' \rangle \rhd l[P_1 \,||\, P_2'] \backslash L$ τότε θα πρέπει

$\langle L' , n' \rangle \rhd l[P_1 \,||\, P_2'] \backslash L \xrightarrow{\widehat{\alpha}} \langle L'' , n'' \rangle \rhd l\,[P_1' \,||\, P_2'] \backslash L$

και $\langle L' , n' \rangle \rhd l[P_1' \,||\, P_2] \backslash L \xrightarrow{\widehat{\beta}} \langle L'' , n'' \rangle \rhd l[P_1' \,||\, P_2'] \backslash L$

Για την περίπτωση του τελεστή αθροίσματος η σχετική απόδειξη θα γίνει σε κάθε αλγόριθμο ξεχωριστά καθώς η διεργασία αθροίσματος δεν είναι σε όλες τις περιπτώσεις συρρέουσα.

Μπορούμε να πούμε ότι οποιαδήποτε διεργασία αποτελεί σύνθεση συρρεόντων διεργασιών με τους πιο πάνω τελεστές είναι επίσης συρρέουσα διεργασία.

Κεφάλαιο 4

Αλγόριθμοι Μετάδοσης Μηνύματος σε Κατανεμημένο Περιβάλλον

4.1 Μετάδοση μηνύματος σε κατανεμημένο περιβάλλον

Περιγραφή Προβλήματος

Θεωρούμε ότι σε ένα σύστημα υπάρχει ένας αποστολέας κάποιου μηνύματος που πρέπει να μεταδοθεί σε ένα σύνολο από n παραλήπτες. Υπάρχουν πολλοί αλγόριθμοι οι οποίοι μπορούν να επιλύσουν το πιο πάνω πρόβλημα όμως σε ένα κατανεμημένο περιβάλλον όπου υπάρχουν σφάλματα επεξεργαστών πρέπει να χρησιμοποιηθούν αλγόριθμοι οι οποίοι να μπορούν να χειριστούν τέτοια σφάλματα και με το πέρας της εκτέλεσης να μας εγγυώνται ότι όσοι επεξεργαστές είναι ζωντανοί έχουν παραλάβει το συγκεκριμένο μήνυμα. Στην περίπτωση που έχουμε πολλά μηνύματα που πρέπει να σταλούν τότε μπορούμε να γενικοποιήσουμε την πιο πάνω πρόταση και να πούμε ότι κάθε αλγόριθμος μας εγγυάται ότι όλοι οι ζωντανοί επεξεργαστές συμφωνούν στο σύνολο των μηνυμάτων που παρέλαβαν πληρώντας την συνθήκη της ομοφωνίας. Οι οντότητες μεταξύ τους θεωρούμε ότι μπορούν να επικοινωνήσουν πάνω στο κανάλι $send_{i,j}$ όπου $i \leq j$. Έτσι στο σύστημα υπάρχουν n+ (n-1)+ (n-2)+ … 1 κανάλια δηλαδή $O(n^2)$ συνολικά κανάλια. Όλοι οι επεξεργαστές γνωρίζουν τα κανάλια τους με τα οποία μπορούν να επικοινωνήσουν. Επίσης θεωρούμε ότι η κάθε διεργασία γνωρίζει την

ταυτότητα της που είναι ένας μοναδικός αριθμός από το 1 μέχρι το n και ο αποστολέας του συστήματος έχει την ταυτότητα 0.

4.2 Ο αλγόριθμος Basic Broadcast

Ο αλγόριθμος [2] λειτουργεί σε κατανεμημένα συστήματα όπου υπάρχει ένας αποστολέας S ο οποίος θέλει να στείλει ένα μήνυμα m σε ένα σύνολο από παραλήπτες {R}. Ο αποστολέας επικοινωνεί με τους παραλήπτες μέσω κοινών καναλιών που ενώνουν τον αποστολέα με κάθε παραλήπτη ξεχωριστά. Το μήνυμα στέλνεται ταυτόχρονα σε όλους τους παραλήπτες και θεωρούμε ότι τα κανάλια δεν έχουν σφάλματα και άρα αν ο αποστολέας δεν είναι εσφαλμένος με το πέρας της εκτέλεσης ενός επικοινωνιακού γύρου του αλγορίθμου όλες οι μη εσφαλμένες οντότητες στο σύστημα θα λάβουν το μήνυμα ακόμα και ο ίδιος ο αποστολέας. Σε διαφορετική περίπτωση αν ο αποστολέας υποπέσει σε σφάλμα τότε δεν έχουμε οποιεσδήποτε εγγυήσεις για το ποιες οντότητες θα παραλάβουν το μήνυμα και σ' αυτήν την περίπτωση το σύστημα δεν παρέχει αξιοπιστία.

Ενέργειες αποστολέα: Ο αποστολέας μπορεί να κάνει broadcast ένα μήνυμα και μπορεί να κάνει και παραλαβή του ίδιου μηνύματος.
Ενέργειες παραλήπτη: Ο παραλήπτης μπορεί μόνο να κάνει παραλαβή κάποιου μηνύματος ακούοντας συνεχώς στο κανάλι που τον ενώνει με τον αποστολέα.

Αλγόριθμος

```
Upon event ⟨beb, Broadcast | m⟩ do
        Forall q ∈ Π do
                Trigger ⟨pl, Send | q,m ⟩;
Upon event ⟨pl, Deliver | p, m⟩ do
        Trigger ⟨beb, Deliver | p,m⟩;
```

Π: Το σύνολο που περιέχει όλους τους παραλήπτες.
m: Το μήνυμα.

Ο αλγόριθμος best effort broadcast λειτουργεί ως εξής:
Στο γεγονός Broadcast όπου ο αποστολέας θέλει να μεταδώσει ένα μήνυμα m τότε για κάθε οντότητα q που ανήκει στο σύνολο Π των παραληπτών εκτελεί την ενέργεια Send με περιεχόμενα την ταυτότητα του παραλήπτη και το μήνυμα m. Στο γεγονός Deliver κάποιου μηνύματος από κάποια οντότητα τότε ο παραλήπτης εκτελεί την ενέργεια Deliver με περιεχόμενα την ταυτότητα του και το περιεχόμενο του μηνύματος m. Ο κάθε παραλήπτης ακούει πάνω στο κανάλι που τον συνδέει με τον αποστολέα S και ο αποστολέας μπορεί να παραλάβει και ο ίδιος το μήνυμα του. Ένα σενάριο στο οποίο μπορούν να μας επηρεάσουν τα σφάλματα είναι καθώς ο αποστολέας στέλνει το μήνυμα στους παραλήπτες να καταρρεύσει και να μην λάβουν το μήνυμα όλοι οι ζωντανοί παραλήπτες και έτσι η παραλαβή να γίνει σε ένα υποσύνολο των παραληπτών.

Ο αλγόριθμος συνοδεύεται από τις πιο κάτω εγγυήσεις σχετικά με την ορθότητα του

1) Συνθήκη εγκυρότητας: Αν ο αποστολέας δεν είναι εσφαλμένος και μεταδώσει ένα μήνυμα τότε όλοι οι μη εσφαλμένοι επεξεργαστές θα παραλάβουν το μήνυμα.

2) Συνθήκη no-duplication: Κανένα μήνυμα δεν θα παραληφθεί περισσότερες από μία φορές.

3) Συνθήκη μη δημιουργίας: Αν κάποιος επεξεργαστής παραλάβει ένα μήνυμα τότε το ίδιο μήνυμα πρέπει να είχε προηγουμένως σταλεί από τον αποστολέα του συστήματος.

Ορθότητα αλγορίθμου

Η ορθότητα του αλγορίθμου ως προς τις πιο πάνω συνθήκες επιχειρηματολογείτε στο [2] ως εξής:

Η ιδιότητα της μη δημιουργίας ενός μηνύματος συνεπάγεται από την αντίστοιχη ιδιότητα που ικανοποιούν τα κανάλια επικοινωνίας.
Το ίδιο ισχύει και για την ιδιότητα no duplication η οποία πηγάζει από την υπόθεση που κάναμε προηγουμένως ότι κάθε μήνυμα που αποστέλλεται από μια οντότητα είναι μοναδικό.

Η συνθήκη εγκυρότητας ισχύει από το γεγονός ότι ο αποστολέας στέλνει το μήνυμα σε κάθε άλλη οντότητα και την συνθήκη της αξιόπιστης παράδοσης ενός μηνύματος από τα κανάλια επικοινωνίας έτσι όλοι οι ορθοί επεξεργαστές θα παραλάβουν το μήνυμα από την στιγμή που ο αποστολέας δεν είναι εσφαλμένος.

Πολυπλοκότητα αλγορίθμου

Για κάθε μήνυμα που μεταδίδεται από τον αποστολέα χρειαζόμαστε O(n) μηνύματα που θα σταλούν προς όλους τους παραλήπτες άρα συνολικά θα έχουμε O(nm) όπου m ο αριθμός των διαφορετικών μηνυμάτων και είναι η περίπτωση στην οποία ο αποστολέας δεν θα υποπέσει σε σφάλμα.

Μοντελοποίηση αλγορίθμου χρησιμοποιώντας την γλώσσα μοντελοποίησης συστημάτων Dπf για ένα μήνυμα

$$BB\langle L,n\rangle = (l_0[S\langle m\rangle] \,||\, l_1[R_1\langle\,\rangle] \,||\, \ldots. \,||\, l_n[R_n\langle\,\rangle])\backslash\{send_{0,1}, send_{0,2}, \ldots, send_{0,n}\}$$

$$S\langle m\rangle = \prod_{0\le i\le n} \overline{send_{0,i}}\langle m\rangle \,||\, send_{0,0}\langle m\rangle \,.\, \overline{deliver_0}\langle m\rangle \,.\, 0$$

$$R_i\langle\,\rangle = send_{0,i}\langle m\rangle.\, \overline{deliver_i}\langle m\rangle.\, 0\ ,0 \le i \le n$$

Υποθέτουμε εδώ ότι ο αποστολέας μεταδίδει ένα μήνυμα και ακολούθως τερματίζει. Ο αποστολέας S στέλνει το μήνυμα ταυτόχρονα στα κανάλια $send_{0,0}$ – $send_{0,n}$ τα οποία είναι δεσμευμένα για επικοινωνία εντός του συστήματος μεταξύ των οντοτήτων. Μόλις ο ίδιος παραλάβει το δικό του μήνυμα τότε τερματίζει αφού δηλώσει πρώτα ότι το παρέλαβε στέλλοντας μήνυμα πάνω στο κανάλι $deliver_0$. Κάθε οντότητα που λαμβάνει ένα μήνυμα στέλνει μήνυμα προς τα έξω στο κανάλι $deliver_i$ για να δηλώσει ότι παρέλαβε το μήνυμα m. Επίσης ο κάθε παραλήπτης μόλις λάβει το μήνυμα από το κανάλι του δηλώνει ότι το παρέλαβε και τερματίζει.

4.3 Ο αλγόριθμος Lazy Reliable Broadcast

Ο αλγόριθμος αυτός [2] λειτουργεί σε κατανεμημένα περιβάλλοντα όπου υπάρχει ένας αποστολέας S κάποιου μηνύματος το οποίο προωθεί σε όλες τις οντότητες του συστήματος και ένα σύνολο από παραλήπτες {R} οι οποίοι λαμβάνουν το μήνυμα πάνω

στο κοινό κανάλι που ενώνει τον καθένα ξεχωριστά με τον αποστολέα. Ο αλγόριθμος αυτός λαμβάνει υπόψη τα σφάλματα που μπορούν να προκύψουν σε ένα τέτοιο περιβάλλον κάνοντας χρήση του perfect failure detector για ανίχνευση σφαλμάτων στο σύστημα. Επίσης ο αλγόριθμος κάνει χρήση του αλγόριθμου basic broadcast για αποστολή των μηνυμάτων μεταξύ των οντοτήτων. Κάθε οντότητα μπορεί να είναι και αποστολέας και παραλήπτης και κάνει χρήση του βασικού αλγόριθμου για μετάδοση μηνύματος παρόλο που ο αποστολέας του αρχικού μηνύματος είναι ένας. Η συνθήκη συμφωνίας του αλγορίθμου μας λέει ότι αν με το πέρας του αλγορίθμου υπάρχει έστω και ένας ζωντανός επεξεργαστής που έκανε deliver το μήνυμα τότε πρέπει όλοι οι ζωντανοί επεξεργαστές μέσα στο σύστημα να κάνουν deliver το μήνυμα.

Αλγόριθμος

Uses BestEffortBroadcast instance beb.

Uses PerfectFailureDetector instance P.

upon event <rb,Init> do

status:=*true*;

From[p]:=[Ø]^N;

upon event <rb, Broadcast | m> **do**

trigger <beb,Broadcast| [Data, self, m]>;

upon event <beb,Deliver | p,[Data, s, m]> **do**

if m ∉ from[s] **then**

trigger<rb,Deliver | s,m>;

from[s]:=from[s] ∪ {m};

if status == *false* **then**

trigger <beb,Broadcast| [Data, s, m]>;

upon event <P, Crash | p> **do**

status=false;

forall m ϵ from[p] **do**

trigger <beb, Broadcast | [Data, p, m]>;

m: η ταυτότητα του μηνύματος.

s: η ταυτότητα του αποστολέα.

Data: τα περιεχόμενα του μηνύματος.

status: μεταβλητή που δηλώνει αν ο αποστολέας είναι ζωντανός ή εσφαλμένος.

from: λίστα που κρατά τα μηνύματα που έχουν ήδη παραληφθεί.

Ενέργειες αποστολέα

Ο αποστολέας μπορεί να μεταδώσει το μήνυμα αρχικά σε όλους και μπορεί ο ίδιος ταυτόχρονα να παραλάβει το δικό του μήνυμα.

Ενέργειες παραλήπτη

Ο παραλήπτης μπορεί να παραλάβει το μήνυμα του αποστολέα είτε από τον ίδιο τον αποστολέα είτε από κάποιο άλλο παραλήπτη αν ο αποστολέας είναι εσφαλμένος. Επιπλέον αν λάβει μήνυμα από τον αποστολέα και ξέρει ότι ο αποστολέας είναι εσφαλμένος τότε θα προωθήσει το μήνυμα αυτό σε όλες τις οντότητες του συστήματος και αν εντοπίσει κάποιο σφάλμα της οντότητας αποστολέας τότε κάνει broadcast όλα τα μηνύματα που έλαβε μέχρι εκείνη την στιγμή.

Επεξήγηση αλγορίθμου:

Όπως προαναφέραμε ο αλγόριθμος κάνει χρήση του αλγόριθμου BestEffortBroadcast στο πιο κάτω επίπεδο σε κάθε οντότητα ξεχωριστά για να μπορούν να μεταδίδουν μηνύματα εσωτερικά του συστήματος οι οντότητες. Επίσης γίνεται χρήση του τέλειου ανιχνευτή σφαλμάτων. Η κάθε οντότητα διαθέτει σε ατομικό επίπεδο μια λίστα με τα μηνύματα που έλαβε από τον αποστολέα και μια μεταβλητή που δηλώνει αν ο αποστολέας είναι ζωντανός(true ή false). Αρχικά η πρώτη λίστα είναι κενή και η μεταβλητή περιέχει την τιμή true. Αν ο αποστολέας χρειαστεί να προωθήσει κάποιο μήνυμα τότε εκτελεί την εντολή BestEffortBroadcast με το περιεχόμενο του μηνύματος και ως αποστολέα τον ίδιο. Αν κάποια οντότητα p παραλάβει κάποιο μήνυμα στο πιο κάτω επίπεδο από την οντότητα S(αποστολέας) τότε ελέγχει αν δεν έχει λάβει ήδη το μήνυμα εκτελεί την ενέργεια Deliver του μηνύματος m από τον αποστολέα S στο πιο πάνω επίπεδο και προσθέτει το μήνυμα στην λίστα με τα μηνύματα που παρέλαβε. Αν ο αποστολέας S είναι εσφαλμένος τότε προωθεί αντίγραφο του μηνύματος στο πιο κάτω επίπεδο σε όλες τις οντότητες όπου εδώ είναι η περίπτωση που ο παραλήπτης εντοπίζει

πρώτα σφάλμα του αποστολέα και μετά παραλαμβάνει το μήνυμα που του έστειλε ο αποστολέας πρώτου καταρρεύσει. Σε περίπτωση που κάποια οντότητα εντοπίσει σφάλμα του αποστολέα ανά πάσα στιγμή τότε για κάθε μήνυμα που παρέλαβε από αυτόν προωθεί ένα αντίγραφο του στο πιο κάτω επίπεδο σε όλες τις οντότητες με αποστολέα τον αρχικό. Άρα με τον τρόπο αυτό όταν μια οντότητα ανακαλύψει σφάλμα του αποστολέα στέλνει σε όλους τους υπόλοιπους τα μηνύματα για να εγγυηθεί ότι τα παρέλαβαν και ότι έχουν όλοι οι ενεργοί επεξεργαστές τα ίδια μηνύματα. Σε περίπτωση που μια οντότητα παρέλαβε ήδη κάποιο συγκεκριμένο μήνυμα τότε αφού ελέγξει στην λίστα της αν δει ότι το έλαβε ήδη τότε απλά το αγνοεί διαφορετικά το παραλαμβάνει και έτσι αποφεύγουμε διπλασιασμό των μηνυμάτων. Ένα σενάριο για ένα μήνυμα θα μπορούσε να ήταν ο αποστολέας να καταφέρει να στείλει σε όλες τις οντότητες το μήνυμα και όταν όλοι οι ζωντανοί επεξεργαστές κάνουν deliver το μήνυμα τότε ο αλγόριθμος τερματίζει. Ένα δεύτερο σενάριο θα μπορούσε να ήταν να στείλει σε ένα υποσύνολο των παραληπτών το μήνυμα και να καταρρεύσει και μόλις παραλάβουν το μήνυμα και εντοπίσουν σφάλμα του τότε θα προωθήσουν το μήνυμα σε όλες τις οντότητες. Όταν όλες οι ζωντανές οντότητες κάνουν deliver το μήνυμα τότε ο αλγόριθμος τερματίζει. Ένα τρίτο σενάριο θα μπορούσε να ήταν πάλι να στείλει σε ένα υποσύνολο των παραληπτών το μήνυμα και ακολούθως να καταρρεύσει και όταν αυτοί παραλάβουν το μήνυμα του πριν προλάβουν να το προωθήσουν να καταρρεύσουν και έτσι κανένας από τους ζωντανούς επεξεργαστές να μην κάνει deliver το μήνυμα.

