FORTH-ICS / TR-057 September 1992

Verification of Real Time Systems: The extension of COSPAN in dense time

Panagiotis N. Tzounakis[*]

Abstract

The often enormous complexity of concurrent real-time systems makes their correct design a very difficult issue to deal without some form of computer assistance. We present an extension of the well known *selection/resolution* or simply S/R model to include timing constrains. Our extension consists of the integration of an existing model of dense time based on timing assumptions and its corresponding algorithms within the S/R model. We describe the use and the implementation of our model in a new automatic verification program for real-time systems based on COSPAN, a well-known verification tool for concurrent systems over discrete time. COSPAN is extended by adding an on-the-fly automatically-generated "monitor" process which keeps track of timing constraints so as to exclude system behaviors that are inconsistent with the timing information. We discuss the S/R model and COSPAN in order to provide some background needed, describe the dense time model, and the theory and implementation of the extensions made in order to handle real-time information. The results of applying the verifier to several examples with real time are discussed.

[*]Institute of Computer Science, F.O.R.T.H., Crete, Greece
E-mail pj@ics.forth.gr

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΠΑΛΗΘΕΥΣΗ ΣΥΣΤΗΜΑΤΩΝ ΠΡΑΓΜΑΤΙΚΟΥ ΧΡΟΝΟΥ: Η ΕΠΕΚΤΑΣΗ ΤΟΥ COSPAN ΣΕ ΣΥΝΕΧΗ ΧΡΟΝΟ

Παναγιώτης Ν. Τζουνάκης

Μεταπτυχιακή Εργασια

ΗΡΑΚΛΕΙΟ, ΚΡΗΤΗ
Σεπτέμβριος 1992

ΕΠΑΛΗΘΕΥΣΗ ΣΥΣΤΗΜΑΤΩΝ ΠΡΑΓΜΑΤΙΚΟΥ ΧΡΟΝΟΥ: Η ΕΠΕΚΤΑΣΗ ΤΟΥ COSPAN ΣΕ ΣΥΝΕΧΗ ΧΡΟΝΟ

Μεταπτυχιακή εργασία
που υποβλήθηκε στην επιτροπή μεταπτυχιακών σπουδών
του τμήματος επιστήμης υπολογιστών
της σχολής θετικών επιστημών
του πανεπιστημίου κρήτης
ως μερική απαίτηση για την απόκτηση του
ΔΙΠΛΩΜΑΤΟΣ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΞΕΙΔΙΚΕΥΣΗΣ
Σεπτέμβριος 1992

Συγγραφέας:

Παναγιώτης Ν. Τζουνάκης
Τμήμα Επιστήμης Υπολογιστών

Εισηγητική Επιτροπή:

Κώστας Κουρκουμπέτης Αναπληρωτής καθηγητής
(Επόπτης)

Πάνος Κωνσταντόπουλος Αναπληρωτής καθηγητής
(Μέλος)

Μανόλης Κατεβαίνης Αναπληρωτής καθηγητής
(Μέλος)

Δεκτή:

Στέλιος Ορφανουδάκης Καθηγητής, πρόεδρος τμήματος
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών

Πρόλογος

Η συχνά τεράστια πολυπλοκότητα παράλληλων συστημάτων πραγματικού χρόνου κάνει πολύ δύσκολη τη σωστή σχεδίαση τους χωρίς κάποιας μορφής βοήθεια μέσω υπολογιστή. Παρουσιάζουμε μια επέκταση του γνωστού μοντέλου «επιλογής/απόφασης» (*selection/resolution* ή απλά S/R) η οποία συμπεριλαμβάνει χρονικούς περιορισμούς. Η επέκταση μας συνίσταται στην ενοποίηση ενός υπάρχοντος μοντέλου για χρονικές υποθέσεις και των αντίστοιχων αλγορίθμων με το S/R μοντέλο. Περιγράφουμε τη χρήση και την υλοποίηση του μοντέλου μας σε ένα νέο πρόγραμμα αυτόματης επαλήθευσης (verification) για συστήματα πραγματικού χρόνου, το οποίο βασίζεται στο COSPAN, ένα πολύ γνωστό πρόγραμμα επαλήθευσης γιά παράλληλα συστηματα σε διακριτό χρόνο. Επεκτείνουμε το COSPAN προσθέτωντας μιά διεργασία «ελεγκτή» η οποία δημιουργείται δυναμικά και αυτόματα και παρακολουθεί τους χρονικούς περιορισμούς, ώστε να αποκλείει συμπεριφορές του συστήματος που είναι χρονικά ασυνεπείς. Παρουσιάζουμε το S/R μοντέλο και το COSPAN ως απαραίτητη υποδομή, το μοντέλο συνεχούς χρόνου, και την θεωρία και υλοποίηση των επεκτάσεων μας μέσω των οποίων χειριζόμαστε πληροφορία πραγματικού χρόνου. Στη συνέχεια συζητάμε τα αποτελέσματα της χρήσης του προγράμματος μας για τη λυση μερικών προβλημάτων.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω πρώτα από όλους τον επόπτη καθηγητή μου κ. Κώστα Κουρκουμπέτη που με τη έμπνευση και την καθοδήγηση του έγινε δυνατή η πραγματοποίηση αυτής της εργασίας. Τη συνεργάτη μου κα. Μαρία Μαμαλάκη που με βοήθησε ουσιαστικά στις πρώτες μου ερευνητικές προσπάθειες στον τομέα της επαλήθευσης και στα πρώτα στάδια της δουλειάς μου. Το συμφοιτητή μου Ευριπίδη Πετράκη για διάφορες χρήσιμες συζητήσεις σε αλγόριθμους κατακερματισμού, και τους συμφοιτητές μου Μάγδα Χατζάκη, Νίκο Λαμπρογιώργο και Γιάννη Μουζάκη για την υπομονή που είχαν να τρέξουν τις πρώτες εκδόσεις του λογισμικού μου και να δώσουν πολύτιμες πληροφορίες για διορθώσεις. Το φίλο και συμφοιτητή στις προπτυχιακές μου σπουδές Κώστα Παπαδά που με τον τρόπο του βοήθησε πολύ χρόνο πριν αρχίσει να φτιάχνεται αυτή η εργασία. Το Ινστιτούτο Πληροφορικής του Ι.Τ.Ε. για την υλικοτεχνική υποδομή και την οικονομική στήριξη που παρείχε κατά τη διάρκεια των μεταπτυχιακών μου σπουδών, και ειδικά τον ερευνητή Στέλιο Σαρτζετάκη που μου έδωσε την ευκαιρία να παραμείνω κοντά στο Πανεπιστήμιο Κρήτης πριν ακόμα αρχίσω μεταπτυχιακές σπουδές.

Περιεχόμενα

Κατάλογος Σχημάτων

Περίληψη

Στην εργασία αυτή παρουσιάζουμε μια επέκταση του γνωστού μοντέλου «επιλογής/απόφασης» (*selection/resolution* ή απλά S/R) η οποία συμπεριλαμβάνει χρονικούς περιορισμούς. Το S/R μοντέλο είναι ένα μαθηματικό μοντέλο για συνδρομικές (concurrent) διεργασίες, όπου ένα πολύπλοκο σύστημα μπορεί να περιγραφεί σαν σύνολο από συζευγμένες μηχανές πεπερασμένων καταστάσεων. Μια ιδιότητα που κάνει το S/R μοντέλο εξαιρετικά ελκυστικό είναι ότι, η σύζευξη των μηχανών περιγράφεται μέσα από κατηγορήματα πάνω στα σήματα επικοινωνίας. Σε πολλές περιπτώσεις αυτό βοηθάει στην παραγωγή σύντομων, περιεκτικών και κατανοητών περιγραφών.

Το S/R μοντέλο είναι ένα μοντέλο διακριτού γραμμικού χρόνου με «σημασιολογία ίχνους» (trace semantics). Ορίζεται ένας λογισμός πάνω στις διεργασίες, οι οποίες μπορούν να συνθέτονται χρησιμοποιώντας την πράξη του σύγχρονου ή ασύγχρονου γινομένου. Το μοντέλο περιέχει συνθήκες αποδοχής μέσω των οποίων μπορεί να εκφραστεί κάθε ω-κανονική (ω-regular) ιδιότητα του συστήματος. Η επαλήθευση συνίσταται στην απόδειξη της έγκλεισης ιχνών.

Η επέκταση μας συνίσταται στην ενοποίηση ενός υπάρχοντος μοντέλου συνεχούς χρόνου με το S/R μοντέλο, η οποία βασίζεται στο COSPAN, ένα γνωστό πρόγραμμα επαλήθευσης γιά παράλληλα συστηματα σε διακριτό χρόνο. Το COSPAN [HK89, JK86, ZH90] είναι ένα εργαλείο λογισμικού για την περιγραφή και επαλήθευση συστημάτων που αποτελούνται από συντονιζόμενες συνιστώσες. Το COSPAN μπορεί να χρησιμοποιηθεί για την ανάπτυξη περιγραφών υψηλού επιπέδου, για την ανάλυση της λογικής και στοχαστικής συμπεριφοράς συστημάτων, και προαιρετικά για την παραγωγή πολύ αποδοτικών υλοποιήσεων σε `C` κατευθείαν από την περιγραφή υψηλού επιπέδου. Το εργαλείο χρησιμοποιεί μια τυπική, ιεραρχική, αναλυτική μέθοδο ανάπτυξης.

Η λογική ανάλυση στο COSPAN επιτυγχάνεται μέσω συμβολικού ελέγχου της περιγραφής του συστήματος ως προς την επιθυμητή συμπεριφορά που καθορίζει ο χρήστης. Το σύστημα δεν προσομοιώνεται ούτε υλοποιείται, αλλά κατασκευάζεται μια μαθηματική απόδειξη που επιβεβαιώνει (ή διαψεύδει) την ύπαρξη της υπό εξέταση συμπεριφοράς. Στους αλγόριθμους ανάλυσης του COSPAN χρησιμοποιούνται τυπικές τεχνικές αναγωγής που μπορούν να ανταπεξέλθουν στο συνήθως τεράστιο πλήθος καταστάσεων των συντονιζόμενων συστημάτων.

Σε αυτή την εργασία το S/R μοντέλο επαυξάνεται ώστε να συμπεριληφθούν περιορισμοί πραγματικού χρόνου. Περιγράφουμε την έννοια των «λογικών μετρητών χρόνου» (*logical timers*) [Dil89] που είναι πλασματικές συνιστώσες και χρησιμοποιούνται για την παρακολούθηση των χρόνων κατά τους οποίους συμβαίνουν γεγονότα μέσα στο σύστημα. Με αυτό τον τρόπο είναι δυνατόν να αποκλειστούν υπολογιστικές ακολουθίες που παραβιάζουν τις

χρονικές παραδοχές. Ετσι ένα σύστημα που μπορεί να παραβίαζε κάποιες προδιαγραφές συμπεριφοράς σε ένα μοντέλο που δεν λαμβάνει υπόψη τις σχετικές ταχύτητες των συνιστωσών, μπορεί να τις πληρεί κάτω από ορισμένες χρονικές παραδοχές.

Το COSPAN επεκτείνεται προσθέτωντας μιά διεργασία «ελεγκτή» η οποία δημιουργείται αυτόματα και παρακολουθεί τους χρονικούς περιορισμούς, ώστε να αποκλείει συμπεριφορές του συστήματος που είναι χρονικά ασυνεπείς. Παρουσιάζουμε όλες τις λεπτομέρειες σχετικά με την υλοποίηση του μοντέλου «διαδικασιών με χρόνο» (timed processes) που κατασκευάζουμε χρησιμοποιώντας το COSPAN. Το νέο εργαλείο RTCOSPAN έχει υλοποιηθεί χωρίς τροποποιήσεις της εσωτερικής δομής του COSPAN ή της γραμματικής του S/R, δείχνοντας έτσι ότι επεκτάσεις καλά σχεδιασμένων εργαλείων επαλήθευσης είναι δυνατές. Το RTCOSPAN είναι συμβατό με το COSPAN.

Παρουσιάζουμε το τμήμα της διασύνδεσης χρήστη-εργαλείου (user-interface) που εισάγει τα νέα χαρακτηριστικά του RTCOSPAN, τη φιλοσοφία σχεδίασης μέσω της οποίας το RTCOSPAN ανταπεξέρχεται το μεγάλο πλήθος καταστάσεων που απαιτούν οι μετρητές χρόνου, και δίνουμε αρκετές λεπτομέρειες των εσωτερικών δομών δεδομένων και αλγορίθμων του εργαλείου. Τέλος αναφέρουμε μερικά προβλήματα και περιορισμούς της τωρινής υλοποίησης μας.

Η μέθοδος μας χρησιμοποιείται για την περιγραφή και επαλήθευση δύο συστημάτων. Το πρώτο μας παράδειγμα αποτελεί μιά επέκταση του γνωστού πρωτοκόλλου επικοινωνίας alternating–bit, η οποία περιέχει πληροφορία συνεχούς χρόνου. Το δεύτερο παράδειγμα χρησιμοποιεί δύο παράλληλα κανάλια επικοινωνίας με ακαθόριστη αλλά πεπερασμένη καθυστέρηση μετάδοσης. Για την ευκολία του αναγνώστη οι περιγραφές μας δίνονται με μορφή γράφων αντί για τη γλώσσα περιγραφής του RTCOSPAN, μια και η μετάφραση είναι απλή.

Εχουμε δείξει ότι το COSPAN μπορεί να επεκταθεί ώστε να χειρίζεται περιορισμούς πεπερασμένων καθυστερήσεων με φυσικό τρόπο, και ότι το νέο σύστημα είναι σε θέση να επαληθεύσει ενδιαφέροντα παραδείγματα συστημάτων. Η βασική ιδέα του διαχωρισμού της περιγραφής σε τμήματα ανάλογα με την εξάρτηση από τον χρόνο, και η χρήση των περιοχών τιμών μετρητών για το κομμάτι που εξαρτάται από το χρόνο, μπορούν πιθανά να εφαρμοστούν και σε άλλες τεχνικές επαλήθευσης επίσης.

Κεφάλαιο 1

Εισαγωγή

Είναι γενικά παραδεκτό ότι η σχεδίαση πρωτόκολλων και παράλληλων συστημάτων και αλγορίθμων αποτελεί περισσότερο τέχνη παρά εφαρμοσμένη τεχνική. Συνήθως είναι δύσκολο για τον άνθρωπο να προβλέψει όλες τις δυνατές αλληλεπιδράσεις μεταξύ συνιστωσών ενός συστήματος οι οποίες ενεργούν παράλληλα. Ως αποτέλεσμα, πολλά αν όχι τα περισσότερα πρωτόκολλα και αλγόριθμοι περιέχουν ελλατώματα σχεδίασης, που οφείλονται σε απαρατήρητες αλληλεπιδράσεις ανάμεσα στις συνιστώσες τους. Τέτοια λάθη στην κατασκευή είναι εξαιρετικά δύσκολο να ανιχνευτούν και να διορθωθούν με προσομοίωση ή ακόμα και με παρατήρηση της λειτουργίας ενός πρωτότυπου του συστήματος. Ο λόγος είναι ότι πολλές φορές τα παράλληλα συστήματα είναι μη αιτιοκρατικά (nondeterministic), οπότε πιθανά προβλήματα μπορεί να είναι τυχαία και μη επαναλαμβανόμενα.

Μια μερική λύση στο παραπάνω πρόβλημα είναι η χρήση προγραμμάτων αυτόματης επαλήθευσης, τα οποία εγγυούνται την ορθή σχεδίαση του συστήμάτος. Τέτοια προγράμματα εκμεταλεύονται το γεγονός ότι πολλά από τα ενδιαφέροντα συστήματα και πρωτόκολλα είναι πεπερασμένων καταστάσεων. Απαριθμούν τις όλες τις καταστάσεις του συστήματος, ψάχνοντας παράλληλα για παραβιάσεις μιας περιγραφής συμπεριφοράς που παρέχει ο χρήστης. Οταν διαπιστωθεί μια τέτοια παραβίαση, το πρόγραμμα είναι σε θέση να δώσει την συγκεκριμένη ιστορία της συμπεριφοράς του συστήματος, η οποία προκάλεσε την παραβίαση.

Σήμερα υπάρχουν αρκετά μοντέλα και προγράμματα για την περιγραφή και επαλήθευση αλγορίθμων ή συστημάτων ψηφιακών κυκλωμάτων. Ενα τέτοιο μοντέλο είναι και το S/R [BG80], όπου συστήματα περιγράφονται σαν σύνολο από μηχανές πεπερασμένων καταστάσεων που συγχρονίζονται μέσω ενός μηχανισμού επικοινωνίας με μεγάλες δυνατότητες. Πάνω στο S/R μοντέλο βασίζεται το COSPAN [HK89, JK86, ZH90], ένα από τα πιο πολύπλοκα και ισχυρά προγράμματα για την ώρα. Το COSPAN αναπτύχθηκε από τον R. Kurshan και άλλους στα AT&T Bell Laboratories, και έχει χρησιμοποιηθεί για την επαλήθευση αρκετών πρωτοκόλλων και σχεδίων ψηφιακών κυκλωμάτων. Το S/R μοντέλο έχει αρκετή εκφραστική ισχύ, ώστε στο COSPAN να μπορεί να εκφραστεί και να επαληθευτεί οποιαδήποτε ω–κανονική (ω–regular) ιδιότητα. [1]

Στη γενική περίπτωση, τα μοντέλα που χρησιμοποιούνται σε προγράμματα επαλήθευσης περιέχουν το χρόνο σαν αφηρημένη μονάχα έννοια. Για παράδειγμα μοντελοποιούν την χρονική διάταξη των διαφόρων γεγονότων, αλλά όχι και τον ακριβή χρόνο που ένα γεγονός συμβαίνει. Ομως αυτή η αντίληψη του χρόνου δεν είναι επαρκής για την επαλήθευση πρωτοκόλλων και αλγορίθμων των οποίων η σωστή λειτουργία εξαρτάται από χρονικούς περιορισμούς, ή τα οποία πρέπει να ικανοποιήσουν χρονικές προθεσμίες που επιβάλονται από εξωτερικούς παράγοντες. Καθώς υπολογιστές και δίκτυα αλληλεπιδρούν όλο και περισσότερο με πραγματικές συσκευές ή φυσικές διεργασίες, γίνεται όλο και δυσκολότερο να κρυφτούν ή να αγνοηθούν ιδιότητες σχετικές με χρόνο. Ακόμα, η σχεδίαση σωστών συστημάτων πραγματικού χρόνου είναι εξαιρετικά δύσκολη, επειδή οι χρονικοί περιορισμοί αυξάνουν σε μεγάλο βαθμό την πολυπλοκότητα των αλληλεπιδράσεων που μπορούν να συμβούν. Είναι λοιπόν φανερό ότι προγράμματα για αυτόματη επαλήθευση συστημάτων πραγματικού χρόνου θα ήταν εξαιρετικά χρήσιμα.

Ενα βασικό ζήτημα που πρέπει να εξεταστεί σε κάθε σύστημα όπου υπάρχουν χρονικές ιδιότητες είναι η υφή του χρόνου. Εχουν προταθεί αρκετά πλαίσια για επαλήθευση ιδιοτήτων σχετικών με χρόνο, τα οποία θα κατατάξουμε σε τρείς κύριες κατηγορίες, ανάλογα με το μοντέλο του χρόνου όπου βασίζονται. Η πρώτη κατηγορία βασίζεται σε μοντέλα διακριτού χρόνου, όπου οι τιμές του χρόνου (στιγμές) είναι σύνολο ισομορφικό με τους ακεραίους ή τους φυσικούς αριθμούς [JM86, HPOG89]. Τέτοια μοντέλα μοιάζουν αρκετά με τα παραδοσιακά μοντέλα παραλληλίας και συγχρονισμού. Για παράδειγμα, στη γραμμική χρονική λογική (linear temporal logic), περιγραφές που περιέχουν χρόνο μπορούν να εκφραστούν με επαναληπτική χρήση του τελεστή «επόμενη στιγμή» (next time).

Κατά την περιγραφή συστημάτων που δεν είναι αμιγώς σύγχρονα με χρήση μοντέλων διακριτού χρόνου, ένας περιορισμός είναι ότι απαιτήται εκ των προτέρων δέσμευση για επιλογή μιάς σταθερής μονάδας χρόνου (time quantum). Αλληλεπιδράσεις που απαιτούν λεπτότερο διαμερισμό του χρόνου από την κλίμακα μονάδας που έχει ήδη επιλεγεί περνούν απαρατήρητες. (Π.χ. ένα γεγονός που συμβαίνει μόνο όταν η διεργασία A εκτελεί πάνω από τρεις φορές μια λειτουργία ανάμεσα σε δύο λειτουργίες της διεργασίας B). Μια λύση σε αυτό το πρόβλημα θα ήταν η επιλογή μιας «αρκετά μικρής» μονάδας χρόνου, αλλά μια τέτοια πρακτική μπορεί να οδηγήσει σε εξαιρετικά μη αποδοτικούς αλγόριθμους επαλήθευσης.

Μια άλλη κατηγορία μοντέλων χρόνου υποθέτει ότι ο χρόνος είναι συνεχής, αλλά ιδιότητες με χρόνο εκφράζονται συγκρίνοντας γεγονότα που συμβαίνουν σε ένα σύστημα με την

[1]Υπενθυμίζουμε ότι οι ω–κανονικές εκφράσεις είναι η επέκταση των κανονικών εκφράσεων για λέξεις απείρου μήκους πάνω σε ένα πεπερασμένο αλφάβητο.

τιμή ενός φανταστικού ρολογιού. Το ρολόι είναι ορατό σε όλο το σύστημα, και κτυπά σε κάποιο γνωστό προκαθορισμένο ρυθμό [AK83, AH90, Bur89, Ost90]. Εδώ ένας περιορισμός είναι ότι η αναπαράσταση της χρονικής πληροφορίας δεν είναι ακριβής. Αν υποθέσουμε ότι το ρολόι κτυπά κάθε δευτερόλεπτο, είναι αδύνατο να εκφράσουμε ακριβώς το γεγονός «Το γεγονός β συμβαίνει το πολύ δύο δευτερόλεπτα μετά το γεγονός α». Αν επιχειρήσουμε να πούμε «Δεν υπάρχουν πάνω από n χτύποι του ρολογιού ανάμεσα στα α και β» τότε : Είτε $n = 1$ οπότε παραβλέπουμε την περίπτωση που το α συμβαίνει σε χρόνο 1.9 και το β σε χρόνο 3.1, όπου α και β απέχουν μόνο 1.2 δευτερόλεπτα, είτε $n = 2$ οπότε επιτρέπουμε την περίπτωση που το α συμβαίνει σε χρόνο 1.1 και το β σε χρόνο 3.9, όπου α και it β απέχουν 2.8 δευτερόλεπτα.

Η τρίτη κατηγορία περιλαμβάνει μοντέλα στα οποία ο χρόνος θεωρείται συνεχές διάστημα με άκρα ακεραίους αριθμούς, χωρίς όμως να υπάρχει ένα ρολόι για όλο το σύστημα το οποίο να παίρνει διακριτές τιμές. Το μοντέλο αυτό πλησιάζει πιο πολύ την πραγματικότητα από αυτά των προηγούμενων δύο κατηγοριών, μια και στην πραγματικότητα ο χρόνος φαίνεται να είναι συνεχής ποσότητα. Εχει χρησιμοποιηθεί λιγότερο συχνά από αυτά, επειδή η εφαρμογή μεθόδων αυτόματης ανάλυσης είναι δυσκολότερη εδώ. Το κύριο πρόβλημα είναι η ενσωμάτωση της πληροφορίας του χρόνου που περνά ανάμεσα στα διάφορα γεγονότα, ως μέρος της συνολικής κατάστασης του συστήματος. Ομως επειδή υπάρχουν άπειρες διαφορετικές χρονικές στιγμές ακόμα και σε πολύ μικρά διαστήματα, χρειάζεται αρκετή εξυπνάδα γιά να πετύχουμε εισαγωγή αυτής της χρονικής πληροφορίας σε πεπερασμένη αναπαράσταση του συστήματος.

Ολες οι γνωστές γενικές μέθοδοι για ανάλυση συστημάτων πραγματικού χρόνου με πεπερασμένες καταστάσεις βασίζονται στην ιδέα της αναπαράστασης ενός απείρου πλήθους καταστάσεων που περιέχουν χρόνο, από ένα πεπερασμένο αριθμό κλάσεων ισοδυναμίας. Η πρώτη προσπάθεια από τους Berthomieu και Menasche έγινε πάνω σε δίκτυα Petri με χρόνο [BM83], όπου σταθερά άνω και κάτω φράγματα καθορίστηκαν σε κάθε μετάβαση για την καθυστέρηση από την ώρα που η μετάβαση επιτρέπεται για πρώτη φορά, μεχρι την ώρα που πραγματοποιείται. Εδώ υπάρχει μια πραγματική μεταβλητή για κάθε μετάβαση, η οποία παρακολουθεί το διάστημα ανάμεσα στην τρέχουσα τιμή του χρόνου και την ώρα που πρέπει να γίνει η μετάβαση. Ο αλγόριθμος ανάλυσης χρησιμοποιεί συστήματα γραμμικών ανισοτήτων που αναπαριστούν σύνολα εκχωρήσεων στις μεταβλητές. Ο κατασκευή του αλγορίθμου είναι δυνατή, γιατί για ασφαλή δίκτυα Petri ο αριθμός τέτοιων συστημάτων είναι πεπερασμένος.

Αργότερα ο Dill περιέγραψε έναν αλγόριθμο για ανάλυση μιας κατηγορίας πεπερασμένων αυτομάτων που αποδέχονται άπειρες λέξεις τα Büchi αυτόματα, μαζί με χρονικούς περιορισμούς που βασίζονται σε παρόμοια συστήματα ανισοτήτων με αυτά που χρησιμοποιεί ο αλγόριθμος των δικτίων Petri [Dil89]. Οι χρονικοί περιορισμοί συσχετίζονται με τα αυτόματα ορίζοντας ειδικά γεγονότα «θέσης» (set event) και «εκπνοής» (expire event), καθώς και σταθερά άνω και κάτω άκρα των χρονικών διαστημάτων ανάμεσα σε γεγονότα θέσης και εκπνοής. Ο αλγόριθμος είναι χρήσιμος για την επαλήθευση ιδιοτήτων ανεξάρτητων από χρόνο σε συστήματα με χρονικούς περιορισμούς. Περίπου την ίδια εποχή, ο Lewis παρουσίασε ένα παρόμοιο αλλά ειδικότερο αλγόριθμο, για βραχυκυκλώματα σε ασύγχρονα κυκλώματα με χρονικούς περιορισμούς.

Αρκετά πρόσφατα οι Alur και Dill [AD90, Alu91] ασχολήθηκαν με προβλήματα αποφάσεων συναφή με κανονικές γλώσσες με χρόνο, και πεπερασμένα αυτόματα με χρόνο. Επίσης οι Alur, Κουρκουμπέτης και Dill [ACD90, ACD91a, ACD91b] ασχολήθηκαν με προβλήματα αποφάσεων συναφή με επεκτάσεις για πραγματικό χρόνο χρονικών λογικών, και αυτομάτων με χρόνο. Υπήρξαν αρκετά ενδιαφέροντα αποτελέσματα, όπως π.χ. αποδείχτηκε ότι η έγκλειση γλωσσών είναι μη αποφασίσιμη για μη αιτιοκρατικά αυτόματα, αλλά αποφασίσιμη για αιτιοκρατικά αυτόματα. Το αποτέλεσμα αυτό είναι ιδιαίτερα χρήσιμο επειδή η έγκλειση γλωσσών είναι βασική έννοια στην επαλήθευση χρονικών ιδιοτήτων. Η ίδια γενική ιδέα των προηγούμενων μεθόδων—δηλ. η χρήση πεπερασμένου αριθμού κλάσεων ισοδυναμίας για αναπαράσταση καταστάσεων με χρόνο—έλυσε αρκετά προβλήματα ακόμα. Ο αριθμός των καταστάσεων είναι συνάρτηση της ακριβούς σχέσης ισοδυναμίας που χρησιμοποιείται. Η συγκεκριμένη σχέση ισοδυναμίας που χρησιμοποιήθηκε δημιούργησε πολύ λιγότερες καταστάσεις από ότι στη θεωρία για την χειρότερη περίπτωση· βέβαια είναι πιθανό ότι οι προηγούμενες μέθοδοι θα δημιουργούσαν πολύ λιγότερες καταστάσεις στις συνηθισμένες περιπτώσεις.

Στη συνέχεια παρουσιάζεται η επέκταση του S/R μοντέλου όπου βασίζεται το COSPAN (δες [CDT92]) με την οποία γίνεται δυνατή η επαλήθευση συστημάτων με περιορισμούς πραγματικού χρόνου. Οι περιορισμοί είναι άγνωστες αλλά πεπερασμένες καθυστερήσεις. Η μέθοδος που χρησιμοποιείται είναι ουσιαστικά αυτή που προτείνει ο Dill [Dil89], που θεωρείται η πλέον κατάλληλη μια και το COSPAN επίσης βασίζεται σε πεπερασμένα αυτόματα πάνω σε άπειρες λέξεις. Το μοντέλο του χρόνου είναι συνεχές: απεριόριστος αριθμός γεγονότων μπορεί να συμβεί ανάμεσα σε δυό διαφορετικές χρονικές στιγμές. Η βασική προσέγγιση στο πρόβλημα είναι η επαύξηση της περιγραφής του συστήματος με μιά αυτόματα κατασκευαζόμενη διεργασία ειδικού τύπου (ελεγκτής—monitor), η οποία αποκλείει ακολουθίες γεγονότων ασυμβίβαστες με ένα δεδομένο σύνολο χρονικών περιορισμών. Ο ελεγκτής λειτουργεί εξετάζοντας συστήματα γραμμικών ανισοτήτων. Η επαλήθευση πραγματοποιείται δυναμικά (on-the-fly), δηλ. η πληροφορία που αφορά το χρόνο δημιουργείται δυναμικά, λύση απαραίτητη για να ανταπεξέλθουμε το μεγάλο πλήθος καταστάσεων που εισάγει ο συνεχής χρόνος. Επίσης έχει υλοποιηθεί μια προσεκτικά σχεδιασμένη στρατηγική διαχείρισης μνήμης, ώστε να αποφύγουμε ελάττωση της απόδοσης εξαιτίας των συχνών ανταλλαγών σελίδων μνήμης (page swaps). Ειδικά χαρακτηριστικά του COSPAN όπως ο τερματισμός ψαξίματος (search truncation) χρησιμοποιούνται επιπλέον για την ελάττωση του αριθμού των καταστάσεων και την επιτάχυνση της ανάλυσης του συστήματος. Η μέθοδος έχει υλοποιηθεί εξ ολοκλήρου σαν μιά ανεξάρτητη ενότητα κώδικα χωρίς να απαιτηθούν αλλαγές στον κώδικα του COSPAN, πράγμα πολύ σημαντικό καθώς δείχνει ότι η μέθοδος μας θα μπορούσε αρκετά εύκολα να εφαρμοστεί και για άλλα εργαλεία επαλήθευσης.