Ορθότητα αλγορίθμου

Η ορθότητα του αλγορίθμου ως προς τις συνθήκες μη δημιουργίας, μη διπλασιασμού και συμφωνίας επιχειρηματολογείτε στο [2] ως εξής:

Συνθήκη μη δημιουργίας μηνύματος: Η συνθήκη αυτή έπεται από την αντίστοιχη συνθήκη που ικανοποιείται από τον βασικό αλγόριθμο διάχυσης μηνύματος όπου χρησιμοποιείται για την μετάδοση των μηνυμάτων στο σύστημα.

Συνθήκη μη διπλασιασμού: Η συνθήκη αυτή ικανοποιείται σε κάθε οντότητα από την στιγμή που υπάρχει η λίστα from με όλα τα μηνύματα που παραλήφθηκαν από τον αποστολέα και έτσι το ίδιο μήνυμα δεν μπορεί να παραληφθεί περισσότερες από μία φορές άσχετα αν φτάνει σε κάποιο παραλήπτη περισσότερες φορές.

Συνθήκη συμφωνίας: Η συνθήκη συμφωνίας μας εγγυάται ότι με το πέρας εκτέλεσης του αλγορίθμου όλοι οι παραλήπτες θα έχουν παραλάβει τα ίδια μηνύματα από τον αποστολέα και αυτό έπεται από τη συνθήκη μη δημιουργίας κάποιου μηνύματος που ικανοποιείται από τον βασικό αλγόριθμο και επίσης από το γεγονός ότι ο κάθε παραλήπτης με τον εντοπισμό κάποιου σφάλματος του αποστολέα προωθεί σε όλους τους υπόλοιπους όλα τα μηνύματα που παρέλαβε από αυτόν μέχρι εκείνη την στιγμή. Επίσης αν υπάρξει σφάλμα του αποστολέα όλοι οι ζωντανοί παραλήπτες θα το ανιχνεύσουν.

Πολυπλοκότητα αλγορίθμου

Για κάθε διαφορετικό μήνυμα που μεταδίδεται σε αυτόν το αλγόριθμο από τον αποστολέα αν δεν υπάρξει σφάλμα του αποστολέα θα χρειαστούν O(n) μηνύματα διαφορετικά θα χρειαστούν $O(n^2)$ μηνύματα αν όλοι παραλάβουν το αρχικό μήνυμα του αποστολέα και μετά εντοπίσουν το σφάλμα του. Αν υπήρξαν προηγούμενα μηνύματα τότε το κόστος για κάθε διαφορετικό μήνυμα θα είναι $O(n^2)$ μηνύματα άρα $O(mn^2)$ μηνύματα όπου m ο αριθμός των διαφορετικών μηνυμάτων. Αν δεν υπάρξει σφάλμα του αποστολέα τότε το κόστος των μηνυμάτων θα είναι O(mn) όπου m ο αριθμός των διαφορετικών μηνυμάτων που έστειλε ο αποστολέας.

4.4 Ο αλγόριθμος Uniform Reliable Broadcast

Με τον αλγόριθμο αυτό [2] εγγυόμαστε ότι το σύνολο των μηνυμάτων που παραλαμβάνονται από τους εσφαλμένους επεξεργαστές είναι υποσύνολο των μηνυμάτων που παραλαμβάνουν οι μη εσφαλμένοι επεξεργαστές για τον λόγο ότι υπάρχει το ενδεχόμενο ένας επεξεργαστής αφού παραλάβει ένα μήνυμα πριν προλάβει να το προωθήσει στους υπόλοιπους να καταρρεύσει και έτσι να μην το παραλάβουν αυτοί που είναι ορθοί με αποτέλεσμα να υπάρχει συμφωνία μεταξύ των επεξεργαστών αλλά με λανθασμένο αριθμό μηνυμάτων.

Περιγραφή αλγορίθμου

Ο αλγόριθμος δουλεύει σε κατανεμημένα συστήματα όπου ένας αποστολέας μεταδίδει ένα μήνυμα σε ένα σύνολο από παραλήπτες. Για να πούμε ότι μια οντότητα παρέλαβε ένα μήνυμα σύμφωνα με τον αλγόριθμο αυτό θα πρέπει να το έχει ήδη λάβει και να

ξέρει ότι όλοι οι ζωντανοί επεξεργαστές έχουν ήδη δει το μήνυμα. Έτσι κάθε οντότητα που λαμβάνει κάποιο συγκεκριμένο μήνυμα το προωθεί σε όλες τις υπόλοιπες. Επίσης κάθε οντότητα κρατά μια λίστα ack με όλες τις υπόλοιπες οντότητες που παρέλαβαν ένα συγκεκριμένο μήνυμα m και μόλις παραλάβουν το μήνυμα από αυτή την οντότητα την προσθέτουν στην λίστα. Για κάθε μήνυμα υπάρχει μια λίστα για αυτό ack[m]. Όταν το παραλάβουν από όλες τις οντότητες τότε είναι σίγουρη ότι όλες το είδαν και μπορεί να το παραλάβει σε αυτό το επίπεδο. Επίσης κάθε οντότητα κρατά μια λίστα pending η οποία περιέχει όλα τα μηνύματα που έχει ήδη λάβει αλλά ακόμα δεν τα έλαβαν όλες οι υπόλοιπες οντότητες και μια λίστα delivered στην οποία κρατούνται όλα τα μηνύματα που παραλήφθηκαν για να μην παραλάβουμε ένα μήνυμα περισσότερες από μία φορές. Ο αλγόριθμος κάνει χρήση του αλγόριθμου BestEffortBroadcast για μετάδοση μηνυμάτων μεταξύ των επεξεργαστών και του ανιχνευτή σφαλμάτων από προηγούμενους αλγόριθμους. Η συνθήκη συμφωνίας του αλγορίθμου μας λέει ότι αν με το πέρας εκτέλεσης του αλγορίθμου υπάρχει έστω και ένας επεξεργαστής εσφαλμένος ή μη που έκανε deliver το μήνυμα τότε και όλοι οι ζωντανοί επεξεργαστές πρέπει να κάνουν deliver το μήνυμα.

Αλγόριθμος

Uses:

BestEffortBroadcast, instance beb

PerfectFailureDetector, instance P

upon event <*urb ,Init* > do

delivered:=Ø;

pending:= Ø;

correct:=*Π*;

forall m **do** *ack[m]*:= Ø;

upon event <*urb, Broadcast | m*> do

pending:=*pending*U{(*self,m*)};

trigger<*beb,Broadcast*|[Data,*self,m*]>;

upon event<*beb,Deliver*|*p*,[Data,*s,m*]>do

ack[m]:=*ack*[m]U{*p*};

If (s,m) ∉ *pending* **then**

pending:=*pending*U{(s,m)};

trigger <*beb,Broadcast*|[Data,*s,m*]};

upon event <P,*Crash*|*p*> **do**

correct:=*correct*\{*p*};

function *candeliver*(*m*) **returns** Boolean **is**

return (*correct*⊆*ack*[*m*]);

upon exists (*s,m*) ϵ *pending* such that *candeliver*(*m*) ∧ *m*∉ *delivered* **do**

delivered:=*delivered*U{*m*};

trigger<*urb,Deliver*|*s,m*>;

m: η ταυτότητα του μηνύματος.

s: η ταυτότητα του αποστολέα.

Data: τα περιεχόμενα του μηνύματος.

ack[m]: λίστα που κρατά τους επεξεργαστές που παρέλαβαν το μήνυμα m.

pending: λίστα που κρατά τα μηνύματα που παραλήφθηκαν αλλά δεν έχουν γίνει ακόμα deliver.

deliver: λίστα που περιέχει όλα τα μηνύματα που έγιναν deliver.

correct: λίστα που περιέχει τους ζωντανούς επεξεργαστές.

Επεξήγηση αλγορίθμου

Αρχικά για κάθε επεξεργαστή η λίστα delivered και pending είναι κενές και η λίστα correct περιέχει όλους τους επεξεργαστές του συστήματος. Επίσης για κάθε μήνυμα η λίστα ack του μηνύματος είναι αρχικά κενή. Σε ένα γεγονός broadcast από τον αποστολέα για κάποιο μήνυμα m τότε ο αποστολέας προσθέτει το συγκεκριμένο μήνυμα στην λίστα pending με την ταυτότητα του ίδιου και εκτελεί τον βασικό αλγόριθμο μετάδοσης μηνύματος για να το στείλει σε όλους τους επεξεργαστές στο σύστημα με αποστολέα την ταυτότητα του ίδιου. Σε ένα γεγονός deliver στο επίπεδο του βασικού αλγόριθμου σε κάποιο παραλήπτη από κάποια οντότητα p τότε γίνεται εισαγωγή του συγκεκριμένου επεξεργαστή στην λίστα ack του μηνύματος και αν το μήνυμα δεν ανήκει ήδη στην λίστα pending τότε προσθέτεται και εκτελείται ο βασικός

αλγόριθμος για να προωθηθεί το μήνυμα στους υπόλοιπους με την ταυτότητα του αρχικού αποστολέα. Αν υπάρξει μήνυμα το οποίο δεν έχει ακόμη προστεθεί στην λίστα delivered και το έχουν δει όλοι οι μη εσφαλμένοι επεξεργαστές τότε προσθέτεται στην λίστα και βγαίνει από την λίστα pending. Για τον πιο πάνω έλεγχο χρησιμοποιούμε την συνάρτηση *candeliver*(*m*) η οποία παίρνει σαν παράμετρο το μήνυμα και μας απαντά αναλόγως. Σε ένα γεγονός Crash από τον επεξεργαστή p τότε εντοπίζεται δυναμικά στο σύστημα και όλοι οι ζωντανοί επεξεργαστές βγάζουν από την λίστα τους τον συγκεκριμένο επεξεργαστή που υπόπεσε σε σφάλμα. Ένα σενάριο εκτέλεσης θα ήταν να μην υποπέσει σε σφάλμα ο αποστολέας και να καταφέρει να στείλει σε όλες τις οντότητες το μήνυμα του και όταν όλες οι ζωντανοί επεξεργαστές κάνουν deliver το μήνυμα τότε ο αλγόριθμος τερματίζει. Ένα δεύτερο σενάριο θα μπορούσε να ήταν να καταρρεύσει ο αποστολέας καθώς στέλνει το μήνυμα και να το παραλάβει μόνο ένας επεξεργαστής ο οποίος με την σειρά του θα το στείλει σε όλες τις υπόλοιπες οντότητες και μόλις όλοι οι ζωντανοί επεξεργαστές το κάνουν deliver τότε ο αλγόριθμος θα τερματίσει. Ένα τρίτο σενάριο θα μπορούσε να ήταν ο αποστολέας να καταρρεύσει καθώς στέλνει το μήνυμα και πάλι να προλάβει να το στείλει σε ένα μόνο επεξεργαστή. Ακολούθως ο επεξεργαστής αυτός να καταρρεύσει πριν προλάβει να το στείλει παρακάτω και έτσι κανένας από τους ζωντανούς επεξεργαστές να μην παραλάβει το μήνυμα.

Ενέργειες οντοτήτων

Οι ενέργειες κάποιου παραλήπτη και του αποστολέα είναι οι ίδιες με την μόνη διαφορά ότι ο αποστολέας του αρχικού μηνύματος είναι ο αποστολέας που υπάρχει στο σύστημα αλλά και τα δύο είδη οντοτήτων μπορούν να κάνουν broadcast ένα μήνυμα και να παραλάβουν κάποιο μήνυμα δηλαδή μπορούν να εκτελέσουν τις πράξεις urb_broadcast και urb_deliver.

Ορθότητα αλγορίθμου

Η ορθότητα του αλγορίθμου ως προς τις συνθήκες εγκυρότητας, μη διπλασιασμού και ομοφωνίας επιχειρηματολογείτε στο [2] ως εξής:

Συνθήκη εγκυρότητας: Η συνθήκη εγκυρότητας που δηλώνει ότι αν μια οντότητα p κάνει broadcast ένα μήνυμα τότε μετά θα το παραλάβει και η ίδια προκύπτει από την

ακρίβεια που υπάρχει στην ανίχνευση των σφαλμάτων και την εγκυρότητα του βασικού αλγορίθμου που χρησιμοποιείται για την μετάδοση των μηνυμάτων.

Συνθήκη μη δημιουργίας: Συνεπάγεται από την αντίστοιχη συνθήκη του βασικού αλγόριθμου.

Συνθήκη μη διπλασιασμού: Η συνθήκη μη διπλασιασμού κάποιου μηνύματος πηγάζει από την λίστα delivered που έχει ο κάθε επεξεργαστής για να κρατά πληροφορία για τα ποια μηνύματα παρέλαβε μέχρι στιγμής.

Συνθήκη ομοφωνίας: Η συνθήκη ομοφωνίας μεταξύ των επεξεργαστών προκύπτει από το γεγονός ότι κάθε επεξεργαστής στέλλει σε όλους το μήνυμα αφού το λάβει και περιμένει μέχρι να ξέρει ότι το έλαβαν και όλοι οι άλλοι για να το παραλάβει. Επίσης σημαντικό παράγοντα παίζει και το γεγονός ότι ο ανιχνευτής σφαλμάτων πρέπει να είναι έγκυρος και αξιόπιστος για να ξέρουν οι μη εσφαλμένοι επεξεργαστές ποιοι δεν είναι έγκυροι και να μην περιμένουν να παραλάβουν το μήνυμα από αυτούς.

Πολυπλοκότητα αλγορίθμου

Ο αλγόριθμος χρειάζεται $O(n^2)$ μηνύματα για να μεταδώσει ένα μήνυμα μεταξύ των επεξεργαστών αφού αρχικά στέλλονται n μηνύματα από τον αποστολέα στους υπόλοιπους συμπεριλαμβανομένου και του ίδιου και ακολούθως ο κάθε επεξεργαστής όταν παραλάβει το μήνυμα στέλλει n-1 μηνύματα στους υπόλοιπους και στον ίδιο εξαιρουμένου του αρχικού αποστολέα άρα συνολικά n(n-1) μηνύματα σε 2 επικοινωνιακούς γύρους για κάθε μήνυμα. Αν υπάρχουν m διαφορετικά μηνύματα τότε ο συνολικός αριθμός μηνυμάτων που στέλλονται στο σύστημα είναι $O(mn^2)$ μηνύματα.

4.5 Συμπεράσματα

Οι αλγόριθμοι LRB και URB κάνουν χρήση του βασικού αλγόριθμου για να μπορέσουν να επικοινωνήσουν οι οντότητες μεταξύ τους με μηνύματα. Ο καθένας χρησιμοποιεί ξεχωριστούς μηχανισμούς που μας εγγυούνται ότι με το πέρας του καθενός από τους πιο πάνω αλγορίθμους όλοι οι ζωντανοί επεξεργαστές θα συμφωνούν στο σύνολο των μηνυμάτων που παρέλαβαν σε περίπτωση που ο αποστολέας του συστήματος καταρρεύσει. Ο αλγόριθμος URB μας εγγυάται μια πιο ισχυρή μορφή αξιόπιστης μετάδοσης καθώς μας εγγυάται ότι το σύνολο των μηνυμάτων που παραλαμβάνουν οι εσφαλμένοι επεξεργαστές είναι πάντα υποσύνολο των μηνυμάτων που παραλαμβάνουν

οι ενεργοί επεξεργαστές για τον λόγο ότι υπάρχουν περιπτώσεις όπου κάποιοι επεξεργαστές μπορεί να λάβουν κάποιο μήνυμα και ακολούθως να καταρρεύσουν προτού προλάβουν να στείλουν το μήνυμα παρακάτω. Ο αλγόριθμος μας το εγγυάται αυτό από την στιγμή που ένας επεξεργαστής περιμένει να πιάσουν το μήνυμα όλοι οι ζωντανοί επεξεργαστές και ακολούθως κάνει ο ίδιος deliver το μήνυμα. Στην περίπτωση του αλγόριθμου LRB μπορεί να υπάρξουν περιπτώσεις όπου οι εσφαλμένοι επεξεργαστές να έκαναν deliver κάποιο μήνυμα το οποίο να μην έχουν οι ζωντανοί επεξεργαστές με το πέρας του αλγορίθμου. Στα επόμενα δύο κεφάλαια θα μοντελοποιήσουμε τους δύο πιο πάνω αλγόριθμους με βάση την Dπf και θα αποδείξουμε την ορθότητα τους.

Κεφάλαιο 5

Μοντελοποίηση και Απόδειξη Αλγορίθμου LRB

5.1 Μοντελοποίηση αλγορίθμου

5.2 Απόδειξη ορθότητας αλγορίθμου

5.1 Μοντελοποίηση αλγορίθμου

$LRB\langle L,n\rangle = (l_0[S\langle m,\langle\,\rangle\rangle] \,||\, l_1[R_1\langle\varepsilon,\langle\,\rangle,true\rangle] \,||\, l_2[R_2\langle\varepsilon,\langle\,\rangle,true\rangle] \,||\, \ldots \,||\, l_n[R_n\langle\varepsilon,\langle\,\rangle,true\rangle])\backslash K$

K={send$_{0,0}$, send$_{0,1}$,…, send$_{1,1}$, … send$_{n,n}$}

$S\langle m,from\rangle = \prod_{0\le i\le n}\overline{send_{0,i}}(m) \;||\; send_{0,0}(m).\overline{deliver_0}(m)$

$R_i\langle from,status\rangle = susp\; l_0\colon (R_i^3\langle m,from\rangle\,;\sum_{0\le k\le n} send_{i,k}(m).R_i^1\langle m,from,status\rangle)$

$$R_i^1\langle m,from,status\rangle = cond(m\notin from \;\&\&\; status=true \Rightarrow \overline{deliver_i}(m)\;||\; R_i\langle from\cup\{m\},status\rangle,$$
$$m\notin from \;\&\&\; status=false \Rightarrow \overline{deliver_i}(m)\;||\; R_i^2\,\langle m,from\cup\{m\},status\rangle,$$
$$m\in from \;\&\&\; status=true \Rightarrow R_i\langle from,status\rangle,$$
$$m\in from \;\&\&\; status=false \Rightarrow R_i^f\,\langle from,status\rangle)$$

$R_i^2\,\langle m,from,status\rangle = \prod_{0\le k\le n}\overline{send_{i,k}}(m)\;||\; R_i^f\,\langle from,status\rangle$

$R_i^3\,\langle from\rangle = cond(m\in from \Rightarrow R_i^2\langle m,from,false\rangle, m\notin from \Rightarrow R_i^f\,\langle from,false\rangle)$

$R_i^f\,\langle from,status\rangle$=$\sum_{0\le k\le n} send_{i,k}(m).R_i^1\langle m,from,status\rangle$

R={R_1, R_2,…, R_n}

Επεξήγηση μοντελοποίησης αλγορίθμου

Συντομογραφίες

l_i: η τοποθεσία στην θέση i.

m: το μήνυμα.

$from$: η λίστα $from$ στην οποία αποθηκεύονται τα μηνύματα που παραλήφθηκαν.

$status$: η μεταβλητή $status$ η οποία δηλώνει αν ο αποστολέας είναι ζωντανός.

S: η διεργασία της οντότητας αποστολέας.

R_i: η διεργασία της οντότητας παραλήπτης με ταυτότητα i.

$send_{i,k}$: το κανάλι στο οποίο μπορούν να επικοινωνήσουν οι επεξεργαστές i και k.

$deliver_i$: το κανάλι μέσω του οποίου ο επεξεργαστής i επικοινωνεί με το περιβάλλον του.

Ξεκινώντας όλες οι οντότητες που βρίσκονται στο σύστημα βρίσκονται στην αρχική τους κατάσταση όπου ο αποστολέας έχει το μήνυμα στην τοπική του μνήμη, κανένας παραλήπτης δεν έχει το μήνυμα στην τοπική του μνήμη και η λίστα from αρχικά για κάθε οντότητα είναι κενή. Επίσης η τοπική μεταβλητή status στους παραλήπτες αρχικά είναι true. Στο σύστημα μπορεί να συμβούν μέχρι και n σφάλματα όπου n είναι ο αριθμός των επεξεργαστών και κάθε οντότητα θεωρούμε ότι βρίσκεται σε μια μοναδική τοποθεσία l_i όπου i είναι η ταυτότητα του επεξεργαστή. Αρχικά L={l_0, $l_{1,...}$ l_n} αφού όλοι οι επεξεργαστές είναι ζωντανοί. Οι μεταβλητές L και n αποτελούν καθολικές μεταβλητές οι οποίες μεταβάλλονται δυναμικά κάθε φορά που εντοπίζεται ένα σφάλμα στο σύστημα.

Η διεργασία του αποστολέα ξεκινά να στέλλει παράλληλα σε όλες τις οντότητες το μήνυμα m πάνω στα κανάλια που τον ενώνουν με την καθεμιά από αυτές συμπεριλαμβανομένου και του εαυτού της. Επίσης παράλληλα μπορεί να παραλάβει το δικό της μήνυμα και να εκτελέσει την εξωτερική ενέργεια $\overline{deliver_0}(m)$ για να δηλώσει ότι το παρέλαβε και ακολούθως τερματίζει.

Η διεργασία R_i η οποία αντιπροσωπεύει την λειτουργία του παραλήπτη i στην αρχική της κατάσταση ελέγχει τη τοποθεσία l_0 που ανήκει στον αποστολέα για να δει αν είναι ζωντανός και αν εντοπίσει σφάλμα του τότε μεταβαίνει στην κατάσταση R_i^3 όπου ελέγχει αν το μήνυμα ανήκει ήδη στην λίστα from τότε μεταβαίνει στην κατάσταση R_i^2 διαφορετικά μεταβαίνει στην κατάσταση R_i^f αλλάζοντας την τιμή της παραμέτρου

status σε false που δηλώνει ότι ο αποστολέας είναι εσφαλμένος. Αν ο αποστολέας δεν είναι εσφαλμένος και στην αρχική του κατάσταση ένας παραλήπτης λάβει μήνυμα πάνω σε οποιοδήποτε κανάλι που τον ενώνει με τις υπόλοιπες οντότητες τότε μεταβαίνει στην κατάσταση R_i^1 όπου ελέγχει αν το μήνυμα που παρέλαβε ανήκει ήδη στην λίστα from τότε ανάλογα με την κατάσταση του αποστολέα μεταβαίνει είτε στην αρχική κατάσταση R_i είτε στην κατάσταση R_i^f. Αν όμως το μήνυμα δεν ανήκει στην λίστα from τότε εκτελεί την εξωτερική ενέργεια $\overline{deliver_i}(m)$ για να δηλώσει ότι το έλαβε και παράλληλα ελέγχει την τιμή της παραμέτρου status. Αν είναι true τότε σημαίνει ότι ο αποστολέας είναι ζωντανός και επιστρέφει στην αρχική της κατάσταση προσθέτοντας στην λίστα from το μήνυμα που παρέλαβε. Αν το status είναι false που σημαίνει ότι ο αποστολέας είναι εσφαλμένος τότε προσθέτει στην λίστα from το μήνυμα που παρέλαβε και μεταβαίνει στην κατάσταση R_i^2 όπου κάνει μετάδοση του μηνύματος σε όλους και παράλληλα μπορεί να μεταβεί στην κατάσταση R_i^f. Στην κατάσταση R_i^f αν παραλάβει το μήνυμα σε κάποιο από τα κανάλια που τον ενώνουν με τις υπόλοιπες οντότητες τότε μεταβαίνει στην κατάσταση R_i^1.

5.2 Απόδειξη ορθότητας αλγορίθμου

Θα αποδείξουμε την ορθότητα του αλγορίθμου δείχνοντας ότι το σύστημα ⟨L,n⟩ ⇒(LRB)\K είναι ασθενώς ισοδύναμο με μια δεύτερη διεργασία η οποία συλλαμβάνει την επιδιωκόμενη συμπεριφορά του συστήματος. Για να γίνει αυτό δυνατό πρέπει να ορίσουμε μια εξωτερική οντότητα που περικλείει το σύστημα και ελέγχοντας τον τρόπο λειτουργίας μας δηλώνει αν το σύστημα πληροί τις προϋποθέσεις που θέσαμε και μας απαντά με ένα μήνυμα επιβεβαίωσης. Παρόμοια μέθοδος χρησιμοποιήθηκε από τους Francalanza και Hennessy για την σχετική απόδειξη ορθότητας στο άρθρο τους. Αυτό που πρέπει να ισχύει με το πέρας του αλγορίθμου είναι ότι αν υπάρχει έστω και ένας ζωντανός επεξεργαστής ο οποίος παρέλαβε το μήνυμα τότε και κάθε ζωντανός επεξεργαστής πρέπει να το παραλάβει. Το περίβλημα μπορεί να επικοινωνεί με το σύστημα μας και να παίρνει την πληροφορία που χρειάζεται. Η απόδειξη που θα κάνουμε θα γίνει για ένα μήνυμα και μπορεί να γενικοποιηθεί και για πολλά. Στην περίπτωση του αλγορίθμου μας ορίζουμε την διεργασία περιβλήματος ως εξής ξεκινώντας για i=0

$A_i(m) = suspl_i: (A_{i+1}(m)\ ;\ \overline{check_i}(m).receive_i(answer).$

$cond(answer = no \Rightarrow A_{i+1}(m),$

$\quad answer = yes \Rightarrow B_0(m))),\ 0 \leq i \leq n$

$B_i(m) = suspl_i: (B_{i+1}(m)\ ;\ deliver_i(m).B_{i+1}(m)), 0 \leq i \leq n$

$A_{n+1}(m) = \overline{ok}.0$

$B_{n+1}(m)\ = \overline{ok}.0$

Η διεργασία περιβλήματος ξεκινώντας από την αρχή ελέγχει για κάθε οντότητα αν είναι ζωντανή τότε στέλνει μήνυμα στην οντότητα μαζί με το συγκεκριμένο μήνυμα για το οποίο κάνει τον έλεγχο και παίρνει πίσω μια απάντηση που δηλώνει αν η συγκεκριμένη οντότητα έχει στην κατοχή της το μήνυμα ή όχι. Αν δεν το έχει συνεχίζει με την επόμενη στη σειρά οντότητα. Αν τις ελέγξει όλες τις οντότητες τότε βγάζει προς τα έξω το μήνυμα ok και τερματίζει. Αν το έχει τότε ξαναρχίζει από την αρχή και για κάθε οντότητα στη σειρά ελέγχει αν είναι εσφαλμένη και αν δεν είναι τότε περιμένει να πιάσει είσοδο στο κανάλι deliver_i που δηλώνει ότι η διεργασία που βρίσκεται στη τοποθεσία l_i παρέλαβε το μήνυμα. Αφού ελέγξουμε και τις n οντότητες τότε η διεργασία βγάζει προς τα έξω το μήνυμα ok και τερματίζει. Η διεργασία περιβλήματος δεν μπορεί να υποπέσει σε σφάλμα γι' αυτό λέμε ότι ανήκει στην αθάνατη τοποθεσία.

Το περίβλημα εκτελείται παράλληλα με το σύστημα νοουμένου ότι δεν μπορούν να προκύψουν άλλα σφάλματα στο σύστημα. Η διεργασία ελέγχει αν υπάρχει έστω και ένας ζωντανός επεξεργαστής ο οποίος παρέλαβε το μήνυμα τότε μόλις όλοι οι ζωντανοί επεξεργαστές παραλάβουν το μήνυμα εκτελεί την εξωτερική ενέργεια ok και τερματίζει. Αν κανένας από τους ζωντανούς επεξεργαστές δεν παρέλαβε το μήνυμα τότε πάλι εκτελεί την εξωτερική ενέργεια ok και τερματίζει. Άρα το ok που μας δίνει ο αλγόριθμος μας εγγυάται ότι ισχύει η συνθήκη συμφωνίας του αλγόριθμου LRB.

Στην μοντελοποίηση του αλγορίθμου προσθέτουμε τις ενέργειες οι οποίες χρησιμοποιούνται για επικοινωνία του συστήματος με την διεργασία περιβλήματος όπου ο αποστολέας αν πιάσει μήνυμα ανά πάσα στιγμή στο κανάλι $check_0$ με παράμετρο το μήνυμα m τότε θα πρέπει να απαντά πάντα με yes στο κανάλι $receive_0$. Ο παραλήπτης αν πιάσει μήνυμα στο κανάλι $check_i$ με παράμετρο το μήνυμα m τότε

αφού ελέγξει στην λίστα from αν υπάρχει το συγκεκριμένο μήνυμα απαντά αναλόγως με yes ή no πάνω στο κανάλι $receive_i$.

$$LRB\langle L,n\rangle = (l_0[S\langle m,\langle\,\rangle\rangle] \,||\, l_1[R_1\langle\varepsilon,\langle\,\rangle,true\rangle] \,||\, l_2[R_2\langle\varepsilon,\langle\,\rangle,true\rangle] \,||\, \dots \,||\, l_n[R_n\langle\varepsilon,\langle\,\rangle,true\rangle])\backslash K$$

K={send$_{0,0}$, send$_{0,1}$,…, send$_{1,1}$, … send$_{n,n}$}

$$S\langle m,from\rangle = \prod_{0\le i\le n}\overline{send_{0,i}}(m) \,||\, send_{0,0}(m).\overline{deliver_0}(m) \,||\, \boldsymbol{check_0(m).\overline{receive}_0(yes)}$$

$$R_i\langle from,status\rangle = susp\ l_0 : (R_i^3(from)\ ;\ \textstyle\sum_{0\le k\le n} send_{i,k}(m).R_i^1(m,from,status)) + \boldsymbol{check_i(m).cond(m\in from \Rightarrow \overline{receive}_i(yes) \,||\, R_i\langle from,status\rangle,\ m\notin from \Rightarrow \overline{receive}_i(no) \,||\, R_i\langle from,status\rangle)}$$

$$R_i^1\langle m,from,status\rangle = cond(m\notin from \,\&\&\, status=true \Rightarrow \overline{deliver_i}(m) \,||\, R_i\langle from\cup\{m\},status\rangle,\ m\notin from \,\&\&\, status=false \Rightarrow \overline{deliver_i}(m) \,||\, R_i^2\,\langle m,from\cup\{m\},status\rangle,\ m\in from \,\&\&\, status=true \Rightarrow R_i\langle from,status\rangle,\ m\in from \,\&\&\, status=false \Rightarrow R_i^f\langle from,status\rangle)$$

$$R_i^2\,\langle m,from,status\rangle = \prod_{0\le k\le n}\overline{send_{i,k}}(m)\ ||\ R_i^f\langle from,status\rangle$$

$$R_i^3\,\langle from\rangle = cond(m\in from \Rightarrow R_i^2\langle m,from,false\rangle, m\notin from \Rightarrow R_i^f\,\langle from,false\rangle)$$

$$R_i^f\,\langle from,status\rangle = \textstyle\sum_{0\le k\le n} send_{i,k}(m).R_i^1(m,from,status) + \boldsymbol{check_i(m).cond(m\in from \Rightarrow \overline{receive}_i(yes) \,||\, R_i^f\langle from,status\rangle,\ m\notin from \Rightarrow \overline{receive}_i(no) \,||\, R_i^f\langle from,status\rangle)}$$

Για να αποδείξουμε ότι το σύστημα ⟨L',0⟩⊳ (LRB' || A$_0$)\C θα τερματίζει πάντα και θα βγάζει το μήνυμα ok σαν έξοδο θα πρέπει να αποδείξουμε ότι

Θεώρημα 1. Αν ⟨L,n⟩⊳(LRB)\K ⇒ ⟨L',0⟩⊳(LRB')\K τότε ⟨L',0⟩⊳(LRB' || A$_0$)\C ≈ $\overline{ok}.0$.

Με λόγια το θεώρημα δηλώνει ότι αν LRB' είναι μια τυχαία κατάσταση του συστήματος από την οποία δεν προκύπτουν άλλα σφάλματα στο σύστημα τότε αν τρέξουμε το σύστημα αυτό παράλληλα με την διεργασία περίβλημα τότε θα εκτελέσει την εξωτερική ενέργεια ok και ακολούθως θα μεταβεί σε τερματική κατάσταση. Ορίζουμε το σύνολο C ως το σύνολο το οποίο περιέχει τα κανάλια τα οποία είναι δεσμευμένα για επικοινωνία εντός του συστήματος και τα οποία είναι τα κανάλια

$deliver_i$, $check_i$, $receive_i$, $send_{0,0}$-$send_{n,n}$ και ως A_0 την αρχική κατάσταση της διεργασίας περιβλήματος.
Για να αποδείξουμε το ζητούμενο η απόδειξη που θα κάνουμε θα χωριστεί σε δύο μέρη Αρχικά θα πρέπει να αποδείξουμε ότι το πιο πάνω σύστημα είναι ικανό να παράξει το επιθυμητό αποτέλεσμα και μετά να τερματίσει. Στη συνέχεια δείχνουμε ότι το σύστημα είναι confluent για να αποδείξουμε ότι οποιαδήποτε εκτέλεση του αλγορίθμου θα μας δώσει το επιθυμητό αποτέλεσμα.

Η μεθοδολογία η οποία θα ακολουθήσουμε είναι αρχικά θα δείξουμε με την μέθοδο της επαγωγής ότι το σύστημα LRB ξεκινώντας από την αρχική του κατάσταση όταν θα φτάσει στο σημείο από το οποίο δεν μπορούν να προκύψουν άλλα σφάλματα θα έχει μια συγκεκριμένη μορφή και ακολούθως όταν τρέξουμε το σύστημα παράλληλα με την διεργασία περιβλήματος θα δείξουμε ότι υπάρχει εκτέλεση η οποία μας δίνει το ζητούμενο αποτέλεσμα βγάζοντας προς τα έξω ok. Μετά θα δείξουμε ότι το καινούριο σύστημα που αποτελείται από το σύστημα LRB' παράλληλα με την διεργασία περιβλήματος είναι confluent και έτσι οποιαδήποτε τυχαία εκτέλεση μπορεί να μας δώσει το επιθυμητό αποτέλεσμα. Η απόδειξη που θα κάνουμε είναι για ένα μήνυμα και μπορεί να γενικοποιηθεί και για πολλά μηνύματα.

Ορίζουμε I_0 το σύνολο των επεξεργαστών που βρίσκονται στην αρχική τους κατάσταση R_i και τρέχουν παράλληλα, ως I_j το σύνολο των επεξεργαστών που βρίσκονται στην κατάσταση R_i^j και τρέχουν παράλληλα και ως K το σύνολο το οποίο περιέχει τα κανάλια $send_{0,0}$-$send_{n,n}$.

Λήμμα 1. ⟨L',0⟩⊳(LRB')\K θα έχει την πιο κάτω μορφή

⟨L',0⟩⊳(LRB')\K=⟨L',0⟩⊳$(l_0[S'] \parallel \prod_{i\epsilon I_f} l_i[{R_i}^f \langle from, status\rangle] \parallel$
$\prod_{i\epsilon I} l_i[R_i \langle from, status\rangle] \parallel \prod_{i\epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{i,k}}(m)])$\K

όπου
$IUI_f = R, I \cap I_f = \emptyset$
$l_0 \in L' \Rightarrow I_2 = I_3 = \emptyset$

$l_0 \notin L' \Rightarrow I_2 \subseteq \{i \mid m \in from_i\}$

$I_3 = R \supseteq \{j \mid m \notin from_j\}$

και το S' μπορεί να έχει μια από τις πιο κάτω μορφές

$\prod_{i \in R} \overline{send_{0,i}}(m) \parallel \overline{deliver_0}(m)$ ή

$check_0(m).\overline{receive}_0(\text{yes})$ ή

$\prod_{i \in R} \overline{send_{0,i}}(m) \parallel send_{0,0}(m).\overline{deliver_0}(m)$

Απόδειξη Λήμματος 1:

Θα δείξουμε με επαγωγή ότι από την αρχή του συστήματος μέχρι το σημείο όπου δεν μπορούν να γίνουν άλλα σφάλματα όποια ακολουθία ενεργειών και να γίνει τότε το σύστημα μας θα καταλήξει στην πιο κάτω μορφή

$\langle L', 0\rangle \triangleright (LRB')\backslash K = \langle L', 0\rangle \triangleright (l_0[S'] \parallel \prod_{i \epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel$

$\prod_{i \epsilon I} l_i[R_i\langle from, status\rangle] \parallel \prod_{i \epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel \prod_{i \epsilon I_2} \prod_{k \epsilon I_3} l_i[\overline{send_{i,k}}(m)])\backslash K$

όπου σε κάθε κατάσταση θα υπάρχουν ένα σύνολο από επεξεργαστές W όπου W⊆R.