Η παρουσίαση οργανώνεται ως εξής: Στα επόμενα δύο κεφάλαια παρέχουμε αρκετή υποδομή στον αναγνώστη ώστε να κατανοήσει τη δουλειά μας. Το κεφάλαιο 2 περιγράφει το μοντέλο μηχανών πεπερασμένων καταστάσεων Ε/Α (S/R) και τη χρήση του ως μέθοδο επαλήθευσης. Το εργαλείο COSPAN παρουσιάζεται στο κεφάλαιο 3, όπου περιγράφονται οι βασικές κατασκευές της γλώσσας του και η δομή του σε υψηλό επίπεδο. Στη συνέχεια μέσα στο κεφάλαιο 4 εξηγείται το μοντέλο συνεχούς χρόνου και ο τρόπος με τον οποίο υλοποιούνται οι χρονικοί περιορισμοί μέσα στο μοντέλο Ε/Α. Στο κεφάλαιο 5 εξηγούμε τη διασύνδεση χρήστη–εργαλείου που παρέχει το νέο εργαλείο μας RTCOSPAN, και συζητάμε σε βάθος την εσωτερική δομή και οργάνωση του. Τέλος στο κεφάλαιο 6 δίνουμε δύο μη τετριμένα παραδείγματα επαλήθευσης συστημάτων με χρονικούς περιορισμούς χρησιμοποιώντας το RTCOSPAN, και παρουσιάζουμε τα συμπεράσματα της εργασίας μας στο κεφάλαιο 7. Αναγνώστες που επιθυμούν να εξετάσουν προσεκτικότερα την εσωτερική δομή του RTCOSPAN θα πρέπει να διαβάσουν τα ευρετήρια A και B, ενώ οι χρήστες του RT-COSPAN θα βρούν το ευρετήριο C αρκετά ενδιαφέρον.

Κεφάλαιο 2

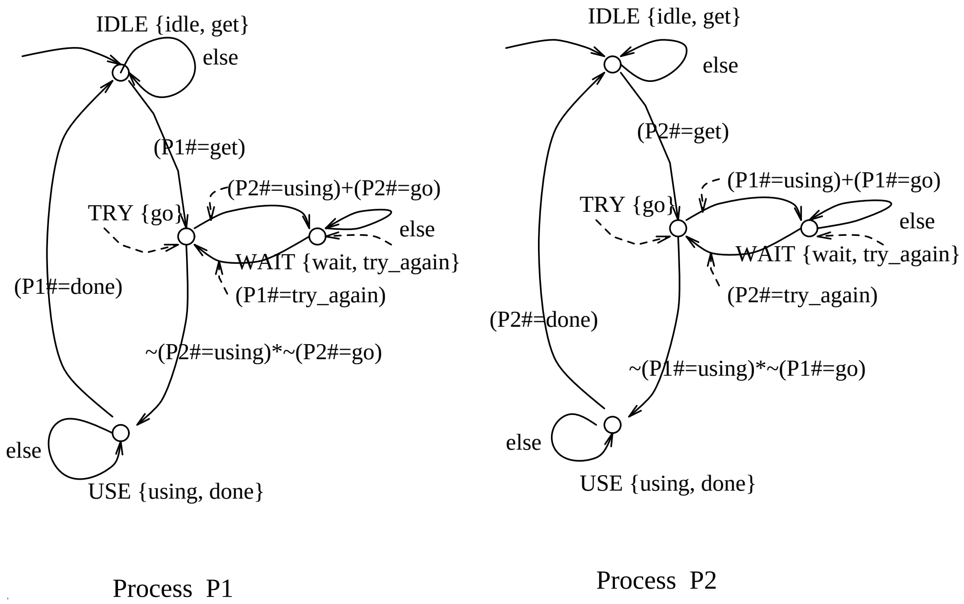
Το μοντέλο επιλογής/απόφασης (S/R model)

Το μοντέλο επιλογής/απόφασης (selection/resolution model) ή απλά μοντέλο Ε/Α (S/R model) [SA83, AC85, BG80, Kur90, SA86] είναι ένα μαθηματικό μοντέλο συγχρονισμού για συνδρομικές (concurrent) διεργασίες, όπου ένα πολύπλοκο σύστημα μπορεί να περιγραφεί σα σύνολο από μηχανές πεπερασμένων καταστάσεων οι οποίες επικοινωνούν μεταξύ τους. Μια ιδιότητα που κάνει το μοντέλο Ε/Α εξαιρετικά ελκυστικό είναι ότι η συνδέση των μηχανών γίνεται μέσω συνθηκών πάνω στα σήματα επικοινωνίας. Σε πολλές περιπτώσεις αυτό βοηθάει στην παραγωγή σύντομων, περιεκτικών και κατανοητών περιγραφών.

2.1 Εισαγωγικές έννοιες και ορισμοί

Ενα σύστημα το οποίο περιγράφεται με χρήση του μοντέλου Ε/Α απαρτίζεται από ένα πεπερασμένο σύνολο απλών συνιστωσών που ονομάζονται *διεργασίες*. Κάθε διεργασία μπορεί να παρασταθεί από ένα κατευθυνόμενο γράφο με ετικέττες πάνω στις ακμές του, όπως στο σχήμα 2.1. Οι *καταστάσεις* της διεργασίας παριστάνονται από τις κορυφές του γράφου και αποτελούν το σύνολο V δυνατών τιμών της *μεταβλητής κατάστασης* της διεργασίας. Μια κατάσταση περιέχει κωδικοποιημένη πληροφορία της ιστορίας της διεργασίας, και στη γενική περίπτωση δεν υπάρχει πρόσβαση σε αυτήν από άλλες διεργασίες. Το διάνυσμα με συνιστώσες τις καταστάσεις όλων των διεργασιών ενός συστήματος ονομάζεται η *ολική κατάσταση* του συστήματος.

Κάθε διεργασία μπορεί να έχει μια *μεταβλητή επιλογής* ή απλά επιλογή με πεπερασμένο πάντα πεδίο τιμών S. Τότε, σε κάθε κατάσταση η διεργασία διαλέγει μία μέσα από ένα σύνολο δυνατών επιλογών (Αυτό το σύνολο είναι υποσύνολο του S). Στο γράφο του σχήματος 2.1 τα σύνολα αυτά εμφανίζονται μέσα σε άγκιστρα δίπλα σε κάθε κορυφή. Η



Σχήμα 2.1: Ενα παράδειγμα αμοιβαίου αποκλεισμού, όπως παρουσιάζεται σε ένα απλό μοντέλο ενός καναλιού επικοινωνίας πολλαπλής πρόσβασης με ανίχνευση συγκρούσεων και μηχανισμό οπισθοχώρησης.

επιλογή γίνεται με μη-αιτιοκρατικό τρόπο, και οι τιμές της μπορούν να θεωρηθούν σήματα εξόδου της διεργασίας ή τμήμα κοινής (shared) μνήμης του συστήματος που χρησιμοποιείται για το συγχρονισμό των διεργασιών. Κάθε διεργασία μπορεί να διαβάζει τις τιμές όλων των μεταβλητών επιλογής των υπολοίπων διεργασιών, ενώ μπορεί να μεταβάλλει την τιμή μόνο της δικής της μεταβλητής επιλογής. Το διάνυσμα με συνιστώσες τις επιλογές όλων των διεργασιών ενός συστήματος ονομάζεται *ολική επιλογή* του συστήματος, και ουσιαστικά είναι η κοινή μνήμη μέσω της οποίας επιτυγχάνεται η συγχρονισμός του συστήματος.

Κάθε κατευθυνόμενη ακμή του γράφου αντιστοιχεί σε μια μετάβαση κατάστασης της διεργασίας, η οποία πραγματοποιήται σε ένα υπολογιστικό βήμα. Ολα τα υπολογιστικά βήματα αποτελούνται από δύο επιμέρους φάσεις. Προηγείται μια φάση επιλογής, όπου κάθε διεργασία αναθέτει μια τιμή στη μεταβλητή επιλογής της, ώστε να σχηματιστεί το τρέχον διάνυσμα της ολικής επιλογής. Ακολουθεί η φάση της απόφασης, όπου κάθε διεργασία εξετάζει ποιές από τις δυνατές μεταβάσεις από την κατάσταση στην οποία βρίσκεται είναι συμβατές με τις επιλογές όλων των διεργασιών. Η διαδικασία είναι ως εξής: Σε κάθε δυνατή μετάβαση αντιστοιχεί μια λογική συνθήκη πάνω στην ολική επιλογή του συστήματος. Οταν η συνθήκη είναι αληθής, η μετάβαση επιτρέπεται να πραγματοποιηθεί. Στο γράφο του σχήματος 2.1 οι συνθήκες μετάβασης αναγράφονται δίπλα στην αντίστοιχη ακμή. Για απλοποίηση στο συμβολισμό όταν δίπλα σε μια ακμή αναγράφεται η δεσμευμένη λέξη **else** η συνθήκη ισούται με την άρνηση της διάζευξης των συνθηκών πάνω στις υπόλοιπες εξερχόμενες ακμές. Στη συνέχεια όλες οι διεργασίες μεταβαίνουν σε μια από τις επιτρεπτές επόμενες καταστάσεις.

Στην περίπτωση που μια διεργασία μπορεί να πραγματοποιήσει πάνω από μια μεταβάσεις, διαλέγει κάποια με μη-αιτιοκρατικό τρόπο.

Από τα παραπάνω παρατηρούμε ότι, κάθε διεργασία (δηλ. κάθε μια από τις συνιστώσες του συστήματος) είναι ουσιαστικά μιά μηχανή πεπερασμένων καταστάσεων. Μπορούμε να θεωρήσουμε ότι κάθε τέτοια μηχανή παράγει μια γλώσσα που απαρτίζεται από απείρου μήκους ακολουθίες των τιμών της μεταβλητής επιλογής της διεργασίας. Η παρατήρηση αυτή είναι ουσιώδης επειδή όπως θα δούμε στη συνέχεια, παρόμοια περιγράφονται και οι ιδιότητες που θέλουμε να ελεγχθούν στο υπό εξέταση σύστημα και ο έλεγχος βασίζεται σε συνθήκες αποδοχής, δηλ. ουσιαστικά χρησιμοποιούνται πεπερασμένα αυτόματα πάνω σε άπειρες λέξεις.

2.2 Διεργασίες Ε/Α

Ας ονομάσουμε $X_1, X_2, \ldots X_M$ τις πεπερασμένες το πλήθος μεταβλητές που απαρτίζουν την κοινή μνήμη του συστήματος. Κάθε X_i παίρνει τιμές από ένα πεπερασμένο σύνολο. Ορίζουμε $\mathcal{L}$ να είναι μια άλγεβρα Bool με στοιχεία τα κατηγορήματα (συνθήκες) πάνω στις μεταβλητές $X_1, X_2, \ldots X_M$, και στην οποία η πράξη ΚΑΙ (γινόμενο) συμβολίζεται με $\cdot$, η πράξη Η (άθροισμα) συμβολίζεται με $+$ και η πράξη ΟΧΙ (άρνηση) με $\neg$. Ως $\mathcal{L}$-διεργασία P (ή απλά διεργασία) ορίζεται η 7-άδα

$$P \;=\; (V,\, S,\, \sigma,\, M,\, I,\, CY,\, RE)$$

όπου

(1) V είναι το σύνολο των καταστάσεων της P.

(2) S είναι το σύνολο των επιλογών της P και $S \subset \mathcal{L}$.

(3) σ είναι η συνάρτηση επιλογής (selector function) της P και $\sigma : V \rightarrow 2^S$.

(4) M είναι ο πίνακας μεταβάσεων (transition matrix) της P και $M : V \times V \rightarrow \mathcal{L}$.

(5) I είναι η αρχική κατάσταση της P και $I \in V$.

(6) $CY = CY_1, \ldots, CY_m$, $CY_i \subset V$, $i = 1, \ldots, m$, είναι ένα σύνολο από σύνολα επαναλαμβανόμενων καταστάσεων τα οποία ονομάζονται *cysets*. Η ονομασία προέρχεται από τον αγγλικό όρο «cycling sets».

(7) $RE \subseteq V \times V$, είναι ένα σύνολο από επαναλαμβανόμενες ακμές *(recurring edges)*. (Δηλαδή ακμές που η P διασχίζει άπειρα συχνά αλλά όχι απαραίτητα με ένα περιοδικό τρόπο).

Η συνάρτηση μετάβασης σ καθορίζει για κάθε κατάσταση v το σύνολο των δυνατών επιλογών $\sigma(v)$ που μπορούν να γίνουν όταν η P βρίσκεται στην κατάσταση v. Ο πίνακας μεταβάσεων M μπορεί να θεωρηθεί πίνακας γειτονιάς (adjacency matrix) ενός κατευθυνόμενου γράφου με σύνολο κορυφών V. Τα μη μηδενικά στοιχεία του είναι οι ετικέττες που περιγράφουν τις συνθήκες που κρίνουν αν μια μετάβαση είναι επιτρεπτή. Για παράδειγμα αν η ετικέττα της ακμής από την κατάσταση v στην w είναι $\ell = M(v,w)$ και η επιλογή της διεργασίας στην κατάσταση v είναι s, τότε η μετάβαση στην κατάσταση w είναι επιτρεπτή <u>ανν</u> $s \cdot \ell \neq 0$. Τα σύνολα επαναλαμβανόμενων καταστάσεων και ακμών στους ορισμούς (6) και (7) τα οποία χρησιμοποιούνται για την εισαγωγή συνθηκών ζωτικότητας στο μοντέλο των $\mathcal{L}$-διεργασιών, και για την επαλήθευση ορθής λειτουργίας του συστήματος, περιγράφονται παρακάτω μαζί με τις αντίστοιχες έννοιες.

Η άλγεβρα $\mathcal{L}$ δημιουργείται από το σύνολο των ατομικών κατηγορημάτων πάνω στις μεταβλητές $X_1, \ldots, X_M$. Εκτός από τις ετικέττες πάνω στις ακμές, και οι επιλογές των διεργασιών μπορούν να θεωρούνται κατηγορήματα της άλγεβρας $\mathcal{L}$. Ας συμβολίσουμε ως $P_i\#$ τη μεταβλητή επιλογής της διεργασίας P_i που αντιστοιχεί σε κάποια μεταβλητή X_i. Οταν η διεργασία P_i επιλέγει την τιμή a το κατηγόρημα $(P_i\# = a)$ είναι αληθές. Οι επιλογές δεν μπορούν να είναι οποιαδήποτε στοιχεία της άλγεβρας $\mathcal{L}$, παρά μόνο ατομικά κατηγορήματα πάνω στη μεταβλητή επιλογής $P_i\#$ της μορφής $(P_i\# = a)$ όπου a κάποια δυνατή τιμή επιλογής της P_i. Αρα το σύνολο των επιλογών μιας διεργασίας P_i αντιστοιχεί στα άτομα της υποάλγεβρας της $\mathcal{L}$ που δημιουργείται από τη μεταβλητή $P_i\#$. Για παράδειγμα το κατηγόρημα $(P_i\# = a) + (P_i\# = b)$ δεν μπορεί να είναι επιλογή και επίσης στην $\mathcal{L}$ $(P_i\# = a) + (P_i\# = b) = 0$ για κάθε a, b, δηλαδή δεν είναι δυνατόν μια μεταβλητή επιλογής να παίρνει δύο (ή περισσότερες) διαφορετικές τιμές ταυτόχρονα. Αυτή η ιδιότητα ονομάζεται «ατομικότητα» *(atomicity property)*. Ακόμη ισχύει ότι για τις διεργασίες $P_1, \ldots, P_n$ και τις αντίστοιχες μεταβλητές επιλογής $(P_i\# = s_i)$, πάντοτε $\prod_{i=1}^{n}(P_i\# = s_i) \neq 0$. Αυτό σημαίνει ότι διαφορετικές διεργασίες διαλέγουν τις τιμές των μεταβλητών επιλογής τους ανεξάρτητα. Αυτή η ιδιότητα ονομάζεται »ανεξαρτησία« *(independence property)*. Ενα κατηγόρημα της μορφής $\prod_{i=1}^{M}(X_i = a_i)$ ονομάζεται *ολική επιλογή*. Εστω S_G το σύνολο όλων των δυνατών ολικών επιλογών.

Στη γενική περίπτωση, οι ετικέττες ακμών μιας διεργασίας P_i είναι κατηγορήματα που περιέχουν και μεταβλητές επιλογής άλλων διεργασιών εκτός από την $P_i\#$, επειδή συνήθως οι μεταβάσεις εξαρτώνται και από το τι κάνουν και οι υπόλοιπες διεργασίες. Μια διεργασία της οποίας τα κατηγορήματα στις ετικέττες ακμών εξαρτώνται μόνο από τις τιμές της δικής της μεταβλητής επιλογής ονομάζεται «ολικά ορισμένη»(completely defined) διεργασία. Μια τέτοια διεργασία παρουσιάζει συμπεριφορά ανεξάρτητη από τις υπόλοιπες διεργασίες του συστήματος.

Στο COSPAN το παραπάνω μοντέλο επεκτείνεται ώστε να επιτρέπεται ένας πιο γενικός ορισμός της άλγεβρας $\mathcal{L}$. Οι ετικέττες που αντιστοιχούν στις μεταβάσεις των διεργασιών επιτρέπεται να είναι κατηγορήματα πάνω και στις τιμές της κατάστασης των διεργασιών εκτός από τις επιλογές τους. Για κάθε διεργασία P_i υπάρχει μια μεταβλητή *κατάστασης* P_i\$ που παίρνει τιμές από το σύνολο καταστάσεων της διεργασίας. Με αυτό τον τρόπο επιτρέπεται σε μια διεργασία να «δει» την κατάσταση των άλλων διεργασιών·αυτό γίνεται ώστε να παράγονται απλούστερες περιγραφές, αλλιώς η πληροφορία της κατάστασης θα έπρεπε να κωδικοποιηθεί στις επιλογές της διεργασίας στην συγκεκριμένη κατάσταση. Πάντως μπορεί να δειχτεί ότι αυτό το χαρακτηριστικό δεν προσθέτει παραπάνω εκφραστική δύναμη στο αρχικό μοντέλο.

2.2.1 Σύνθεση διεργασιών Ε/Α

Ενα από τα σημαντικά χαρακτηριστικά του μοντέλου Ε/Α είναι ότι είναι δυνατό να οριστεί ένας λογισμός πάνω στις διεργασίες, έτσι ώστε το γινόμενο διεργασιών να είναι πάλι διεργασία. Αυτό το γινόμενο διεργασιών αντιπροσωπεύει την κοινή εκτέλεση τους, και καθώς ισχύουν οι ιδιότητες της ατομικότητας και της ανεξαρτησίας, ο σχηματισμός της διεργασίας γινομένου είναι ουσιαστικά το ίδιο με το να θεωρούμε όλες τις συνιστώσες διεργασίες μαζί σαν μία οντότητα. Η διεργασία γινόμενο είναι το γινόμενο των γράφων των ξεχωριστών διεργασιών. Η πράξη του γινομένου τελικά είναι το *τανυστικό γινόμενο* των πινάκων μεταβάσεων των ξεχωριστών διεργασιών.

Εστω $\otimes$ το συνηθισμένο τανυστικό γινόμενο πινάκων, δηλαδή $(A \otimes B)_{(i,j),(k,l)} = A_{(i,k)} \cdot B_{(j,l)}$ όπου $\cdot$ είναι ο τελεστής γινόμενο στην άλγεβρα Bool $\mathcal{L}$. Με δεδομένες τις $\mathcal{L}$ διεργασίες $P_1, \ldots, P_n$, έστω $P_i = (V_i, S_i, \sigma_i, M_i, I_i, CY_i, RE_i)$ και $CY_i = CY_{i1}, \ldots, CY_{im_i}$. Ορίζουμε το γινόμενο τους $P = \otimes_{i=1}^n P_i$ να είναι η $\mathcal{L}$ διεργασία $P = (V, S, \sigma, M, I, CY, RE)$ όπου

(1) $V = \times_{i=1}^n V_i$.

(2) $S = \Pi_{i=1}^n S_i$. Π συμβολίζει το γινόμενο των συνόλων από κατηγορήματα (ή συναρτήσεις πάνω σε κατηγορήματα) ως συνήθως, δηλαδή αν $C, D \in 2^{\mathcal{L}}$, τότε $C \cdot D = \{c \cdot d \,|\, c \in C,\ d \in D\}$.

(3) $\sigma = \Pi_{i=1}^n \sigma_i$.

(4) $M = \otimes_{i=1}^n M_i$.

(5) $I = (I_1, \ldots, I_n)$.

(6) $CY = \{CY'_{ij}\}, i = 1, \ldots, n, j = 1, \ldots, m_i$, όπου το CY'_{ij} περιέχει όλες τις καταστάσεις στο V οι οποίες προβάλονται κατά την i-οστή συνιστώσα σε μια κατάσταση μέσα στο σύνολο CY_{ij}.

(7) Το RE περιέχει όλες τις μεταβάσεις στο $V \times V$ οι οποίες προβάλονται κατά κάποια συνιστώσα i σε μια μετάβαση στο RE_i, για κάποιο $i = 1, \ldots, n$.

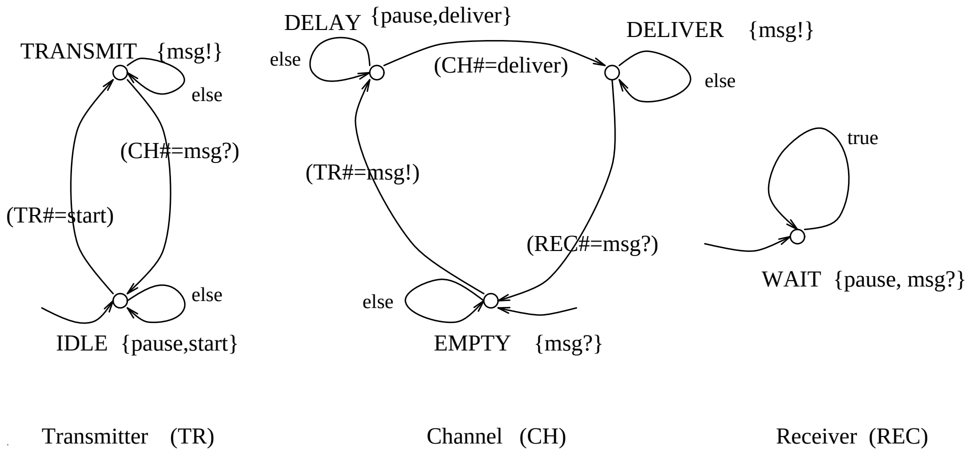
2.3 Σημασιολογία διεργασιών και επαλήθευση

2.3.1 Συμπεριφορές διεργασιών

Το μοντέλο Ε/Α είναι ένα μοντέλο *διακριτού χρόνου*, με άλλα λόγια, το πεδίο ορισμού και των δύο μεταβλητών κατάστασης και επιλογής μιας διεργασίας είναι οι μη-αρνητικοί ακέραιοι. Μια ακολουθία από ζεύγη κατάστασης-επιλογής είναι αρκετή για να περιγράψει μια *συμπεριφορά διεργασίας*. Ας υποθέσουμε ότι η διεργασία κατά τη χρονική στιγμή k βρίσκεται στην κατάσταση $v = v(k)$, και διαλέγει μια επιλογή $s = s(k)$ όπου $s \in \sigma(v)$. Τότε, στην επόμενη χρονική στιγμή $k + 1$, η κατάσταση $v' = v(k + 1)$ και η επιλογή $s' = s(k+1)$ μπορούν να βρεθούν ως εξής. Υπολογίζουμε το $s \cdot M(v, w)$ για όλα τα $w \in V$. Εστω το σύνολο των δυνατών επόμενων καταστάσεων $N(v, s) = \{w | s \cdot M(v, w) \neq 0, w \in V\}$. Διαλέγουμε κάποιο στοιχείο $v' \in N(v, s)$ σαν επόμενη κατάσταση. Εαν υπάρχουν παραπάνω από μια επόμενες καταστάσεις στο $N(v, s)$, διαλέγουμε μια οποιαδήποτε. Εαν το $N(v, s)$ είναι κενό, θεωρούμε τη συγκεκριμένη ανάθεση ετικέττας λανθασμένη. Σε αυτή την περίπτωση έχουμε μια κατάσταση γνωστή ως *μπλοκάρισμα* (lockup), όπου δεν ορίζεται καμία επόμενη κατάσταση και καμία μελλοντική συμπεριφορά δεν είναι δυνατή. [1] Αυτή δεν είναι νόμιμη συμπεριφορά για μια διεργασία Ε/Α αφού όλες οι συμπεριφορές διεργασιών Ε/Α είναι άπειρες. Από αυτό συμπεραίνουμε ότι στο μοντέλο Ε/Α η διάζευξη των ετικεττών πάνω στις δυνατές μεταβάσεις που ξεκινούν από μια κατάσταση είναι πάντα αληθής (*true*). Μετά τον καθορισμό μιας επόμενης κατάστασης, η διεργασία καθορίζει μία νέα επιλογή $s' \in \sigma(v')$. Παρόμοια, αν υπάρχουν παραπάνω από μια δυνατές επιλογές σε μια κατάσταση v', πραγματοποιείται μια μη αιτιοκρατική εκλογή επιλογής.

Εχοντας ολοκληρώσει τους ορισμούς πάνω στη συμπεριφορά μιας μοναδικής διεργασίας P, τώρα εξετάζουμε την από κοινού συμπεριφορά πολλών διεργασιών. Θεωρώντας τις διεργασίες $P_1, \ldots, P_n$, ορίζουμε την από κοινού συμπεριφορά αυτών των διεργασιών ως εξής: Κατασκευάζουμε την $P = \otimes_{i=1}^{n} P_i$ και μετά παίρνουμε προβολές για να καθορίσουμε τις

[1] Δεν πρέπει να συγχέεται το μπλοκάρισμα με το αδιέξοδο (deadlock). Το αδιέξοδο στο μοντέλο Ε/Α προκύπτει σε μια κατάσταση v όταν για όλες τις δυνατές επιλογές $s_1, \ldots, s_n$ σε αυτή την κατάσταση, $N(v, s_i) = \{v\} i = 1, \ldots, n$.



Σχήμα 2.2: Ενα παράδειγμα καναλιού με καθυστέρηση. Η διεργασία κανάλι καθυστερεί το μήνυμα επιλέγοντας παύση.

συμπεριφορές διεργασίας των συνιστωσών. Αυτές οι από κοινού συμπεριφορές είναι μονοπάτια δείγματα της από κοινού συμπεριφοράς όλων των διεργασιών. Αυτές οι πράξεις ορίζουν το *σύγχρονο γινόμενο* των $P_1, \ldots, P_n$. Ομως υπάρχουν περιπτώσεις όπου θέλουμε να μοντελοποιήσουμε συσκευές που έχουν «απροσδιόριστη» έξοδο για μια χρονική περίοδο. Ασύγχρονες συσκευές αυτού του τύπου μπορούν να μοντελοποιηθούν προσθέτοντας μια επιλογή που ονομάζεται «παύση» και παρέχοντας κατάλληλη ανάθεση ετικεττών σχηματίζοντας με αυτό τον τρόπο ένα *ασύγχρονο γινόμενο.* Θεωρήστε το παράδειγμα του σχήματος 2.2, όπου έχουμε το μοντέλο ενός εκπομπού, ενός δέκτη και ένα κανάλι επικοινωνίας με ακαθόριστη καθυστέρηση μετάδοσης. Αν διαλέξουμε μια επιλογή παύσης, συμβαίνει ένας «βρόγχος στην ίδια κατάσταση» (self-loop) ή αλλιώς «μετάβαση παύσης» (pause transition). Τέτοιες μεταβάσεις παύσης μπορούν να πραγματοποιηθούν μηδέν ή περισσότερες φορές. Μια διεργασία μπορεί να πραγματοποιήσει ένα οποιονδήποτε αριθμό μεταβάσεων μη-παύσης, την ώρα που μια άλλη διεργασία βρίσκεται σε παύση (Αυτή είναι και η ουσία της ασύγχρονης συμπεριφοράς). Στο παραπάνω παράδειγμα μπορεί να περάσει οσοσδήποτε χρόνος ανάμεσα στις στιγμές που γενιούνται δυό συνεχόμενα μηνύματα. Για όλοκληρο αυτό το χρονικό διάστημα η διεργασία εκπομπός βρίσκεται σε παύση. Το κανάλι μπορεί να δημιουργήσει οσοδήποτε μεγάλες καθυστερήσεις μετάδοσης επιλέγοντας παύση, και ο δέκτης μπορεί να καταναλώσει κάποιο χρονικό διάστημα όπου επεξεργάζεται ένα μηνυμα που παραδώθηκε από το κανάλι.

Ενας εναλλακτικός τρόπος για την περιγραφή των «αλυσίδων» (chain) ενός γινομένου διεργασιών το οποίο περιγράφει ένα ολοκληρωμένο σύστημα είναι ο ακόλουθος: Εστω $s = s(k) = s_1(k) \cdot s_2(k) \cdots s_n(k)$ η ολική επιλογή n διεργασιών, όπου η διεργασία P_i είναι στην κατάσταση $v_i(k)$ κατά την χρονική στιγμή k. Τότε μια δυνατή επόμενη κατάσταση $v_i(k+1)$ της διεργασίας P_i καθορίζεται διαλέγοντας ένα στοιχείο $v_i' \in N_i$, όπου $N_i(v_i, s) = \{w_i | s \cdot M_i(v_i, w_i) \neq 0, w_i \in V_i\}$. Αυτό είναι το ίδιο με το να πολλαπλασιάσουμε κάθε ετικέττα ακμής ℓ εξερχόμενης από την κατάσταση v_i με την ολική επιλογή s και να ελέγξουμε αν $s \cdot \ell \neq 0$. Μια δυνατή ολική επόμενη κατάσταση είναι τότε η $(v'_1, v'_2, \ldots, v'_n)$ όπου $v'_i \in N_i(v_i, s)$.

2.3.2 Σημασιολογία διεργασιών: Ιχνη και αλυσίδες

Η σημασιολογία μιας διεργασίας Ε/Α δίνεται μέσα από τα ίχνη (traces), τα οποία είναι παρατηρήσιμες συμπεριφορές του συστήματος. Ιχνος είναι μια άπειρη ακολουθία από ολικές επιλογές, δηλ. είναι ένα στοιχείο του S_G^{ω}.[2] Ενα *τρέξιμο* (run) μιας διεργασίας P_i που αντιστοιχεί σε ένα ίχνος $s(0)s(1)\ldots$ είναι μια απείρου μήκους ακολουθία από καταστάσεις $v_i(0)v_i(1)\ldots$, τέτοια ώστε :

(a) $v_i(0) = I_i$ (Η αρχική κατάσταση της διεργασίας P_i),

(b) Αν $s_i(k)$ είναι η προβολή του $s(k)$ πάνω στη μεταβλητή επιλογής και $P_i\#$ ($s_i(k)$ είναι η επιλογή της P_i στην ολική επιλογή, τότε $s_i(k) \in \sigma(v_i(k))$,

(c) $v_i(k+1) \in N(v_i(k), s(k))$ for $k > 0$.

Ενα ίχνος είναι ίχνος της διεργασίας P_i εαν υπάρχει ένα αντίστοιχο τρέξιμο της P_i. Διαισθητικά, τα ίχνη της διεργασίας P είναι το σύνολο με στοιχεία απείρου μήκους ακολουθίες κοινής μνήμης που μπορεί να εμφανιστούν στο σύστημα, δεδομένου ότι η διεργασία P είναι μία από τις συντονιζόμενες διεργασίες, δηλαδή το συνολο με στοιχεία ιστορίες κοινής μνήμης (παρατηρήσιμη συμπεριφορά) οι οποίες είναι συνεπείς με την P.

Σημαντική παρατήρηση: Στο COSPAN αντίθετα με το «καθαρό» μοντέλο Ε/Α, επιτρέπεται οι ετικέττες στις μεταβάσεις των διεργασιών να εξαρτώνται και από τις μεταβλητές κατάστασης εκτός από τις μεταβλητές επιλογής. Αυτό το σχήμα μπορεί να μεταφραστεί στο παραδοσιακό μοντέλο ορίζοντας ξανά την επιλογή της διεργασίας έτσι ώστε να συμπεριλάβει τις τιμές της κατάστασης της. Σε αυτό το ισοδύναμο σύστημα, η νέα μεταβλητή επιλογής της P_i είναι η $(P_i\$, P_i\#)$, και η κοινή μνήμη περιέχει τις μεταβλητές κατάστασης των διεργασιών (όχι και των ελεγκτών).

Στο υπόλοιπο αυτής της εργασίας θα συμβολίζουμε με μια *αλυσίδα* (chain) (αντί για ίχνος) μια άπειρη ακολουθία από ολικές καταστάσεις και ολικές επιλογές. Ετσι οι αλυσίδες είναι ίχνη σε αυτό το σύστημα επαυξημένης μνήμης, όπου οι επιλογές των διεργασιών ορίζονται ξανά ώστε να συμπεριλάβουν τις τιμές των καταστάσεων των διεργασιών. Μια σημαντική παρατήρηση είναι ότι οι αλυσίδες έχουν «ολοκληρωμένη πληροφορία» όσον αφορά την συμπεριφορά του συστήματος, ενώ τα ίχνη στη γενική περίπτωση είναι προβολές των

[2]Συμβολίζουμε με ω την άπειρη επανάληψη για ω-κανονικές εκφράσεις.

αλυσίδων πάνω στην συνιστώσα επιλογή, και παραπάνω από μια αλυσίδα θα μπορούσε να αντιστοιχεί στο ίδιο ίχνος. Στα παρακάτω όλοι οι ορισμοί μας θα χρησιμοποιούν σημασιολογία με αλυσίδες αντί για σημασιολογία με ίχνη, μια και ασχολούμαστε με το COSPAN. Σημειώστε ότι το σύνολο των αλυσίδων της P ορίζεται παρόμοια με την περίπτωση των ιχνών. Ο μόνος πρόσθετος περιορισμός είναι ότι το τρέξιμο της P πάνω σε μια αλυσίδα πρέπει να συμπίπτει με την τιμή της συνιστώσας κατάστασης της κατά την διάρκεια του τρεξίματος.

2.3.3 Ιχνη των $\mathcal{L}$-διεργασιών και συνθήκες ζωτικότητας

Ας εξετάσουμε την περίπτωση όπου ένα κανάλι επικοινωνίας καθυστερεί για κάποιο απροσδιόριστο αλλά πεπερασμένο χρονικό διάστημα πρίν το μεταδώσει. Είναι φανερό ότι μια διεργασία που ορίζεται με χρήση μόνο των ορισμών (1)-(5) της ενότητας 2.2 δεν μπορεί να μοντελοποιήσει μια τέτοια διάταξη επακριβώς. Ο λόγος είναι ότι αφού το μοντέλο του καναλιού επιτρέπεται να κρατήσει το μήνυμα για απροσδιόριστο χρονικό διάστημα, μπορεί να κρατήσει το μήνυμα και για πάντα. Προφανώς οι αλυσίδες της διεργασίας καναλιού στο παράδειγμα του σχήματος 2.2 περιέχουν αυτό το είδος συμπεριφοράς. Μια παρόμοια παρατήρηση ισχύει για το παράδειγμα του σχήματος 2.1, στο οποίο δεν μπορούμε να μοντελοποιήσουμε την ιδιότητα ότι μια διεργασία πάντα τελικά αποδεσμεύει τον πόρο αφού τον χρησιμοποιήσει (για κάποιο απροσδιόριστο αλλά πεπερασμένο χρονικό διάστημα). Είναι δυνατό να εκφράσουμε τέτοιες ιδιότητες *ζωτικότητας* μόνο εάν το μοντέλο μας γίνει περισσότερο δυνατό προσθέτωντας *συνθήκες αποδοχής* (acceptance conditions) στη βασική διεργασία. Ιδιότητες ζωτικότητας μπορούν να εκφραστούν στο COSPAN χρησιμοποιώντας τις ποσότητες CY και RE, που εισήγχθησαν αλλά δεν εξηγήθηκαν με λεπτομέρειες στην ενότητα 2.2. Η διατύπωση ιδιοτήτων ζωτικότητας σχετίζεται με τη θεωρία των αυτομάτων πάνω σε άπειρες λέξεις, βλέπε [Buc62, McN66, Rab72]. Η ιδέα είναι ότι μπορούμε να διακρίνουμε κατάλληλα το σύνολο των αλυσίδων που περιγράφουν μια διεργασία απαιτώντας να ισχύει η ακόλουθη συνθήκη αποδοχής:

Μια αλυσίδα c της P είναι *αποδεκτή* ανν η συνιστώσα κατάστασης της που αντιστοιχεί στην P ικανοποιεί τις παρακάτω συνθήκες:

(a) Δεν μένει τελικά μέσα σε κάποιο cyset $CY_i,\ i = 1, \ldots, m$, και

(b) Δεν πραγματοποιεί άπειρα συχνά μεταβάσεις που ανήκουν στο σύνολο RE.

Ορίζουμε ως $\mathcal{L}_P$ το σύνολο των αλυσίδων της P που είναι αποδεκτές από την P. Από τους παραπάνω ορισμούς εύκολα συνεπάγεται ότι $\mathcal{L}_{P \otimes Q} = \mathcal{L}_P \cap \mathcal{L}_Q$.

Οι συνθήκες αποδοχής στο COSPAN παρέχουν σε μια $\mathcal{L}$-διεργασία την ίδια εκφραστική δύναμη με τα ω-κανονικά σύνολα και την διευρυμένη χρονική λογική (Extended Temporal

Logic, ETL), βλέπε [Kur90, Wol83]. Αυτό υπονοεί ότι κάθε ω-κανονική ιδιότητα μπορεί να περιγραφεί ως μια $\mathcal{L}$-διεργασία. Εχει επίσης αποδειχτεί στο [Kur90] ότι οι αιτιοκρατικές $\mathcal{L}$-διεργασίες είναι εξίσου εκφραστικές με τις μη αιτιοκρατικές. Διαφορετικοί τύποι συνθηκών αποδοχής μπορούν επίσης να χρησιμοποιηθούν, βλεπε [ACW90].

Μπορούμε να περιγράψουμε την κατάλληλη ιδιότητα ζωτικότητας στο παράδειγμα του σχήματος 2.2 προσθέτωντας το cyset {DELAY} στη διεργασία κανάλι. Αυτό το κανάλι πάντα τελικά μεταδίδει τα μηνύματα που δέχεται. Οι αποδεκτές αλυσίδες είναι της μορφής [3]

$$((a|b)^*bc^*de^*)^*(a|b)^\omega \mid ((a|b)^*bc^*de^*)^*(a|b)^*(a|b)^*bc^*de^\omega \mid ((a|b)^*bc^*de^*)^\omega$$

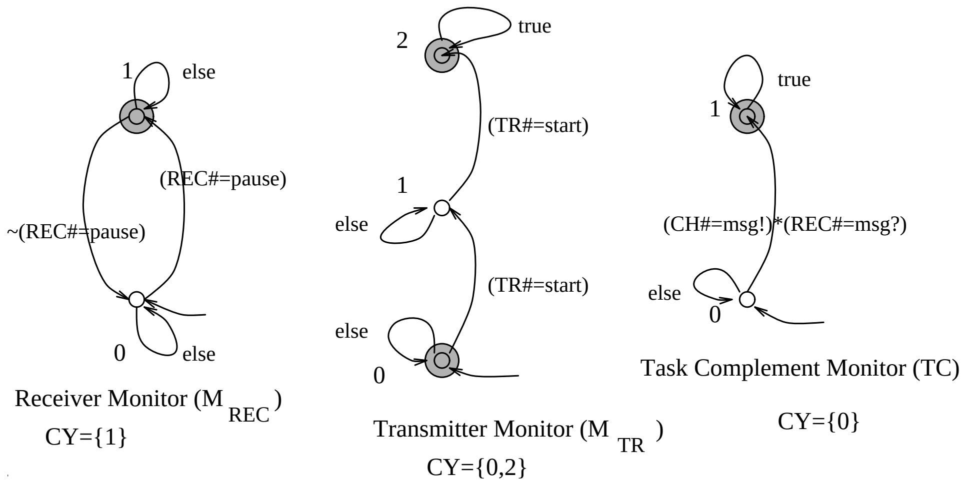
, όπου a = (EMPTY, pause), b = (EMPTY, msg?), c = (DELAY, pause), d = (DELAY, deliver), e = (DELIVER, msg!). Παρατηρήστε ότι χωρίς τη συνθήκη αποδοχής, τίποτα δεν θα εμπόδιζε την διεργασία από το να μείνει για πάντα στην κατάσταση DELAY. Παρόμοια, στο παράδειγμα του αμοιβαίου αποκλεισμού οι δύο συνθήκες ότι η διεργασία πάντα τελικά φεύγει από τις καταστάσεις WAIT και USE μπορεί να μοντελοποιηθεί προσθέτωντας τα cyset {WAIT,USE} σε κάθε διεργασία.

Οπως και στο παράδειγμα του καναλιού, ακολουθούμε τη συνθήκη να εξαλείψουμε ένα ζευγάρι άγκιστρα όταν το CY περιέχει ένα μόνο σύνολο. Ετσι γράφουμε $CY = \{\text{DELAY}\}$ αντί για $CY = \{\{\text{DELAY}\}\}$. Επίσης, αν $CY = \emptyset$, τότε είναι φανερό ότι η $\mathcal{L}$-διεργασία αποδέχεται όλες τις αλυσίδες $c \in C$. Σε τέτοιες περιπτώσεις το σύνολο CY δεν εμφανίζεται καθόλου.

Το σχήμα 2.3 παρουσιάζει ένα παράδειγμα του πώς εφαρμόζεται η μέθοδος μας για την επαλήθευση κάποιων ιδιοτήτων ζωτικότητας για το παράδειγμα του καναλιού καθυστέρησης στο σχήμα 2.2. Οι προδιαγραφές του συστήματος υποθέτουν ότι (α) ο δέκτης πάντοτε τελικά ελέγχει για εισερχόμενα μηνύματα, (β) ο εκπομπός πάντοτε τελικά εκπέμπει ένα μοναδικό μήνυμα, και (γ) το κανάλι δεν καθυστερεί ένα μήνυμα για πάντα. Αυτές οι συνθήκες περιγράφονται μέσω των ελεγκτών M_{REC}, M_{TR} στο σχήμα 2.3, και προσθέτωντας το cyset {DELAY} στη διεργασία κανάλι CH. Το «καθήκον» (task) που πρέπει να αποδειχτεί συνίσταται στο να δείξουμε ότι το μήνυμα τελικά θα παραληφθεί από το δέκτη. Αυτό κωδικοποιείται μέσω του ελεγκτή για το συμπλήρωμα του καθήκοντος M_{TC}.

Μπορούμε εύκολα να παράγουμε τις συνθήκες αποδοχής για ένα γινόμενο διεργασιών μέσω των συνθηκών αποδοχής των συνιστωσών διεργασιών. Μια αλυσίδα της διεργασίας γινόμενο είναι αποδεκτή ανν η προβολή της πάνω σε κάθε μεμονωμένη διεργασία είναι αποδεκτή από εκείνη τη διεργασία. Για παράδειγμα, έστω ότι η P_1 έχει καταστάσεις V_1 και συνθήκες αποδοχής $CY_1 = \{CY_1^1, CY_1^2\}$, και η P_2 έχει καταστάσεις V_2 και $CY_2 = CY_2^1$.

[3] Χρησιμοποιούμε το σύμβολο $\star$ για να δηλώσουμε πεπερασμένη επανάληψη, και το ω για να δηλώσουμε άπειρη επανάληψη, για κανονικές εκφράσεις και ω-κανονικές εκφράσεις αντίστοιχα.



Σχήμα 2.3: Οι συνθήκες ζωτικότητας για την περιγραφή του καναλιού καθυστέρησης. Το M_{REC} αποδέχεται όλες τις αλυσίδες στις οποίες δεν ισχύει ότι τελικά πάντοτε $REC\# = pause$. Το M_{TR} αποδέχεται όλες τις αλυσίδες στις οποίες ο εκπομπός τελικά στέλνει ένα μήνυμα και μετά σταματά. Το M_{TC} αποδέχεται όλες τις αλυσίδες στις οποίες το μήνυμα ποτέ δεν παραλαμβάνεται από τον δέκτη.

Τότε η συνθήκη αποδοχής CY της $P = P_1 \otimes P_2$ είναι $CY = \{CY_1^1 \times V_2, CY_1^2 \times V_2, V_1 \times CY_2^1\}$. Παρόμοια, $RE = (RE_1 \times V_2^2) \cup (V_1^2 \times RE_2)$.

2.3.4 Διεργασίες ελεγκτές και επαλήθευση ορθής λειτουργίας

Για την επαλήθευση της ορθής λειτουργίας ενός συστήματος και για να αποδείξουμε ότι συγκεκριμένες απαιτήσεις ικανοποιούνται, χρειαζόμαστε μια τυπική περιγραφή των επιθυμητών ιδιοτήτων του συστήματος. Στο μοντέλο Ε/Α, όπως και σε άλλα μοντέλα συμπεριφοράς βασισμένα σε *σημασιολογία ιχνών*, μια διεργασία ικανοποιεί μια ιδιότητα αν όλες οι αλυσίδες της ικανοποιούν την ιδιότητα. Στην ορολογία του COSPAN (του εργαλείου μας κάτω από το μοντέλο Ε/Α), αυτές οι ιδιότητες ονομάζονται το *καθήκον (task)* που πρέπει να επιτελέσει αυτή η διεργασία. Ενας τρόπος να καθοριστεί το καθήκον είναι να δημιουργηθεί μια διεργασία η οποία παρατηρεί, αλλά δεν συμμετέχει στην εκτέλεση του συστήματος. Μια τέτοια διεργασία χρησιμοποιείται για να τεθεί μια ερώτηση όσον αφορά το σύστημα, και θα την ονομάσουμε *ελεγκτή* (monitor). Με την αφηρημένη έννοια, ένας ελεγκτής είναι ένα πεπερασμένο αυτόματο πάνω σε άπειρες λέξεις του οποίου το αλφάβητο αποτελείται από όλα τα δυνατά ζεύγη κατάστασης-επιλογής της διεργασίας ή των διεργασιών που ελέγχει. Στο COSPAN ένας ελεγκτής είναι ακριβώς όπως μια $\mathcal{L}$-διεργασία, με την εξαίρεση ότι δεν έχει επιλογές (ή έχει μια τετριμένη επιλογή που δεν χρησιμοποιείται από άλλες διεργασίες), καθώς μονάχα παρατηρεί. Βέβαια ένα αυτόματο χρειάζεται επίσης και κάποια έννοια αποδοχής. Τα σύνολα επαναλαμβανόμενων καταστάσεων και οι επαναλαμβανόμενες ακμές (για συντομία θα τα αναφέρουμε ως cysets kai recur edges αντίστοιχα) χρησιμοποιούνται για τον ορισμό συνθηκών αποδοχής.

Ακόμα μια χρήση των ελεγκτών μπορεί να είναι ο περιορισμός της συμπεριφοράς άλλων διεργασιών. Αν ο ελεγκτής M δεν αποδέχεται όλες τις αλυσίδες μιας διεργασίας P, η $P \otimes M$ θα περιέχει μόνο τις αλυσίδες που περιέχονται και στην P και στην M. Αυτό το χαρακτηριστικό μας παρέχει επίσης ένα τρόπο να ενσωματώσουμε και χρονικούς περιορισμούς μέσα σε μια διεργασία, όπως θα δούμε στην αντίστοιχη ενότητα.

Οταν χρησιμοποιούμε το COSPAN για την επαλήθευση ενός συστήματος, αρχίζουμε με μια τυπική περιγραφή του συστήματος P που αποτελείται από ένα σύνολο συνιστωσών διεργασιών $P_1, \ldots, P_n$, όπου κάθε P_i είναι μια $\mathcal{L}$-διεργασία, ή μια διεργασία ελεγκτής που αντιστοιχεί σε κάποιους επιθυμητούς περιορισμούς. Ετσι, το σύστημα μας είναι $P = \otimes_{i=1}^{n} P_i$. Η επαλήθευση πραγματοποιείται παρέχοντας ένα ελεγκτή που περιγράφει το σύνολο των *ανεπιθύμητων* αλυσίδων. Συνήθως ονομάζουμε ένα ελεγκτή αυτού του είδους TC (από το Task Complement ή συμπλήρωμα του καθήκοντος). Επειτα, αντί να ελέγχουμε ότι κάθε αλυσίδα της P είναι επιθυμητή, είναι αρκετό να ελέγχουμε εαν *υπάρχει* μια ανεπιθύμητη αλυσίδα·με άλλα λόγια να ελέγχουμε κατά πόσον το σύνολο των αλυσίδων που αποδέχεται η $P \otimes TC$ είναι κενό. Η κενότης μιας $\mathcal{L}$-διεργασίας P μπορεί να εξεταστεί με χρήση μιας απλής γραφοθεωρητικής συνθήκης: [4]

Η P έχει μια αλυσίδα αποδοχής ανν υπάρχει μια ισχυρά συνεκτική συνιστώσα στο γράφο της διεργασίας προσβάσιμη από τη αρχική κατάσταση, η οποία περιέχει κάποιο ισχυρά συνεκτικό υποσύνολο καταστάσεων S τέτοιο ώστε:

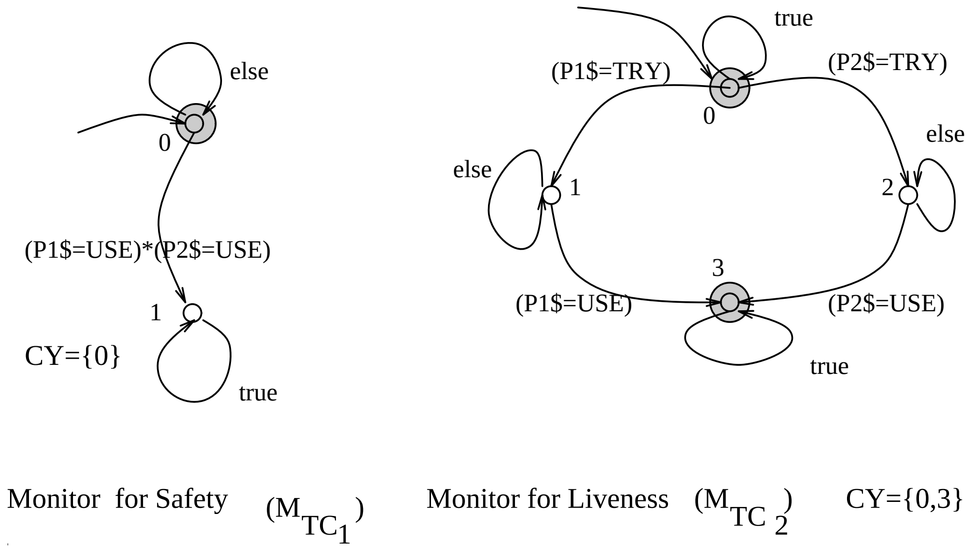
(a) Το S παραμένει ισχυρά συνεκτικό αν διαγράψουμε όλες τις μεταβάσεις που αντιστοιχούν σε επαναλαμβανόμενες ακμές, και

(β) Το S δεν περιέχεται σε κάποιο σύνολο επαναλαμβανόμενων ακμών.

Ο έλεγχος της παραπάνω συνθήκης μπορεί να γίνει σε χρόνο γραμμικό ως προς το μέγεθος του προσβάσιμου μέρους του γράφου της διεργασίας, με μια απλή τροποποίηση ενός συνηθισμένου αλγόριθμου για την εύρεση των ισχυρά συνεκτικών συνιστωσών ενός γράφου (Βλέπε [AHU74]).

Συνήθως, είναι πιο βολικό να εκφράζεται ένα καθήκον T σα γινόμενο αρκετών μικρότερων καθηκόντων, $T = T_1 \otimes T_2 \otimes \cdots \otimes T_k$. Σε αυτή την περίπτωση, μπορούμε να ελέγχουμε ανεξάρτητα εαν $P \otimes TC_i$ είναι κενό για $i = 1, \ldots, k$ για κάθε συμπλήρωμα καθήκοντος

[4] Το COSPAN μπορεί επίσης να χειρίζεται καθήκοντα που δεν είναι σε μορφή συμπληρώματος. Σε αυτή την περίπτωση μια συμπεριφορά είναι αποδεκτή εάν (α) Οι επαναλαμβανόμενες ακμές διασχίζονται άπειρες φορές, ή (β) η κατάσταση του(ων) ελεγκτή(ών) τελικά περιέχεται σε κάποιο σύνολο επαναλαμβανόμενων καταστάσεων. Μια και πάντα χρησιμοποιούμε συμπληρώματα καθηκόντων στην προσέγγιση των προβλημάτων, δεν θα συζητήσουμε παραπάνω αυτό το χαρακτηριστικό.



Σχήμα 2.4: Το συμπλήρωμα καθήκοντος για το πρόβλημα του αμοιβαίου αποκλεισμού. Η M_{TC_1} αποδέχεται όλες τις αλυσίδες που τελικά παραβιάζουν τη συνθήκη αμοιβαίου αποκλεισμού. Η M_{TC_2} αποδέχεται όλες τις αλυσίδες όπου μια διεργασία ζητάει τον πόρο χωρίς ποτέ να τον παίρνει.

TC_i. Αυτή η πρακτική είναι περισσότερο αποδοτική από το να ελέγχουμε την κενότητα της $P \otimes TC$, μια και ελατώνει το μέγεθος των συνόλων ολικών καταστάσεων τα οποία πρέπει να κατασκευαστούν. Επιπλέον η κατασκευή κάθε TC_i είναι πολύ ευκολότερη από την κατασκευή του TC. Αυτό είναι ένα πολύ σημαντικό ζήτημα, καθώς ο ορισμός ενός «καλού» καθήκοντος (ή του αντίστοιχου συμπληρώματος) μπορεί μερικές φορές να είναι μπερδεμένη δουλειά. Κάποιος που αναπτύσσει ένα σύστημα θα πρέπει να σκεφτεί αρκετά πριν αποφασίσει ότι το καθήκον που έχει υπόψην του είναι κατάλληλο για την απαιτούμενη αξιολόγηση του συστήματος.

Στο σχήμα 2.4 παρουσιάζουμε τους δύο ελεγκτές που αποτελούν το συμπλήρωμα του καθήκοντος για το πρόβλημα αμοιβαίου αποκλεισμού στο σχήμα 2.1. Καταστάσεις που ανήκουν σε κάποιο cyset περικλείονται από ένα γκρίζο κυκλικό δίσκο. Ο πρώτος ελεγκτής περιγράφει το συμπλήρωμα της ιδιότητας ασφάλειας για τον αμοιβαίο αποκλεισμό, δηλ. ότι μόνο μια διεργασία έχει πρόσβαση στον κοινόχρηστο πόρο κάθε χρονική στιγμή. Ο δεύτερος ελεγκτής περιγράφει το συμπλήρωμα της ακόλουθης συνθήκης «ζωτικότητας»: «Αν μια διεργασία χρειάζεται τον πόρο, τελικά θα αποκτήσει προοβαση σ'αυτόν». Συνεπώς η M_{TC_2} αποδέχεται κάθε συμπεριφορά κατά την οποία κάποια διεργασία υποβάλει μια αίτηση που ποτέ δεν ικανοποιείται. Είναι αρκετά εύκολο να δούμε ότι η περιγραφή στο σχήμα 2.1 ικανοποιεί την ιδιότητα ασφάλειας αλλά δεν ικανοποιεί την ιδιότητα ζωτικότητας, ακόμα κι αν προσθέσουμε το cyset $\{$WAIT,USE$\}$ στην περιγραφή κάθε διεργασίας. Θα δείξουμε αργότερα χρησιμοποιώντας ένα παράδειγμα ότι είναι δυνατόν αυτή η συνθήκη ζωτικότητας να ικανοποιηθεί εάν συνδέσουμε κάποια χρονική πληροφορία με τις διεργασίες.

Κεφάλαιο 3

Το εργαλείο COSPAN

Το COSPAN [HK89, JK86, ZH90] είναι ένα εργαλείο λογισμικού για την περιγραφή και επαλήθευση συντονιζόμενων συστημάτων. Το COSPAN μπορεί να χρησιμοποιηθεί για την ανάπτυξη περιγραφών σε υψηλό επίπεδο, για την ανάλυση της λογικής και στοχαστικής συμπεριφοράς ενός συντονιζόμενου συστήματος και προαιρετικά για την παραγωγή πολύ αποδοτικών υλοποιήσεων στη γλώσσα C οι οποίες παράγονται κατευθείαν από την περιγραφή υψηλού επιπέδου. Το εργαλείο χρησιμοποιεί μια τυπική, ιεραρχική, αναλυτική (top/down) διαδικασία ανάπτυξης.

Η λογική ανάλυση στο COSPAN επιτυγχάνεται μέσω συμβολικού ελέγχου του περιγραφόμενου συστήματος απέναντι σε μια συμπεριφορά που καθορίζει ο χρήστης. Το σύστημα ούτε προσομοιώνεται, ούτε εκτελείται, αλλά παράγεται μια μαθηματική απόδειξη (ή διάψευση) της εν λόγω συμπεριφοράς. Τυπικές μέθοδοι αναγωγής χρησιμοποιούνται στους αλγόριθμους ανάλυσης του COSPAN, ώστε να ανταπεξέλθουμε το συνήθως τεράστιο χώρο καταστάσεων που συναντάται σε συντονιζόμενα συστήματα.

3.1 Η γλώσσα περιγραφής του COSPAN

Η γλώσσα περιγραφής του COSPAN παρέχει το απαραίτητο συντακτικό για την περιγραφή των διεργασιών Ε/Α και των ελεγκτών. Πολλές κατασκευές αυτής της γλώσσας είναι παρόμοιες με εκείνες στις γλώσσες προγραμματισμού C και PASCAL. Παρουσιάζουμε τις βασικές κατασκευές της γλώσσας ώστε ο αναγνώστης να μπορεί να καταλάβει τα παραδείγματα περιγραφών που περιέχονται στο κεφάλαιο 6. Ενα τετριμένο παράδειγμα στο σχήμα 3.1 δείχνει πως μοιάζουν τα προγράμματα του COSPAN. Ο πλήρης ορισμός της γλώσσας μπορεί να βρεθεί στο [JK86].

Τα σχόλια περικλείονται ανάμεσα στους χαρακτήρες /* και */ και όπως στη C δεν μπορεί να είναι φωλιασμένα. Μπορούν επίσης να περιέχονται όλες οι εντολές που αναγνωρίζει ο

```
proc   COUNTER                                /* Two state switch */
       selvar      #        :(off, on)
       stvar       $        :(0..1)
       init        0
       trans

       0                                      {off, on}
                   -> 1                       : # = on
                   -> 0                       else;

       1                                      {off, on}
                   -> 0                       : # = off
                   -> 1                       else;

end    /* COUNTER */
```

Σχήμα 3.1: Ενα μικρό S/R πρόγραμμα.

C-preprocessor, όπως οι **#define** και **#include**.

Η βασική οντότητα της γλώσσας περιγραφής είναι η *διεργασία (process)*. Οι διεργασίες αρχίζουν με τη δεσμευμένη λέξη **proc** (ή **monitor** προκειμένου για διεργασία ελεγκτή), και τελειώνουν με τη δεσμευμένη λέξη **end** ακολουθούμενη προαιρετικά από το όνομα της διεργασίας. Ο ορισμός του ελεγκτή στο COSPAN επεκτείνει το μοντέλο που παρουσιάστηκε στην ενότητα 2.2 επιτρέποντας ένα γενικότερο ορισμό της άλγεβρας $\mathcal{L}$. Μια διεργασία ελεγκτής επιτρέπεται να έχει κατηγορήματα πάνω στις τιμές των καταστάσεων των διεργασιών εκτός από των επιλογών τους στις ετικέττες των μεταβάσεων. Ετσι μια διεργασία ελεγκτής έχει τη δυνατότητα να «βλέπει» επίσης τις καταστάσεις των άλλων διεργασιών· αυτό είναι χρήσιμο για να απλοποιούνται οι περιγραφές, αλλιώς η πληροφορία της κατάστασης θα μπορούσε να κωδικοποιηθεί μέσα στην επιλογή της διεργασίας σε εκείνη την κατάσταση. Μπορεί να δειχτεί ότι αυτό το χαρακτηριστικό του COSPAN δεν προσθέτει καθόλου εκφραστική δύναμη στο αρχικό μοντέλο Ε/Α. Οι διεργασίες μπορούν επίσης να είναι φωλιασμένες, παρέχοντας έτσι ένα τρόπο για την κατασκευή δομημένων περιγραφών.

Οι δεσμευμένες λέξεις **selvar** και **stvar** χρησιμοποιούνται για να περιγράψουν το όνομα και το πεδίο ορισμού των μεταβλητών επιλογής και κατάστασης μιας διεργασίας αντίστοιχα. Μια κοινή πρακτική ανάμεσα στους χρήστες του COSPAN είναι να ονομάζεται η μεταβλητή επιλογής # και η μεταβλητή κατάστασης \$. Σε αυτή την περίπτωση $P.\#$ ($P.\$$) συμβολίζει την επιλογή (κατάσταση) της διεργασίας P. Για απλότητα, στην περιγραφή της διεργασίας P χρησιμοποιούμε το σύμβολο # (\$) για να αναφερθούμε στη δικιά της μεταβλητή επιλογής (κατάστασης). Η δεσμευμένη λέξη **import** χρησιμοποιείται για να δηλώσει μια λίστα από εξωτερικές επιλογές (και/ή καταστάσεις προκειμένου για ελεγκτή) που η διεργασία θα χρησιμοποιήσει ως είσοδο. Μια διεργασία πρέπει να χρησιμοποιήσει ένα «όνομα μονοπατιού» (pathname) της μορφής process_name.variable_name για να έχει πρόσβαση στη μεταβλητή μιας άλλης διεργασίας. (Π.χ. αν P_1 και P_2 είναι δύο μη-φωλιασμένες διεργασίες, η P_1 πρέπει να περιέχει μια δήλωση "**import** $P_2.\#$" για να διαβάζει τη μεταβλητή επιλογής της P_2).