Βασικό βήμα

Στην αρχική του κατάσταση το σύστημα θα έχει την πιο κάτω μορφή για τον γύρο M=0

$\langle L,n\rangle \triangleright LRB\backslash K = \langle L,n\rangle \triangleright (l_0[S] \parallel \prod_{i \epsilon I} l_i[R_i\langle from, status\rangle])\backslash K$ όπου ικανοποιεί την πιο πάνω συνθήκη,

$I_f = I_1 = I_2 = I_3 = \emptyset$

Επαγωγική υπόθεση

Υποθέτουμε ότι ισχύει για τον γύρο M=m και ότι το σύστημα LRB είναι στην ζητούμενη μορφή

$\langle L^1, l\rangle \triangleright (l_0[S'] \parallel \prod_{i \epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i \epsilon I} l_i[R_i\langle from, status\rangle] \parallel$

$\prod_{i \epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel \prod_{i \epsilon I_2} \prod_{k \epsilon I_3} l_i[\overline{send_{i,k}}(m)])\backslash K$, $L^1 \subseteq L$ και $l \leq n$.

Επαγωγικό βήμα

Οι πιθανές μεταβάσεις οι οποίες μπορούν να γίνουν στο γύρο M=m+1 είναι οι εξής

$i \in I_2, j \in I$ ή $j \in I_f$

$\langle L^1,l\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle from, status\rangle] \parallel$

$\prod_{i\epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{i,k}}(m)])\backslash K \xrightarrow{\tau(send_{i,j})}$

$\langle L^2,k\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle from, status\rangle] \parallel$

$\prod_{i\epsilon I_1 \cup k} l_i[\overline{deliver_i}(m)] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{i,k}}(m)])\backslash K$ ή

$\langle L^2,k\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle from, status\rangle] \parallel$

$\prod_{i\epsilon I_1 \cup k} l_i[\overline{deliver_i}(m)] \parallel \prod_{i\epsilon I_2 \cup k} \prod_{k\epsilon I_3} l_i[\overline{send_{i,k}}(m)])\backslash K$

$\langle L^1,l\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle from, status\rangle] \parallel$

$\prod_{i\epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{i,k}}(m)])\backslash K \xrightarrow{susp\ l_0}$

$\langle L^2,k\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I_f \cup k} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i\epsilon I-k} l_i[R_i\langle from, status\rangle] \parallel$

$\prod_{i\epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel \prod_{i\epsilon I_2 \cup k} \prod_{k\epsilon I_3} l_i[\overline{send_{i,k}}(m)])\backslash K$ ή

$\langle L^2,k\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I_f \cup k} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i\epsilon I-k} l_i[R_i\langle from, status\rangle] \parallel$

$\prod_{i\epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{i,k}}(m)])\backslash K$

Επομένως η μορφή του συστήματος $\langle L^2,k\rangle \triangleright (LRB^{m+1})\backslash K$ θα διατηρηθεί η ίδια όπου $L^2 \subseteq L$ και $k \leq n$.

Έτσι αποδείξαμε με επαγωγή ότι η μορφή του συστήματος από το σημείο όπου δεν μπορούν να προκύψουν άλλα σφάλματα θα είναι:

$\langle L',0\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle from, status\rangle] \parallel$

$\prod_{i\epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{i,k}}(m)])\backslash K$

όπου $L' \subseteq L$.

Λήμμα 2. ⟨L',0⟩⊳ (LRB' || A_0) \C⇒ $\xrightarrow{\overline{ok}}$ ⇒≈0

Σύμφωνα με το Λήμμα 2 υπάρχει εκτέλεση στην οποία θα εκτελεστεί η εξωτερική ενέργεια ok και ακολούθως θα μεταφερθούμε σε τερματική κατάσταση.

Απόδειξη Λήμματος 2:

Συνεχίζοντας την ροή εκτέλεσης του συστήματος παίρνοντας μια τυχαία εκτέλεση θα δείξουμε πως μπορούμε να φτάσουμε στην τερματική κατάσταση του καινούριου συστήματος που αποτελείται από το σύστημα LRB′ παράλληλα με την διεργασία περιβλήματος. Ανάλογα με την εξέλιξη των γεγονότων υπάρχουν 2 πιθανές περιπτώσεις.

1) Περίπτωση όπου ο αποστολέας είναι εσφαλμένος και όλοι οι ζωντανοί επεξεργαστές με το πέρας του αλγορίθμου έχουν το μήνυμα.

Υποθέτουμε ότι ο αποστολέας πριν καταρρεύσει στην πρώτη φάση στέλνει το μήνυμα στον επεξεργαστή j.

⟨L',0⟩⊳$(l_0[S'] \parallel \prod_{i\epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle from, status\rangle] \parallel \prod_{i\epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel \prod_{i\epsilon I_2}\prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel A_0)$\C

Φάση 1: Οι διεργασίες $i\epsilon I$ εκτελούν την ενέργεια $susp\ l_0$

⟨L',0⟩⊳$(l_0[S'] \parallel \prod_{i\epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle from, status\rangle] \parallel \prod_{i\epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel \prod_{i\epsilon I_2}\prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel A_0)$\C$\xrightarrow{susp\ l_0}\dots\xrightarrow{susp\ l_0}$

Όλες οι ζωντανές οντότητες εντοπίζουν το σφάλμα του αποστολέα.

⟨L',0⟩⊳$(l_0[S'] \parallel \prod_{i\epsilon I_f \cup L'} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i\epsilon I-L'} l_i[R_i\langle from, status\rangle] \parallel \prod_{i\epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel \prod_{i\epsilon I_2\cup j}\prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel A_0)$\C

Φάση 2: Η διεργασία R_j στέλνει σε όλες τις οντότητες το μήνυμα m πάνω στο κανάλι που τους ενώνει.

$\langle$L',0$\rangle \triangleright (l_0[S'] \parallel \prod_{i \epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i \epsilon I} l_i[R_i\langle from, status\rangle] \parallel$

$\prod_{i \epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel \prod_{i \epsilon I_2} \prod_{k \epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$ A$_0$)\C$\xrightarrow{\tau(send_{j,0}(m))} \dots$

$\xrightarrow{\tau(send_{j,n}(m))}$

$\langle$L',0$\rangle \triangleright (l_0[S'] \parallel \prod_{i \epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i \epsilon I_1 \cup \{L'-j\}} l_i[\overline{deliver_i}(m)] \parallel$

$\prod_{i \epsilon I_2 - j \cup \{L'-j\}} \prod_{k \epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$ A$_0$)\C

Οι διεργασίες $i \epsilon I \cup I_f$ που έχουν $from = \emptyset$ θα πάρουν το μήνυμα

Φάση 3: Θα εκτελεστούν όλες οι εσωτερικές ενέργειες $send_{i,k}$

$\langle$L',0$\rangle \triangleright (l_0[S'] \parallel \prod_{i \epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i \epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel$

$\prod_{i \epsilon I_2} \prod_{k \epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$ A$_0$)\C$\xrightarrow{\tau(send_{i,j}(m))} \dots \xrightarrow{\tau(send_{j,k}(m))}$

Όλες οι ζωντανές οντότητες πλην της διεργασίας j στέλνουν σε όλες τις υπόλοιπες οντότητες στο σύστημα το μήνυμα πάνω στο κανάλι που τους ενώνει.

$\langle$L',0$\rangle \triangleright (l_0[S'] \parallel \prod_{i \epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i \epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel$

$\prod_{i \epsilon I_2 - \{L'-j\}} \prod_{k \epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$ A$_0$)\C

Φάση 4: Γίνονται οι επικοινωνίες με το περίβλημα

$\langle$L',0$\rangle \triangleright (l_0[S'] \parallel \prod_{i \epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i \epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel$

$\prod_{i \epsilon I_2} \prod_{k \epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$ A$_0$)\C$\xrightarrow{susp\ l_0}$

$\langle$L',0$\rangle \triangleright (l_0[S'] \parallel \prod_{i \epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i \epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel$

$\prod_{i \epsilon I_2} \prod_{k \epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$ A$_1$)\C$\xrightarrow{\tau(check_1(m))} \xrightarrow{\tau(receive_1(answer))}$

$\langle$L',0$\rangle \triangleright (l_0[S'] \parallel \prod_{i \epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i \epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel$

$\prod_{i \epsilon I_2} \prod_{k \epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$ B$_0$)\C$\xrightarrow{susp\ l_0}$

$\langle$L',0$\rangle \triangleright (l_0[S'] \parallel \prod_{i \epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i \epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel$

$\prod_{i \epsilon I_2} \prod_{k \epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$ B$_1$)\C$\xrightarrow{\tau(deliver_1(m))} \dots \xrightarrow{susp\ l_n}$

Σε αυτό το σημείο η διεργασία περίβλημα ελέγχει τις υπόλοιπες οντότητες μέχρι και την οντότητα με ταυτότητα n και σε κάθε μια από αυτές αν δεν εντοπίσει σφάλμα της τότε παραλαμβάνει το μήνυμα deliver από αυτή και συνεχίζει με την επόμενη.

$\langle L',0\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i\epsilon I_1 - L'} l_i[\overline{deliver_i}(m)] \parallel$

$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel B_{n+1})\backslash C \xrightarrow{\overline{ok}}$

$\langle L',0\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I_f} l_i[{R_i}^f\langle from, status\rangle] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel 0)\backslash C \approx 0$

Στην πιο πάνω περίπτωση το περίβλημα αφού παραλάβει το μήνυμα deliver από όλους τους ζωντανούς επεξεργαστές τότε βγάζει ως έξοδο το μήνυμα ok και τερματίζει. Ακολούθως το σύστημα φτάνει σε μια κατάσταση όπου όλοι περιμένουν να λάβουν μήνυμα σε κάποιο κανάλι και κανένας δεν μπορεί να στείλει και έτσι η κατάσταση του συστήματος είναι ισοδύναμη με την τερματική κατάσταση.

2) Περίπτωση όπου φτάνουμε στο σημείο που δεν μπορούν να προκύψουν άλλα σφάλματα και ο αποστολέας που υπάρχει στο σύστημα είναι ζωντανός

$\langle L',0\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I} l_i[R_i\langle from, status\rangle] \parallel \prod_{i\epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel$

$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel A_0)\backslash C$

Φάση 1: Η οντότητα αποστολέας στέλνει στις υπόλοιπες οντότητες το μήνυμα πάνω στα κανάλια που τους ενώνουν.

$\langle L',0\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I} l_i[R_i\langle from, status\rangle] \parallel \prod_{i\epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel$

$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel A_0)\backslash C \xrightarrow{\tau(send_{0,0}(m))} \dots \xrightarrow{\tau(send_{0,n}(m))}$

Η διεργασία αποστολέας στέλνει σε όλες τις οντότητες το μήνυμα πάνω στο κανάλι που τους ενώνει.

$\langle L',0\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I} l_i[R_i\langle from, status\rangle] \parallel \prod_{i\epsilon I_1 \cup L'} l_i[\overline{deliver_i}(m)] \parallel$

$\prod_{i\epsilon I_2 - 0} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel A_0)\backslash C$

Οι διεργασίες $i\epsilon I \cup I_f$ που έχουν $from = \emptyset$ θα πάρουν το μήνυμα

Φάση 2: Γίνονται οι επικοινωνίες με το περίβλημα

$\langle L',0\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I} l_i[R_i\langle from, status\rangle] \parallel \prod_{i\epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel$

$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel A_0)\backslash C \xrightarrow{\tau(check_0(m))} \xrightarrow{\tau(receive_0(answer))}$

$\langle L',0\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I} l_i[R_i\langle from, status\rangle] \parallel \prod_{i\epsilon I_1} l_i[\overline{deliver_i}(m)] \parallel$

$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel B_0)\backslash C \xrightarrow{\tau(deliver_0(m))} \dots \xrightarrow{\tau(deliver_n(m))}$

Στο σημείο αυτό η διεργασία περίβλημα ελέγχει όλες τις οντότητες μέχρι την n και αν δεν εντοπίσει σφάλμα μιας διεργασίας τότε λαμβάνει το μήνυμα deliver από την συγκεκριμένη διεργασία και προχωρά με την επόμενη στην σειρά.

$\langle L',0\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I} l_i[R_i\langle from, status\rangle] \parallel \prod_{i\epsilon I_1 - L'} l_i[\overline{deliver_i}(m)] \parallel$

$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel B_{n+1})\backslash C \xrightarrow{\overline{ok}}$

$\langle L',0\rangle \triangleright (l_0[S'] \parallel \prod_{i\epsilon I} l_i[R_i\langle from, status\rangle] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel 0)\backslash C \approx 0$

Αφού ο αποστολέας είναι ζωντανός τότε θα καταφέρει να στείλει το μήνυμα σε όλες τις οντότητες πάνω στα κανάλια $send_{0,i}$ και κατά συνέπεια όσες οντότητες είναι ζωντανές από αυτό το σημείο και μετά θα κάνουν όλες deliver το μήνυμα. Όταν το περίβλημα λάβει το μήνυμα deliver από όλους τους ζωντανούς θα εκτελέσει την ενέργεια $\overline{ok}$ και θα τερματίσει. Το σύστημα θα φτάσει σε μια κατάσταση όπου όλες οι ζωντανές οντότητες θα περιμένουν να λάβουν μήνυμα σε κάποιο κανάλι και κανένας δεν θα μπορεί να στείλει. Το σύστημα είναι ισοδύναμο με την τερματική διεργασία.

Λήμμα 3. $\langle L',0\rangle \triangleright$ (LRB'||A_0) \C είναι confluent

Απόδειξη Λήμματος 3:

Καταρχήν θα δείξουμε ότι τα πιο κάτω αθροίσματα στις καταστάσεις R_i και ${R_i}^f$ αντίστοιχα είναι συρρέουσες διεργασίες

1)

$$\sum_{0\le k\le n} send_{i,k}(m).R_i^1\langle m, from, status\rangle \; + \; check_i(m).cond(m \in from \Rightarrow \overline{receive}_\iota(yes) \parallel R_i\langle from, status\rangle, \; m \notin from \Rightarrow \overline{receive}_\iota(no) \parallel R_i\langle from, status\rangle)$$

2)

$$\sum_{0\le k\le n} send_{i,k}(m).R_i^1\langle m, from, status\rangle \; +$$

$$check_i(m).cond(m \in from \Rightarrow \overline{receive}_\iota(yes) \;||\; R_i^f\langle from, status\rangle,$$

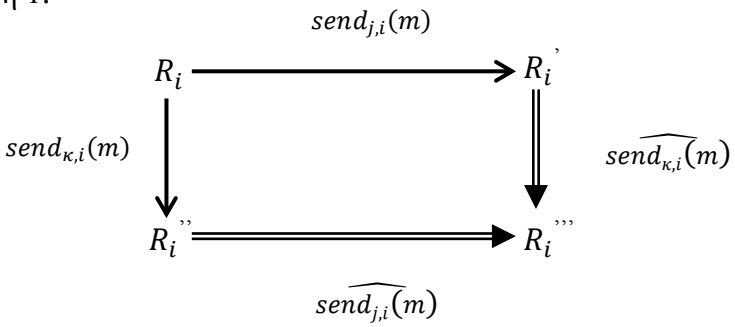
$$m \notin from \Rightarrow \overline{receive}_\iota(no) \;||\; R_i^f\langle from, status\rangle)$$

Ξεκινούμε με το (1)

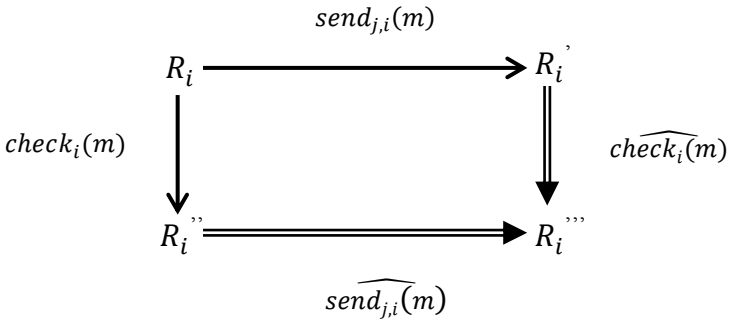
Για να δείξουμε ότι το άθροισμα
$\sum_{0\le k\le n} send_{i,k}(m).R_i^1\langle m, from, status\rangle \; +$
$check_i(m).cond(m \in from \Rightarrow \overline{receive}_\iota(yes) \;||\; R_i\langle from, status\rangle,$
$m \notin from \Rightarrow \overline{receive}_\iota(no)) \;||\; R_i\langle from, status\rangle)$

στην κατάσταση R_i είναι συρρέουσα διεργασία θα πρέπει να δείξουμε ότι

Περίπτωση 1:



Περίπτωση 2:



Απόδειξη περίπτωσης 1:

Σύμφωνα με την μοντελοποίηση του αλγορίθμου όποια ενέργεια από τις δύο εκτελεστεί στην κατάσταση R_i τότε από την κατάσταση R_i^1 μπορεί να επανέλθει πίσω στην αρχική κατάσταση R_i όπου μπορεί να εκτελεστεί η δεύτερη ενέργεια ικανοποιώντας τον ορισμό της συρροής.

Απόδειξη περίπτωσης 2:

Αν εκτελεστεί η ενέργεια $send_{j,i}(m)$ τότε ακολούθως από την κατάσταση R_i^1 μπορεί να επανέλθει πίσω στην κατάσταση R_i όπου θα μπορεί να εκτελεστεί η ενέργεια $check_i(m)$. Αν εκτελεστεί η ενέργεια $check_i(m)$ στην κατάσταση R_iακολούθως θα μεταβεί πίσω στην R_i όπου θα μπορεί να εκτελεστεί η ενέργεια $send_{j,i}(m)$..

Συνεχίζουμε με το (2)

Για να δείξουμε ότι το άθροισμα

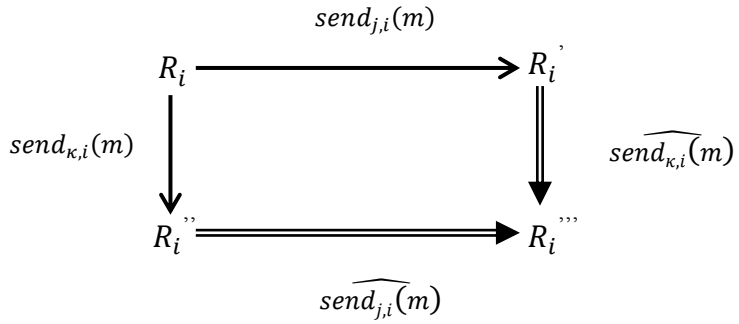
$\sum_{0\le k\le n} send_{i,k}(m).R_i^1\langle m, from, status\rangle$ +

$check_i(m_j).cond(m \in from \Rightarrow \overline{receive}_\iota(yes) \,||\, R_i^f\langle from, status\rangle,$

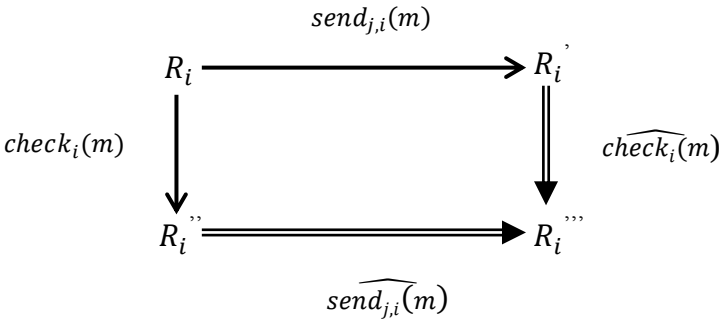
$m \notin from \Rightarrow \overline{receive}_\iota(no) \,||\, R_i^f\langle from, status\rangle)$

στην κατάσταση $R_i{}^f$ είναι συρρέουσα διεργασία θα πρέπει να δείξουμε ότι

Περίπτωση 1:



Περίπτωση 2:



Απόδειξη περίπτωσης 1:

Σύμφωνα με την μοντελοποίηση του αλγορίθμου όποια ενέργεια από τις δύο εκτελεστεί στην κατάσταση $R_i{}^f$ τότε από την κατάσταση R_i^1 μπορεί να επανέλθει πίσω στην

κατάσταση $R_i{}^f$ όπου μπορεί να εκτελεστεί η δεύτερη ενέργεια ικανοποιώντας τον ορισμό της συρροής.

Απόδειξη περίπτωσης 2:
Παρατηρούμε ότι στην κατάσταση $R_i{}^f$ αν εκτελεστεί η ενέργεια $send_{j,i}(m)$ τότε μπορεί να μεταβεί πίσω στην κατάσταση $R_i{}^f$ είτε από την κατάσταση R_i^1 είτε από την κατάσταση R_i^2 όπου μπορεί να εκτελεστεί η ενέργεια $check_i(m)$. Αν εκτελεστεί η ενέργεια $check_i(m)$ στην κατάσταση $R_i{}^f$ τότε ακολούθως θα επιστρέψει πίσω στην κατάσταση $R_i{}^f$ όπου μπορεί να εκτελεστεί η ενέργεια $send_{j,i}(m)$.

Συνεχίζουμε δείχνοντας ότι κάθε μια από τις καταστάσεις του συστήματος αποτελεί συρρέουσα διεργασία.
Για να δείξουμε ότι το σύστημα μας είναι confluent θα ελέγξουμε μια-μια τις καταστάσεις του συστήματος.

$A_{n+1}(m) = \overline{ok}.0$

Η διεργασία $A_{n+1}(m)$ κάνει χρήση του τελεστή της διαδοχής η απόδειξη του οποίου έγινε στο υποκεφάλαιο 3.5.

$B_{n+1}(m) = \overline{ok}.0$

Η διεργασία $B_{n+1}(m)$ κάνει χρήση του τελεστή της διαδοχής η απόδειξη του οποίου έγινε στο υποκεφάλαιο 3.5.

$B_i(m) = suspl_i: \left(B_{i+1}(m)\ ;\ deliver_i(m).B_{i+1}(m)\right)$

Η διεργασία $B_i(m)$ κάνει χρήση του τελεστή susp και του ακολουθιακού τελεστή οι σχετικές αποδείξεις των οποίων έγιναν στο υποκεφάλαιο 3.5.

$A_i(m) = suspl_i: (A_{i+1}(m)\ ;\ \overline{check_i}(m).receive_i(answer).$

$cond(answer = no \Rightarrow A_{i+1}(m),$

$answer = yes \Rightarrow B_0(m)))$

Η διεργασία $A_i(m)$ κάνει χρήση του τελεστή $susp$, του ακολουθιακού τελεστή και του τελεστή συνθήκης οι σχετικές αποδείξεις των οποίων έγιναν στο υποκεφάλαιο 3.5.

$$S\langle m, from\rangle = \prod_{0\le i\le n} \overline{send_{0,i}}(m) \;||\; send_{0,0}(m).\overline{deliver_0}(m) \;||$$
$$check_0(m).\overline{receive}_0(\text{yes})$$

Η διεργασία S βλέπουμε ότι αποτελεί σύνθεση διεργασιών με τον τελεστή του ταυτοχρονισμού και της διαδοχής για τους οποίους αποδείξαμε στο υποκεφάλαιο 3.5 ότι διατηρούν την έννοια της συρροής.

$$R_i\langle from, status\rangle = susp\ l_0 : (R_i^3\langle from\rangle\ ; \sum_{0\le k\le n} send_{i,k}(m).R_i^1\langle m, from, status\rangle) +$$
$$check_i(m).cond(m \in from \Rightarrow \overline{receive}_i(yes) \;||\; R_i\langle from, status\rangle,$$
$$m \notin from \Rightarrow \overline{receive}_i(no) \;||\; R_i\langle from, status\rangle)$$

Η διεργασία R_i βλέπουμε ότι αποτελεί σύνθεση διεργασιών με τον τελεστή της επιλογής, της διαδοχής, της συνθήκης, του ταυτοχρονισμού και του τελεστή susp. Για τους τελευταίους τέσσερις αποδείξαμε στο υποκεφάλαιο 3.5 ότι διατηρούν την έννοια της συρροής ενώ για τον τελεστή της επιλογής η σχετική απόδειξη έγινε στο υποκεφάλαιο 5.2 για τον αλγόριθμο LRB. Για τις διεργασίες R_i^1 και R_i^3 θα δείξουμε στη συνέχεια ότι είναι συρρέουσες.

$$R_i^1\langle m, from, status\rangle = cond(m \notin from \;\&\&\; status = true \Rightarrow \overline{deliver_i}(m) \;||$$
$$R_i\langle from \cup \{m\}, status\rangle$$
$$m \notin from \;\&\&\; status = false \Rightarrow \overline{deliver_i}(m) \;||$$
$$R_i^2\ \langle m, from \cup \{m\}, status\rangle,$$
$$m \in from \;\&\&\; status = true \Rightarrow R_i\langle from, status\rangle$$
$$m \in from \;\&\&\; status = false \Rightarrow R_i^f\langle from, status\rangle)$$

Η διεργασία R_i^1 βλέπουμε ότι αποτελεί σύνθεση διεργασιών με τον τελεστή της συνθήκης και του ταυτοχρονισμού. Οι σχετικές αποδείξεις για τους δύο τελεστές έγιναν στο υποκεφάλαιο 3.5. Για την διεργασία R_i^f δείχνουμε στη συνέχεια ότι είναι συρρέουσα διεργασία όπως και για την διεργασία R_i^2.

$$R_i^2\ \langle m, from, status\rangle = \prod_{0\le k\le n} \overline{send_{i,k}}(m) \;||\; R_i^f\langle from, status\rangle$$

Η διεργασία R_i^2 κάνει χρήση του τελεστή ταυτοχρονισμού για τον οποίο αποδείξαμε ότι διατηρεί την έννοια της συρροής στο υποκεφάλαιο 3.5. Για την διεργασία R_i^f δείχνουμε στη συνέχεια ότι είναι συρρέουσα διεργασία.

$$R_i^3\ \langle from\rangle = cond(m \in from \Rightarrow R_i^2\langle m, from, false\rangle, m \notin from \Rightarrow R_i^f\langle from, false\rangle)$$

Η διεργασία R_i^3 κάνει χρήση του τελεστή συνθήκης για τον οποίο αποδείξαμε ότι διατηρεί την έννοια της συρροής στο υποκεφάλαιο 3.5. Για την διεργασία R_i^f δείχνουμε στη συνέχεια ότι είναι συρρέουσα διεργασία.

$$R_i^f\langle from, status\rangle = \sum_{0\le k\le n} send_{i,k}(m).\, R_i^1\langle m, from, status\rangle + check_i(m).\, cond(m \in from \Rightarrow \overline{receive}_i(yes) \,||\, R_i^f\langle from, status\rangle,\ m \notin from \Rightarrow \overline{receive}_i(no) \,||\, R_i^f\langle from, status\rangle)$$

Η διεργασία R_i^f κάνει χρήση των τελεστών επιλογής, διαδοχής, συνθήκης και ταυτοχρονισμού όπου η σχετική απόδειξη του πρώτου έγινε στο υποκεφάλαιο 5.2 για τον αλγόριθμο LRB και των υπόλοιπων στο υποκεφάλαιο 3.5.

$$\text{(LRB')}\backslash K = (l_0[S'] \,||\, \prod_{i\epsilon I_f} l_i[R_i{}^f\langle from, status\rangle] \,||\, \prod_{i\epsilon I} l_i[R_i\langle from, status\rangle] \,||\, \prod_{i\epsilon I_1} l_i[\overline{deliver_i}(m)] \,||\, \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{i,k}}(m)])\backslash K$$

Η διεργασία LRB' αποτελεί σύνθεση διεργασιών με τον τελεστή ταυτοχρονισμού για τον οποίο αποδείξαμε στο υποκεφάλαιο 3.5 ότι διατηρεί την έννοια της συρροής. Οι διεργασίες μεταξύ τους δεν έχουν κοινά κανάλια εισόδου ή εξόδου.

(LRB'||A_0)\C: Η διεργασία αποτελεί σύνθεση δύο διεργασιών με τον τελεστή του ταυτοχρονισμού η απόδειξη του οποίου έγινε στο υποκεφάλαιο 3.5. Οι διεργασίες μεταξύ τους δεν έχουν κοινά κανάλια εισόδου ή εξόδου.

Για να αποδείξουμε ότι το σύστημα ⟨L',0⟩⊳(LRB' || A_0)\C είναι confluent βλέπουμε ότι το σύστημα αποτελεί σύνθεση πολλών συρρεόντων διεργασιών και επίσης κάθε κανάλι μέσα στο σύστημα μας είναι μοναδικό και κάθε δύο οντότητες ενώνονται πάνω σε ένα μοναδικό κανάλι. Έτσι κάθε πιθανή εκτέλεση του αλγορίθμου μπορεί να μας δώσει το ζητούμενο αποτέλεσμα.

Απόδειξη θεωρήματος 1

Το πιο πάνω σύστημα πάντοτε θα τερματίζει βγάζοντας ως έξοδο το μήνυμα $\overline{ok}$ και ακολούθως τερματίζει. Επομένως τα δύο συστήματα συνδέονται με σχέση ασθενής διπροσομοίωσης. ⟨L',0⟩⊳ (LRB' || A_0)\C ≈ $\overline{ok}.0$.

Η τερματική κατάσταση στην οποία τερματίζει το σύστημα θα είναι

⟨L',0⟩⊳ (LRB'' || 0)\C όπου

⟨L',0⟩⊳(LRB'')\K=⟨L',0⟩⊳($l_0[S']$ || $\prod_{i\epsilon I_f} l_i[{R_i}^f\langle from, status\rangle]$ ||

$\prod_{i\epsilon I_2}\prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)]$)\K ή

⟨L',0⟩⊳(LRB'')\K=⟨L',0⟩⊳($l_0[S']$ || $\prod_{i\epsilon I_f} l_i[R_i\langle from, status\rangle]$ ||

$\prod_{i\epsilon I_2}\prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)]$)\K

Οι ενέργειες $\prod_{i\epsilon I_2}\prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)]$ αποτελούν τις ενέργειες στις οποίες δεν έγινε ο συγχρονισμός για τον λόγο ότι είτε ο επεξεργαστής i είτε ο επεξεργαστής k κατέρρευσαν.

Μπορούμε να γενικοποιήσουμε την απόδειξη μας και στην περίπτωση που υπάρχουν περισσότερα από ένα μηνύματα στο σύστημα μας. Επίσης η πιο πάνω απόδειξη που έγινε μπορούμε να πούμε ότι ισχύει για ένα τυχαίο αριθμό σφαλμάτων άρα μπορούμε να πούμε ότι ισχύει για οποιοδήποτε αριθμό σφαλμάτων.

Κεφάλαιο 6

Μοντελοποίηση και Απόδειξη Αλγορίθμου URB

6.1 Μοντελοποίηση αλγορίθμου

$URB\langle L, n\rangle = (l_0[S\langle m, \langle\,\rangle, \Pi, \langle\,\rangle\rangle] \,||\, l_1[R_1\langle \varepsilon, \langle\,\rangle, \Pi, \langle\,\rangle\rangle] \,||\, \dots \,||\, l_n[R_n\langle \varepsilon, \langle\,\rangle, \Pi, \langle\,\rangle\rangle])\backslash K$

K ={send$_{0,0}$, send$_{0,1}$,…, send$_{1,1}$,…, send$_{1,n}$,… send$_{n,n}$}

$S\langle m, p, c, d, ack\rangle = \prod_{0\le k\le n} \overline{send_{0,k}}\,(m)\,||$

` $(\sum_{0\le k\le n} send_{i,k}(m).$

$cond(m \notin d \wedge c \subseteq ack \cup \{p_k\} \Rightarrow \overline{deliver_0}(m) \,||\, 0$

$c \not\subseteq ack \cup \{p_k\} \Rightarrow A_0^0\langle p, c, d, ack \cup \{p_k\}\rangle)))$

$R_i\langle p, c, d, ack\rangle = \sum_{0\le k\le n} send_{i,k}(m).$

$cond(m \notin p \Rightarrow \prod_{0\le k\le n} \overline{send_{i,k}}\,(m) \,||\, R_i\langle p \cup \{m\}, c, d, ack \cup \{p_k\}\rangle,$

$m \notin d \wedge c \subseteq ack \cup \{p_k\} \Rightarrow \overline{deliver_i}(m) \,||\, 0$

$c \not\subseteq ack \cup \{p_k\} \Rightarrow A_i^0\langle p, c, d, ack \cup \{p_k\}\rangle)$

$A_i^j\langle p, c, d, ack \cup \{p_k\}\rangle = susp\ l_j\colon (A_i^{j+1}\langle p, c - \{p_j\}, d, ack\rangle\ ;\ A_i^{j+1}\langle p, c, d, ack\rangle), 0 \le j \le n$

$A_i^{n+1} = cond(i = 0 \Rightarrow S\langle p, c, d, ack\rangle, i\ != 0 \Rightarrow R_i\langle p, c, d, ack\rangle)$

R={R$_1$, R$_2$,…, R$_n$}

Επεξήγηση συντομογραφιών
l_i: η τοποθεσία στην θέση i.
m: το μήνυμα m.
p: η λίστα pending που κρατά τα μηνύματα που δεν έγιναν ακόμη deliver.
c: η λίστα correct περιέχει το σύνολο των ζωντανών επεξεργαστών.
d: η λίστα delivered που περιέχει τα μηνύματα που έγιναν deliver.
p_i: ο επεξεργαστής με ταυτότητα i.
ack: η λίστα acknowledge για το μήνυμα m που περιέχει τους επεξεργαστές οι οποίοι έλαβαν ήδη το μήνυμα.
S: η διεργασία της οντότητας αποστολέας.
R_i: η διεργασία της οντότητας παραλήπτης με ταυτότητα i.
$send_{i,k}$: το κανάλι που ενώνει τις διεργασίες i και k.
$deliver_i$: το κανάλι που ενώνει την οντότητα i με το εξωτερικό περιβάλλον του συστήματος.

Επεξήγηση μοντελοποίησης αλγορίθμου
Το σύστημα μας μπορεί να ανεχτεί μέχρι και n σφάλματα όπου n ο αριθμός των οντοτήτων που υπάρχουν στο σύστημα μας. Επίσης κάθε διεργασία ανήκει σε μια μοναδική τοποθεσία l_i όπου i η ταυτότητα του κάθε επεξεργαστή. Η ταυτότητα του αποστολέα είναι το 0 και του κάθε παραλήπτη είναι ένας μοναδικός αριθμός από το 1 μέχρι το n. Αρχικά οι τοποθεσίες όλων των οντοτήτων βρίσκονται στο σύνολο L που περιέχει τις τοποθεσίες των επεξεργαστών που είναι ζωντανοί. Οι μεταβλητές αυτές είναι global και μεταβάλλονται δυναμικά κάθε φορά που συμβαίνει ένα σφάλμα στο σύστημα μας. Η οντότητα αποστολέας περιέχει το πρώτο μήνυμα στην τοπική μνήμη ενώ οι παραλήπτες δεν περιέχουν κανένα μήνυμα στην τοπική τους μνήμη αφού δεν έλαβαν ακόμα κάποιο. Επίσης οι λίστες delivered, pending και ack αρχικά είναι κενές για όλες τις οντότητες και η λίστα correct περιέχει όλες τις οντότητες.

Ξεκινώντας η διεργασία αποστολέας στέλνει το μήνυμα στις υπόλοιπες οντότητες πάνω στα κανάλια που τους ενώνουν μεταξύ τους. Επιπλέον ο ίδιος παράλληλα μπορεί να λάβει το μήνυμα m πάνω σε κάποιο από τα κανάλια που τον ενώνουν με τις υπόλοιπες οντότητες και ακολούθως εκτελεί τον έλεγχο αν το μήνυμα που παρέλαβε δεν ανήκει στην λίστα delivered και όλοι οι ζωντανοί επεξεργαστές έλαβαν ήδη το συγκεκριμένο μήνυμα τότε εκτελεί την εξωτερική ενέργεια $\overline{deliver_0}(m)$ και παράλληλα τερματίζει. Για να γίνει αυτό ελέγχει την τοπική του λίστα ack που περιέχει τους επεξεργαστές που έλαβαν ήδη το συγκεκριμένο μήνυμα. Αν όμως δεν ισχύει η πιο πάνω συνθήκη τότε συνεχίζει σύμφωνα με την διεργασία A_0^0 προσθέτοντας στην λίστα ack την ταυτότητα του επεξεργαστή από τον οποίο έλαβε το μήνυμα.