Η δεσμευμένη λέξη **init** δηλώνει το αρχικό σύνολο καταστάσεων της αντίστοιχης διεργασίας. Η δομή **trans** είναι η ενότητα με τις μεταβάσεις και αποτελείται από κομμάτια (blocks) που περιγράφουν τις μεταβάσεις από μια κατάσταση στις επόμενες της. (Η παρούσα κατάσταση μπορεί να είναι και επίσης ένα σύνολο καταστάσεων αν όλες οι μεταβάσεις είναι πανομοιότυπες). Το σχήμα ενός κομματιού είναι ως εξής:

$$\begin{array}{rrl} & \text{current state} & \{\text{selection list}\} \\ -> & \text{next state} & : \text{condition} \\ & \ldots & \\ & \ldots & \\ -> & \text{next state} & : \text{condition;} \end{array}$$

Προσέξτε ότι η παρούσα κατάσταση, λίστα επιλογών, επόμενη κατάσταση, και συνθήκη μπορεί να είναι εκφράσεις. Οι εκφράσεις αποτελούνται από μεταβλητές επιλογής και κατάστασης, τις δεσμευμένες λέξεις **true** και **false**, και τους συνηθισμένους αριθμητικούς και λογικούς τελεστές. Το $+$ χρησιμοποιείται για πρόσθεση ακεραίων ή λογική διάζευξη, το $\star$ χρησιμοποιείται για πολλαπλασιασμό ακεραίων ή λογική σύζευξη και το ~ για λογική άρνηση. Οι σχεσιακοί τελεστές είναι οι $<=, <, >=, >, =, \tilde{} =$ για μικρότερο ή ίσο, μικρότερο, μεγαλύτερο ή ίσο, μεγαλύτερο, ίσο και διάφορο αντίστοιχα. Το σύμβολο $->$ συμβολίζει τη μετάβαση καταστάσεως. Επίσης η δεσμευμένη λέξη **else** μπορεί να χρησιμοποιείται για να δείχνει την *άρνηση* του *λογικού ή* των προηγούμενων συνθηκών σε ένα κομμάτι.

Ενα χαρακτηριστικό του COSPAN χρήσιμο στον προχωρημένο προγραμματιστή του S/R είναι η ικανότητα παρεμβολής καθαρού κώδικα `C` μέσα στον ορισμό μιας διεργασίας μετά τη δομή μετάβασης. Αυτό το προαιρετικό κομμάτι `C` κώδικα αρχίζει με τη δεσμευμένη λέξη **code** ακολουθούμενη από προαιρετικές δηλώσεις κώδικα. Ο χαρακτήρας $ χρησιμοποιείται ως διαχωριστικό που υποδηλώνει την αρχή και το τέλος των εντολών σε `C`. Τα κομμάτια κώδικα (**code** blocks) θα συζητηθούν εκτεταμένα στην ενότητα 5.2.

Μπορούμε να προσθέσουμε συνθήκες αποδοχής σε μια διεργασία χρησιμοποιώντας τις δηλώσεις "**cyset** *state list*" και/ή "**recur** *edge list*", όπου *state list* είναι της μορφής $state_1, \ldots, state_n$, και *edge list* είναι της μορφής $state_1 -> state_2, \ldots, state_{n-1} -> state_n$.

Τέλος ένας παραμετρικός τύπος διεργασίας μπορεί να οριστεί με μια δήλωση της μορφής:

proctype type_name(*formal parameters list*)
...
...
end

Μια παραμετρική διεργασία μπορεί να οριστεί ως εξής:

proc proc_name : type_name(*actual parameters list*)

και ένας πίνακας παραμετρικών διεργασιών ως εξής:

proc proc_name[index_range] : type_name(*actual parameters list*)

3.2 Δομή του COSPAN

Το COSPAN στην πραγματικότητα αποτελείται από ένα αρχείο με εντολές Bourne shell που ονομάζεται `cospan`, ένα μεταγλωτιστή S/R, αρχεία δηλώσεων (header files) και μια βιβλιοθήκη από ρουτίνες σε γλώσσα μηχανής. Οι χρήστες καλούν το πρόγραμμα `cospan` δίνοντας διάφορες εκλογές (options) και ένα όνομα αρχείου σαν τελευταίο όρισμα. Κάθε εργασία του COSPAN μπορεί να περιέχει μέχρι τρια το πολύ ξεχωριστά βήματα. Αυτά είναι:

1. Επεξεργασία των οδηγιών του `cpp` και μετάφραση μιας S/R περιγραφής που περιέχεται σε ένα αρχείο με όνομα π.χ. foo**.sr** το οποίο δίνεται σαν το τελευταίο όρισμα, για να δημιουργηθεί ένα αρχείο με `C` κώδικα που ονομάζεται foo**.c**. Το `cospan` καλεί το `cpp` και τον S/R μεταφραστή `sr` (ή `sr_D` αν δώθηκε και η εκλογή −v).

2. Μετάφραση του `C`-κώδικα που περιέχεται στο αρχείο foo**.c** που δημιουργήθηκε στο προηγούμενο βήμα και συνένωση (linking) με τη βιβλιοθήκη από ρουτίνες σε γλώσσα μηχανής, για να δημιουργηθεί ένα εκτελέσιμο αρχείο που ονομάζεται foo**.an**. Σε αυτή την περίπτωση το `cospan` καλεί ένα μεταφραστή της `C` (το `CC` αν δεν έχει καθοριστεί άλλη εκλογή) ο οποίος χρησιμοποιεί τα αρχεία δηλώσεων `crank.h` ή `crunch.h` και την βιβλιοθήκη `libsr.a`.

3. Ανάλυση της περιγραφής με εκτέλεση του αρχείου foo**.an** που δημιουργήθηκε στο βήμα 2. Αν κατά την εκτέλεση βρεθεί μια παραβίαση του καθορισμένου καθήκοντος, ένα αποτύπωμα (track) της υπεύθυνης συμπεριφοράς του συστήματος καταχωρείται στο αρχείο foo**.T**

Ανάλογα με το αν το επίθεμα του αρχείου που έχει ως όρισμα είναι **.sr**, **.c** ή **.an**, το `cospan` αρχίζει την ακολουθία εκτέλεσης από το βήμα 1,2 ή 3 αντίστοιχα.

Το πιο σημαντικό βήμα είναι προφανώς το βήμα 3. Η κυρίως δραστηριότητα του τμήματος του COSPAN που κάνει την ανάλυση (το οποίο περιέχεται στη `libsr.a`) είναι ο έλεγχος αν η διεργασία γινόμενο P, που ορίζεται στην S/R περιγραφή και της οποίας η επεξεργασία γίνεται στο βήμα 1, είναι κενή. Για την βελτιστοποίηση της απόδοσης αυτό γίνεται «καθ'οδόν»(on-the-fly), δηλαδή η διεργασία γινόμενο κατασκευάζεται από τις συνιστώσες διεργασίες ενώ ερευνάται ο χώρος των ολικών καταστάσεων. Υπάρχουν πολλά πραγματικά παραδείγματα από πρωτόκολλα επικοινωνίας και ψηφιακά κυκλώματα τα οποία έχουν αναλυθεί χρησιμοποιώντας την παραπάνω μέθοδο, βλέπε [Kur90]. Ο χρόνος που ξοδεύει το λογισμικό του COSPAN για να κάνει την ανάλυση μπορεί να εξαρτάται από πολλούς παράγοντες της περιγραφής όπως τον αριθμό των διαφορετικών συνιστωσών, τον αριθμό των συνόλων επαναλαμβανόμενων καταστάσεων, τη μορφή του γράφου (αριθμός ακμών ανά κόμβο). Τυπικά, μπορεί να αναλύσει περίπου ένα εκατομμύριο καταστάσεις την ώρα σε ένα σταθμό εργασίας της σειράς SUN 3.

Οπως θα εξηγήσουμε αργότερα με λεπτομέρειες στην ενότητα 5.2, όλα τα βήματα του COSPAN έχουν τροποποιηθεί για να εισαχθούν οι χρονικοί περιορισμοί και να υλοποιηθεί το RTCOSPAN. Το `cospan` έχει μεταβληθεί, νέα αρχεία σε γλώσσα `sr` έχουν δημιουργηθεί, το αρχείο δηλώσεων `crank.h` έχει αλλάξει και επίσης χρησιμοποιείται μια νέα βιβλιοθήκη με όνομα `libRT.a`. Παρόλα αυτά το τμήμα του COSPAN που κάνει την ανάλυση χρησιμοποιείται όπως είναι.

Κεφάλαιο 4

Προσθήκη ιδιοτήτων πραγματικού χρόνου

Μέχρι τώρα έχουμε συζητήσει ένα μαθηματικό μοντέλο και ένα εργαλείο λογισμικού κατάλληλο για επαλήθευση ανεξάρτητη από την ταχύτητα συστημάτων που αποτελούνται από συντονιζόμενες συνιστώσες. Τώρα επαυξάνουμε το μοντέλο Ε/Α ώστε να συμπεριλάβει περιορισμούς πραγματικού χρόνου, και περιγράφουμε την έννοια των *μετρητών* (timers) οι οποίοι είναι ιδεατές συνιστώσες που παρακολουθούν τους δυνατούς χρόνους κατά τους οποίους μπορούν να συμβούν γεγονότα. [1] Με αυτό τον τρόπο είναι δυνατόν να εξαιρεθούν ακολουθίες υπολογισμών που παραβιάζουν τις χρονικές υποθέσεις. Ετσι ένα σύστημα που θα παραβίαζε κάποιες προδιαγραφές σε ένα μοντέλο καθαρά ανεξάρτητο από την ταχύτητα, μπορεί να τις ικανοποιήσει κάτω από συγκεκριμένες χρονικές υποθέσεις.

4.1 Περιγραφή της προσέγγισης

Η προσέγγιση που χρησιμοποιήθηκε για να ενσωματωθεί χρονική πληροφορία μέσα στο μοντέλο Ε/Α είναι η μετάφραση των χρονικών περιορισμών σε ένα σύνολο από διεργασίες ελεγκτές που αποδέχονται μόνο αλυσίδες οι οποίες είναι συνεπείς με τους καθορισμένους χρονικούς περιορισμούς. Με αυτό τον τρόπο η επαλήθευση της ορθής λειτουργίας μιας περιγραφής συστήματος ανάγεται στο να ελεγχθεί εαν η γλώσσα που αποδέχεται αυτό το καινούργιο μεγαλύτερο σύνολο από $\mathcal{L}$-διεργασίες είναι κενή. Αυτό είναι ένα καθήκον που μπορεί να επιτευχθεί χρησιμοποιώντας τις συνηθισμένες ευκολίες που παρέχει το COSPAN.

[1] Δεν θα 'πρεπε να δημιουργείται σύγχιση με τους μετρητές που χρησιμοποιούνται σε πρωτόκολλα επικοινωνιών ή ψηφιακά κυκλώματα. Οποτε η διάκριση δεν είναι καθαρή από τα συμφραζόμενα, θα χρησιμοποιούμε τον όρο *λογικός μετρητής* (logical timer).

Τα στοιχεία του νέου αυτού συνόλου από $\mathcal{L}$-διεργασίες που περιέχουν χρονική πληροφορία θα ονομάζονται *διεργασίες με χρόνο* (timed processes). Μια περιγραφή διεργασίας με χρόνο στο μοντέλο μας αποτελείται από δύο μέρη: Το *χρονικά-ανεξάρτητο* μέρος και το *χρονικά-εξαρτημένο* μέρος. Το χρονικά-ανεξάρτητο μέρος (που παρουσιάστηκε στο κεφάλαιο 2 και την ενότητα 3.1) καθορίζει ένα υπερσύνολο των δυνατών αλυσίδων που μπορούν να εμφανιστούν στο πραγματικό σύστημα, το οποίο περιέχει κάποιες αλυσίδες που δεν είναι συνεπείς με τους χρονικούς περιορισμούς. Για παράδειγμα, ας εξετάσουμε την περίπτωση ενός πρωτόκολλου επικοινωνίας με επιλεκτική επανάληψη (selective-repeat). Στην περιγραφή του πρωτοκόλλου χωρίς χρόνο οι μετρητές που σηματοδοτούν τα γεγονότα που σημαίνουν «πέρας χρόνου» (timeout) μπορούν να εκπνεύσουν σε οποιονδήποτε χρόνο αφού έχουν τεθεί. Στο πραγματικό σύστημα πολλές από αυτές τις επικαλύψεις δεν είναι δυνατές εξαιτίας των ορίων στις καθυστερήσεις των μηνυμάτων και των τιμών όπου τίθενται οι μετρητές.

Το χρονικά-εξαρτημένο μέρος στην πραγματικότητα απαρτίζεται από ένα σύνολο περιορισμών πραγματικού χρόνου. Αυτοί οι περιορισμοί είναι της μορφής «Αν το $c(i)$ ικανοποιεί την ιδιότητα A και το $c(j)$ ικανοποιεί την ιδιότητα B, με $i < j$, τότε $l \leq t(j) - t(i) \leq u$», όπου $c(i)$ και $c(j)$ ανήκουν στην αλυσίδα της χρονικά-ανεξάρτητης διεργασίας, l και u είναι ακέραιες σταθερές και $t(i)$, $t(j)$ είναι οι χρόνοι κατά τους οποίους οι καταστάσεις $c(i)$ και $c(j)$ εμφανίζονται στο ίχνος της διεργασίας. Ενα πλεονέκτημα αυτής της μορφής χρονικών περιορισμών είναι ότι τα γεγονότα στο σύστημα και οι χρονικές καθυστερήσεις μπορούν να συνδεθούν με πολύ ευέλικτους τρόπους. Για παράδειγμα είναι προφανώς εφικτό να κατασκευαστεί ένα μοντέλο για την περίπτωση «Το γεγονός β συμβαίνει όχι περισσότερο από δύο δευτερόλεπτα μετά το γεγονός α» που αναφέρθηκε στο κεφάλαιο 1. Για παράδειγμα ας θεωρήσουμε ότι κατασκευάζουμε ένα μοντέλο ενός σηματοδότη για τον οποίο ξέρουμε ότι η διάρκεια του πράσινου σήματος είναι μεταξύ 2 και 3 δευτερολέπτων. Τότε αν το A οριστεί ως «στη $c(i)$ το φως γίνεται πράσινο» και το B οριστεί ως «Η $c(j)$ είναι η πρώτη κατάσταση της αλυσίδας μετά την $c(i)$ στην οποία το φως είναι κόκκινο», παίρνουμε ένα χρονικό περιορισμό της παραπάνω μορφής με $l = 2, u = 3$.

Ενας *λογικός μετρητής* T χρησιμοποιείται για να κωδικοποιήσει κάθε χρονικό περιορισμό. Ατυπα κάθε μετρητής T έχει μια *συνθήκη θέσης* (set condition) η οποία είναι η ιδιότητα A, μια *συνθήκη εκπνοής* (expire condition) η οποία είναι η ιδιότητα B, μια *συνθήκη ακύρωσης* (cancel condition) η οποία θα εξηγηθεί παρακάτω, και μια *περιγραφή διαστήματος χρόνου* (time interval specification) η οποία (στο παραπάνω παράδειγμα) αντιστοιχεί στο διάστημα $[2, 3]$. Ενα τέτοιο διάστημα είναι πανω στους πραγματικούς αριθμούς και έχει άκρα ακεραίους. Η ερμηνεία είναι ότι όταν η A είναι αληθής ο T τίθεται, και όταν η B γίνεται αληθής εκπνέει. Η ακριβής τιμή του χρόνου που έχει περάσει ανάμεσα στα γεγονότα θέσης και εκπνοής δεν καθορίζεται ακριβώς, αλλά ανήκει στο διάστημα $[2, 3]$. Τα γεγονότα μετρητών (ονομαστικά τα γεγονότα θέσης, εκπνοής και ακύρωσης) πάρέχουν τη διασύνδεση ανάμεσα στο χρονικά-εξαρτημένο και το χρονικά-ανεξάρτητο τμήμα της περιγραφής ενός συστήματος. Μια τυπική περιγραφή της διασύνδεσης θα παρουσιαστεί στην ενότητα 4.3.

Το χρονικά-εξαρτημένο τμήμα της περιγραφής μεταφράζεται αυτόματα μέσω της επέκτασης μας στο αρχικό λογισμικό του COSPAN σε μια διεργασία ελεγκτή M_T χρησιμοποιώντας την προσέγγιση στο [Dil89] (η οποία παρουσιάζεται περιληπτικά στην ενότητα 4.4). Αυτός ο ελεγκτής δεν αποδέχεται τις αλυσίδες του χρονικά-ανεξάρτητου μέρους της περιγραφής που παραβιάζουν τους περιορισμούς πραγματικού χρόνου, και επιβάλει περιορισμούς διάταξης στα γεγονότα μετρητών. Προφανώς οι περιορισμοί διάταξης είναι επακόλουθο των περιορισμών πραγματικού χρόνου. Χρησιμοποιώντας τα αποτελέσματα από το [Dil89] είναι δυνατόν να αποδειχτεί ότι το σύνολο όλων των αλυσίδων που είναι συνεπείς με το χρόνο είναι η τομή των χρονικά-ανεξάρτητων με τις χρονικά-εξαρτημένες διεργασίες, με άλλα λόγια, οι αλυσίδες της $P \otimes M_T$. Μια διεργασία με χρόνο δεν έχει συμπεριφορά με χρόνο την οποία να αποδέχεται (ιστορία σε πραγματικό χρόνο) ανν δεν υπάρχει αλυσίδα συνεπής με το χρόνο στην $P \otimes M_T$, με άλλα λόγια ανν η γλώσσα που αποδέχεται η διεργασία $P \otimes M_T$ είναι κενή.

Ο ελεγκτής λειτουργεί συμπεραίνοντας πληροφορία χρόνου για την ιστορία των γεγονότων μετρητών στις αλυσίδες. Αποθηκεύει την πληροφορία χρόνου στις καταστάσεις του και την χρησιμοποιεί για να αποφασίσει ποιά γεγονότα χρόνου επιτρέπεται να πραγματοποιηθούν στη συνέχεια. Αν μια μετάβαση που συνδέεται με ένα συγκεκριμένο σύνολο γεγονότων χρόνου επιτρέπεται, ο ελεγκτής ενημερώνει την κατάσταση του ώστε να αντανακλά τις αλλαγές της κατάστασης των μετρητών. Αν τα γεγονότα δεν επιτρέπεται να πραγματοποιηθούν, ο ελεγκτής μεταβαίνει και παραμένει σε μια κατάσταση «λάθους» στην οποία η αλυσίδα δεν είναι αποδεκτή (ένα cyset).

Για παράδειγμα, στην περίπτωση του σηματοδότη που αναφέρθηκε προηγουμένως, όσο τα στιγμιότυπα του συστήματος ικανοποιούν την ιδιότητα ότι το φανάρι είναι πράσινο, ο χρόνος δεν μπορεί να προχωρήσει παραπάνω από τρία δευτερόλεπτα. Αυτό εξαιρεί την πραγματοποίηση κάποιων άλλων γεγονότων πριν το φανάρι γίνει κόκκινο, αν η πληροφορία χρόνου που εξάγεται ως πόρισμα της πραγματοποίησης αυτών των γεγονότων έρχεται σε αντίφαση με την παραπάνω πληροφορία. Οι λεπτομέρειες της κατασκευής τέτοιων ελεγκτών μπορούν να βρεθούν στην ενότητα 4.4..

Διεργασίες με χρόνο μπορούν να συνθεθούν εκτελώντας την συνηθισμένη πράξη $\otimes$ στα χρονικά-ανεξάρτητα τμήματα (ώστε να παραχθεί ένα νέο χρονικά-ανεξάρτητο τμήμα) και παίρνοντας την *ένωση* των μετρητών, ώστε να κατασκευαστεί το σύνολο μετρητών του γινομένου.

Οταν η M_T έχει κατασκευαστεί και προστεθεί στην περιγραφή του συστήματος, χρονικά-ανεξάρτητες ιδιότητες, οι οποίες καθορίζουν μόνο διατάξεις των γεγονότων του συστήματος αλλά όχι και τον χρόνο πραγματοποίησης, μπορούν να επαληθευθούν αμέσως χρησιμοποιώντας το COSPAN με τον τρόπο που περιγράψαμε στην προηγούμενη ενότητα.

Πιο ενδιαφέρουσα περίπτωση αποτελεί η επαλήθευση χρονικών ιδιοτήτων, για παράδειγμα, «ο χρόνος μεταξύ οποιονδήποτε δύο συνεχόμενων λήψεων ενός μηνύματος από τον δέκτη είναι πάντα μικρότερος από δύο δευτερόλεπτα.», ή «δεν στέλνονται δύο διακοπές μέσα σε 1msec».

Στη γενική περίπτωση η επαλήθευση τέτοιων χρονικών ιδιοτήτων είναι μη αποφασίσιμη (undecidable) — υπάρχουν καποια καθήκοντα με χρόνο των οποίων δεν μπορεί να βρεθεί η συμπληρωματική συμπεριφορά [AD90], δηλαδή το συμπλήρωμα τους δεν μπορεί να εκφραστεί σαν διεργασία με χρόνο. Ως τέχνασμα σε αυτό το ζήτημα παρέχουμε το συμπλήρωμα του καθήκοντος *κατευθείαν* περιγράφοντας ανεπιθυμήτη συμπεριφορά με χρόνο μέσω μιας διεργασίας με χρόνο, και μετά ελέγχοντας αν η σύνθεση της διεργασίας με χρόνο που αντιστοιχεί στην περιγραφή του συστήματος και του συμπληρώματος του καθήκοντος είναι κενή. Αυτή η προσέγγιση είναι έγκυρη μια και αν υπάρχει κάποια «κακή» συμπεριφορά με χρόνο, πρέπει να ικανοποιεί και την περιγραφή με χρόνο του συστήματος και την περιγραφή με χρόνο του συμπληρώματος του καθήκοντος. Φυσικά η προσέγγιση μας είναι εφαρμόσιμη μόνο όταν το συμπλήρωμα του καθήκοντος μπορεί να εκφραστεί σα διεργασία με χρόνο.

Στις περισσότερες περιπτώσεις η παραγωγή ενός συμπληρώματος καθήκοντος με χρόνο είναι απλή. Στο προηγούμενο παράδειγμα, οι ανεπιθύμητες αλυσίδες με χρόνο είναι εκείνες στις οποίες ο δέκτης λαμβάνει δύο συνεχόμενα μηνύματα με χρονική απόσταση μεγαλύτερη από δύο δευτερόλεπτα.

4.2 Το μοντέλο διεργασιών με χρόνο

Σε αυτή την ενότητα θα ορίσουμε το μοντέλο της διεργασίας με χρόνο. Για το σκοπό αυτό χρειαζόμαστε πρώτα να ορίσουμε τυπικά την έννοια των λογικών μετρητών που εισαγάγαμε άτυπα στην προηγούμενη ενότητα.

4.2.1 Λογικοί Μετρητές

Ορίζουμε ένα λογικό μετρητή T ως μια τετράδα $(set, expire, cancel, I)$ όπου set, $expire$, $cancel \in \mathcal{L}$ και I είναι ένα διάστημα με άκρα από το σύνολο $\{0, 1, \ldots, \infty\}$. Ενας μετρητής που έχει τεθεί και δεν έχει εκπνεύσει ούτε έχει ακυρωθεί ονομάζεται *ενεργός* (active) μετρητής. Οταν και το set και το $expire$ είναι αληθή την ίδια χρονική στιγμή η ερμηνεία που επιλέγουμε να δώσουμε είναι ότι ο μετρητής πρώτα εκπνέει και κατόπιν τίθεται στιγμιαία.

Σε αυτή την περίπτωση ο μετρητής θεωρείται συνέχεια ενεργός. Οπως θα δούμε είναι κατάλληλο για τους σκοπούς μας να ορίσουμε το I χρησιμοποιώντας τα άνω και κάτω *άκρα* (bounds) του. Για να χειριζόμαστε ομοιόμορφα άπειρα και πεπερασμένα άκρα, καθώς και αυστηρές και μη αυστηρές ανισότητες, ορίζουμε *ζεύγη άκρων* (bound pairs), τα οποία είναι διατεταγμένα ζεύγη από $\mathcal{Z} \times \{<, \leq\} \cup \{(\infty, <), (-\infty, <)\}$ ($\mathcal{Z}$ είναι το σύνολο των ακεραίων). Τα σύμβολα $<$ και $\leq$ είναι ολικά διατεταγμένα: Το $<$ θεωρείται αυστηρά μικρότερο του $\leq$. Μια μερική διάταξη ορίζεται ως $(x, r) \leq (x', r')$ αν $x < x'$ ή $x = x'$ και $r \leq r'$ (λεξικογραφική διάταξη).

Για παράδειγμα αν προσθέσουμε στο παράδειγμα του σχήματος 2.2 ένα λογικό μετρητή με περιγραφή $set = (TR\# = msg!) \cdot (CH\# = msg?)$, $expire = (CH\# = deliver)$, $cancel = false$, $I = (6, 11]$, κωδικοποιούμε τον περιορισμό ότι ένα μήνυμα απαιτεί $6 < t \leq 11$ μονάδες χρόνου από τη στιγμή που παραλαμβάνεται από το κανάλι μέχρι να είναι έτοιμο να γίνει διαθέσιμο στον δέκτη. Εδώ το κάτω άκρο l του I είναι $(6, <)$ και το άνω άκρο u είναι $(11, \leq)$.

Η πρώτη συνιστώσα ενός ζεύγους άκρου στο σύστημα μας πρέπει να είναι ένα ακέραιο πολλαπλάσιο κάποιας μονάδας χρόνου. Αυτό δεν αλλάζει την συνεχή χρονική υφή του μοντέλου μια και οι μεταβάσεις στο σύστημα μπορούν να συμβούν σε οποιαδήποτε σημεία στο συνεχή χρόνο. Η επιλογή της κατάλληλης μονάδας χρόνου είναι κομμάτι της σωστής περιγραφής του συστήματος. Σημειώστε ότι θα μπορούσαμε να χρησιμοποιήσουμε ρητούς αριθμούς αντί για ακεραίους. Μια και υπάρχουν πεπερασμένοι τον αριθμό περιορισμοί, θα μπορούσαμε να διαλέξουμε τον μέγιστο κοινό διαιρέτη των παραπάνω σταθερών σαν μονάδα χρόνου. Αυτή είναι μια σημαντική υπόθεση για την αναγωγή του προβλήματος της επαλήθευσης σε πεπερασμένο πρόβλημα.

Τώρα ορίζουμε ένα *σύστημα μετρητών* (timer system) να είναι μια τετράδα (T, l, u, A_0), όπου T είναι ένα πεπερασμένο σύνολο από μετρητές, $l : T \to B$ και $u : T \to B$ είναι απεικονίσεις που αντιπροσωπεύουν σταθερά κάτω και άνω άκρα αντίστοιχα στις τιμές των μετρητών. Οταν $u_i = (\infty, <)$ δεν υπάρχει πεπερασμένο άνω άκρο για το διάστημα του μετρητή i. Το A_0 είναι ένα σύνολο μετρητών οι οποίοι έχουν τεθεί αρχικά. Μπορεί να δειχτεί ότι το A_0 δεν προσθέτει εκφραστική δύναμη στο μοντέλο μας, οπότε συνήθως θα θεωρούμε το A_0 να είναι το κενό σύνολο ($A_0 = \emptyset$). Ισχύει πάντοτε ότι $(0, <) \leq l_i \leq u_i \leq (\infty, <), \forall i \in T$.

Οι παρακάτω τρείς συνθήκες καθορίζουν ποτε μια ακολουθία γεγονότων σε ένα σύστημα μετρητών είναι «*καλά ορισμένη*» (well formed).

1. Οι ενεργοί μετρητές τίθενται *μόνο εαν* εκπνέουν στο ίδιο σύνολο γεγονότων.

2. Οι μετρητές εκπνέουν μόνο όταν είναι ενεργοί.

3. Κάθε ενεργός μετρητής κάποτε εκπνέει.

Σε μερικές περιπτώσεις θεωρούμε χρήσιμο να χαλαρώσουμε τον περιορισμό 1, ορίζοντας έτσι ένα νέο τύπο μετρητή που ονομάζεται «*επανατιθέμενος μετρητής*» (resetable timer). Ενας τέτοιος μετρητής μοιάζει περισσότερο με ξυπνητήρι που μπορεί να το ξαναθέσουμε πριν χτυπήσει.

4.2.2 Διεργασίες με χρόνο

Είναι πλέον καιρός να ορίσουμε τυπικά την έννοια της διεργασίας με χρόνο. Μια $\mathcal{L}$-διεργασία *με χρόνο* είναι μια ν-άδα (P, T), όπου

(1) P είναι μια $\mathcal{L}$-διεργασία,

(2) $T = \{T_1, \ldots, T_l\}$ είναι ένα σύνολο από λογικούς μετρητές.

Η σημασιολογία μιας $\mathcal{L}$-διεργασίας με χρόνο (P, T) δίνεται μέσω ενός συνόλου από *αλυσίδες με χρόνο*. Μια αλυσίδα με χρόνο είναι μια ακολουθία από διατεταγμένα ζεύγη $(c(i), t(i))$ σε διακριτό χρόνο

$$c \;=\; (c(0), t(0))\ (c(1), t(1))\ \ldots,$$

όπου $c(i)$ είναι ένα ζεύγος ολικής κατάστασης–επιλογής του συστήματος και $t(i)$ είναι ένας μη αρνητικός πραγματικός αριθμός. Η ακολουθία διακριτού χρόνου $1, 2, \ldots$ ορίζεται ως ο *λογικός χρόνος* της αλυσίδας, ενώ $t(i)$ είναι η ακολουθία των τιμών του πραγματικού χρόνου (συνεχές πεδίο) που αντιστοιχούν στην ακολουθία των λογικών χρόνων. Διαισθητικά, ο λογικός χρόνος συνδέεται με την ακολουθία των διακριτών γεγονότων που αντιστοιχούν σε μεταβάσεις του συστήματος, και η ακολουθία πραγματικού χρόνου συνδέει ένα αποτύπωμα (timestamp) πραγματικού χρόνου σε καθένα από αυτά τα γεγονότα.

Για παράδειγμα, η πληροφορία μιας αλυσίδας με χρόνο συνεπάγεται ότι για κάθε λογικό χρόνο μέσα στο διάστημα πραγματικού χρόνου $[t(i-1), t(i))$ η τιμή της κοινής μνήμης (ολική κατάσταση–επιλογή) είναι $c(i-1)$, και ότι η μετάβαση $c(i-1) \to c(i)$ συμβαίνει σε χρόνο $t(i)$. Σημειώστε ότι αυτός ο ορισμός απαιτεί να ισχύει $t(i) > t(i-1)$. Οπως θα αναφέρουμε αργότερα, μπορούμε να χαλαρώσουμε αυτή την υπόθεση και να απαιτήσουμε να ισχύει $t(i) \geq t(i-1)$· αυτή η σημασιολογία συνεπάγεται ότι μπορούμε να έχουμε ένα αριθμό από μεταβάσεις οι οποίες συμβαίνουν σε μηδέν χρόνο. Μια αλυσίδα με χρόνο της P πρέπει να ικανοποιεί τις παρακάτω συνθήκες συνέπειας.

(1) $c(0)\ c(1)\ \ \ldots$ είναι μια αλυσίδα στην $\mathcal{L}_P$,

(2) $t(i-1) < t(i), i = 0, 1, \ldots$ (ο χρόνος αυξάνεται αυστηρά), [2]

[2] Θα μπορούσαμε επίσης να χρησιμοποιήσουμε τη σημασιολογία ότι $t(i-1) \leq t(i)$. Σε αυτή την περίπτωση το διάστημα του μετρητή «χρονικής προόδου» στην ενότητα 4.3 θα είναι το $[0, \infty)$.