Η διεργασία του παραλήπτη στην αρχική της κατάσταση μπορεί να λάβει μήνυμα πάνω σε κάποιο κανάλι και σε περίπτωση που λάβει κάποιο μήνυμα m ελέγχει αν το μήνυμα ανήκει ήδη στην λίστα pending και αν δεν ανήκει που σημαίνει το πήρε για πρώτη φορά τότε το κάνει broadcast στο σύστημα και παράλληλα επιστρέφει στην αρχική της κατάσταση προσθέτοντας στην λίστα pending το μήνυμα m και στην λίστα ack τον επεξεργαστή από τον οποίο έλαβε το μήνυμα. Αν δεν ισχύει η πιο πάνω συνθήκη τότε ελέγχει αν το μήνυμα δεν ανήκει ήδη στην λίστα delivered και όλοι οι ζωντανοί επεξεργαστές έλαβαν ήδη το μήνυμα τότε εκτελεί την εξωτερική ενέργεια $\overline{deliver_i}(m)$ και παράλληλα τερματίζει. Σε περίπτωση που δεν έλαβαν ακόμη το μήνυμα όλοι οι ζωντανοί επεξεργαστές τότε συνεχίζει σύμφωνα με την διεργασία A_i^0 προσθέτοντας στην λίστα ack την ταυτότητα του επεξεργαστή από τον οποίο έλαβε το μήνυμα.

Η διεργασία A_i^j ξεκινώντας από την αρχή ελέγχει μια-μια τις τοποθεσίες των διεργασιών και σε περίπτωση που εντοπίσει σφάλμα κάποιας την βγάζει από την λίστα c που περιέχει τις ζωντανές οντότητες και συνεχίζει με την επόμενη. Όταν τις ελέγξει όλες επιστρέφει στην αρχική της κατάσταση.

6.2 Απόδειξη ορθότητας αλγορίθμου

Η μεθοδολογία η οποία θα χρησιμοποιήσουμε είναι η ίδια με αυτή που χρησιμοποιήθηκε για τον αλγόριθμο LRB όπου θα πρέπει να ορίσουμε την επιθυμητή συμπεριφορά του συστήματος και ακολούθως να αποδείξουμε ότι το σύστημα σε κάθε πιθανή εκτέλεση μπορεί να παράξει το επιθυμητό αποτέλεσμα. Για να το καταφέρουμε αυτό ορίζουμε πάλι μια διεργασία περιβλήματος ως εξής:

Στον αλγόριθμο αυτό ορίζουμε το περίβλημα με δύο φάσεις όπου η πρώτη θα τρέξει παράλληλα με το σύστημα από την αρχή μέχρι το σημείο από το οποίο δεν μπορούν να προκύψουν άλλα σφάλματα. Η δεύτερη φάση θα τρέξει από το σημείο αυτό παράλληλα με το σύστημα μέχρι το σημείο που θα τερματίσει η διεργασία περίβλημα δίνοντας μας ως απάντηση το μήνυμα ok.

<u>1η φάση</u>

Αρχικά list=∅

$A\langle list\rangle = \sum_{0\le k\le n} deliver_k(m).A\langle list \cup \{p_k\}\rangle$

<u>2η φάση</u>

$B\langle list\rangle = cond(list = \emptyset \Rightarrow C_0(list), list\ != \emptyset \Rightarrow D_0\langle list\rangle)$

$C_i\langle list\rangle = suspl_i: (C_{i+1}(m)\ ;\ \overline{check_\iota}(m).receive_i(answer).$
$cond(answer = no \Rightarrow C_{i+1}(m),$
$\quad answer = yes \Rightarrow D_0(m))), 0 \le i \le n$
$C_{n+1}\langle list\rangle = \overline{ok}.0$

$D_i\langle list\rangle = susp\ l_i: (D_{i+1}\langle list\rangle\ ;\ cond(p_i \in list \Rightarrow D_{i+1}\langle list\rangle,$
$p_i \notin list \Rightarrow deliver_i(m).D_{i+1}\langle list\rangle)$,0≤i≤n
$D_{n+1}\langle list\rangle = \overline{ok}.0$

Το περίβλημα έχει ως παράμετρο την λίστα list όπου αρχικά είναι κενή. Στην πρώτη φάση η διεργασία για κάθε μήνυμα που παίρνει $deliver_k(m)$ απλά προσθέτει τον επεξεργαστή k ο οποίος παρέλαβε το μήνυμα στην λίστα list. Στην δεύτερη φάση αρχικά ελέγχει αν η λίστα list είναι κενή τότε μεταβαίνει σε μια καινούρια κατάσταση όπου για κάθε ένα από τους επεξεργαστές αν δεν είναι εσφαλμένος του στέλνει ένα μήνυμα check και παίρνει πίσω μια απάντηση η οποία δηλώνει αν ο επεξεργαστής έχει ή όχι το μήνυμα. Αν δεν το έχει συνεχίζει με την επόμενη στην σειρά οντότητα. Αν φτάσει μέχρι το τέλος και κανένας από τους ζωντανούς δεν το έχει τότε βγάζει προς τα έξω ok και τερματίζει. Αν κάποιος από τους ζωντανούς επεξεργαστές έχει το μήνυμα τότε μεταβαίνει σε μια κατάσταση όπου περιμένει από όλους τους ζωντανούς επεξεργαστές να κάνουν deliver το μήνυμα με τη σειρά και ακολούθως τερματίζει δίνοντας μας ok. Αν η λίστα αρχικά δεν είναι κενή τότε ξεκινώντας από την αρχή για κάθε ένα από τους επεξεργαστές εκτελεί τον εξής έλεγχο: Αν ο επεξεργαστής είναι εσφαλμένος τότε συνεχίζει με τον επόμενο στην σειρά. Σε περίπτωση που δεν είναι εσφαλμένος ελέγχει αν ανήκει στην λίστα list τότε σημαίνει πήρε ήδη το μήνυμα deliver από αυτόν και άρα συνεχίζει με τον επόμενο στην σειρά. Αν δεν ανήκει στην λίστα τότε περιμένει μέχρι να πιάσει μήνυμα deliver από αυτόν και μετά να συνεχίσει με τον επόμενο. Αφού ελέγξουμε όλες τις οντότητες τότε είμαστε σίγουροι ότι όλοι οι ζωντανοί επεξεργαστές παρέλαβαν το μήνυμα και το περίβλημα τερματίζει δίνοντας μας ως μήνυμα επιβεβαίωσης το ok.

Το περίβλημα με βάση τον τρόπο λειτουργίας του μας εγγυάται ότι αν οποιοσδήποτε επεξεργαστής κάνει deliver το μήνυμα τότε ο κάθε ζωντανός επεξεργαστής θα πρέπει να κάνει deliver το μήνυμα με το πέρας του αλγορίθμου. Αυτό ικανοποιεί την συνθήκη συμφωνίας του αλγορίθμου URB και άρα το ok που μας δίνει σαν αποτέλεσμα το περίβλημα αποδεικνύει ορθότητα του αλγορίθμου.

Στην μοντελοποίηση του αλγορίθμου προσθέτουμε τις ενέργειες οι οποίες χρησιμοποιούνται για επικοινωνία του συστήματος με την διεργασία περιβλήματος όπου ο αποστολέας αν πιάσει μήνυμα ανά πάσα στιγμή στο κανάλι $check_0$ με παράμετρο το μήνυμα m τότε θα πρέπει να απαντά πάντα με yes στο κανάλι $receive_0$. Ο παραλήπτης αν πιάσει μήνυμα στο κανάλι $check_i$ με παράμετρο το μήνυμα m και βρίσκεται στην κατάσταση όπου δεν έκανε ακόμη deliver το μήνυμα τότε ελέγχει αν το

έχει στη λίστα pending απαντά με yes διαφορετικά απαντά με no. Αν βρίσκεται σε κατάσταση στην οποία έκανε ήδη deliver το μήνυμα τότε απαντά με yes.

$$URB\langle L,n\rangle = (l_0[S\langle m,\langle\rangle,\Pi,\langle\rangle\rangle] \,||\, l_1[R_1\langle\varepsilon,\langle\rangle,\Pi,\langle\rangle\rangle] \,||\, \dots \,||\, l_n[R_n\langle\varepsilon,\langle\rangle,\Pi,\langle\rangle\rangle])\backslash K$$

K={send$_{0,0}$, send$_{0,1}$,…, send$_{1,1}$,…, send$_{1,n}$,… send$_{n,n}$}

$$S\langle m,p,c,d,ack[m]\rangle = \prod_{0\le k\le n}\overline{send_{0,k}}(m)\,||$$
$$(\sum_{0\le k\le n} send_{0,k}(m).$$
$$cond(m \notin d \wedge c \subseteq ack(m)\cup\{p_k\} \Rightarrow \overline{deliver_i}(m) \,\|\, \mathbf{0},$$
$$c \not\subseteq ack\cup\{p_k\} \Rightarrow A_0^0\langle p,c,d,ack\cup\{p_k\}\rangle)$$
$$||\ \boldsymbol{check_0(m).\overline{receive_0}(yes)})$$

$$R_i\langle p,c,d,ack\rangle = \sum_{0\le k\le n} send_{i,k}(m).$$
$$cond(m \notin p \Rightarrow \prod_{0\le k\le n}\overline{send_{i,k}}(m) \,||\, R_i\langle p\cup\{m\},c,d,ack\cup\{p_k\}\rangle,$$
$$m \notin d \wedge c \subseteq ack\cup\{p_k\} \Rightarrow \overline{deliver_i}(m) \,\|\, \boldsymbol{R_i^1},$$
$$c \not\subseteq ack\cup\{p_k\} \Rightarrow A_i^0\langle p,c,d,ack\cup\{p_k\}\rangle)$$
$$+\ \boldsymbol{check_i(m).cond(m \in p \Rightarrow \overline{receive_0}(yes) \,||\, R_i\langle p,c,d,ack\rangle,}$$
$$\boldsymbol{m \notin p \Rightarrow \overline{receive_0}(no) \,||\, R_i\langle p,c,d,ack\rangle)}$$

$$\boldsymbol{R_i^1 = check_i(m).\overline{receive_i}(yes).R_i^1 + \sum_{0\le k\le n} send_{i,k}(m).R_i^1}$$

$$A_i^j\langle p,c,d,ack\rangle = susp\ l_j\colon (A_i^{j+1}\langle p,c-\{p_j\},d,ack\rangle\ ;\ A_i^{j+1}\langle p,c,d,ack\rangle), 0 \le j \le n$$

$$A_i^{n+1} = cond(i = 0 \Rightarrow S\langle p,c,d,ack\rangle, i\ != 0 \Rightarrow R_i\langle p,c,d,ack\rangle)$$

Σκοπός μας είναι να δείξουμε ότι το σύστημα ⟨L',0⟩⊳ (URB' || B⟨list⟩)\C σε κάθε πιθανή εκτέλεση θα τερματίζει και θα μας δίνει ok στην έξοδο.

Θεώρημα 2: Αν ⟨L,n⟩⊳(URB || A[list])\C⇒⟨L',0⟩⊳(URB'|| A[list])\C τότε
⟨L',0⟩⊳(URB' || B⟨list⟩)\C ≈ $\overline{ok}.0$
Το θεώρημα δηλώνει πως αν URB' μια τυχαία κατάσταση του συστήματος από την οποία δεν προκύπτουν άλλα σφάλματα στο σύστημα τότε αν τρέξουμε το σύστημα παράλληλα με την δεύτερη φάση της διεργασίας περίβλημα θα εκτελεστεί η εξωτερική ενέργεια ok και ακολούθως το καινούριο σύστημα θα μεταβεί σε τερματική κατάσταση.

Ορίζουμε το σύνολο C ως το σύνολο το οποίο περιέχει τα κανάλια τα οποία είναι δεσμευμένα για επικοινωνία εντός του συστήματος και τα οποία είναι τα κανάλια $deliver_i$, $check_i$, $receive_i$, $send_{0,0}$-$send_{n,n}$, ως A⟨list⟩ την αρχική κατάσταση της διεργασίας περιβλήματος στην α' φάση εκτέλεσης της και ως B⟨list⟩ την αρχική κατάσταση της διεργασίας περιβλήματος στην β' φάση εκτέλεσης της. Η λίστα list για την δεύτερη φάση εκτέλεσης είναι η ίδια λίστα που προέρχεται από την α' φάση εκτέλεσης.

Για να αποδείξουμε το ζητούμενο η απόδειξη που θα κάνουμε θα χωριστεί σε δύο μέρη Αρχικά θα πρέπει να αποδείξουμε ότι το πιο πάνω σύστημα είναι ικανό να παράξει το ζητούμενο αποτέλεσμα και μετά να τερματίσει. Στη συνέχεια δείχνουμε ότι το σύστημα είναι confluent για να αποδείξουμε ότι οποιαδήποτε εκτέλεση του αλγορίθμου θα μας δώσει το επιθυμητό αποτέλεσμα.

Η μεθοδολογία η οποία θα ακολουθήσουμε είναι η ίδια με αυτή που χρησιμοποιήσαμε στον αλγόριθμο LRB. Αρχικά θα δείξουμε με την μέθοδο της επαγωγής ότι το σύστημα (URB || A[list])\C ξεκινώντας από την αρχική του κατάσταση όταν θα φτάσει στο σημείο από το οποίο δεν μπορούν να προκύψουν άλλα σφάλματα θα έχει μια συγκεκριμένη μορφή και ακολούθως όταν τρέξουμε το σύστημα παράλληλα με την δεύτερη φάση της διεργασίας περίβλημα θα δείξουμε ότι υπάρχει εκτέλεση η οποία μας δίνει το ζητούμενο αποτέλεσμα βγάζοντας προς τα έξω ok. Μετά θα δείξουμε ότι το καινούριο σύστημα που αποτελείται από το σύστημα URB' παράλληλα με την δεύτερη φάση της διεργασίας περίβλημα είναι confluent και έτσι οποιαδήποτε τυχαία εκτέλεση μπορεί να μας δώσει το επιθυμητό αποτέλεσμα. Η απόδειξη που θα κάνουμε είναι για ένα μήνυμα και μπορεί να γενικοποιηθεί και για πολλά μηνύματα.

Ορίζουμε I_0 το σύνολο των επεξεργαστών που βρίσκονται στην αρχική τους κατάσταση R_i και τρέχουν παράλληλα και ως I_s το σύνολο των επεξεργαστών που βρίσκονται στην κατάσταση R_i^s και τρέχουν παράλληλα.

Λήμμα 4. Το ⟨L',0⟩⊳(URB' || A[list])\C θα έχει την πιο κάτω μορφή

⟨L',0⟩⊳(URB' || A[list])\C = ⟨L',0⟩⊳ $(l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel \prod_{\iota\epsilon I_4} l_i[\overline{deliver_i}(m)]$ || A[list])\C

όπου $I \cup I_1 = R$, $I \cap I_1 = \emptyset$,

$I_2 \subseteq \{i \mid m \in pending_i\}$,

$I_3 = R \supseteq \{j \mid m \notin pending_j\}$

και το S μπορεί να έχει μια από τις πιο κάτω μορφές

$\prod_{0\leq k\leq n} \overline{send_{0,k}}\,(m)$

ή

$\sum_{0\leq k\leq n} send_{0,k}(m)$.

$cond(m \notin d \wedge c \subseteq ack(m) \cup \{p_k\} \Rightarrow \overline{deliver_i}(m).0,$

$c \not\subseteq ack(m) \cup \{p_k\} \Rightarrow A_0^0\langle p, c, d, ack \cup \{p_k\}\rangle)$

ή

$check_0(m).\overline{receive}_0(yes)$

Απόδειξη Λήμματος 4:

Θα δείξουμε με επαγωγή ότι από την αρχή του συστήματος μέχρι το σημείο από το οποίο δεν μπορούν να προκύψουν άλλα σφάλματα όποια ακολουθία ενεργειών και να γίνει τότε το σύστημα μας θα καταλήξει στην πιο κάτω μορφή όπου στις καταστάσεις R_i, R_i^1 θα υπάρχουν ένα σύνολο από επεξεργαστές W όπου W⊆R.

⟨L',0⟩⊳$(l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel \prod_{\iota\epsilon I_4} l_i[\overline{deliver_i}(m)]$ || A[list])\C

<u>Βασικό βήμα</u>

Στην αρχική του κατάσταση το σύστημα θα έχει την πιο κάτω μορφή για τον γύρο M=0

⟨L,n⟩⊳(URB || A[list])\C = ⟨L,n⟩⊳(S$\langle m, p, c, d, ack\rangle \parallel \prod_{i\epsilon I} R_i\langle p, c, d, ack\rangle$|| A[list])\C

όπου ικανοποιεί την πιο πάνω συνθήκη,

$I_1 = I_2 = I_3 = I_4 = \emptyset$

Επαγωγική υπόθεση

Υποθέτουμε ότι ισχύει για τον γύρο M=m και ότι το σύστημα $\langle L^1,l\rangle \triangleright (URB^m)\backslash K$ είναι στην ζητούμενη μορφή

$\langle L^1,l\rangle \triangleright (l_0[S\langle m,p,c,d,ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p,c,d,ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel$
$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel \prod_{\iota\in I_4} l_i[\overline{deliver_i}(m)] \parallel A[list])\backslash C$, $L^1 \subseteq L$ και $l \leq n$.

Επαγωγικό βήμα

Οι πιθανές μεταβάσεις οι οποίες μπορούν να γίνουν στο γύρο M=m+1 είναι οι εξής

$\langle L^1,l\rangle \triangleright (l_0[S\langle m,p,c,d,ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p,c,d,ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel$
$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel \prod_{i\in I_4} l_i[\overline{deliver_i}(m)] \parallel A[list])\backslash C \xrightarrow{\tau(send_{i,j}(m))}$
$\langle L^2,k\rangle \triangleright (l_0[S\langle m,p,c,d,ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p,c,d,ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel$
$\prod_{i\epsilon I_2 \cup j} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel \prod_{\iota\in I_4} l_i[\overline{deliver_i}(m)] \parallel A[list])\backslash C$ ή
$\langle L^2,k\rangle \triangleright (l_0[S\langle m,p,c,d,ack\rangle] \parallel \prod_{i\epsilon I-j} l_i[R_i\langle p,c,d,ack\rangle] \parallel \prod_{i\epsilon I_1 \cup j} l_i[R_i^1] \parallel$
$\prod_{i\epsilon I} \prod_{k\epsilon K} l_i[\overline{send_{\iota,k}}(m)] \parallel \prod_{\iota\in I_4 \cup j} l_i[\overline{deliver_i}(m)] \parallel A[list])\backslash C$ ή
$\langle L^2,k\rangle \triangleright (l_0[S\langle m,p,c,d,ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p,c,d,ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel$
$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel \prod_{\iota\in I_4} l_i[\overline{deliver_i}(m)] \parallel A[list])\backslash C$

$\langle L^1,l\rangle \triangleright (l_0[S\langle m,p,c,d,ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p,c,d,ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel$
$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel \prod_{i\in I_4} l_i[\overline{deliver_i}(m)] \parallel A[list])\backslash C \xrightarrow{\tau(deliver_i(m))}$
$\langle L^2,k\rangle \triangleright (l_0[S\langle m,p,c,d,ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p,c,d,ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel$
$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel \prod_{i\in I_4 \cup i} l_i[\overline{deliver_i}(m)] \parallel A[list \cup i])\backslash C$

$\langle L^1,l\rangle \triangleright (l_0[S\langle m,p,c,d,ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p,c,d,ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel$
$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel \prod_{\iota\in I_4} l_i[\overline{deliver_i}(m)] \parallel A[list])\backslash C \xrightarrow{susp\ \ l_i}$
$\langle L^2,k\rangle \triangleright (l_0[S\langle m,p,c,d,ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p,c,d,ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel$
$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel \prod_{\iota\in I_4} l_i[\overline{deliver_i}(m)] \parallel A[list])\backslash C$

Από όλα τα πιο πάνω μπορούμε να πούμε ότι το σύστημα $\langle L^2,k\rangle \triangleright (URB^{m+1})\backslash C$ θα διατηρήσει την μορφή του όπου $L^2 \subseteq L$ και $k \leq n$.