Ορίζουμε ως $\mathcal{L}^t_{(P,T)}$ το σύνολο των αλυσίδων με χρόνο της (P,T). Εστω $\tau^s_j(i)$ και $\tau^e_j(i)$, $i = 1, 2, \ldots$, οι ακολουθίες των λογικών χρόνων κατά τους οποίους οι συνθήκες set_j και $expire_j$ του μετρητή T_j ικανοποιούνται στη $c(i)$. Αν $\tau^s_j(i)$ $(\tau^e_j(i))$ δεν ορίζεται για $i > k$, θέτουμε $\tau^s_j(i) = \infty$ $(\tau^e_j(i) = \infty)$ για $i > k$. Τότε για κάθε μετρητή T_j

(3) $\tau^s_j(i) < \infty$ συνεπάγεται $\tau^e_j(i) < \infty$,

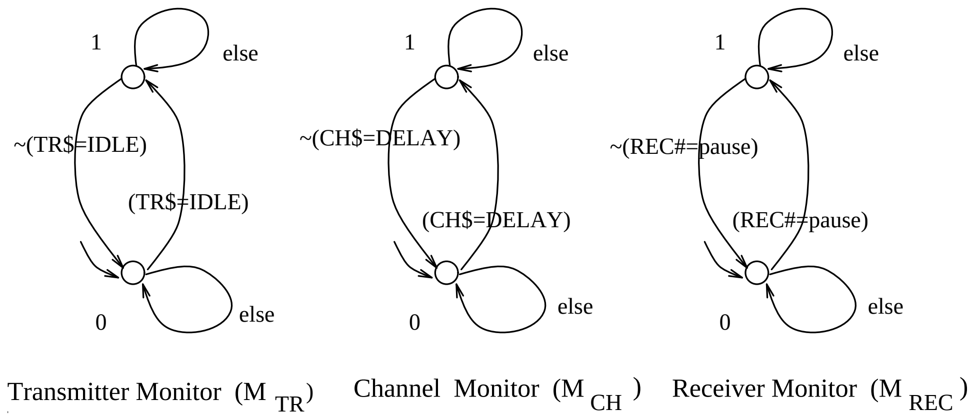
(4) $\tau^s_j(i) < \tau^e_j(i) \leq \tau^s_j(i+1), i = 1, 2, \ldots,$ όταν $\tau^s_j(i), \tau^e_j(i) < \infty$,

(5) $t(\tau^e_j(i)) - t(\tau^s_j(i)) \in I_j$.

Η συνθήκη (3) είναι μια συνθήκη ζωτικότητας: Ενας μετρητής που τίθεται πρέπει κάποτε να εκπνεύσει. Η συνθήκη (4) απαγορεύει στο μετρητή να τεθεί και να εκπνεύσει στην ίδια κατάσταση της αλυσίδας. Οταν κάτι τέτοιο συμβαίνει, σημαίνει ότι η εκπνοή αντιστοιχεί σε μια προηγούμενη θέση του μετρητή και ο μετρητής τίθεται ξανά αμέσως. Αυτή η συνθήκη δεν μπορεί να εκπληρωθεί αν $\tau^s_j(1) = \tau^e_j(1)$ κάτι που απαγορεύεται από την (4). Η συνθήκη (5) είναι ο χρονικός περιορισμός που επιβάλεται από τον μετρητή. Το σύνολο $\mathcal{L}_{(P,T)}$ των αλυσίδων με χρόνο μιας διεργασίας με χρόνο (P,T) είναι το σύνολο των ιστοριών της κοινής μνήμης οι οποίες είναι συνεπείς με την πληροφορία του χρόνου. Η σύνθεση των διεργασιών με χρόνο είναι τώρα φανερή. Οι αλυσίδες της σύνθεσης της (P_1, T_1) με την (P_2, T_2) είναι οι αλυσίδες της διεργασίας με χρόνο $(P_1 \otimes P_2, T_1 \cup T_2)$. Σημειώστε ότι στο μοντέλο μας υποθέτουμε ότι τα ονόματα των μετρητών είναι μοναδικά, και ότι ο ίδιος μετρητής δεν μπορεί να συνδεθεί με παραπάνω από μια διεργασία.

Χρονικοί περιορισμοί μπορούν να προστεθούν σε ένα σύστημα συνάπτωντας ένα ελεγκτή με χρόνο, δηλαδή, ένα ελεγκτή με τους δικούς του ιδιωτικούς χρονικούς περιορισμούς. Θεωρήστε το παράδειγμα στο σχήμα 2.2 με τους ακόλουθους χρονικούς περιορισμούς: (α) Η διεργασία εκπομπός δεν επιλέγει την τιμή *pause* για παραπάνω από d_1 μονάδες χρόνου, (β) όταν η διεργασία κανάλι λαμβάνει ένα μήνυμα το διαθέτει στο λήπτη σε χρόνο t τέτοιο ώστε $d_2 \leq t \leq d_3$, (γ) η διεργασία λήπτης πρέπει να ελέγχει για εισερχόμενα μηνύματα τουλάχιστον κάθε d_4 μονάδες χρόνου.

Για την κωδικοποίηση του πρώτου περιορισμού πρέπει να προσθέσουμε ένα μετρητή T_{TR} ο οποίος τίθεται την πρώτη φορά που η διεργασία εκπομπός TR είναι στην κατάσταση IDLE, εκπνέει όταν η διεργασία αλλάζει κατάσταση, και έχει μια περιγραφή διαστήματος της μορφής $(0, d_1]$. Παρόμοιοι μετρητές T_{CH} και T_{TR} πρέπει να προστεθούν για να κωδικοποιηθούν οι περιορισμοί στην συμπεριφορά των διεργασιών του καναλιού και του δέκτη. Για να εκφράσουμε τους παραπάνω περιορισμούς θέσης κατασκευάζουμε τις διεργασίες ελεγκτές M_{TR}, M_{CH}, M_{REC} οι οποίες περιγράφονται στο σχήμα 4.1. Πάρτε για παράδειγμα την M_{TR}. Κάθε φορά που αυτός ο ελεγκτής είναι στην κατάσταση 0,



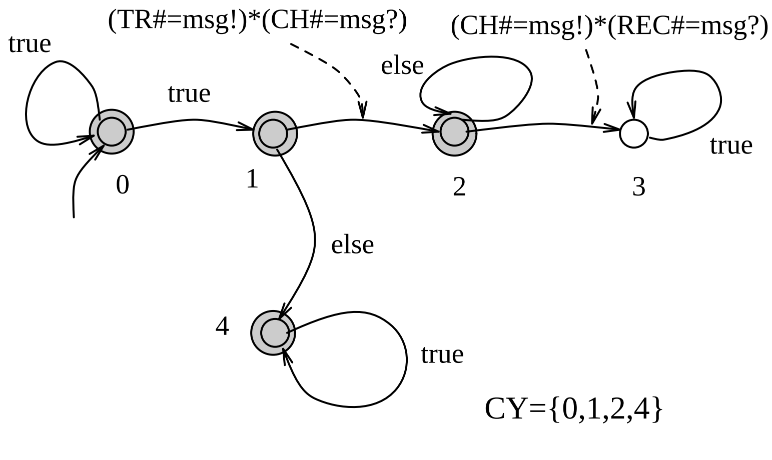
Σχήμα 4.1: Περιγραφη με χρόνο του καναλιού με καθυστέρηση. Οι καταστάσεις του M_{TR} παρέχουν την πληροφορία για την πρώτη φορά που η TR βρίσκεται στις καταστάσεις DELAY και DELIVER. Παρόμοιες ιδιότητες ισχύουν για τις M_{TC} και M_{REC}.

η κατάσταση της αντίστοιχης ελεγχόμενης διεργασίας είναι διαφορετική από την IDLE, ή αυτό αντιστοιχεί στην πρώτη φορά που η διεργασία είναι στην κατάσταση IDLE. Παρόμοιες παρατηρήσεις ισχύουν για τους άλλους δύο ελεγκτές. Μια και η συνθήκη θέσης του T_{TR} είναι $set = (M_{TR}\$ = 0) \cdot (TR\$ = IDLE)$, και η συνθήκη εκπνοής είναι $expire = (TR\$ = TRANSMIT) * (M_{TR}\$ = 1)$.[3] Σημειώστε ότι αυτός ο μετρητής πρέπει να εκπνεύσει μόνο την πρώτη φορά που η διεργασία TR είναι στην νέα κατάσταση·διαφορετικά μπορεί να εκπνέει σε διαδοχικά βήματα χωρίς να έχει τεθεί, που σίγουρα δεν είναι το είδος της συμπεριφοράς που θέλουμε να μοντελοποιήσουμε. Αυτό συμβαίνει όταν $M_{TR}\$ = 1$. Οι ορισμοί των T_{CH}και T_{TR} είναι παρόμοιοι με του T_{TR}. Η διεργασία με χρόνο για το ολοκληρωμένο σύστημα είναι $(TR \otimes CH \otimes REC \otimes M_{TR} \otimes M_{REC}, \{T_{TR}, T_{CH}, T_{REC}\})$.

Καταλήγουμε δείχνοντας την προσέγγιση μας για επαλήθευση. Ας υποθέσουμε ότι το καθήκον προς επαλήθευση στο παραπάνω παράδειγμα είναι το ακόλουθο: «Για όλα τα μηνύματα, η διαφορά του χρόνου κατά τον οποίο ένα μήνυμα στέλνεται και αυτού κατά τον οποίο παραλαμβάνεται, είναι μικρότερη από d_5 μονάδες χρόνου». Αυτό το συμπλήρωμα καθήκοντος περιγράφεται απο τον ελεγκτή M_{TC} στο σχήμα 4.2. Ο M_{TC} διαλέγει μη αιτιοκρατικά ένα ζευγάρι αποστολής και λήψης μηνύματος για το οποίο ο περιορισμός χρόνου παραβιάζεται. Αυτό κωδικοποιήται από το μετρητή T_{TC} με περιγραφή $set = (M_{TC}\$ = 1) \cdot (TR\# = msg!) \cdot (CH\# = msg?)$, $expire = (M_{TC}\$ = 2) \cdot (CH\# = msg!) \cdot (REC\# = msg?)$, $I = [d_5, \infty)$.

Με παρόμοιο τρόπο θα μπορούσαμε να κωδικοποιήσουμε το καθήκον «Ο χρόνος μεταξύ διαδοχικές λήψεις μηνυμάτων είναι πάντα μικρότερος από d_6 μονάδες χρόνου». Θα συζητήσουμε περισσότερο αυτά τα παραδείγματα στο κεφάλαιο 6.

[3]Οταν δεν υπάρχει συνθήκη ακύρωσης στην περιγραφή ενός μετρητή, εννοείται ότι είναι πάντα ψευδής.

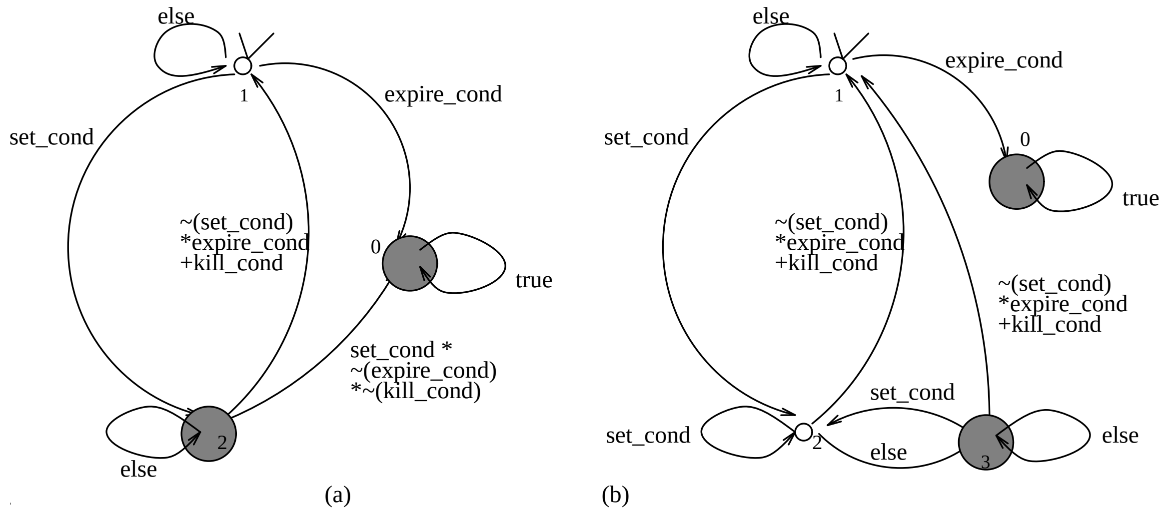


Σχήμα 4.2: Περιγραφή με χρόνο του συμπληρώματος καθήκοντος. Ο M_{TC} διαλέγει μη αιτιοκρατικά ένα ζευγάρι αποστολής και λήψης μηνύματος, και ο T_{TC} διαβεβαιώνει ότι αυτό συμβαίνει σε χρόνο περισσότερο από d_3 μονάδες χρόνου.

4.3 Υλοποίηση διεργασιών Ε/Α με χρόνο

Ως τώρα έχουμε παρουσιάσει την ακριβή υφή των χρονικών περιορισμών και την έννοια των $\mathcal{L}$-διεργασιών με χρόνο. Η προσέγγιση που έχουμε ακολουθήσει για επαλήθευση είναι να περιγράφουμε ένα σύστημα ως μια διεργασία γινόμενο $P = P_1 \otimes P_2 \ldots \otimes P_n$, το σύνολο των ανεπιθύμητων συμπεριφορών ως μια διεργασία συμπληρώματος καθήκοντος TC, τους χρονικούς περιορισμούς $\{T_1, T_2, \ldots, T_m\}$ (του συστήματος και του συμπληρώματος καθήκοντος) ως μια διεργασία M_T, και μετά να ελέγχουμε εαν η $P \otimes TC \otimes M_T$ είναι κενή. Τώρα συζητούμε την μετάφραση των περιορισμών πραγματικού χρόνου σε κατασκευές Ε/Α που αντιστοιχούν στη διεργασία ελεγκτή M_T,και εξετάζουμε την δομή της $.M_T$.

Ως πρώτο βήμα, το σύνολο $\{T_1, T_2, \ldots, T_m\}$ επαυξάνεται με την προσθήκη του μετρητή «χρονικής προόδου» TA. Αυτός ο μετρητής κωδικοποιεί το γεγονός ότι περνάει χρόνος ανάμεσα σε διαδοχικές μεταβάσεις του συστήματος, ακόμα κι αν αυτές δεν αντιστοιχούν σε συγκεκριμένα γεγονότα μετρητών. Οι συνθήκες θέσης και εκπνοής του TA ικανοποιούνται πάντοτε εκτός από τον λογικό χρόνο 0, όπου μόνο η συνθήκη θέσης ικανοποιείται. Αυτό επιτυγχάνεται προσθέτωντας τον ελεγκτή M_{TA} του σχήματος 4.4(α) και ορίζοντας set_{TA} =**true**, $expire_{TA} = (M_{TA}\$ = 2)$. Το διάστημα του TA είναι είτε $(0, \infty)$ αν διαλέξουμε τη σημασιολογία ότι πάντα περνάει μη μηδενικός χρόνος ανάμεσα στις μεταβάσεις, ή $[0, \infty)$ αν μπορεί να περάσει μηδέν χρόνος ανάμεσα στις μεταβάσεις. Οπως θα αναφέρουμε στην λεπτομερή περιγραφή της υλοποίησης του RTCOSPAN, δεν είναι αναγκαίο να υλοποιήσουμε τον TA σαν συνηθισμένο μετρητή. Χρειάζεται να λάβουμε υπόψην την επίδραση



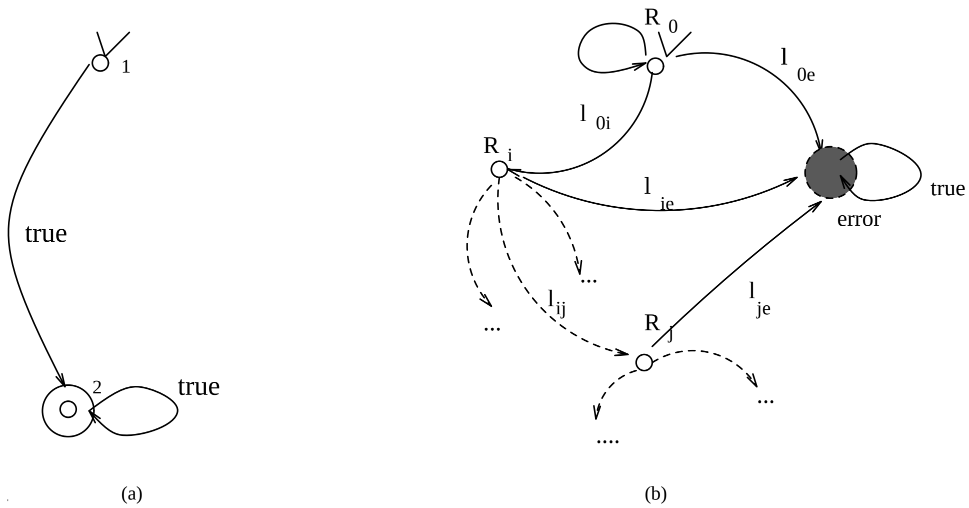
Σχήμα 4.3: Διεργασίες ελεγκτές που περιγράφουν (α) μετρητές και (β) μετρητές που μπορούν να επανατεθούν.

του μόνο αν δεν υπάρχει άλλο γεγονός μετρητή. Σε μια τέτοια περίπτωση, αν υπάρχει μια ακολουθία μεταβάσεων που όλες αντιστοιχούν σε γεγονότα που δεν έχουν σχέση με μετρητές, χρειάζεται να λάβουμε υπόψην μόνο την πρώτη εκπνοή του, μια και όλες οι εκπνοές που ακολουθούν δεν παρέχουν παραπάνω χρονική πληροφορία. Ο M_T κωδικοποιεί τους περιορισμούς των $\{T_1, T_2, \ldots, T_m, T_{TA}\}$, και είναι ο ίδιος μια σύνθετη διεργασία που απαρτίζεται από τις ακόλουθες απλούστερες διεργασίες:

$$M_T = WF_1 \otimes WF_2 \otimes \ldots \otimes WF_m \otimes WF_{TA} \otimes M_{TA} \otimes TR,$$

όπου m είναι ο αριθμός των λογικών μετρητών στην περιγραφή. Για κάθε λογικό μετρητή i, απαιτείται μια διεργασία ελεγκτής WF_i. Η WF_i εγγυάται ότι η ακολουθία των γεγονότων που αντιστοιχεί στον μετρητή i είναι «*καλώς ορισμένη*». Η δομή του παρουσιάζεται στα σχήματα 4.3(α) και 4.3(β) για ένα απλό μετρητή και ένα μετρητή που μπορεί να επανατεθεί αντίστοιχα. Η κατάσταση 1 είναι η αρχική κατάσταση, όπου ο μετρητής θεωρείται ανενεργός. Η κατάσταση 2 (και η κατάσταση 3 για μετρητή που μπορεί να επανατεθεί) είναι η «ενεργός» κατάσταση του μετρητή, ενώ η κατάσταση 0 είναι μια κατάσταση σφάλματος. Για τον μετρητή στο σχήμα 4.3(α) οι συνθήκες 1 και 2 της ενότητας 4.2.1 είναι εγγυημένες συμπεριλαμβάνοντας την κατάσταση 0 σε ενα σύνολο επαναλαμβανόμενων καταστάσεων, ενώ η συνθήκη 3 είναι εγγυημένη συμπεριλαμβάνοντας την κατάσταση 2 στο ίδιο σύνολο επαναλαμβανόμενων καταστάσεων. Για τον μετρητή που μπορεί να επανατεθεί στο σχήμα 4.3(β) η συνθήκη 2 είναι εγγυημένη από τον οριομό του συνόλου επαναλαμβανόμενων καταστάσεων της κατάστασης 0 και η συνθήκη 3 είναι εγγυημένη από τον ορισμό του συνόλου επαναλαμβανόμενων καταστάσεων της κατάστασης 3.

Η διεργασία Ε/Α ελεγκτής TR που παρουσιάζεται στο σχήμα 4.4(β) έχει καταστάσεις



Σχήμα 4.4: Διεργασίες ελεγκτές που περιγράφουν (α) Διεργασία ελεγκτής M_{TA} για τον μετρητή «χρονικής προόδου» και (β) Ελεγκτής περιοχής μετρητών TR.

που αντιστοιχούν στα σύνολα τιμών των μετρητών. Τέτοιες καταστάσεις ονομάζονται *περιοχές μετρητών*, και η συνάρτηση μετάβασης τους είναι η υλοποίηση του αλγόριθμου του Dill [Dil89] που περιγράφεται στην ενότητα 4.4. Η αρχική κατάσταση του TR είναι η περιοχή μετρητών όπου όλοι οι μετρητές (εκτός του μετρητή 0) είναι ανενεργοί. Δεδομένης μιας περιοχής μετρητών R_i και κάποιας πληροφορίας σχετικής με τους μετρητές, ο αλγόριθμος υπολογίζει την επόμενη περιοχή R_j. Η πληροφορία αποτελείται από το σύνολο E_{ij} των μετρητών που εξέπνευσαν και το σύνολο των μετρητών S_{ij} οι οποίοι τέθηκαν στην τρέχουσα κατάσταση της αλυσίδας. Στην ορολογία του μοντέλου Ε/Α μπορούμε να θεωρούμε τις μεταβάσεις ανάμεσα στις καταστάσεις R_i και R_j να έχουν ως ετικέτα l_{ij}, την σύζευξη των συνθηκών εκπνοής των μετρητών στο E_{ij} και των συνθηκών θέσης των μετρητών στο S_{ij}.

Οπως θα δούμε στις λεπτομέρειες της υλοποίησης, δεν είναι απαραίτητο να κατασκευάσουμε ολόκληρο τον TR σαν μια διεργασία Ε/Α ελεγκτή. Αντίθετα θα το κατασκευάζουμε δυναμικά ανάλογα με τις ανάγκες κατά την ανάλυση. Η TR είναι πραγματικά μια διεργασία πεπερασμένων καταστάσεων μια και ο αριθμός των προσβάσιμων περιοχών είναι πεπερασμένος. Ακόμα μια παρατήρηση είναι η ακόλουθη: όταν γεγονότα μετρητών παράγουν μια περιοχή μετρητών ασυνεπή με τους περιορισμούς πραγματικού χρόνου που επιβάλονται στο σύστημα, δεν είναι σημαντικό να γνωρίζουμε το ακριβές γεγονός που προκάλεσε την ασυνέπεια. Ετσι για τους σκοπούς μας είναι αρκετό να συμπυκνώσουμε όλες τις ασυνεπείς καταστάσεις μετρητών σε μια μοναδική κατάσταση που θα ονομάζουμε κατάσταση «σφάλμα« και να συμπεριλάβουμε αυτή την κατάσταση σε ένα σύνολο επαναλαμβανόμενων καταστάσεων (βλέπε σχήμα 4.4(β)).

Σημειώστε ότι οι λογικές συνθήκες (ονομαστικά οι *set*, *expire*, *cancel*) σε όλες τις συνιστώσες της M_T είναι όλες κατηγορήματα που ανήκουν στην $\mathcal{L}$, και η M_T είναι μια $\mathcal{L}$-διεργασία ευαίσθητη μόνο στα γεγονότα του συστήματος που σχετίζονται με μετρητές.

4.4 Περιοχές μετρητών και ο ελεγκτής TR

Περιγράφουμε τώρα τη μέθοδο [Dil89] που χρησιμοποιείται από τον ελεγκτή TR για να κωδικοποιεί πληροφορία χρόνου μέσα στις καταστάσεις του. Αυτή η μέθοδος βασίζεται στο διαμερισμό των τιμών των μετρητών σε ένα πεπερασμένο σύνολο από κλάσεις. Αυτές οι κλάσεις περιέχουν πληροφορία η οποία συνοψίζει την ιστορία του παρελθόντος του συστήματος σε σχέση με το χρόνο, και είναι κυρτές γραμμικές περιοχές στο θετικό m-διάστατο ορθοκανονικό υπόχωρο, οι οποίες είναι εύκολο να υλοποιηθούν.

Μια «*αποτίμηση μετρητή*» ορίζεται να είναι μια απεικόνιση από ένα σύνολο ενεργών (που έχουν τεθεί αλλά δεν έχουν εκπνεύσει ή ακυρωθεί) λογικών μετρητών στους πραγματικούς αριθμούς. Αυτή η απεικόνιση καταγράφει την ποσότητα του χρόνου μέχρι κάθε ενεργός μετρητής να εκπνεύσει. Ορίζουμε μια «*περιοχή μετρητών*» να είναι μια απεικόνιση από ενα σύνολο ενεργών μετρητών σε ένα σύνολο από αποτιμήσεις μετρητών. Οι περιοχές μετρητών είναι καταστάσεις του TR.

Παριστάνουμε περιοχές μετρητών σαν ένα σύνολο απο γραμμικές ανισότητες. [4] Αυτοί έχουν τη μορφή $t_i - t_j \leq d_{ij}$ όπου d_{ij} είναι ακέραιος, ή ∞. Σημειώστε ότι $t_j - t_i \leq d_{ij}$ είναι ισοδύναμο με $-d_{ji} \leq t_i - t_j$,το οποίο σημαίνει ότι τέτοιες ανισότητες μπορούν επίσης να παρέχουν κάτω φράγματα σε διαφορές. Ενας πλασματικός λογικός μετρητής T_0 ο οποίος πάντοτε έχει την τιμή 0 χρησιμοποιείται για να αναπαραστήσει ανισότητες όπως η $t_i < 10$. Σε αυτή την περίπτωση οι $t_i - t_0 < d_{i0}$ και $t_0 - t_i < d_{0i}$ παρέχουν σταθερά άνω και κάτω φράγματα αντίστοιχα.

Ενας κατάλληλος τρόπος για το χειρισμό ενός συστήματος ανισοτήτων αυτής της μορφής είναι να το αναπαραστήσουμε με ένα τετραγωνικό πίνακα D, που ονομάζεται ***πίνακας φραγμάτων διαφορών*** ή ΠΦΔ για συντομία (difference bounds matrix or DBM). Σημειώστε ότι μπορεί να υπάρχουν υπονοούμενοι περιορισμοί σε ένα σύστημα ανισοτήτων, έτσι μια περιοχή μετρητών μπορεί να αναπαρασταθεί από πολλούς ΠΦΔ. Για παράδειγμα θεωρείστε τα δυο συστήματα ανισοτήτων:

$$\begin{aligned} t_1 - t_2 &< 1 \\ t_2 - t_3 &< 1 \\ t_1 - t_3 &< 100 \end{aligned} \qquad\qquad \begin{aligned} t_1 - t_2 &< 1 \\ t_2 - t_3 &< 1 \\ t_1 - t_3 &< 2 \end{aligned}$$

[4] Εναλλακτικά θα μπορούσαμε να θεωρήσουμε μια περιοχή μετρητών σαν ένα πολύτοπο στον $k+1$-διάστατο χώρο, όπου k είναι ο αριθμός των μετρητών (χρονικοί περιορισμοί).

Είναι φανερό ότι και τα δυο αναπαριστούν την ίδια περιοχή, μια και οι πρώτες δυο ανισότητες υπονοούν ότι $t_1 - t_3 < 2$. Στην πραγματικότητα αντικατάσταση της τελευταίας ανισότητας $t_1 - t_3 < c$ για κάθε $c \geq 2$ δεν θα αλλάξει την περιοχή που παριστάνουν αυτές οι ανισότητες.

Ευτυχώς υπάρχει μια πράξη πάνω σε ένα ΠΦΔ που ονομάζεται *κανονικοποίηση* (canonicalization), η οποία ανάγει κάθε ΠΦΔ που αναπαριστά την ίδια περιοχή σε ένα και μοναδικό ΠΦΔ. Συμβολίζουμε την κανονικοποίηση του ΠΦΔ D ως $\mathbf{cf}(D)$. Η κανονικοποίηση παρέχει ένα εύκολο τρόπο να ελέγχουμε αν η περιοχή ενός ΠΦΔ είναι κενή, ή αν δυο ΠΦΔ περιγράφουν την ίδια περιοχή.

Η πράξη της κανονικοποίησης βασίζεται στην ακόλουθη παρατήρηση: Οι τιμές των d_{ij} μπορούν να ελαχιστοποιηθούν χωρίς να αλλάξει η περιοχή του ΠΦΔ, αν χειριστούμε τον ΠΦΔ D σαν τον πίνακα ακμών ενός γράφου, όπου οι τιμές των d_{ij} είναι μήκη ακμών, και μετα λύσουμε το πρόβλημα «συντομότερης διαδρομής για όλα τα ζευγάρια». Ενας συνηθισμένος αλγόριθμος όπως ο Floyd-Warshall μπορεί να χρησιμοποιηθεί.

Τα στοιχεία του D είναι ζεύγη άκρων. Αν $i > j$, d_{ij} είναι ένα άνω φράγμα του $t_i - t_j$ και $-d_{ji}$ είναι ένα κάτω φράγμα του $t_i - t_j$. Τα στοιχεία της κυρίας διαγωνίου είναι πάντα ίσα με $(0, \leq)$. Ο εσωτερικός βρόγχος του αλγόριθμου Floyd-Warshall περιέχει την εκχώρηση $d_{ij} := \min(d_{ik} + d_{kj}, d_{ij})$. Μια και d_{ij} είναι ζεύγος άκρων και όχι ακέραιος, είναι απαραίτητος ένας ορισμός των πράξεων min και + πάνω σε ζεύγη άκρων:

$$\begin{aligned} (x, r) + (x', r') &= (x + x', \min(r, r')) \\ \min((x, r), (x', r')) &= \begin{cases} (x, r) & \text{if } (x, r) \leq (x', r') \\ (x', r') & \text{otherwise} \end{cases} \end{aligned}$$

Αν D αντιπροσωπεύει ένα επιλύσιμο σύνολο από ανισότητες, ο αλγόριθμος Floyd-Warshall θα παράγει ένα μοναδικό ΠΦΔ με ελάχιστες τιμές σε όλα τα στοιχεία d_{ij}. Διαφορετικά, ένα στοιχείο της διαγωνίου d_{ii} θα πάρει μια τιμή μικρότερη από $(0, \leq)$ κάποια στιγμή. Οταν βρεθεί ένας τέτοιος κύκλος με αρνητικό κόστος, ο αλγόριθμος θα έπρεπε να τερματίσει (αντί να πέσει σε ένα άπειρο βρόγχο).

Κάθε κατάσταση του TR είναι ένας ΠΦΔ, ο οποίος αναπαριστά ένα σύνολο από δυνατές αποτιμήσεις μετρητών. Με κάθε ΠΦΔ συνδέουμε ένα σύνολο δεικτών που περιέχει τον T_0 και το αντίστοιχο σύνολο ενεργών μετρητών. Το σύνολο δεικτών στην αρχική κατάσταση είναι $\{0\}$.