Έτσι αποδείξαμε με επαγωγή ότι στο σημείο από το οποίο δεν μπορούν να προκύψουν άλλα σφάλματα η μορφή του συστήματος θα είναι η εξής

$\langle L',0\rangle \triangleright (l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel$
$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel \prod_{\iota\in I_4} l_i[\overline{deliver_i}(m)] \parallel A[list])\backslash C$, $L' \subseteq L$

Λήμμα 5. $\langle L',0\rangle \triangleright$ (URB' $\parallel B\langle list\rangle)\backslash C \Rightarrow \xrightarrow{\overline{ok}} \Rightarrow \approx 0$

Το πιο πάνω λήμμα δηλώνει πως αν τρέξουμε το σύστημα URB' παράλληλα με την δεύτερη φάση της διεργασίας περίβλημα τότε υπάρχει εκτέλεση η οποία μας δίνει το μήνυμα ok και ακολούθως τερματίζει.

Απόδειξη Λήμματος 5:
Συνεχίζοντας την ροή εκτέλεσης του συστήματος URB' το οποίο θα τρέξει παράλληλα με την δεύτερη φάση της διεργασίας περίβλημα υπάρχουν 3 πιθανές περιπτώσεις ανάλογα με την εξέλιξη των γεγονότων στο καινούριο σύστημα.

1) Η πρώτη περίπτωση αποτελεί την περίπτωση στην οποία στην πρώτη φάση του περιβλήματος υπάρχει τουλάχιστον ένας επεξεργαστής ο οποίος έκανε deliver το μήνυμα και έτσι στο τέλος το περίβλημα θα πρέπει να απαιτήσει από όλους τους ζωντανούς επεξεργαστές να κάνουν deliver το μήνυμα.

$\langle L',0\rangle \triangleright (l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel$
$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel \prod_{\iota\in I_4} l_i[\overline{deliver_i}(m) \parallel B\langle list\rangle)\backslash C$

Φάση 1: Ο αποστολέας στέλνει το μήνυμα σε όλους πάνω στα κανάλια που τους ενώνουν

$\langle L',0\rangle \triangleright (l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$
$\prod_{\iota\in I_4} l_i[\overline{deliver_i}(m) \parallel B\langle list\rangle)\backslash C \xrightarrow{\tau(send_{0,0}(m))} \ldots \xrightarrow{\tau(send_{0,n}(m))}$

Ο αποστολέας στέλνει το μήνυμα σε όλους πάνω στα κανάλια που τους ενώνουν

$\langle \text{L'},0\rangle \triangleright (l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I-\{L'-0\}} l_i[R_i\langle p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_1 \cup \{L'-0\}} l_i[R_i^1] \parallel$
$\prod_{i\epsilon I_2 - 0 \cup \{L'-0\}} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel \prod_{\iota\epsilon I_4} l_i[\overline{deliver_i}(m) \parallel \text{B}\langle\text{list}\rangle)\backslash \text{C}$
Οι διεργασίες $i \in I$ όπου $m \notin pending_i$ θα πάρουν το μήνυμα και θα το προωθήσουν

Φάση 2: Εκτελούνται όλες οι εσωτερικές ενέργειες $send_{i,j}$
$\langle \text{L'},0\rangle \triangleright (l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$
$\prod_{\iota\epsilon I_4} l_i[\overline{deliver_i}(m) \parallel \text{B}\langle\text{list}\rangle)\backslash \text{C} \xrightarrow{\tau(send_{i,j}(m))} \ldots \xrightarrow{\tau(send_{j,k}(m))}$
Όλες οι ζωντανές οντότητες πλην του αποστολέα στέλνουν σε όλες τις υπόλοιπες οντότητες το μήνυμα πάνω στα κανάλια που τους ενώνουν.
$\langle \text{L'},0\rangle \triangleright (l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel \prod_{i\epsilon I_2 - \{L'-0\}} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$
$\prod_{\iota\epsilon I_4 \cup \{L'-0\}} l_i[\overline{deliver_i}(m) \parallel \text{B}\langle\text{list}\rangle)\backslash \text{C}$

Φάση 3: Γίνονται οι επικοινωνίες με το περίβλημα
$\langle \text{L'},0\rangle \triangleright (l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$
$\prod_{\iota\epsilon I_4} l_i[\overline{deliver_i}(m) \parallel \text{B}\langle\text{list}\rangle)\backslash \text{C} \rightarrow$
$\langle \text{L'},0\rangle \triangleright (l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$
$\prod_{\iota\epsilon I_4} l_i[\overline{deliver_i}(m) \parallel \text{D}_0\langle\text{list}\rangle)\backslash \text{C} \xrightarrow{\tau(deliver_0(m))} \ldots \xrightarrow{susp\ l_n}$
Στο σημείο αυτό το περίβλημα θα ελέγξει μια-μια τις διεργασίες μέχρι και την n και για κάθε μια που δεν είναι εσφαλμένη αν δεν έλαβε ήδη deliver από αυτή μόλις λάβει συνεχίζει με την επόμενη.
$\langle \text{L'},0\rangle \triangleright (l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$
$\prod_{\iota\epsilon I_4 - \{L'-0\}} l_i[\overline{deliver_i}(m) \parallel \text{D}_{\text{n+1}}\langle\text{list}\rangle)\backslash \text{C} \xrightarrow{\overline{ok}}$
$\langle \text{L'},0\rangle \triangleright (l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel 0)\backslash \text{C} \approx 0$

Η διεργασία περίβλημα αφού ελέγξει την λίστα list μετά τον πρώτο γύρο και δει ότι δεν είναι κενή τότε θα μεταβεί στην κατάσταση D_0 όπου για κάθε επεξεργαστή αν δεν είναι εσφαλμένος θα περιμένει μέχρι να κάνει deliver το μήνυμα αν δεν το έκανε ήδη και ακολούθως συνεχίζει με την επόμενη στην σειρά οντότητα. Όταν όλοι οι ζωντανοί κάνουν deliver το μήνυμα τότε το περίβλημα τερματίζει δίνοντας μας το ok. Ακολούθως το σύστημα φτάνει σε μια κατάσταση που είναι ισοδύναμη με την

τερματική διεργασία όταν δεν μπορούν να εκτελεστούν άλλες ενέργειες εσωτερικά του συστήματος.

2) Η δεύτερη περίπτωση αποτελεί την περίπτωση στην οποία καμιά από τις οντότητες δεν έκανε deliver το μήνυμα στην πρώτη φάση και το περίβλημα αφού ελέγξει όλους τους ζωντανούς επεξεργαστές να βρει κάποιο ο οποίος έχει στην κατοχή του το μήνυμα και ακολούθως να απαιτήσει από όλους τους ζωντανούς να το κάνουν deliver.

Οι πρώτες δύο φάσεις είναι οι ίδιες με την προηγούμενη περίπτωση με την μόνη διαφορά ότι εκτελείται η ενέργεια deliver από τον αποστολέα.

Φάση 3: Γίνονται οι επικοινωνίες με το περίβλημα

$\langle \mathrm{L}',0\rangle \triangleright (l_0[S\langle m,p,c,d,ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i{}^1] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$

$\prod_{\iota\epsilon I_4} l_i[\overline{deliver_i}(m) \parallel \mathrm{B}\langle \mathrm{list}\rangle)\backslash \mathrm{C} \xrightarrow{\tau(check_0(m))} \xrightarrow{\tau(receive_0(answer))}$

$\langle \mathrm{L}',0\rangle \triangleright (l_0[S\langle m,p,c,d,ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$

$\prod_{\iota\epsilon I_4} l_i[\overline{deliver_i}(m) \parallel \mathrm{D}_0\langle \mathrm{list}\rangle)\backslash \mathrm{C} \xrightarrow{\tau(deliver_0(m))} \ldots \xrightarrow{susp\ l_n}$

Στο σημείο αυτό το περίβλημα θα ελέγξει μια-μια τις διεργασίες μέχρι και την n και για κάθε μια που δεν είναι εσφαλμένη μόλις λάβει deliver από αυτή θα συνεχίζει με την επόμενη.

$\langle \mathrm{L}',0\rangle \triangleright (l_0[S\langle m,p,c,d,ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel$

$\prod_{\iota\epsilon I_4 - L'} l_i[\overline{deliver_i}(m) \parallel \mathrm{D}_{n+1}\langle \mathrm{list}\rangle)\backslash \mathrm{C} \xrightarrow{\overline{ok}}$

$\langle \mathrm{L}',0\rangle \triangleright (l_0[S\langle m,p,c,d,ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i{}^1] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel 0)\backslash \mathrm{C} \approx 0$

Η διεργασία περίβλημα μόλις εντοπίσει κάποιο ζωντανό επεξεργαστή ο οποίος έχει στην κατοχή του το μήνυμα τότε μεταβαίνει στην κατάσταση D_0 στην οποία για κάθε επεξεργαστή από την αρχή αν δεν είναι εσφαλμένος τότε περιμένει μέχρι να κάνει deliver το μήνυμα και ακολούθως συνεχίζει με τον επόμενο στη σειρά. Όταν όλοι οι ζωντανοί κάνουν deliver το μήνυμα τότε η διεργασία περίβλημα τερματίζει δίνοντας προς τα έξω το ok. Ακολούθως το σύστημα φτάνει σε μια κατάσταση που είναι ισοδύναμη με την τερματική διεργασία όταν δεν μπορούν να εκτελεστούν άλλες ενέργειες εσωτερικά του συστήματος.

3) Η τρίτη περίπτωση αποτελεί την περίπτωση στην οποία καμιά από τις οντότητες δεν έκανε deliver το μήνυμα στην πρώτη φάση και ακολούθως αφού ελέγξει όλους τους ζωντανούς επεξεργαστές να μην βρει κάποιο που έχει στην κατοχή του το μήνυμα και έτσι το περίβλημα να τερματίσει εκτελώντας την εξωτερική ενέργεια ok. Υποθέτουμε ότι στο σύστημα δεν υπάρχουν μεταβιβαζόμενα μηνύματα.

$\langle L',0\rangle \triangleright (l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel B\langle list\rangle)\backslash C \rightarrow$

$\langle L',0\rangle \triangleright (l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel C_0\langle list\rangle)\backslash C \xrightarrow{susp\ l_0}$

$\langle L',0\rangle \triangleright (l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel C_1\langle list\rangle)\backslash C \xrightarrow{\tau(check_1(m))} \xrightarrow{\tau(receive_1(answer))} \ldots \xrightarrow{\tau(check_n(m))} \xrightarrow{\tau(receive_n(answer))}$

Στο σημείο αυτό το περίβλημα θα ελέγξει μια-μια τις διεργασίες μέχρι και την n και για κάθε μια που δεν είναι εσφαλμένη θα της στείλει μήνυμα στο κανάλι $check_i$ και θα λάβει απάντηση στο κανάλι $receive_i$ που στην περίπτωση αυτή θα είναι no σε όλες τις περιπτώσεις.

$\langle L',0\rangle \triangleright (l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel C_{n+1}\langle list\rangle)\backslash C \xrightarrow{\overline{ok}}$

$\langle L',0\rangle \triangleright (l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel 0)\backslash C \approx 0$

Η διεργασία περίβλημα αφού δει ότι η λίστα list είναι κενή τότε πάει στην κατάσταση C_0 όπου ξεκινώντας από την αρχή ελέγχει για κάθε οντότητα αν δεν είναι εσφαλμένη τότε στέλνει μήνυμα πάνω στο κανάλι $check_i$ και παίρνει μια απάντηση στο κανάλι $receive_\iota$ που δηλώνει αν ο συγκεκριμένος επεξεργαστής έχει το μήνυμα. Αφού δει ότι κανένας από τους ζωντανούς επεξεργαστές δεν έχει το μήνυμα τότε τερματίζει δίνοντας προς τα έξω το μήνυμα ok. Ακολούθως το σύστημα θα μεταβεί σε μια κατάσταση που είναι ισοδύναμη με την τερματική διεργασία.

Λήμμα 6. ⟨L',0⟩⊳(URB' || B⟨list⟩)\C είναι confluent

Απόδειξη Λήμματος 6:

Θα πρέπει να δείξουμε ότι τα πιο κάτω αθροίσματα αποτελούν συρρέουσες διεργασίες στις καταστάσεις S, R_i, R_i^1 αντίστοιχα.

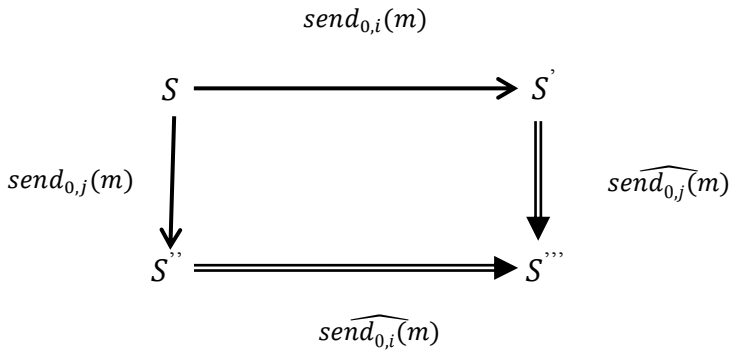
1) $\sum_{0\le k\le n} send_{0,k}(m)$

2) $\sum_{0\le k\le n} send_{i,k}(m) + check_i(m)$

3) $check_i(m).\overline{receive}_i(yes).R_i^1 + \sum_{0\le k\le n} send_{i,k}(m)\,.\,R_i^1$

Ξεκινούμε με το (1):

Για να δείξουμε ότι το άθροισμα $\sum_{0\le k\le n} send_{0,k}(m)$ στην διεργασία S είναι συρρέουσα διεργασία θα πρέπει να δείξουμε ότι



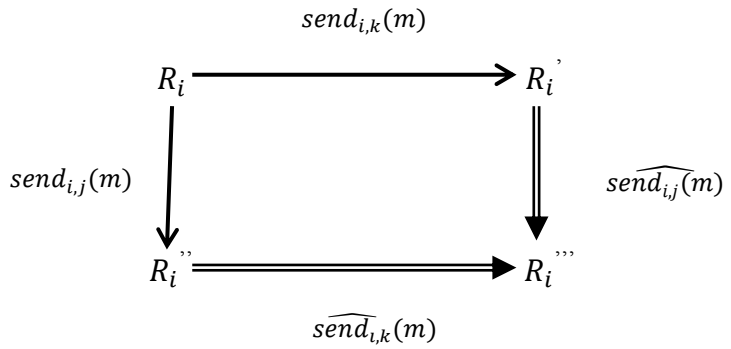
Απόδειξη:

Οποιαδήποτε από τις δύο ενέργειες εκτελεστεί στη συνέχεια η διεργασία θα επιστρέψει στην αρχική της κατάσταση όπου θα μπορεί να εκτελέσει την δεύτερη ενέργεια.

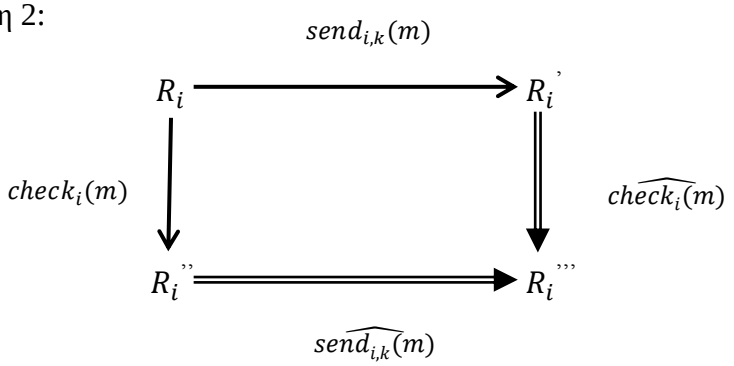
Απόδειξη για το (2):

Για να δείξουμε ότι το άθροισμα $\sum_{0\le k\le n} send_{i,k}(m) + check_i(m)$ στην διεργασία R_i είναι συρρέουσα διεργασία θα πρέπει να δείξουμε ότι

Περίπτωση 1:



Περίπτωση 2:



Απόδειξη περίπτωσης 1:

Σύμφωνα με την μοντελοποίηση του αλγορίθμου όποια ενέργεια από τις δύο εκτελεστεί στην κατάσταση R_i τότε ακολούθως θα επιστρέψει στην αρχική κατάσταση όπου θα μπορεί να εκτελέσει την δεύτερη ενέργεια.

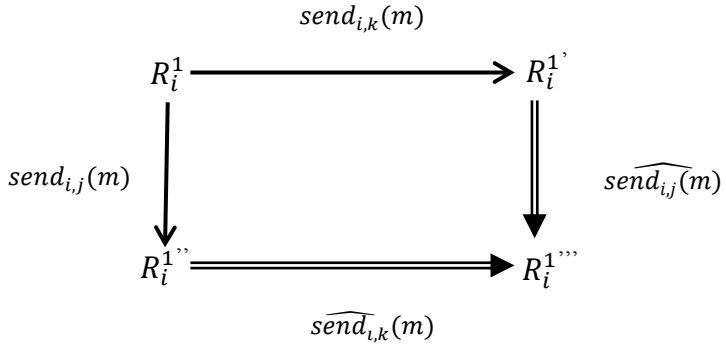
Απόδειξη περίπτωσης 2:

Ισχύει ότι και στην περίπτωση 1.

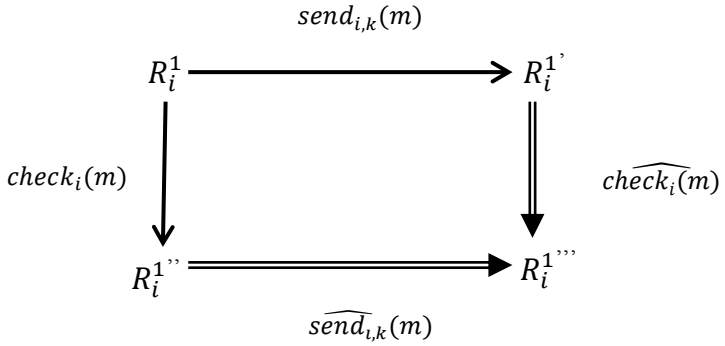
Απόδειξη για το (3):

Για να δείξουμε ότι το άθροισμα $check_i(m).\overline{receive}_i(yes).R_i^1 + \sum_{0\le k\le n} send_{i,k}(m).R_i^1$ στην διεργασία R_i^1 είναι συρρέουσα διεργασία θα πρέπει να δείξουμε ότι

Περίπτωση 1:



Περίπτωση 2:



Απόδειξη περίπτωσης 1:

Σύμφωνα με την μοντελοποίηση του αλγορίθμου όποια ενέργεια από τις δύο εκτελεστεί στην κατάσταση R_i^1 τότε ακολούθως θα επιστρέψει στην αρχική κατάσταση όπου θα μπορεί να εκτελέσει την δεύτερη ενέργεια.

Απόδειξη περίπτωσης 2:

Ισχύει ότι και στην περίπτωση 1.

Ακολούθως θα δείξουμε ότι το σύστημα αποτελεί σύνθεση πολλών συρρεόντων διεργασιών ελέγχοντας μια-μια τις καταστάσεις του συστήματος.

$$S\langle m, p, c, d, ack\rangle = \prod_{0\le k\le n} \overline{send_{0,k}}(m) \,||\, (\sum_{0\le k\le n} send_{0,k}(m).\ cond(m \notin d \wedge c \subseteq ack \cup \{p_k\} \Rightarrow \overline{deliver_i}(m) \,\|\, 0,\ c \not\subseteq ack \cup \{p_k\} \Rightarrow A_0^0\langle p, c, d, ack \cup \{p_k\}\rangle) \,||\, check_0(m).\overline{receive_0}(yes))$$

Η πιο πάνω διεργασία αποτελεί σύνθεση διεργασιών χρησιμοποιώντας τους τελεστές της επιλογής, της συνθήκης του ακολουθιακού και του ταυτοχρονισμού. Για τον πρώτο η σχετική απόδειξη έγινε στο υποκεφάλαιο 6.2 για τον αλγόριθμο URB και για τους άλλους τρείς στο υποκεφάλαιο 3.5. Για την διεργασία A_0^0 δείχνουμε πιο κάτω ότι είναι συρρέουσα διεργασία.

$$R_i^1 = check_i(m).\overline{receive}_i(yes).R_i^1 + \sum_{0\le k\le n} send_{i,k}(m)\,.\,R_i^1$$

Η διεργασία R_i^1 κάνει χρήση του τελεστή διαδοχής η απόδειξη του οποίου έγινε στο υποκεφάλαιο 3.5. Επίσης γίνεται χρήση του τελεστή επιλογής η απόδειξη του οποίου έγινε στο υποκεφάλαιο 6.2 για τον αλγόριθμο URB.

$$R_i\langle p,c,d,ack\rangle = \sum_{0\le k\le n} send_{i,k}(m).$$
$$cond(m \notin p \Rightarrow \prod_{0\le k\le n} \overline{send_{\iota,k}}(m) \,||\, R_i\langle p\cup\{m\},c,d,ack\cup\{p_k\}\rangle,$$
$$m \notin d \wedge c \subseteq ack\cup\{p_k\} \Rightarrow \overline{deliver_i}(m) \,\|\, R_i^1,$$
$$c \nsubseteq ack\cup\{p_k\} \Rightarrow A_i^0\langle p,c,d,ack\cup\{p_k\}\rangle)$$
$$+ check_i(m).cond(m \in p \Rightarrow \overline{receive}_0(yes) \,||\, R_i\langle p,c,d,ack\rangle,$$
$$m \notin p \Rightarrow \overline{receive}_0(no) \,||\, R_i\langle p,c,d,ack\rangle)$$

Η πιο πάνω διεργασία αποτελεί σύνθεση διεργασιών χρησιμοποιώντας τους τελεστές της επιλογής, της συνθήκης, του ταυτοχρονισμού και του ακολουθιακού. Για τον πρώτο η σχετική απόδειξη έγινε στο υποκεφάλαιο 6.2 για τον αλγόριθμο URB και για τους υπόλοιπους στο υποκεφάλαιο 3.5. Για την διεργασία A_i^0 δείχνουμε πιο κάτω ότι είναι συρρέουσα διεργασία.

$$A_i^{n+1} = cond(i = 0 \Rightarrow S\langle p,c,d,ack\rangle, i\,!=0 \Rightarrow R_i\langle p,c,d,ack\rangle)$$

Η διεργασία A_i^{n+1} κάνει χρήση του τελεστή συνθήκης του οποίου η σχετική απόδειξη έγινε στο υποκεφάλαιο 3.5.

$$A_i^j\langle p,c,d,ack\rangle = susp\ l_j\colon (A_i^{j+1}\langle p,c-\{p_j\},d,ack\rangle\ ;\ A_i^{j+1}\langle p,c,d,ack\rangle)$$

Η διεργασία A_i^j κάνει χρήση του τελεστή susp του οποίου η σχετική απόδειξη έγινε στο υποκεφάλαιο 3.5.

$$D_{n+1}\langle list\rangle = \overline{ok}.0$$

Η διεργασία $D_{n+1}\langle list\rangle$ κάνει χρήση του τελεστή διαδοχής η απόδειξη του οποίου έγινε στο υποκεφάλαιο 3.5.

$$D_i\langle list\rangle = susp\ l_i\colon (D_{i+1}\langle list\rangle\ ;\ \ cond(p_i \in list \Rightarrow D_{i+1}\langle list\rangle,\\ p_i \notin list \Rightarrow deliver_i(m).\, D_{i+1}\langle list\rangle))$$

Η διεργασία $D_i\langle list\rangle$ κάνει χρήση των τελεστών της συνθήκης, της διαδοχής και του τελεστή susp οι αποδείξεις των οποίων έγιναν στο υποκεφάλαιο 3.5.

$$C_{n+1}\langle list\rangle = \overline{ok}.0$$

Η διεργασία $C_{n+1}\langle list\rangle$ χρησιμοποιεί τον τελεστή διαδοχής η απόδειξη του οποίου έγινε στο υποκεφάλαιο 3.5.

$$C_i\langle list\rangle = susp l_i\colon (C_{i+1}(m)\ ;\ \overline{check_\iota}(m).receive_i(answer).\\ cond(answer = no \Rightarrow C_{i+1}(m),\\ \quad answer = yes \Rightarrow D_0(m)))$$

Η διεργασία $C_i\langle list\rangle$ κάνει χρήση των τελεστών συνθήκης, διαδοχής και του τελεστή susp οι σχετικές αποδείξεις των οποίων έγιναν στο υποκεφάλαιο 3.5.

$$B\langle list\rangle = cond(list = \emptyset \Rightarrow C_0(list), list\ !\ = \emptyset \Rightarrow D_0\langle list\rangle)$$

Η διεργασία $B\langle list\rangle$ κάνει χρήση του τελεστή συνθήκης η απόδειξη του οποίου έγινε στο υποκεφάλαιο 3.5.

$$URB' = \langle \text{L'},0\rangle \rhd (l_0[S\langle m\ , p, c, d, ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel\\ \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)] \parallel \prod_{\iota\epsilon I_4} l_i[\overline{deliver_i}(m)])\backslash \text{K}$$

Το σύστημα URB' αποτελεί σύνθεση διεργασιών χρησιμοποιώντας τον τελεστή του ταυτοχρονισμού η σχετική απόδειξη του οποίου έγινε στο υποκεφάλαιο 3.5. Οι διεργασίες μεταξύ τους δεν έχουν οποιαδήποτε κοινά κανάλια εισόδου ή εξόδου.

(URB' || B⟨list⟩)\C: Το σύστημα κάνει χρήση του τελεστή ταυτοχρονισμού ο οποίος αποδείξαμε στο υποκεφάλαιο 3.5 ότι διατηρεί την έννοια της συρροής και οι δύο επιμέρους διεργασίες δεν έχουν οποιαδήποτε κοινά κοινά κανάλια εισόδου ή εξόδου.

Για να αποδείξουμε ότι το σύστημα ⟨L',0⟩⊳(URB' || B⟨list⟩)\C είναι confluent βλέπουμε ότι το σύστημα αποτελεί σύνθεση πολλών συρρεόντων διεργασιών και επίσης κάθε κανάλι μέσα στο σύστημα μας είναι μοναδικό και κάθε δύο οντότητες ενώνονται πάνω σε ένα μοναδικό κανάλι. Έτσι κάθε πιθανή εκτέλεση του αλγορίθμου μπορεί να μας δώσει το ζητούμενο αποτέλεσμα.

Απόδειξη θεωρήματος 2:
Το πιο πάνω σύστημα πάντοτε θα τερματίζει βγάζοντας ως έξοδο το μήνυμα $\overline{ok}$ και ακολούθως μεταβαίνοντας σε τερματική κατάσταση. Επομένως τα δύο συστήματα συνδέονται με σχέση ασθενής διπροσομοίωσης. Πιο συγκεκριμένα ⟨L',0⟩⊳ (URB' || B⟨list⟩)\C ≈ $\overline{ok}.0$. Η τερματική κατάσταση στην οποία τερματίζει το σύστημα θα είναι ⟨L',0⟩⊳ (URB'' || 0)\C όπου

⟨L',0⟩⊳ (URB'')\K= ⟨L',0⟩⊳($l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_1} l_i[R_i^1] \parallel$
$\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)]$)\C
ή
⟨L',0⟩⊳($l_0[S\langle m, p, c, d, ack\rangle] \parallel \prod_{i\epsilon I} l_i[R_i\langle p, c, d, ack\rangle] \parallel \prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)]$)\C

Οι ενέργειες $\prod_{i\epsilon I_2} \prod_{k\epsilon I_3} l_i[\overline{send_{\iota,k}}(m)]$ αποτελούν τις ενέργειες στις οποίες δεν έγινε ο συγχρονισμός για τον λόγο ότι είτε ο επεξεργαστής i είτε ο επεξεργαστής k κατέρρευσαν.

Μπορούμε να γενικοποιήσουμε την απόδειξη μας και στην περίπτωση που υπάρχουν περισσότερα από ένα μηνύματα στο σύστημα μας. Επίσης η πιο πάνω απόδειξη που έγινε μπορούμε να πούμε ότι ισχύει για ένα τυχαίο αριθμό σφαλμάτων άρα μπορούμε να πούμε ότι ισχύει για οποιοδήποτε αριθμό σφαλμάτων.

Κεφάλαιο 7

Συμπεράσματα

7.1 Σύνοψη Διπλωματικής

Στο Κεφάλαιο 1 καθορίστηκαν οι στόχοι της διπλωματικής εργασίας όπου ο κύριος στόχος ήταν η ανάπτυξη μεθοδολογίας για την τυπική ανάλυση και επαλήθευση κατανεμημένων αλγορίθμων στην παρουσία σφαλμάτων. Σαν επιμέρους στόχος είχε καθοριστεί η μοντελοποίηση και η απόδειξη ορθότητας δύο αλγορίθμων μετάδοσης μηνύματος σε κατανεμημένο περιβάλλον.

Στο Κεφάλαιο 2 εξηγήθηκαν οι βασικές έννοιες της άλγεβρας διεργασιών CCS και μέσα από την κατανόηση των βασικών της χαρακτηριστικών χρησιμοποιήσαμε την γνώση αυτή για να επεκτείνουμε περαιτέρω την γλώσσα μας ούτως ώστε να προσθέσουμε κάποιες λειτουργίες οι οποίες να μας επιτρέπουν την μοντελοποίηση των αλγορίθμων στην παρουσία σφαλμάτων.

Στο Κεφάλαιο 3 παρουσιάσαμε την άλγεβρα διεργασιών Dπf-calculus η οποία αποτελεί επέκταση της CCS και εξηγήσαμε τις βασικές έννοιες της γλώσσας την οποία χρησιμοποιήσαμε στην συνέχεια για την μοντελοποίηση των αλγορίθμων και την απόδειξη της ορθότητας τους.

Στη συνέχεια στο Κεφάλαιο 4 μελετήθηκε το πρόβλημα μετάδοσης ενός μηνύματος σε κατανεμημένο περιβάλλον και έγινε περιγραφή του τρόπου λειτουργίας 3 αλγορίθμων που επιλύουν το πιο πάνω πρόβλημα του Best Effort Broadcast, του Lazy Reliable

Broadcast και του Uniform Reliable Broadcast όπου οι δύο τελευταίοι μας προσφέρουν εγγύηση συμφωνίας στην παρουσία σφαλμάτων.

Ακολούθως στο Κεφάλαιο 5 έγινε η μοντελοποίηση του αλγόριθμου LRB και αποδείχτηκε η ορθότητα του χρησιμοποιώντας την έννοια της συρροής μιας μορφής ντετερμινισμού η οποία μας εγγυάται ότι αν ένα σύστημα είναι ικανό να παράξει το επιθυμητό αποτέλεσμα και το ίδιο σύστημα αποτελεί σύνθεση συρρεόντων διεργασιών τότε το σύστημα είναι ικανό να μας δώσει το επιθυμητό αποτέλεσμα σε κάθε πιθανή εκτέλεση. Σημαντική ήταν η έννοια της ασθενής διπροσομοίωσης για να δείξουμε ότι ένα σύστημα είναι ικανό να μας δώσει το επιθυμητό αποτέλεσμα δείχνοντας ότι το σύστημα είναι ισοδύναμο με την προδιαγραφή που του ορίσαμε.

Στο Κεφάλαιο 6 έγινε η σχετική μοντελοποίηση και απόδειξη ορθότητας για τον αλγόριθμο URB χρησιμοποιώντας την ίδια μεθοδολογία με τον αλγόριθμο LRB.

7.2 Συμπεράσματα Διπλωματικής

Μέσα από την δουλειά που έγινε καταλήξαμε σε κάποια συμπεράσματα τα οποία αναφέρουμε πιο κάτω.

Αρχικά μπορούμε να πούμε ότι παρουσιάστηκε αδυναμία της άλγεβρας ccs για τον χειρισμό των σφαλμάτων και έτσι μπορούμε να πούμε ότι δεν είναι η κατάλληλη για μοντελοποίηση τέτοιου είδους αλγορίθμων. Επίσης μπορούμε να πούμε ότι για την απόδειξη ορθότητας τέτοιου είδους αλγορίθμων απαραίτητη είναι η χρήση των σχέσεων ισοδυναμίας που ορίζουν μια γλώσσα όπως επίσης και της συρροής αφού σε τέτοιου είδους συστήματα υπάρχει μη πεπερασμένος αριθμός εκτελέσεων. Έτσι μπορούμε να πούμε ότι συνίσταται η χρήση αυτών των μεθόδων για την απόδειξη ορθότητας παρόμοιων συστημάτων νοουμένου ότι πληρούν τους περιορισμούς των ντετερμινιστικών συστημάτων και πιο συγκεκριμένα της συρροής. Στην δική μας περίπτωση οι κυρίες δυσκολίες οι οποίες συναντήσαμε ήταν η κατανόηση των κύριων χαρακτηριστικών της γλώσσας και η επέκταση της για τους σκοπούς της μοντελοποίησης όπως επίσης και η απόδειξη ότι το σύστημα μας χαρακτηρίζεται από συρροή. Έτσι η δυσκολία σε παρόμοιους αλγορίθμους μπορούμε να πούμε ότι θα υπάρξει στα πιο πάνω σημεία.

Συμπερασματικά μπορούμε να πούμε ότι μέσω της άλγεβρας προσφέρονται μηχανισμοί οι οποίοι μας επιτρέπουν να απλοποιήσουμε την διαδικασία απόδειξης ορθότητας των αλγορίθμων για κάθε πιθανή εκτέλεση τους.

7.3 Μελλοντικές Εργασίες

Σε μελλοντικές εργασίες υπάρχει η δυνατότητα να γίνει η μοντελοποίηση και η απόδειξη ορθότητας για αλγόριθμους σε διαφορετική τοπολογία ή σε συστήματα τα οποία χρησιμοποιούν διαφορετικό ανιχνευτή σφαλμάτων από την δική μας περίπτωση.

Βιβλιογραφία

[1] Robin Milner, Communication and Concurrency, Prentice-Hall, London, 1989.

[2] Christian Cachin, Rachid Guerraoui, Luís Rodrigues, Introduction to Reliable and Secure Distributed Programming (Second edition), Springer, New York, 2011.

[3] Adrian Francalanza, Matthew Hennessy, "A Fault Tolerance Bisimulation Proof for Consensus (Extended Abstract)", ESOP, pp. 395-410, 2007.

[4] Anna Philippou, George Michael, "Verification Techniques for Distributed Algorithms", OPODIS, pp. 172 -186 , 2006.

[5] Anna Philippou, Chryssis Georgiou, Marina Gelastou, "On the Application of Formal Methods for Specifying and Verifying Distributed Protocols", NCA, pp. 195-204, 2008.

[6] Άννα Φιλίππου, Διάλεξη "Άλγεβρες Διεργασιών", Σημειώσεις μαθήματος ΕΠΛ664: "Ανάλυση και Επαλήθευση Συστημάτων", Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου, Γενάρης 2012.

[7] Άννα Φιλίππου, Διάλεξη "Άλγεβρες Διεργασιών και Σχέσεις Ισοδυναμίας", Σημειώσεις μαθήματος ΕΠΛ664: "Ανάλυση και Επαλήθευση Συστημάτων", Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου, Γενάρης 2012.

[8] Νικόλας Νικολάου, Διάλεξη "Εισαγωγή στα Κατανεμημένα Συστήματα", Σημειώσεις μαθήματος ΕΠΛ432: "Κατανεμημένοι Αλγόριθμοι", Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου, Σεπτέμβριος 2013.