Θα περιγράψουμε τώρα τη συνάρτηση μετάβασης του TR. Κάθε κατάσταση της αλυσίδας (τιμές της ολικής κατάστασης—ολικής επιλογής) καθορίζουν μοναδικά το σύνολο των γεγονότων που σχετίζονται με μετρητές και των οποίων τα κατηγορήματα ικανοποιούνται από την τιμή του c. Από κάθε δεδομένη κατάσταση του TR με ΠΦΔ D και για κάθε σύνολο γεγονότων που σχετίζονται με μετρητές B (τα οποία όλα συμβαίνουν ταυτόχρονα), ορίζονται μια μοναδική επόμενη κατάσταση και ο αντίστοιχος ΠΦΔ D'. Η ετικέτα της μετάβασης $D \to D'$ είναι η σύζευξη των κατηγορημάτων των γεγονότων στο B. Ετσι αν D και D' αντιστοιχούν στις καταστάσεις R_i και R_j στο γράφο του TR στο σχήμα 4.4, η παραπάνω ετικέττα είναι l_{ij}. Εστω A το σύνολο των ενεργών μετρητών στο D (χωρίς να συμπεριλαμβάνεται ο 0), $E = \{i \mid expire_i \in B\}$ και $S = \{i \mid set_i \in B\}$. Ο D' αντιπροσωπεύει το σύνολο όλων των δυνατών αποτιμήσεων μετρητών κατά τη στιγμή που έχουν συμβεί τα γεγονότα στο B. Πρώτα γίνεται η επεξεργασία των γεγονότων $Expire$, ακολουθούμενη από τα γεγονότα $cancel$ και set. Υποθέστε ότι συμβαίνει ένα μόνο γεγονός $expire_i$. Τότε υποθέτουμε ότι έχει περάσει ένα άγνωστο χρονικό διάστημα $\delta > 0$ πριν αυτό το γεγονός εκπνοής. Η τιμή του δ περιορίζεται ανάμεσα στα σταθερά άνω και κάτω φράγματα της τιμής του T_i $[-d_{0i}, d_{i0}]$ στην περιοχή της προηγούμενης κατάστασης.

Η κατασκευή του D' αποτελείται από τα ακόλουθα βήματα. Πρώτα πρέπει να ορίσουμε ακριβώς την υποπεριοχή των αποτιμήσεων μετρητών στο D όπου γεγονότα από το B είναι δυνατά. Μια συνάρτηση $\mathbf{rte}(D, B)$ (από το «περιορισμός συνόλου γεγονότων», restrict to event set) χρησιμοποιήται για να μετασχηματίσει το D σε ένα νέο πίνακα D^1 που αναπαριστά ακριβώς αυτή την υποπεριοχή. Για να συμβεί το B, όλοι οι εκπνέοντες μετρητές στο B πρέπει να έχουν την ίδια τιμή (αφού εκπνέουν ταυτόχρονα) και κάθε μετρητής που εκπνέει πρέπει να έχει τιμή μικρότερη από αυτή κάθε μετρητή που δεν εκπνέει. Σημειώστε ότι η $\mathbf{rte}$ δεν επηρεάζεται από μετρητές που τίθενται τώρα. Η συνάρτηση $\mathbf{rte}$ ορίζεται ως ακολούθως:

$$\begin{aligned}
d^1_{ij} &= \min(d_{ij}, (0, \leq)) \qquad \text{όταν } i, j \in E \\
d^1_{ij} &= \min(d_{ij}, (0, <)) \qquad \text{όταν } i \in E \text{ και } j \in A - E\ {}^5 \\
d^1_{ij} &= d_{ij} \qquad \text{αλλιώς.}
\end{aligned}$$

Ο D^1 δεν είναι απαραίτητα σε κανονική μορφή, έτσι η $\mathbf{rte}$ πρέπει μετά να εφαρμόσει την $\mathbf{cf}$ σε αυτόν. Η κανονικοποίηση μπορεί να παράγει ένα μη ικανοποιήσιμο πίνακα, που σημαίνει ότι το σύνολο γεγονότων B δεν μπορεί να εμφανιστεί σε αυτό το σημείο σε οποιοδήποτε ίχνος που είναι συνεπές με τους χρονικούς περιορισμούς. Η εμφάνιση μη ικανοποιήσιμων πινάκων είναι το κρίσιμο σημείο στη μέθοδο μας, όπου συγκεκριμένες αλυσίδες της χρονικά-ανεξάρτητης περιγραφής του συστήματος απορίπτονται. Αν το σύνολο γεγονότων B παράγει ένα μη ικανοποιήσιμο πίνακα, τότε η επόμενη κατάσταση του TR πρέπει να είναι η κατάσταση $error$. Ετσι στο γράφο του TR, η ετικέτα l_{ie} (όπου R_i αντιστοιχεί στο D) είναι η διάζευξη—για όλα τα δυνατά γεγονότα B τα οποία παράγουν μη ικανοποιήσιμους πίνακες—των συζεύξεων των κατηγορημάτων των γεγονότων στο B.

Αν η $\mathbf{rte}$ παράγει ένα ικανοποιήσιμο πίνακα, το επόμενο βήμα είναι η ελλάτωση της τιμής κάθε μετρητή που δεν εκπνέει κατά την τιμή των μετρητών που εκπνέουν. Ονομάζουμε αυτή τη συνάρτηση **elapse**(D, B), και ορίζεται ως εξής:

$$\begin{aligned} d^1_{i0} = d_{ij} \wedge d^1_{0i} = d_{ji} &\quad \text{when } i \in A - E \text{ και } j \in E \\ d^1_{ij} = d_{ij} &\quad \text{αλλιώς} \end{aligned}$$

όπου D^1 είναι το αποτέλεσμα της **elapse**(D, B). Αυτός ο μετασχηματισμός κάνει την τιμή των μετρητών που εκπνέουν ίση με 0 μετονομάζοντας τον μετρητή i σε μετρητή 0 (ο οποίος είναι πάντα μηδέν), ενώ διατηρεί τις διαφορές ανάμεσα στους μετρητές.

Σαν επόμενο βήμα οι μετρητές που εκπνέουν διαγράφονται από τον A. Η παρακάτω ιδιότητα των κανονικών ΠΦΔ αποδεικνύεται πολύ χρήσιμη. Αν D' είναι ο D αφού διαγραφούν η γραμμή και η στήλη i, η περιοχή που παριστάνεται από τον D' είναι η προβολή της περιοχής του D πάνω σε όλες τις συντεταγμένες εκτός της i. Αυτό είναι ισοδύναμο με τη διαγραφή των μετρητών που εκπνέουν.

Τέλος πρέπει να ασχοληθούμε με τους μετρητές που ακυρώνονται ή τίθενται τώρα. Ενα γεγονός $cancel_i$ απλά προκαλεί τη διαγραφή της γραμμής και στήλης i (αυτή είναι μια ακόμα πράξη προβολής).

Για ένα γεγονός set_i π.χ., αν $i \in S$, θέτουμε το d_{i0} στο άνω άκρο του I_i, το d_{0i} στην αρνητική τιμή του κάτω άκρου του I_i, και $d_{ij} = d_{ji} = (\infty, <)$ για κάθε $j \in A - E$, και μετά εφαρμόζουμε τη **cf**. Σημειώστε ότι η εφαρμογή της **cf** είναι το ίδιο με το να θέσουμε $d_{ij} = d_{i0} + d_{0j}$ όποτε i ή j ανήκει στο S.

Συνοψίζοντας τη μέθοδο μας πρέπει να πούμε ότι, μια και οι ΠΦΔ είναι τμήμα των καταστάσεων του TR, θα έπρεπε να υπήρχε πεπερασμένος αριθμός από αυτές που να είναι προσβάσιμες από την αρχική κατάσταση του συστήματος. Αυτό αποδεικνύεται στο [Dil89]. Μια σημαντική παρατήρηση είναι ότι ο παραπάνω αλγόριθμος μπορεί να χρησιμοποιηθεί για να δημιουργήσει το TR δυναμικά, ενώ ψάχνει το γράφο του γινομένου των άλλων συνιστωσών του συστήματος.

Κεφάλαιο 5

Το εργαλείο RTCOSPAN

Σε αυτό το κεφάλαιο θα δείξουμε πως το COSPAN επεκτείνεται προσθέτωντας μια δυναμικά δημιουργούμενη διεργασία «ελεγκτή»η οποία παρακολουθεί τους χρονικούς περιορισμούς ώστε να εξαιρεί συμπεριφορές του συστήματος που είναι ασυνεπείς με την πληροφορία του χρόνου. Παρουσιάζουμε όλες τις λεπτομέρειες της υλοποίησης των διαδικασιών επαλήθευσης για διεργασίες με χρόνο όπως αυτές παρουσιάστηκαν στα προηγούμενα κεφάλαια. Το νέο εργαλείο RTCOSPAN που προκύπτει έχει υλοποιηθεί χωρίς να υπάρξει ανάγκη να μεταβληθούν η εσωτερική δομή του COSPAN ή η S/R γραμματική, δείχνοντας ότι επεκτάσεις σε συνεχή χρόνο καλά σχεδιασμένων εργαλείων επαλήθευσης είναι εφικτές. Το RTCOSPAN είναι συμβατό με το COSPAN.

Παρουσιάζουμε το τμήμα της διασύνδεσης χρήστη-εργαλείου (user-interface) που εισάγει τα νέα χαρακτηριστικά του RTCOSPAN, τη φιλοσοφία σχεδίασης μέσω της οποίας το RTCOSPAN ανταπεξέρχεται το μεγάλο πλήθος καταστάσεων που απαιτούν οι μετρητές χρόνου, και δίνουμε αρκετές λεπτομέρειες των εσωτερικών δομών δεδομένων και αλγορίθμων του εργαλείου. Η επεξεργασία της πληροφορίας του χρόνου γίνεται από ένα ανεξάρτητο τμήμα κώδικα έξω από τον κώδικα του COSPAN, και χρησιμοποιούνται ειδικές δυνατότητες του COSPAN όπως ο τερματισμός ψαξίματος (search truncation) για να μειώσουν τον χώρο του προβλήματος που πραγματικά ψάχνουμε και να επιταχύνουν ακόμα περισσότερο την ανάλυση του συστήματος. Τέλος αναφέρουμε μερικά προβλήματα και περιορισμούς της τωρινής υλοποίησης μας.

5.1 Διασύνδεση χρήστη–εργαλείου

Προσπαθήσαμε να κρατήσουμε τη διασύνδεση χρήστη–εργαλείου του RTCOSPAN όσο το δυνατό πιο κοντά σε αυτή του COSPAN, μια που κάτι τέτοιο θα ήταν εξαιρετικά βολικό για τους χρήστες του COSPAN. Το RTCOSPAN καλείται ακριβώς όπως το COSPAN. Οι

```
proctype R_TIMER(tid, lr, L, U, ur : integer; set_cond, expire_cond, kill_cond:
boolean)
        import                  tid, lr, L, U, ur, set_cond, expire_cond, kill_ cond
        selvar      #           : (inactive, active, error)
        stvar       tstate      : (0..2)
        cyset       2           /* Timers should always eventually expire */
        init        1
        trans

        1                       {inactive}
                    -> 0        : expire_cond
                    -> 2        : else * set_cond
                    -> 1        : else;

        2                       {active}
                    -> 2        : expire_cond * set_cond
                    -> 1        : else * expire_cond
                    -> 1        : else * kill_cond
                    -> 0        : else * set_cond
                    -> 2        : else;

        0                       {error}
                    -> 0        : true;
        ...
    end /* R_TIMER()
```

Σχήμα 5.1: Τμήμα του αρχείου δηλώσεων `RT.h` που περιέχει τον S/R ορισμό για τον τύπο διεργασίας R_TIMER ο οποίος αντιστοιχεί στον ελεγκτή WF.

εκλογές (options) που αποδέχεται το RTCOSPAN είναι οι ίδιες, αν και ο χρήστης εκετελεί την εντολή "`rtcospan` [option] ... file" αντί για την "`cospan`". Καθαρά COSPAN περιγραφές γίνονται αποδεκτές και τρέχουν σωστά (μολονότι υπάρχει ένα πολύ μικρό επιπλέον κόστος όταν χρησιμοποιήται το RTCOSPAN).

Σε μια S/R περιγραφή του RTCOSPAN η χρήση της εντολής "`#include <RT.h>`" πριν δηλωθούν οποιηδήποτε μετρητές είναι υποχρεωτική. Το αρχείο δηλώσεων `RT.h` περιέχει την S/R περιγραφή των τύπων διεργασίας (proctype) R_TIMER και R_CLOCK. Ο R_TIMER ορίζει τον τύπο του συνηθισμένου λογικού μετρητή, ενώ ο R_CLOCK είναι ο τύπος του μετρητή που μπορεί να επανατεθεί. Μετρητές τύπου R_CLOCK αποδεικνύονται χρήσιμοι σε ορισμένες περιπτώσεις όπως θα δούμε στην ενότητα 6.1. Μερικές φορές μπορεί να αποκαλούμε ένα μετρητή που μπορεί να επανατεθεί «*ρολόι*».

Το σχήμα 5.1 παρουσιάζει ένα κομμάτι σε περιγραφή COSPAN μιας διεργασίας Ε/Α από το αρχείο `RT.h` η οποία υλοποιεί ελεγκτές WF σαν αυτόν που παρουσιάστηκε στο σχήμα 4.3(α). Ο ορισμός του τύπου διεργασίας R_CLOCK έιναι πολύ παρόμοιος γι'αυτό και δεν τον δείχνουμε εδώ. Πρώτα συζητάμε τις παραμέτρους των τύπων διεργασίας R_TIMER και R_CLOCK. Αυτές οι παράμετροι είναι οι ίδιες και για τους δυο τύπους διεργασίας και συμπεριλαμβάνουν:

1. Την παράμετρο `tid` που καθορίζει την ταυτότητα κάθε μετρητή. Παίρνει τιμές από 1 ως τον αριθμό των μετρητών που έχουν δηλωθεί. Η πραγματική τιμή της `tid` για κάθε μετρητή εξαρτάται μόνο από την σειρά εμφάνισης του μετρητή στην S/R περιγραφή. Η `tid` είναι η υπογεγραμμένη του μετρητή στον πίνακα ΠΦΔ της ενότητας 4.4. Είναι φανερό ότι αν διαφορετικοί μετρητές έχουν μοναδικές τιμές της `tid`, η σειρά εμφάνισης των μετρητών δεν είναι σημαντική. Αυτή η παράμετρος underlineδεν δίδεται από το χρήστη στο COSPAN πρόγραμμα όταν ο χρήστης δηλώνει ένα μετρητή, αλλά εισάγεται αυτόματα από τον στοιχειώδη μεταφραστή του RTCOSPAN πριν κληθεί ο S/R μεταφραστής.

2. Οι παράμετροι 2 εως 5 καθορίζουν το διάστημα πραγματικών, το οποίο αντιστοιχεί στις τιμές όπου πάντα τίθεται ο μετρητής.

   - Η `lr` είναι είτε «\(» ή «\[» καθορίζοντας αν το κάτω άκρο του διαστήματος είναι ανοικτό ή κλειστό.
   - Η `L` είναι ένας ακέραιος που καθορίζει το κάτω άκρο του διαστήματος.
   - Η `U` είναι ένας ακέραιος πιθανώς και άπειρο [1] που καθορίζει το άνω άκρο του διαστήματος.
   - `ur` είναι είτε «\)» ή «\]» καθορίζοντας αν το άνω άκρο του διαστήματος είναι ανοικτό ή κλειστό.

   Για παράδειγμα ένα δείγμα περιγραφής διαστήματος είναι κάπως έτσι: «`\(,5,14,\]`» (όσον αφορά τις παραμέτρους δύο εως πέντε).

3. Η `set_cond` είναι ένα λογικό κατηγόρημα πάνω στις μεταβλητές ολικής επιλογής και ολικής κατάστασης του μοντέλου Ε/Α. Αυτό το κατηγόρημα καθορίζει τις συνθήκες που επιτρέπουν στο γεγονός θέσης ενός μετρητή να συμβεί. Για ένα μετρητή που μπορεί να επανατεθεί αυτό είναι επίσης η συνθήκη που επιτρέπει την επανάθεση. Υποθέστε για παράδειγμα ότι όπως πριν $P_i.\#$ συμβολίζει την μεταβλητή επιλογής της διεργασίας P_i, και έστω οι διεργασίες `machine1`, `machine2` και `assemblyline`. Τότε μια συνθήκη θέσης για ένα μετρητή έστω `assemblyidle` θα μπορούσε να είναι η εξής:

[1] άπειρο στην πραγματικότητα σημαίνει ίσο με μια σταθερά που εξαρτάται από τη μηχανή και ονομάζεται `infty`.

`(machine1.#=jammed)*(machine2.#~ =idle)+(assemblyline.#=reset)`

4. Η `expire_cond` είναι ένα λογικό κατηγόρημα παρόμοιο με το `set_cond` που καθορίζει την συνθήκη εκπνοής ενός μετρητή (που μπορεί να έχει δυνατότητα επανάθεσης ή όχι).

5. Η `kill_cond` (cancel_condition) είναι επίσης ένα λογικό κατηγόρημα σε μεταβλητές κατάστασης και επιλογής, το οποίο καθορίζει μια συνθήκη ακύρωσης για ένα μετρητή. Οταν ένας μετρητής ακυρώνεται θεωρείται ανενεργός και δεν επηρεάζει τον έλεγχο συνέπειας των περιοχών μετρητών, εκτός εαν τεθεί εκ νέου η μεταβεί στην κατάσταση σφάλματος όπου ο χώρος καταστάσεων περικόπτεται.

Ο χρήστης δηλώνει ένα λογικό μετρητή με τις εντολές

1. **proc** *name*:**R_TIMER**(`lr, L, U, ur, set_cond, expire_cond, cancel_cond`)

2. **proc** *name*:**R_CLOCK**(`lr, L, U, ur, set_cond, expire_cond, cancel_cond`)

όπου *name* είναι ένα έγκυρο όνομα διεργασίας στο COSPAN και όλες οι επτά παράμετροι είναι όπως ορίστηκαν παραπάνω. Στην περίπτωση που ένας μετρητής δεν ακυρώνεται ποτέ, η `cancel_cond` θα πρέπει να είναι ίση με 0 ή **false**. Η συνθήκες γεγονότων που έχουν σχέση με μετρητές θα είναι πάντα αληθείς εαν οι αντίστοιχες παράμετροι έχουν την τιμή 1 ή **true**.

Μελετώντας τη συνάρτηση μετάβασης στον ορισμό του τύπου διεργασίας R_TIMER (σχήμα 5.1), παρατηρούμε ότι μερικές μερικές συνθήκες έχουν προτεραιότητα έναντι άλλων. Η ισχυρότερη είναι η συνθήκη εκπνοής μετρητή. Αν ισχύει η `expire_cond` και ο μετρητής είναι ανενεργός (κατάσταση 1), η επόμενη κατάσταση θα είναι η κατάσταση σφάλματος (κατάσταση 0) ανεξάρτητα αν ισχύει η `set_cond`. Αν ένας μετρητής είναι ενεργός, η `expire_cond` θα είναι η πρώτη συνθήκη που θα εξεταστεί, η `kill_cond` θα εξεταστεί μετά, και η τελευταία που θα εξεταστεί θα είναι η `set_cond`.

Το RTCOSPAN δεν παρέχει τρόπο για να οριστεί ένα σύνολο από μετρητές οι οποίοι αρχικά να έχουν τεθεί όπως το σύνολο A_0 στην ενότητα 4.2.1. Ο λόγος είναι ότι έτσι φτιάχνεται μια πιο αποδοτική υλοποίηση, ενώ το μοντέλο δεν χάνει καθόλου εκφραστική δύναμη. Αn ο χρήστης πρέπει να χρησιμοποιήσει ένα σύνολο A_0, πρέπει να δημιουργήσει μια νέα αρχική κατάσταση συστήματος έστω S'. Τοτε όλοι οι μετρητές που ανήκουν στο A_0 θα έπρεπε να τεθούν όταν το σύστημα μεταβαίνει από την S' σε όλες τις καταστάσεις που ήταν προηγούμενα αρχικές.

Η σωστή κατασκευή μοντέλων για συστήματα πραγματικού χρόνου χρησιμοποιώντας μετρητές δεν είναι καθόλου τετριμένη υπόθεση. Οι χρήστες θα πρέπει να είναι πολύ προσεκτικοί με τα κατηγορήματα θέσης, εκπνοής και ακύρωσης. Ο λόγος είναι ότι, όταν για παράδειγμα η συνθήκη εκπνοής ενός ενεργού μετρητή ισχύει σε πολλές διαδοχικές ολικές καταστάσεις κατά το ψάξιμο κατά βάθος του γράφου της διεργασίας που περιγράφει το σύστημα, μπορεί να προκύψει το ακόλουθο σενάριο. Την πρώτη φορά που η διεργασία βρίσκεται σε αυτές τις καταστάσεις ο μετρητής είναι ενεργός και ισχύει η συνθήκη εκπνοής του, έτσι στην επόμενη μετάβαση ο μετρητής πάει στην ανενεργή κατάσταση (1). Μια και στις επόμενες καταστάσεις του συστήματος της διεργασίας η συνθήκη εκπνοής του μετρητή ακόμα ισχύει, ο μετρητής θα εκπνεύσει ξανά. Η εκπνοή ενός μη ενεργού μετρητή οδηγεί στην κατάσταση σφάλματος (0), περικόπτοντας όλο το χώρο καταστάσεων. Αυτό δεν είναι πάντα ότι επιδιώκει ο χρήστης. Ευτυχώς είναι αρκετά εύκολο να κατασκευάσουμε μια απλή διεργασία ελεγκτή ώστε η συνθήκη εκπνοής να ισχύει μόνο μια φορά την πρώτη φορά που η διεργασία μπαίνει σε αυτό το σύνολο καταστάσεων.

Αν ο χρήστης χρειάζεται παραπάνω πληροφορία σχετικά με την επεξεργασία της πληροφορίας πραγματικού χρόνου και την ακριβή μορφή των περιοχών μετρητών, παρέχονται κάποιες χρήσιμες επιλογές για διόρθωση. Εκτελώντας την εντολή `rtcospan -DDEBUG file.sr` αρκετές βοηθητικές πληροφορίες θα εμφανιστούν στο αρχείο stderr, ενώ συνένωση του αρχείου `file.o` με τη βιβλιοθήκη `librt_D.a` αντί για την `librt.a` θα τυπώσει όλες τις περιοχές μετρητών που συναντώνται κατά την ανάλυση του συστήματος.

5.2 Υλοποίηση

Οπως αναφέραμε προηγουμένως, η περιγραφή με χρόνο μεταφράζεται σε ένα σύνολο από $\mathcal{L}$-διεργασίες (ή ελεγκτές) M_T, και κάποιες από αυτές (ονομαστικά οι WF_i και η TR) περιέχουν πρόσθετο κώδικα σε `C` (τμήματα (**code**) ο οποίος υλοποιεί τον αλγόριθμο που καθορίζει τις μεταβάσεις ανάμεσα σε περιοχές μετρητών όπως περιγράψαμε στην ενότητα 4.4. Ο χρήστης παρέχει τις δηλώσεις των μετρητών στο εργαλείο και το RTCOSPAN αυτόματα παράγει τον ελεγκτή TR, την ώρα που ελέγχει αν το γινόμενο όλων των διεργασιών είναι κενό. Οπως έχουμε ήδη αναφέρει σε προηγούμενες ενότητες, για λόγους επίδοσης διαλέξαμε να υλοποιήσουμε την κατασκευή του TR δυναμικά, την ώρα που ερευνάται ο χώρος καταστάσεων του γινομένου (θυμηθήτε ότι ο αλγόριθμος ανάλυσης συνίσταται βασικά στην κατασκευή των ισχυρά συνεκτικών συνιστωσών του γράφου της διεργασίας γινόμενο). Ενας σημαντικός λόγος για αυτό είναι ότι η M_T μπορεί να έχει πολύ μεγάλο αριθμό καταστάσεων, οι οποίες δεν είναι όλες προσβάσιμες στο γινόμενο με τις άλλες συνιστώσες του συστήματος. Ετσι η μετάφραση του M_T σε μια διεργασία ελεγκτή και στη συνέχεια τρέξιμο του COSPAN θα ήταν εντελώς μη αποδοτικό.

Ενα σημαντικό χαρακτηριστικό της υλοποίησης του αλγόριθμου ανάλυσης στο COSPAN είναι η δυνατότητα περικοπής. Οταν κατά τη διάρκεια του ψαξίματος απαντάται μια κατάσταση για την οποία ξέρουμε ότι η ίδια και όλες οι διάδοχοι καταστάσεις ανήκουν σε ένα

```
/* Lists are always sorted in increasing order */
typedef struct list {                  /* Active timers */
                struct list     *next;
                int             timer_id;
} list;

typedef struct timer_state {           /* Timers' state */
                list    *active_timers;
                int     **D;
} timer_state;

typedef struct bucket                  /* Hash table entries */
{
        struct bucket   *next;
        timer_state     *b_data;
} bucket;
```

Σχήμα 5.2: Σύνθετοι τύποι δεδομένων του RTCOSPAN.

σύνολο επαναλαμβανόμενων καταστάσεων, τίθεται η μεταβλητή του COSPAN «truncate» και το ψάξιμο σε βάθος τερματίζεται σε αυτή την κατάσταση. Είναι φανερό ότι αυτός ο γράφος που έχουμε περικόψει είναι ισοδύναμος με τον αρχικό σε σχέση με το πρόβλημα έγκλεισης γλωσσών, και στη γενική περίπτωση είναι μικρότερος.

5.2.1 Δομές δεδομένων

Τώρα εισάγουμε τους τύπους και τις δομές δεδομένων που χρησιμοποιούνται για την υλοποίηση των μετρητών και των περιοχών μετρητών. Το σχήμα 5.2 παρουσιάζει όλες τις δηλώσεις σε `C` των σύνθετων τύπων δεδομένων που χρησιμοποιεί το RTCOSPAN.

Κάθε μετρητής διακρίνεται εσωτερικά στο RTCOSPAN από ένα πεδίο ακέραιου τύπου που ονομάζεται `timer_id`. Σε κάθε περίπτωση σε αυτό εκχωρείται η τιμή της αντίστοιχης παραμέτρου του μετρητή `tid` η οποία αναφέρθηκε στην ενότητα 5.1. Τρείς λίστες μετρητών (timers_to_set, timers_to_expire, timers_to_kill) ορίζονται στη γραμμή 136 του αρχείου `crank.h` (Βλέπε παράρτημα Α). Αυτές οι λίστες περιέχουν τους μετρητές οι οποίοι πρέπει να τεθούν, εκπνεύσουν ή πρέπει να ακυρωθούν σε κάθε βήμα μετάβασης. Αν μια από τις λίστες είναι κενή, αυτό σημαίνει ότι δεν υπάρχει γεγονός που να σχετίζεται με μετρητές (του αντίστοιχου τύπου) το οποίο να συμβαίνει σε εκείνο το βήμα.

Οι περιοχές μετρητών υλοποιούνται σαν δομές του τύπου `timer_state`. Κάθε τέτοια δομή περιέχει μια λίστα από ενεργούς μετρητές και ένα τετραγωνικό πίνακα D με στοιχεία ακέραιους (πεδίο `**D`). Ο `D` είναι διαστάσεων (`ntimers` $+2$) $\times$ (`ntimers` $+2$), όπου `ntimers`

```
w_star (a, b)           /* actually + (or mult) operator */
int a, b;
{
register int na, nb, nr;
        if ((a == infinity) || (b == infinity))
                return(infinity);
        na = a & ~(int)1;
        nb = b & ~(int)1;
        nr = (a & b) % 2;
        return( (na + nb) | ( nr >= 0 ? nr : -nr) );
}
```

Σχήμα 5.3: Υλοποίηση σε C του τελεστή + για ζεύγη άκρων.

είναι μια ακέραια μεταβλητή η οποία ορίζεται στη γραμμή 134 του crank.h και αντιστοιχεί στον αριθμό των μετρητών που δηλώνει ο χρήστης. Για λόγους απόδοσης διαλέξαμε να υλοποιήσουμε την TA προσθέτωντας τον κατάλληλο κώδικα, χωρίς να χρειαστεί να τον αναφέρουμε ρητά στη λίστα των μετρητών. Ετσι ο AT είναι ένας μετρητής «ενσωματομένος» στον κώδικα της υλοποίησης. Αφού τα στοιχεία του D πρέπει να υλοποιούν ζεύγη άκρων, χρησιμοποιήται το ακόλουθο σχήμα κωδικοποίησης: Το λιγότερο σημαντικό δυαδικό ψηφίο αναπαριστά τα σύμβολα $<, \leq$ (Το 0 συμβολίζει το $<$ και το 1 συμβολίζει το $\leq$), ενώ τα υπόλοιπα δυαδικά ψηφία του ακέραιου παριστάνουν την πρώτη συνιστώσα του άκρου. Ετσι αν $(5, <)$ είναι ένα ζεύγος άκρων, η εσωτερική του αναπαράσταση είναι 10 ($10 = 5 << 1|0$, όπου το $<<$ σημαίνει ολίσθηση αριστερά κατά ένα και $|$ είναι ο τελεστής της πράξης «ή» πάνω σε δυαδικά ψηφία). Στην υλοποίηση μας ∞ σημαίνει μια σταθερά που εξαρτάται από τη μηχανή και ονομάζεται infinity. infinity = MAXINT$div2$, όπου MAXINT είναι ο μέγιστος ακέραιος που μπορεί να παραστήσει η μηχανή. Αυτό είναι ένα εξαιρετικά αποδοτικό σχήμα κωδικοποίησης μια και, η ακέραια αναπαράσταση των άκρων διατηρεί τη διάταξη τους σωστά·έτοι από τις πράξεις πάνω σε άκρα που ορίστηκαν στην ενότητα 4.4 μόνο η + πρέπει να υλοποιηθεί με ειδικό τρόπο. Η υλοποίηση είναι απλή όπως φαίνεται στο σχήμα 5.3.

Είναι φανερό ότι η μνήμη που χρειάζεται για την αποθήκευση κάθε αναπαράστασης περιοχής μετρητών είναι $O(m^2)$, όπου m είναι ο αριθμός των μετρητών. Είναι επίσης σίγουρο ότι κάθε περιοχή μετρητών θα την απαντήσουμε σε παραπάνω από μια ολικές καταστάσεις. Σε μια πολύπλοκη S/R περιγραφή μπορεί να είναι προσβάσιμος ένας μεγάλος αριθμός περιοχών μετρητών. Οταν οι απαιτήσεις σε μνήμη φτάνουν (ή ακόμα ξεπερνούν) την ποσότητα της φυσικής μνήμης του υπολογιστή, παρατηρείται μια δραματική πτώση της απόδοσης του εργαλείου, λόγω της συχνής ανάγκης για αντιμετάθεση σελίδων (page swaps). Οπότε, είναι φανερή η ανάγκη για μια αποδοτική πολιτική διαχείρισης μνήμης. Η προσέγγιση μας είναι η ακόλουθη: Χρησιμοποιείται ένας πίνακας κατακερματισμού (hash table) ο οποίος ονομάζεται `State_Table` για να αποθηκεύει όλες τις περιοχές μετρητών που συναντούμε κατά το ψάξιμο κατά βάθος του γράφου του γινομένου. Κάθε στοιχείο του `State_Table` είναι μια λιστα τύπου `bucket` (Βλέπε σχήμα 5.2). Κάθε στοιχείο της λίστας περιέχει ένα δείκτη σε μια δομή `timer_state`. Το μέγεθος του πίνακα κατακερματισμού είναι ένας πρώτος αριθμός που εξαρτάται από το πλήθος των μετρητών που χρησιμοποιούνται στην περιγραφή και καθορίζεται δυναμικά σε χρόνο εκτέλεσης κατά την αρχική φάση εκτέλεσης. Ενας μονοδιάστατος πίνακας που περιέχει πρώτους αριθμούς χρησιμοποιείται για αυτό το σκοπό. Αυτή τη στιγμή το μεγαλύτερο μέγεθος πίνακα κατακερματισμού είναι 32771. Οι τιμές της συνάρτησης κατακερματισμού που χρησιμοποιείται εξαρτώνται από τις τιμές εκείνων των στοιχείων του `D` που αντιστοιχούν σε μετρητές ενεργούς σε εκείνη την κατάσταση. Οταν ανιχνεύεται μια σύγκρουση κατακερματισμού, η νέα δομή τύπου `timer_state` εισάγεται στην κεφαλή της λίστας τύπου `bucket`. Η διεργασία Ε/Α ελεγκτής `Timer_Region_Check` η οποία υλοποιεί τον ελεγκτή TR που αναφέρθηκε στην ενότητα 4.3 έχει ένα δείκτη σε μια `timer_state` δομή για μεταβλητή κατάστασης. Αυτός ο δείκτης είναι στην πραγματικότητα αυτός που φυλάσεται στο αντίστοιχο στοιχείο τύπου `bucket` στον πίνακα κατακερματισμού.

Τέλος χρησιμοποιούμε δύο μονοδιάστατους πίνακες ακεραίων με μέγεθος `ntimers` που ονομάζονται `UTLIM` και `LTLIM` για την αποθήκευση των άνω και κάτω άκρων κάθε μετρητή αντίστοιχα. Αυτοί οι πίνακες δηλώνονται στην γραμμή 137 του αρχείου `crank.h`. Οταν ο μετρητής `i` τίθεται, οι τιμές των `LTLIM[i]` και `UTLIM[i]` εκχωρούνται στα στοιχεία `D[0][i]` και `D[i][0]` του πίνακα περιοχών μετρητών αντίστοιχα (θυμηθήτε ότι D_{0i} είναι το άνω φράγμα για τη διαφορά $t_0 - t_i$ και $-D_{i0}$ είναι το κάτω φράγμα για τη διαφορά $t_i - t_0$).

5.2.2 Εσωτερική δομή και οργάνωση

Το RTCOSPAN τελικά χρησιμοποιεί όλες τις αρχικές συνιστώσες του COSPAN συν μερικές καινούργιες. Μερικές από τις συνιστώσες του COSPAN έχουν μεταβληθεί όπου αυτό υπήρξε αναγκαίο. Κάθε εργασία του RTCOSPAN περιέχει τα ίδια τρία βήματα που αναφέρθηκαν στην ενότητα 3.2 αλλά το αρχείο `rtcospan` έχει μεταβληθεί ώστε:

(a) Να μεταφράζει τους ορισμούς στο βήμα 1, και να προσθέτει κάποιο βοηθητικό `sr` κώδικα στο τέλος της περιγραφής του χρήστη πριν καλέσει τον S/R μεταφραστή.

(b) Να μεταφράζει και να συνενώνει τον κώδικα σε `C` που παράγει το προηγούμενο βήμα επιπλέον με μια νέα βιβλιοθήκη `librt.a` η οποία περιέχει όλες τις ρουτίνες που υλοποιούν τον αλγόριθμο που επεξεργάζεται τις περιοχές μετρητών.

```
switch(f){
case BEG_INIT:
       ...
case CONT_INIT:
       ...
       break;
case BEG_SELECT:
       ...
case CONT_SELECT:
       ...
       break;
case BEG_RESOLVE:
       ...
case CONT_RESOLVE:
       ...
}
```

Σχήμα 5.4: Γενική δομή ενός δομικού λίθου για την `crank()`.

```
0:      Adjustment_Timer
1:      Timer_Region_Check
2:      user_timer_one
   ...
n+1:    user_timer_n

n+2:    system_process_one

n+k+2:  Hash_Init
```

Σχήμα 5.5: Ιεραρχία των διεργασιών στο RTCOSPAN.

Οι ρουτίνες της `librt.a` καλούνται στο `code` τμήμα κώδικα (`code`) της διεργασίας `Timer_Region_Check` η οποία δηλώνεται αυτόματα από την τροποποιημένη μας έκδοση του αρχείου δηλώσεων `crank.h`. Ο κύριος σκοπός του αρχείου `crank.h` είναι να παρέχει τις δηλώσεις για το αρχικό και το τελικό τμήμα της συνάρτησης `crank()`. Αυτή η συνάρτηση καλείται σε κάθε βήμα επιλογής απόφασης από τη βιβλιοθήκη του COSPAN `libsr.a` κατά την εκτέλεση του αρχείου σε γλώσσα μηχανής που το COSPAN παράγει στο βήμα 2. Δεδομένης μιας συγκεκριμένης ολικής κατάστασης, η `crank()` παράγει μια επόμενη κατάσταση κάθε φορά που καλείται. Οταν εξαντληθούν όλες οι επόμενες καταστάσεις, απλά ανακοινώνει το γεγονός μέσω της τιμής που επιστρέφει (`NO_MORE`). Η συνάρτηση `crank()` αποτελείται από μια σειρά από τμήματα (blocks) από εντολές switch σαν αυτή που παρουσιάζεται στο σχήμα 5.4. Υπάρχει ένα τέτοιο τμήμα ανά διεργασία της περιγραφής. Αν υπάρχει ένα τμήμα `C` κώδικα στην περιγραφή μιας διεργασίας Ε/Α, αυτό μπαίνει σε μια δεύτερη εντολή switch της ίδιας δομής η οποία ακολουθεί την εντολή switch που αντιστοιχεί στη δομή μετάβασης της διεργασίας. Συμβολίζουμε με `TR_BLOCK` και `CODE_BLOCK` τα παραπάνω τμήματα των εντολών switch (βλέπε σχήμα 5.6). Η μεταβλητή `f` είναι μια μεταβλητή του COSPAN που αρχικά τίθεται στην τιμή `BEG_INIT`, μετά στην `CONT_INIT` (αρχική φάση του συστήματος), και στη συνέχεια εναλλάσεται ανάμεσα στις τιμές `BEG_SELECT, CONT_SELECT` ή `BEG_RESOLVE, CONT_RESOLVE`. Αν η `f` είναι ίση με `BEG_SELECT` ή `CONT_SELECT` ο κώδικας στις εντολές switch κάθε διεργασίας διαλέγει μια επιλογή, και όταν εκτελεστούν όλες οι εντολές switch έχει κατασκευαστεί η ολική επιλογή. Παρόμοια όταν η `f` ισούται με `BEG_RESOLVE` ή `CONT_RESOLVE`, παράγεται η επόμενη ολική κατάσταση. Η πρώτη και η τελευταία εντολή switch που ορίζονται στις γραμμές 156 και 193 του αρχείου β`crank.h` (βλέπε ευρετήριο Α) είναι πάντα οι ίδιες, ενώ οι υπόλοιπες παράγονται από τον S/R μεταφραστή, μια για κάθε διεργασία Ε/Α που ορίζει ο χρήστης.

Η ιεραρχία των διεργασιών μέσα στο `.c` αρχείο που παράγει το RTCOSPAN (δηλ. η σειρά εμφάνισης στον `C` κώδικα) είναι λίγο πιο πολύπλοκη από ότι στο COSPAN, και παρουσιάζεται στο σχήμα 5.5. Εδώ υποθέτουμε ότι ο χρήστης δηλώνει `n` διεργασίες μετρητών και ότι το κομμάτι της περιγραφής χωρίς χρόνο αποτελείται από `k` διεργασίες. Η `Hash_Init` είναι μια εικονική διεργασία η οποία περιέχει μόνο ένα τμήμα κώδικα που δίνει αρχικές τιμές στις δομές δεδομένων του πίνακα κατακερματισμού. Η διεργασία `Adjustment_Timer` αντιστοιχεί στον ελεγκτή M_{TA} της ενότητας 4.3 και η διεργασία `Timer_Region_Check` στον ελεγκτή TR. Αν και οι `Adjustment_Timer` και `Timer_Region_Check` εμφανίζονται πρώτες σε αυτή την ιεραρχία, τα κομμάτια κώδικα τους εκτελούνται μετά την `Hash_Init` (βλέπε γραμμές 191 και 192 του αρχείου `crank.h` στο ευρετήριο Α). Αυτό είναι υποχρεωτικό μια και αυτές οι δύο διεργασίες πρέπει να είναι οι τελευταίες που θα αποφασίσουν την επόμενη

```
TR_BLOCK     of Adjustment_Timer
TR_BLOCK     of Timer_Region_Check
TR_BLOCK     of user_timer_one
CODE_BLOCK   of user_timer_one
    ...
TR_BLOCK     of user_timer_n
CODE_BLOCK   of user_timer_n
    ...
TR_BLOCK     of system_process_one
CODE_BLOCK   of system_process_one
    ...
CODE_BLOCK   of Hash_Init
CODE_BLOCK   of Adjustment_Timer
CODE_BLOCK   of Timer_Region_Check
```

Σχήμα 5.6: Δομή της crank() στο RTCOSPAN.

κατάσταση τους κατά το βήμα απόφασης, επειδή όλα τα γεγονότα που έχουν σχέση με μετρητές πρέπει να αποτιμηθούν πριν υπολογιστεί η επόμενη περιοχή μετρητών και ελεγχθεί η συνέπεια της. Οι διεργασίες τύπου `R_TIMER` και `R_CLOCK` πρέπει να είναι οι πρώτες που εκτελούν κώδικα, λόγω της παρουσίας εντολών `goto` στον `C` κώδικα που παράγει το COSPAN οι οποίες μπορεί να εμποδίσουν την σωστή αποτίμηση των γεγονότων σχετικά με τους μετρητές τα οποία συμβαίνουν.

Γεγονότα σχετικά με μετρητές που πρόκειται να συμβούν αποτιμούνται μέσα στον κώδικα της αντίστοιχης διεργασίας τύπου `R_TIMER`. Το σχήμα 5.7 παρουσιάζει τη μορφή αυτού του τμήματος κώδικα. Χρησιμοποιούμε ψευδοκώδικα αφού οι συντακτικές λεπτομέρειες δεν είναι σημαντικές για την κατανόηση της σχεδίασης μας. Κατά την αρχική φάση του συστήματος πραγματοποιείται ένας στοιχειώδης έλεγχος της εγκυρότητας των άκρων των διαστημάτων των μετρητών και υπολογίζεται ο ολικός αριθμός των μετρητών στην περιγραφή. Στη συνέχεια, υπολογίζεται η εσωτερική αναπαράσταση των άκρων και αποθηκεύεται στους πίνακες `LTLIM` και `UTLIM`. Δεν χρειάζεται να γίνει κανένας υπολογισμός κατά το βήμα της επιλογής.

Στο βήμα απόφασης, πρώτα ελέγχουμε αν αυτή η διεργασία μετρητή βρίσκεται στην κατάσταση σφάλματος (`tstate` είναι η μεταβλητή κατάστασης του ελεγκτή). Σε αυτή την περίπτωση τίθενται η εσωτερική μεταβλητή του COSPAN `truncate` και η μεταβλητή `bad_timer_region`. Οταν τίθεται η μεταβλητή `truncate`, το COSPAN περικόπτει την ανάλυση σε αυτό το σημείο, ενώ η `bad_timer_region` χρησιμοποιείται αργότερα από τον κώδικα `Timer_Region_Check`. Αν η μεταβλητή κατάστασης `tstate` δεν είναι 0, η βασική ιδέα είναι ότι, οι συνθήκες που επιτρέπουν την πραγματοποίηση γεγονότων σχετικών με το συγκεκριμένο μετρητή χρειάζεται να εξεταστούν μόνο στο συγκεκριμένο κομμάτι κώδικα. Αν ένα γεγονός σχετικό με το μετρητή πρόκειται να συμβεί άμεσα, η αντίστοιχη τιμή της `tid` εισάγεται στην κατάλληλη λίστα μετρητών. Η τιμή της μεταβλητής `tid` εκχωρείται αυτόματα από το RTCOSPAN όπως αναφέραμε στην ενότητα 5.1. Η σειρά με την οποία εξετάζονται οι συνθήκες είναι συνεπής με την συνάρτηση μετάβασης της `R_TIMER` στο σχήμα 5.1. Αυτές οι λίστες είναι διαθέσιμες στον κώδικα της διεργασίας `Timer_Region_Check`, ώστε δεν υπάρχει ανάγκη να υπολογιστεί η συνθήκη του γεγονότος ξανά. Η μόνη διαφορά με τον κώδικα του τύπου διεργασίας `R_CLOCK` είναι ότι στην τελευταία περίπτωση η συνθήκη στη γραμμή 24 του σχήματος 5.7 θα έπρεπε να αντικατασταθεί με «`if ( tstate = 2 or tstate = 3 ) then`».

Τώρα είναι καιρός να στρέψουμε την προσοχή μας στο τμήμα `C` κώδικα της διεργασίας `Timer_Region_Check` που παρουσιάζεται στο σχήμα 5.8. Κατά την αρχική φάση αυτή η διεργασία αποθηκεύει μια αρχική δομή `timer_state` χωρίς ενεργούς μετρητές στην αρχική της κατάσταση (μεταβλητή `trst`). Στην αρχή κάθε βήματος επιλογής ανακτάται η δομή `timer_state` που αντιστοιχεί στην παρούσα ολική κατάσταση του συστήματος. Οι πράξεις

```
case ( f ) in :
   BEG_INIT:
             if ( U < L ) then { Invalid bounds interval }
                     Error ;
                     abort ;
             endif
             ntimers <- ntimers + 1 ;
             break;
   CONT_INIT:
             LTLIM[tid] <- ( - L << 1 | lr ) ; { Lower_bound }
             UTLIM[C_tid] <- ( U << 1 | ur ) ; { Upper_bound }
             break;
   BEG_RESOLVE:
             if ( tstate = 0) then      { Error state }
                     truncate <- 1 ;
                     bad_timer_region <- true ;
                     break;
             endif
   CONT_RESOLVE:
             if ( (tstate = 1) and set_cond ) then  { Idle state }
                     insert(timers_to_set, tid) ;
             endif
             if ( tstate = 2 ) then                  { Active state }
                     if ( expire_cond ) then
                             insert(timers_to_expire, tid) ;
                             if ( set_cond ) then
                                insert(timers_to_set, tid) ;
                             endif
             else
                      if ( kill_cond ) then
                           insert(timers_to_kill, tid);
                      endif
             endif
endcase
```

Σχήμα 5.7: Ψευδοκώδικας ισοδύναμος με το τμήμα C κώδικα για τον τύπο διεργασίας E/A R_TIMER στο αρχείο δηλώσεων RT.h αποθήκευσης και επανάκτησης υλοποιούνται μέσω των χαμηλού επιπέδου μακροεντολών σε C `PACK()` και `unpkse()` του COSPAN, οι οποίες ορίζονται στις γραμμές 98 και 105 του αρχείου δηλώσεων `crank.h`.

Σε κάθε βήμα απόφασης πρώτα ελέγχουμε εαν η `Timer_Region_Check` είναι ήδη στη κατάσταση σφάλματος 0, οπότε τίθεται η `truncate` (γραμμή 176) και ο έλεγχος του προγράμματος μεταφέρεται στην γραμμή 199 όπου καθαρίζονται όλες οι λίστες των μετρητών ώστε να είναι έτοιμες να χρησιμοποιηθούν στο επόμενο βήμα απόφασης. Αλλιώς αν η μεταβλητή `bad_timer_region` έχει τεθεί προηγουμένως από κάποια άλλη διεργασία τύπου `R_TIMER`, η `trst` δείχνει σε μια δομή περιοχής μετρητών στον πίνακα κατακερματισμού και γίνονται τα ακόλουθα βήματα. Πρώτα ελέγχουμε αν κάποιες άλλες διεργασίες τύπου `R_TIMER` ή `R_CLOCK` θα έχουν την κατάσταση σφάλματος ως την επόμενη κατάσταση τους εξετάζοντας την μεταβλητή `bad_timer_region`, και αν υπάρχουν κάποια γεγονότα σχετικά με τους μετρητές εξετάζοντας τις λίστες `timers_to_kill`, `timers_to_set`, `timers_to_expire`. Πρώτα επεξεργαζόμαστε τα γεγονότα εκπνοής, καλώντας τη συνάρτηση `expire_timer()`. Αυτή η συνάρτηση περιέχεται στην βιβλιοθήκη του RTCOSPAN `libRT.a`, και η περιγραφή της σε ψευδοκώδικα περιέχεται στο παράρτημα B. Τα ορίσματα της είναι ένα αντίγραφο της τρέχουσας κατάστασης μετρητών και η λίστα των μετρητών που εκπνέουν, και η τιμή που επιστρέφει μπορεί να θέσει την `bad_timer_region` αν η νέα περιοχή μετρητών που προκύπτει είναι ασυνεπής. Η `kill_timers()` υλοποιεί τα γεγονότα ακύρωσης απλά σβύνοντας τα στοιχεία της `timers_to_kill` από τη λίστα `active_timers` της `tmp_state`. Αν μέχρι στιγμής δεν έχουν βρεθεί ασυνέπειες, η συνάρτηση `reset_timers` (η οποία επίσης περιγράφεται στο ευρετήριο B) ασχολείται με τους μετρητές που τίθενται τώρα. Στη συνέχεια εισάγουμε τη νέα περιοχή μετρητών `tmp_state` στον πίνακα κατακερματισμού μας με τον `t1` να δείχνει στη νέα περιοχή μετρητών, καθαρίζουμε τις λίστες μετρητών και αποθηκεύουμε τον `t1` σαν τη νέα κατάσταση της διεργασίας. Τελικά η `bad_timer_region` γίνεται ψευδής, και είμαστε έτοιμοι για το επόμενο βήμα επιλογής/απόφασης.

5.2.3 Γνωστά προβλήματα

[2] Η πλατφόρμα που απαιτήται για να τρέξει το RTCOSPAN είναι ακόμα αρκετά ειδική. Το RTCOSPAN εγγυημένα τρέχει σε μηχανές sun3 που τρέχουν λειτουργικό σύστημα SunOS Release 4.1.1 με την έκδοση του COSPAN 6.4.4 (4/2/90). Το RTCOSPAN θα πρέπει να τρέχει σε μηχανές sun4 με την έκδοση του COSPAN 6.4.4 (αν και δεν έχει δοκιμαστεί), αλλά παράγει λανθασμένα αποτελέσματα αν χρησιμοποιηθεί η έκδοση του COSPAN 7.1.

[2]Πολλά από τα παρακάτω προβλήματα αναιρούνται στην δεύτερη έκδοση του RTCOSPAN (βλέπε ευρετήριο D)

```
 case ( f ) in :
   BEG_INIT:
   CONT_INIT:
            store initial trst ;
            break;
   BEG_SELECT:
            retrieve current trst ;
            break;
   BEG_RESOLVE:
   CONT_RESOLVE:
    { Default if no timer events }
    t1 <- trst ;
    if ( trst = 0 ) then
         truncate <- 1 ;
         goto clean ;
    endif
    if ( not bad_timer_region and (timers_to_kill <> NIL or
    timers_to_set <> NIL or timers_to_expire <> NIL)) then
      tmp_state <- copy_timer_state( trst ) ;
      if ( timers_to_expire <> NIL ) then
       bad_timer_region <- expire_timer(tmp_state,timers_to_expire)
;
       if ( bad_timer_region ) then
                free( tmp_state ) ;
       endif
      endif
      if ( timers_to_kill <> NIL ) then
              kill_timers(tmp_state, timers_to_kill) ;
      endif
      if ( timers_to_set <> NIL ) then
              if ( not bad_timer_region ) then
                     reset_timers(tmp_state, timers_to_set) ;
              endif
      endif
      if ( not bad_timer_region ) then
        t1 = hash_install(tmp_state,hash_timer_state(tmp_state)) ;
      endif
    endif
clean:  { Cleanup all lists here }
            free_list( timers_to_set ) ;
            free_list( timers_to_expire ) ;
            free_list( timers_to_kill ) ;
    store t1 as next state ;
    bad_timer_region <- false ;
 endcase
```

Σχήμα 5.8: Ψευδοκώδικας ισοδύναμος με το τμήμα C κώδικα για την διεργασία E/A Timer_Region_Check στο αρχείο δηλώσεων trcheck.c

Ο `C` κώδικας που παράγει το COSPAN χρησιμοποιεί ένα σωρό εντολές `goto`, και κάνει διάφορες βελτιστοποιήσεις κώδικα. Αφού έχουμε υλοποιήσει όλο τον κώδικα που σχετίζεται με τους μετρητές και τον αλγόριθμο των περιοχών μετρητών εντελώς έξω από την ρουτίνα ανάλυσης του COSPAN (`analyze()`), εγγυώμαστε ότι ο κώδικας μας λειτουργεί σωστά μόνο όταν οι διεργασίες Ε/Α των μετρητών και των μετρητών με επανάθεση είναι οι πρώτες διεργασίες Ε/Α στην περιγραφή.

Ενα τμήμα του στοιχειώδους μεταφραστή μας (και συγκεκριμένα το αρχείο εντολών `awk` `RT.awk`) δεν χειρίζεται σωστά δηλώσεις μετρητών μέσα σε κατασκευές `#ifdef`. Το λάθος είναι ότι η παράμετρος `tid` θα αυξηθεί λανθασμένα σε αυτή την περίπτωση. Μια σωστή λύση σε αυτό το πρόβλημα θα ήταν η κατασκευή ενός μεταφραστή με `yacc`. Μια προσωρινή λύση είναι η χρήση κατασκευών της παρακάτω μορφής.

```
proc
#ifdef  DELAYS
DELIV_TIME
#endif  DELAYS
#ifdef  DUPLICATE
ZERORECDELAY
#endif  DUPLICATE
:R_TIMER(
#ifdef  DELAYS
 \[,1,DEL_TIME,\], (S.# = 2)*(DELIVERY_TIME.$ = 1),
                       (R.# = 2)*(DELIVERY_TIME.$ = 2), 0)
#endif  DELAYS
#ifdef  DUPLICATE
 \[,0,0,\], M.# > 0, RB.# ~= 0, 0 )
#endif  DUPLICATE
```

Κεφάλαιο 6

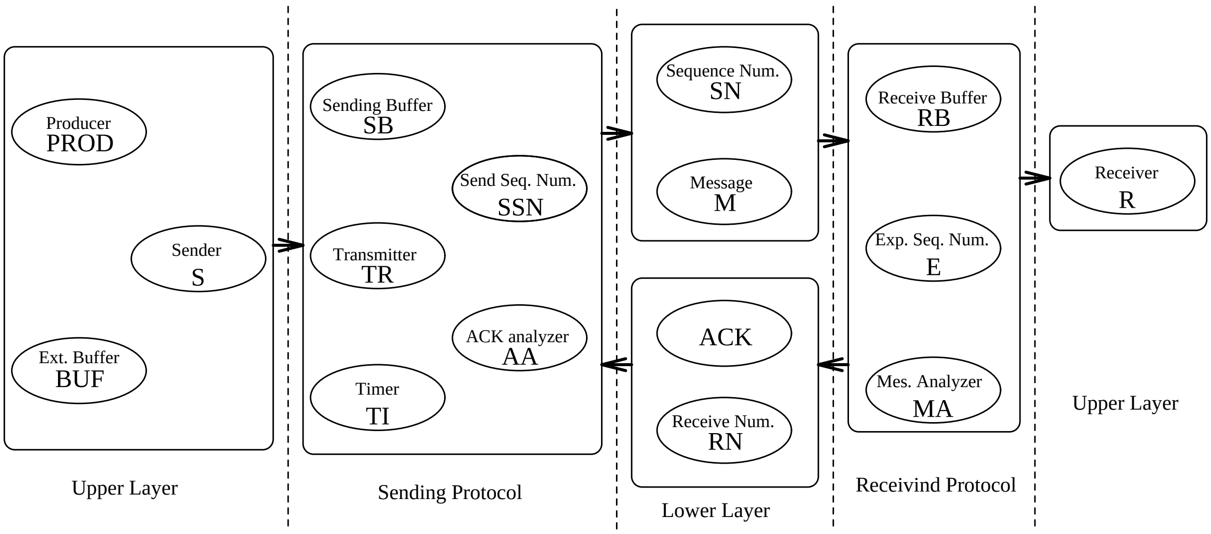
Παραδείγματα

Σε αυτό το κεφάλαιο δείχνουμε πως να χρησιμοποιηθεί η μέθοδος μας για την περιγραφή και επαλήθευση δύο συστημάτων. Το πρώτο παράδειγμα είναι μια επέκταση του γνωστού πρωτοκόλλου «εναλλασόμενου–ψηφίου» (alternating–bit) για επικοινωνία, η οποία περιέχει πληροφορία πραγματικού χρόνου, και το δεύτερο παράδειγμα επεκτείνει το κανάλι με καθυστέρηση της ενότητας 2.3.1. Για την ευκολία του αναγνώστη οι περιγραφές μας θα δωθούν υπό μορφή γράφων αντί για τη γλώσσα περιγραφής του RTCOSPAN, μια και όλη η προηγούμενη συζήτηση καθιστά την μετάφραση απλή.

6.1 Το πρωτόκολλο εναλλασόμενου–ψηφίου

Το μοντέλο μας για το πρωτόκολλο επικοινωνίας εναλλασόμενου–ψηφίου (alternating–bit protocol ή ABP) ουσιαστικά βασίζεται σε εκείνο που παρουσιάζεται στο [ACW90]. Για λόγους πληρότητας περιγράφουμε σύντομα το πρωτόκολλο. Αν ο εκπομπός επιθυμεί να στείλει ένα μήνυμα, απλά το παράγει και το πρωτόκολλο μεταφοράς το παίρνει. Το μήνυμα αποστέλεται μαζί με ένα δυαδικό ψηφίο 0 ή 1 και τίθεται ένας μετρητής. Μετά, το πρωτόκολλο αποστολής περιμένει για μια επιβεβαίωση (acknowledgment) ότι το μήνυμα παραλήφθηκε σωστά. Αν έρθει μια σωστή επιβεβαίωση, στέλνει το επόμενο μήνυμα. Τυχόν λανθασμένες επιβεβαιώσεις αγνοούνται. Αν ο μετρητής εκπνεύσει, το ίδιο μήνυμα ξαναστέλνεται. Το πρωτόκολλο λήψης επιβεβαιώνει το ψηφίο του τελευταίου μηνύματος που έλαβε σωστά. Ενα μοναδικό αντίτυπο παραδίδεται στον δέκτη και οποιαδήποτε διπλά μηνύματα πετιούνται. Χρησιμοποιούμε τη σημασιολογία κατά την οποία ο χρόνος μεταξύ διαδοχικών μεταβάσεων μπορεί να είναι μηδενικός (Θέτωντας τον κατάλληλο λογικό μετρητή AT).

Αρκετές παραλλαγές αυτού του μοντέλου χρησιμοποιήθηκαν για να ελεγχθούν διάφορες ιδιότητες. Η δομή του πρωτοκόλλου σε υψηλό επίπεδο παρουσιάζεται στο σχήμα 6.1. Η σχεδίαση του πρωτοκόλλου εγγυάται παραλλαβή μηνυμάτων από τον πομπό στο δέκτη με



Σχήμα 6.1: Δομή σε υψηλό επίπεδο του πρωτοκόλλου επικοινωνίας εναλλασόμενου–ψηφίου.

τη σωστή σειρά, μέσα από κανάλια επικοινωνίας τα οποία μπορεί να χάσουν μερικά από τα μηνύματα που μεταφέρουν. Τα άνω στρώματα αυτού του πρωτοκόλλου μοντελοποιούνται από δυο διεργασίες που ονομάζονται S και R, και παράγουν και λαμβάνουν μηνύματα αντίστοιχα. Σε μια από τις παραλλαγές ένας εξωτερικός παραγωγός μηνυμάτων PROD και ένας εξωτερικός ενταμιευτής BUF έχουν συμπεριληφθεί στο άνω στρώμα του πρωτοκόλλου αποστολής. Δύο ομάδες διεργασιών μοντελοποιούν τα τμήματα αποστολής και λήψης του πρωτοκόλλου σε επίπεδο μεταφοράς. Το τμήμα μεταφοράς περιέχει ένα ενταμιευτή μηνυμάτων SB, ένα εκπομπό TR, ένα μετρητή επανεκπομπής TI, μια διεγασία SSN που παρακολουθεί τον αριθμό ακολουθίας που στέλνεται μαζί με το επόμενο μήνυμα και τελικά μια διεργασία αναλυτή επιβεβαίωσης AA. Το τμήμα λήψης αποτελείται από ένα ενταμιευτή λήψης RB, τη διεργασία E που παρακολουθεί τον αριθμό ακολουθίας που αναμένεται με το επόμενο μήνυμα, kai τον αναλυτή μηνυμάτων MA. Τελικά τα κάτω επίπεδα αποτελούνται από δύο μονόδρομα (half–duplex) κανάλια επικοινωνίας τα οποία μεταφέρουν (και μπορεί και να χάνουν) μηνύματα και επιβεβαιώσεις αντίστοιχα. Κάθε κανάλι μοντελοποιήται από δυο διεργασίες. Το κανάλι που μεταφέρει μηνύματα (ας το ονομάσουμε το «εξερχόμενο κανάλι») αποτελείται από μια διεργασία μήνυμα M και ένα αριθμό ακολουθίας SN. Οι επιβεβαιώσεις επιστρέφουν μέσα από ένα «εισερχόμενο κανάλι» το οποίο μοντελοποιήται από μια διεργασία επιβεβαίωση ACK και ένα λαμβανόμενο αριθμό ακολουθίας RN. Μια πλήρης περιγραφή σε RTCOSPAN παρουσιάζεται στο ευρετήριο C.

Εχουμε εισάγει αρκετούς μετρητές για να προσθέσουμε χρονικούς περιορισμούς. Αυτοί περιλαμβάνουν:

1. Τον μετρητή με επανάθεση TICLK. Αυτός ο λογικός μετρητής χρησιμοποιήται για

να παρέχει την πληροφορία πραγματικού χρόνου στον μετρητή επανεκπομπής TI του πρωτοκόλλου. Η συνθήκη θέσης του είναι η ίδια με τη συνθήκη θέσης του TI, και η συνθήκη εκπνοής του είναι η εκπνοή του TI, δηλ. (TI.# = to). Η περιγραφή του διαστήματος του είναι $[TIME, TIME]$, που υπονοεί ότι η εκπνοή συμβαίνει ακριβώς $TIME$ μονάδες χρόνου μετά το γεγονός θέσης. Τελικά, αν ληφθεί η σωστή επιβεβαίωση πριν ο TI εκπνεύσει (και ως εκ τούτου ο TICLK εκπνεύσει), τότε ο TICLK ακυρώνεται.

2. Τον λογικό μετρητή CHO_TIMER. Αυτός ο λογικός μετρητής μοντελοποιεί την καθυστέρηση μετάδοσης του εξερχόμενου καναλιού όπου στέλνονται τα μηνύματα. Ο CHO_TIMER τίθεται όποτε ένα μήνυμα παραδίδεται στο εξερχόμενο κανάλι και εκπνέει όποτε το εξερχόμενο κανάλι παραδίδει ή χάνει το μήνυμα.

3. Τον λογικό μετρητή CHI_TIMER. Αυτός ο λογικός μετρητής μοντελοποιεί την καθυστέρηση μετάδοσης του εισερχόμενου καναλιού όπου στέλνονται οι επιβεβαιώσεις. Τίθεται όποτε μια επιβεβαίωση παραδίδεται στο εισερχόμενο κανάλι και εκπνέει όποτε η επιβεβαίωση παραδίδεται ή χάνεται.

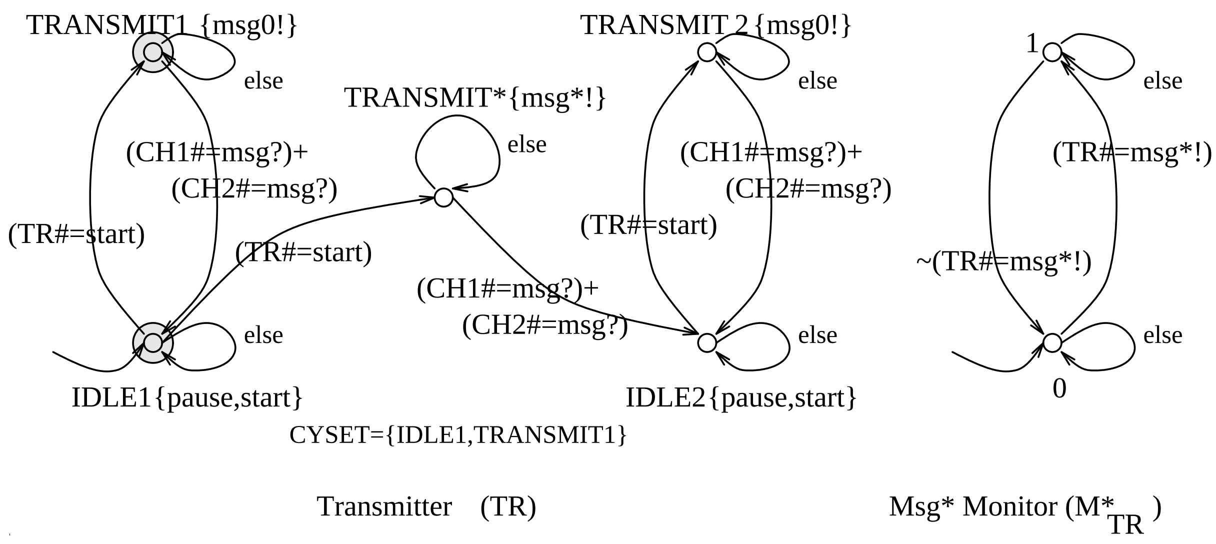
Αυτοί οι τρεις μετρητές αρκούν για να αποδειχτεί η ορθότητα του πρωτοκόλλου, δηλ. η παράδοση των μηνυμάτων με τη σωστή σειρά. Το συμπλήρωμα καθήκοντος που χρησιμοποιείται σε αυτή την περίπτωση είναι το TC όπως είναι στο [ACW90]. Ομως για να αποδείξουμε ότι ένα μήνυμα παραδίδεται μέσα σε καθορισμένα χρονικά όρια, πρέπει να χρησιμοποιήσουμε άλλο ένα λογικό μετρητή DELIV_TIME, και ένα νέο ελεγκτή συμπληρώματος καθήκοντος DELIVERY_TIME. Αυτός ο μετρητής τίθεται όταν ο πομπός αποφασίζει να στείλει ένα σημαδεμένο μήνυμα, και εκπνέει όταν το μήνυμα φτάνει στον δέκτη. Αλλη μια ιδιότητα για έλεγχο είναι αν στέλνονται ποτέ διπλά αντίτυπα ενός μηνύματος δεδομένου ότι το εισερχόμενο κανάλι δεν χάνει επιβεβαιώσεις. Εχουμε δείξει ότι η παρουσία διπλών αντίτυπων εξαρτάται από τη σχέση της τιμής εκπνοής TIME με το άθροισμα των καθυστερήσεων των καναλιών επικοινωνίας. Για να το αποδείξουμε αυτό προσθέσαμε ένα νεό λογικό μετρητή ZERORECDELAY ο οποίος κάνει σίγουρο ότι οι διεργασίες του πρωτοκόλλου λήψης επεξεργάζονται τα μηνύματα σε μηδενικό χρόνο, και χρησιμοποιήσαμε τον ελεγκτή DUPLICATES ως συμπλήρωμα καθήκοντος. Αυτός ο ελεγκτής μεταβαίνει σε μια κατάσταση σφάλματος όταν ένα αντίτυπο μηνύματος φτάνει στον πρωτόκολλο λήψης.

Επειτα χρησιμοποιήσαμε μια επέκταση του ABP που συμπεριλαμβάνει τα πάνω επίπεδα του πρωτοκόλλου, για να μελετήσουμε την πληρότητα του ενταμιευτή στη διασύνδεση πάνω από το ABP. Προσθέσαμε ένα εξωτερικό παραγωγό μηνυμάτων PROD, ένα εξωτερικό ενταμιευτή μηνυμάτων BUF, και αλλάξαμε τον πομπό S να στέλνει μηνύματα που παράγει ο PROD (αυτή είναι η έκδοση της διεργασίας πομπού στο ευρετήριο C). Σε αυτό το μοντέλο ένα μήνυμα προερχόμενο από τον PROD είτε αποθηκεύεται προσωρινά στο BUF αν τα κανάλια επικοινωνίας είναι απασχολημένα, είτε παραδίδεται κατευθείαν στο πρωτόκολλο αποστολής χρησιμοποιώντας ένα μηχανισμό παρεμβολής (cut-through). Ενας λογικός μετρητής PROD_TIMER ελέγχει το ρυθμό με τον οποίο ο PROD παράγει μηνύματα και ένα νέο συμπλήρωμα καθήκοντος BUFTHRESHOLD ελέγχει αν η πληρότητα του ενταμιευτή ξεπερνάει ποτέ ένα συγκεκριμένο όριο κάνοντας την υπόθεση ότι δεν χάνονται μηνύματα ή επιβεβαιώσεις (αυτή είναι η έκδοση των καναλιών επικοινωνίας στο ευρετήριο C). Φανερά, αυτό εξαρτάται από το ρυθμό με τον οποίο η διεργασία παραγωγός δημιουργεί μηνύματα και από τα χαρακτηριστικά καθυστέρησης των καναλιών. Εχουμε ελέγξει την πληρότητα του ενταμιευτή για ένα μεγάλο εύρος παραμέτρων του συστήματος.

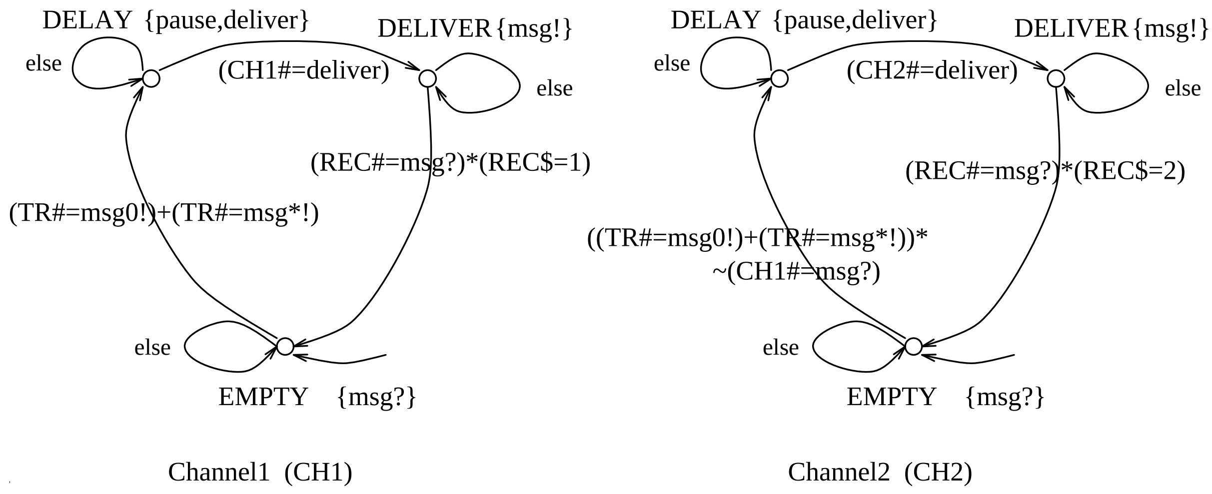
6.2 Δυο παράλληλα κανάλια με καθυστέρηση

Αυτό το παράδειγμα επεκτείνει το κανάλι με καθυστέρηση στο σχήμα 2.2. Το σύστημα αποτελείται από ένα εκπομπό (TR), δυο παράλληλα κανάλια με καθυστέρηση ($CH1, CH2$), και ένα δέκτη (REC). Η περιγραφή με χρόνο του εκπομπού είναι παρόμοια με αυτή της προηγούμενης ενότητας. Οταν ο εκπομπός είναι έτοιμος να εκπέμψει, παραδίδει το μήνυμα στο πρώτο διαθέσιμο κανάλι. Αν και τα δυο κανάλια είναι διαθέσιμα, τότε το μήνυμα στέλνεται στο κανάλι $CH1$ (προτεραιότητα του $CH1$ πάνω στο $CH2$). Ο δέκτης ρωτάει τα κανάλια με ένα κυκλικό τρόπο (round-robin polling), και υπάρχει περιορισμός στο χρόνο ανάμεσα στις διαδοχικές ερωτήσεις. Κάθε κανάλι είναι παρόμοιο με το κανάλι με καθυστέρηση στην ενότητα 2.3.1, και οι περιορισμοί στην καθυστέρηση των καναλιών είναι γενικά διαφορετικοί. Θα θέλαμε να επαληθεύσουμε ιδιότητες για την καθυστέρηση από ακρό σε άκρο των μηνυμάτων, για όλα τα μηνύματα στο σύστημα. Για παράδειγμα υποθέστε ότι θέλουμε να επιβεβαιώσουμε ότι για όλα τα μηνύματα που στέλνονται, ο χρόνος μεταξύ της στιγμής που το μήνυμα είναι έτοιμο για μετάδοση από τον εκπομπό και της στιγμής που αυτό το μήνυμα παραλαμβάνεται από τον δέκτη, είναι μικρότερος (μεγαλύτερος) από κάποια προκαθορισμένη σταθερά.

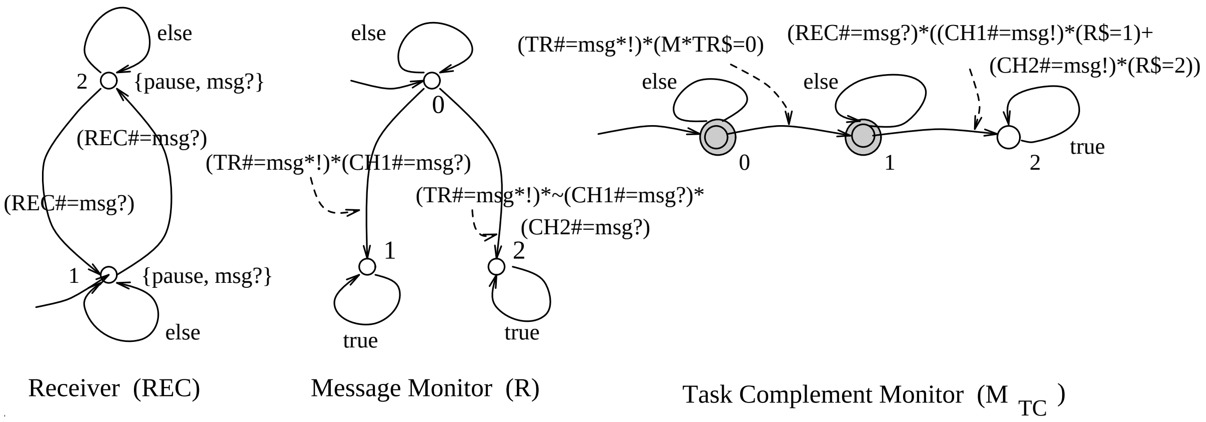
Για να επαληθεύσουμε μια τέτοια ιδιότητα χρησιμοποιούμε ένα τέχνασμα που δανειζόμαστε από τις τεχνικές που παρουσιάζονται στο [ACW90]. Μια και χρειάζεται να αποδείξουμε μια ιδιότητα που σχετίζεται με όλα τα μηνύματα, είναι αρκετό να αποδείξουμε αυτή την ιδιότητα για ένα συγκεκριμένο μήνυμα που διαλέγουμε τυχαία. Για το σκοπό αυτό συμβολίζουμε όλα τα μηνύματα με $msg0$ εκτός από ένα ειδικό μήνυμα που συμβολίζουμε με $msg*$. Ο εκπομπός αποφασίζει μη αιτιοκρατικά πότε να στείλει $msg*$ (Το $msg*$ πάντα στέλνεται κάποτε), και το καθήκον μας ορίζεται μέσω των ιδιοτήτων της καθυστέρησης από άκρη σε



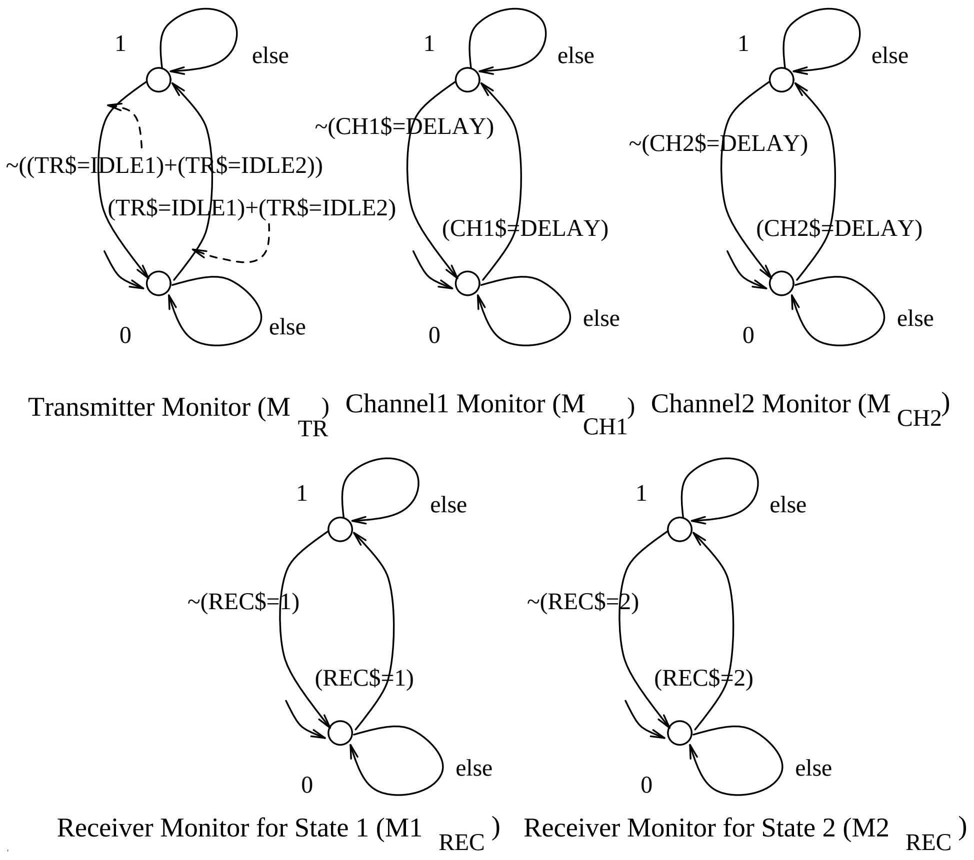
Σχήμα 6.2: Η διεργασία εκπομπός. Διαλέγει μη αιτιοκρατικά να στείλει το ειδικό μήνυμα msg* (όλα τα άλλα μηνύματα αντιστοιχούν σε msg).



Σχήμα 6.3: Τα κανάλια. Το κανάλι 1 έχει προτεραιότητα έναντι του καναλιού 2, και ένα κανάλι παραδίδει ένα μήνυμα μόνο όταν ερωτάται.



Σχήμα 6.4: Ο δέκτης και το συμπλήρωμα καθήκοντος. Ο δέκτης ρωτάει τα δυο κανάλια με κυκλικό τρόπο. Ο ελεγκτής μηνύματος αποθηκεύει το όνομα του καναλιού που έλαβε το msg*.



Σχήμα 6.5: Διεργασία ελεγκτής για την περιγραφή με χρόνο στο παράδειγμα 2.

άκρη του $msg*$. Δηλαδή αν αυτό το καθήκον ικανοποιήται, αυτές οι ιδιότητες πρέπει να ισχύουν για κάθε μήνυμα. Η περιγραφή του εκπομπού παρουσιάζεται στο σχήμα 6.2, και τα κανάλια με καθυστέρηση παρουσιάζονται στο σχήμα 6.3.

Το σχήμα 6.4 περιλαμβάνει τον δέκτη και το καθήκον. Οταν ο δέκτης είναι στην κατάσταση i ρωτάει το κανάλι i. Υπάρχει μια συμπληρωματική διεργασία (η διεργασία ελεγκτής μηνύματος R) την οποία εισάγουμε για να κωδικοποιήσουμε το όνομα του καναλιού που μεταφέρει το $msg*$. Ετσι $R\$ = i$ συμβολίζει ότι το $msg*$ είναι τώρα στο κανάλι i. Ο ελεγκτής του συμπληρώματος καθήκοντος σημειώνει τα δυο ενδιαφέροντα γεγονότα, δηλαδή την παρουσία του $msg*$ στον εκπομπό, και την λήψη του $msg*$ από τον δέκτη.

Για να περιγράψουμε τους λογικούς μετρητές που περιγράφουν τους χρονικούς περιορισμούς προσθέτουμε τους ελεγκτές στο σχήμα 6.5 και το σχήμα 6.2. Ο ελεγκτής M^*_{TR} χρειάζεται για να καταγράψει την πρώτη φορά που το $msg*$ είναι διαθέσιμο στον εκπομπό. Οι μετρητές ορίζονται με παρόμοιο τρόπο όπως και πρίν. Οι μετρητές T_{TR}, T_{CH1}, T_{CH2} είναι παρόμοιοι με τους μετρητές στο παράδειγμα της προηγούμενης ενότητας, και πρέπει να προσθέσουμε ένα επιπλέον μετρητή για τον δέκτη·κάθε μετρητής $T_{i,REC}$ μετρά το χρόνο που περνά η REC στην κατάσταση i χρησιμοποιώντας τον ελεγκτή $M_{i,REC}$. Σημειώστε ότι ο δέκτης μπορεί να αλλάξει κατάσταση μόνο ρωτώντας το αντίστοιχο κανάλι. Ο μετρητής T_{TC} που συνδέεται με το συμπλήρωμα καθήκοντος έχει συνθήκη θέσης $(TR\# = msg{*}!) * (M^*_{TR}\$ = 0)$ και συνθήκη εκπνοής $(REC\# = msg?) * ((CH1\# = msg!) * (R\$ = 1) + (CH2\# = msg!) * (R\$ = 2)) * (M_{TC}\$ = 1)$.

Οταν το μεγαλύτερο άκρο του διαστήματος μετρητή σε αυτό το παράδειγμα δεν ξεπέρασε το 10 το εργαλείο μας εκτέλεσε το βήμα της επαλήθευσης σε 596 δευτερόλεπτα σε ένα σταθμό εργασίας `SUN 3/60` με 8 Mbytes RAM, 15587 καταστάσεις ερευνήθηκαν, 80882 ακμές διασχίστηκαν, και περίπου 10 Mbytes μνήμη χρησιμοποιήθηκαν.

Κεφάλαιο 7

Συμπεράσματα

Δείξαμε ότι το COSPAN μπορεί να επεκταθεί ώστε να χειρίζεται περιορισμούς καθυστερήσεων με φυσικό τρόπο, και πως το σύστημα που προέκυψε μπορεί να επαληθεύσει ενδιαφέροντα παραδείγματα. Η προσέγγιση μας είναι ευέλικτη ώστε να χειρίζεται διαφορετικές σημασιολογίες συνεχούς χρόνου όπως όταν ο χρόνος που περνά ανάμεσα στις μεταβάσεις του συστήματος μπορεί ή πρέπει να είναι αυστηρά θετικός. Η βασική προσέγγιση του διαχωρισμού του χρονικά ανεξάρτητου από το χρονικά εξαρτημένο τμήμα της περιγραφής και η χρήση της κατασκευής των περιοχών μετρητών μπορεί πιθανώς να εφαρμοστεί και σε άλλες μεθόδους επαλήθευσης με την ίδια επιτυχία. Για παράδειγμα, μια παρόμοια ιδέα έχει χρησιμοποιηθεί για την ανάλυση δικτύων Petri με χρόνο [BM83].

Η μεγαλύτερη πρόκληση που απομένει είναι το πρόβλημα της εκτίναξης του αριθμού των καταστάσεων: Ακόμα και σε συστήματα χωρίς χρόνο, ο χώρος καταστάσεων των προβλημάτων μπορεί να μεγαλώσει πολύ γρήγορα στο μέγεθος της περιγραφής του προβλήματος. Οταν λαμβάνονται υπόψην εξαρτήσεις από το χρόνο, το προβλημα επιδεινώνεται·στη χειρότερη περίπτωση, ο αριθμός των περιοχών στη διεργασία ελεγκτή που εξαρτάται από το χρόνο μπορεί να είναι $O(c^{n^2})$, όπου c είναι το μέγιστο εύρος όλων των πεπερασμένων διαστημάτων που εμφανίζονται στους χρονικούς περιορισμούς και n είναι ο αριθμός των περιορισμών. Επιτύχαμε στο να αποφύγουμε την κατασκευή άχρηστων περιοχών δημιουργώντας τις περιοχές δυναμικά. Ομως περισσότερα μέτρα πρέπει να παρθούν για το χειρισμό μεγαλύτερων προβλημάτων, για η χρησιμοποίηση μόνο εκείνων των χρονικών περιορισμών που είναι απαραίτητοι για να επαληθευτεί ένα συγκεκριμένο σύστημα και περιγραφή, και να αγνοούμε άσχετους περιορισμούς που μπορεί να αυξήσουν τον αριθμό των περιοχών. Αυτό θα απαιτούσε συνδιασμό με τις ιδέες από το [AIKY92].

Για την ώρα βελτιώνουμε την υλοποίηση μας ώστε να επιτρέπουμε στο χρήστη να περιγράφει κατευθείαν σε COSPAN συνθήκες της μορφής «το κατηγόρημα Q είναι αληθές στο παρών βήμα και ήταν ψευδές στο προηγούμενο βήμα» στον ορισμό των μετρητών (π.χ.

ξεχώριοε την πρώτη φορά που ισχύει το Q). Κάτι τέτοιο Θα απλοποιήσει την πολυπλοκότητα της περιγραφής μια και στο βασικό μοντέλο χρειάζεται να εισαχθεί ένας ελεγκτής για να λάβει υπόψην μια τέτοια ιδιότητα που έχει να κάνει με την πρώτη φορά που γίνεται κάτι. Επίσης πειραματιζόμαστε με τη μοντελοποίηση και την επαλήθευση μεγαλύτερων συστημάτων όπως οι αλγόριθμοι του FDDI, και την απόδειξη φραγμάτων στις καθυστερήσεις σε ένα ολοκληρωμένο κύκλωμα διακόπτη για ATM δίκτυα.

Παράρτημα A

Αποσπάσματα από το αρχείο δηλώσεων crank.h

Αυτό το ευρετήριο περιέχει συγκεκριμένα τμήματα του αρχείου δηλώσεων `crank.h` τα οποία χρειάστηκαν αλλαγές που απαιτούσε η υλοποίηση του RTCOSPAN. Για την διευκόλυνση του αναγνώστη μπροστά από κάθε γραμμή υπάρχει ο αριθμός της και οι νέες γραμμές περιέχουν το σχόλιο `/* rtsr */`.

```
  1  /*@(#)crank.h   6.4 (AT&T-BL)   3/3/90  13:54:17*/
.......
 89  #include"types.h"       /* rtsr */
 90  /*
 91  * bit manipulation routines
 92  */
.......
 96  #define mask(size)((1<<(size))-1)/*works for size<BITS only*/
 97  #define gmask(size)(size>=BITS?BIG:mask(size))
 98  #define PACK(x,size){*wd= *wd&~(gmask(size)<<bit)|((unsigned)(x)
&gmask(size))<<bit;\
 99  if((bit+=(size))>=BITS&&(++wd,bit-=BITS))*wd= *wd&~mask(bit)|\
100  ((unsigned)(x)>>(size)-bit)&mask(bit);}
101  /* unpack without sign extension */
102  #define unpk(size)(acc= (unsigned)*wd>>bit&gmask(size),(bit+=(si
ze))>=BITS&&\
103 (++wd,bit-=BITS)?acc|((*wd&mask(bit))<<(size)-bit):acc)
104  /* unpack with sign extension */
```

```
#define unpkse(size)((acc=unpk(size))&1<<(size)-1?acc|~gmask(size):acc)
/*
* crank header
*/
#ifdef PAUSE
# define CLEARC c1=0;c2=0;c3=0;
#else
# define CLEARC
#endif
#if defined(HOMVERIFY)&&defined(PAUSE)
# define CLEARH c1h=0;c2h=0;c1f=0;c2f=0;
#else
# define CLEARH
#endif
#ifdef MERGING
# define LINEDEF char *Cline;
# define SEQDEF int seqbit;
# define LINESET Cline=mtrans;
# define SEQSET seqbit=0;
#else
# define LINEDEF
# define SEQDEF
# define LINESET
# define SEQSET
#endif
#define ERROR(loc,err) {errloc=loc;errline=__LINE__;return err;}

#define BEGIN \
int *oldstate,newstate[STATESIZE],statesize=STATESIZE;\
/* rtsr */ int ntimers=0;\
/* rtsr */ int badtr=0;\
/* rtsr */ list *timers_to_set, *timers_to_expire, *timers_to_kill;\
/* rtsr */ extern int *UTLIM, *LTLIM;\
/* rtsr */ extern timer_state *null_timer_state, *hash_install(),\
/* rtsr */ *reset_timers(), *expire_timer(), *copy_timer_state();\
```

```
int c1, c2, c3, c1h, c2h, c1f, c2f, plus, truncate, errloc=(-1), errline;\
char *errfile=__FILE__;\
Action f;\
Retcode \
crank(){\
        LINEDEF SEQDEF\
        extern Cyset cyset[];\
        static Cyset *cy=cyset;\
        static skip,more,loop,eof;\
        register bit,*wd,acc;\
        int i;\
        if(eof){\
                eof=0;\
                return NO_MORE;\
        }\
begin:\
        switch(f){\
        case BEG_RESOLVE:\
                plus=0;truncate=0;CLEARC CLEARH LINESET\
        case CONT_RESOLVE:\
                SEQSET\
        case BEG_INIT:\
        case CONT_INIT:\
                bit=0;wd=newstate;break;\
        case BEG_SELECT:\
                bit=0;wd=oldstate;break;\
        }
/*
* crank trailer
*/
#ifdef PAUSE
# ifdef HOMVERIFY
#  define CHECKC c1&&c2||c3&&!c2||c1f&&(c2f||c2h)
# else
#  define CHECKC c1&&c2||c3&&!c2
```

```
# endif
#else
# define CHECKC 0
#endif
#ifdef BIGM
# define END000 break
# define END001
#else
# define END000 goto begin
# define END001 }
#endif

#include        "trcheck.c"

#define END \
end:\
        Adjustment_Timer_Code\
        Timer_Region_Check_Code\
        switch(f){\
        case BEG_INIT:\
                f=CONT_INIT;\
                return NO_ERR;\
        case CONT_INIT:\
                if(!skip){\
                        f=BEG_SELECT;\
                        return NO_MORE;\
                }\
                skip=0;\
                return NO_ERR;\
        case BEG_SELECT:\
        case CONT_SELECT:\
                f=BEG_RESOLVE;\
                skip=0;\
                END000;\
        case BEG_RESOLVE:\
        case CONT_RESOLVE:\
```

```
                if(loop)f=CONT_RESOLVE;\
                else if(more)f=CONT_SELECT;\
                else {\
                        f=BEG_SELECT;\
                        if(CHECKC)return NO_MORE;\
                        eof=1;\
                }\
                if(CHECKC)goto begin;\
                return NO_ERR;\
        }\
END001
.......
```

Παράρτημα Β

Υλοποίηση των πράξεων σε περιοχές μετρητών

Αυτό το ευρετήριο παρέχει την υλοποίηση των τριών κύριων πράξεων πάνω σε περιοχές μετρητών (ονομαστικά *expire*, *reset*, *canonicalize*) οι οποίες περιέχονται στην βιβλιοθήκη `libRT.a`.

```
expire_timer (ts, E)
ts : timer_state ;
E : list ;
begin
     i, j : integer;
     al, el1, el2 : list;
                { rte(D, B) }
 for ( el1 <- E ; el1 <> NIL ; el1 <- el1.next ) do
        i <- el1.timer_id ;
        for ( el2 <- E ; el2 <> NIL ; el2 <- el2.next ) do
                j <- el2.timer_id ;
                ts.D[i][j] <- w_plus( ts.D[i][j], bound(0, LESSEQU) );
        endfor
        for( al <- ts.active_timers ; al <> NIL ; al <- al.next ) do
                j <- al.timer_id;
                if( member( E, j) ) then
                        continue ;
                endif
                ts.D[i][j] <- w_plus( ts.D[i][j], bound(0, LESSEQU) );
```

```
        endfor
 endfor
 if ( canonicalize( ts ) ) then
        return( -1 );
 endif
                { elapse(D, B) }
 for ( al <- ts.active_timers ; al <> NIL ; al <- al.next ) do
        i <- al.timer_id ;
        if ( member( E, i) ) then
                continue;
        endif
        for ( el2 <- E ; el2 <> NIL ; el2 <- el2.next ) do
                j <- el2.timer_id ;
                ts.D[i][0] <- ts.D[i][j] ;
                ts.D[0][i] <- ts.D[j][i] ;
        endfor
 endfor
                 /* delete expired timers from active timer list */
 for ( el1 <- E ; el1 <> NIL ; el1 <- el1.next ) do
        if ( delete( ts.active_timers, el1.timer_id ) ) then
                return( -1 );
        endif
 endfor
 return( 0 ) ;
end

reset_timers (ts, S)
ts : timer_state ;
S : list ;
{
i, j : integer ;
sl, al, prev : list ;
  { insert elements of S in active timers list, while keeping it sorted }
 for ( sl <- S ; sl <> NIL ; sl <- sl.next ) do
        if ( member( ts.active_timers, sl.timer_id ) ) then
                continue ;                      { Handle clock reset }
```

```
        endif
        if ( ts.active_timers = NIL ) then { First time }
                insert( ts.active_timers, sl.timer_id ) ;
                continue ;
        endif
        prev <- al <- ts.active_timers ;
        while ( (al <> NIL) and ( al.timer_id < sl.timer_id ) ) do
                prev <- al ;
                al <- al.next ;
        endwhile
        if ( prev = al ) then
                insert( ts.active_timers), sl.timer_id ) ;
        else
                prev.next <- new_list_element() ;
                prev.next.timer_id <- sl.timer_id ;
                prev.next.next <- al ;
        endif
 endfor
    { initialize l, u vectors }
 for ( sl <- S ; sl <> NIL ; sl <- sl.next ) do
        i <- sl.timer_id ;
        ts.D[0][i] <- LTLIM[i] ;
        ts.D[i][0] <- UTLIM[i] ;
 endfor
    { init d_{ij} where i or j reset. Same as appling canonicalize }
 for ( sl <- S ; sl <> NIL ; sl <- sl.next ) do
        i <- sl.timer_id ;
        for ( al <- ts.active_timers ; al <> NIL ; al <- al.next ) do
                j <- al.timer_id ;
                ts.D[i][j] <- w_star( ts.D[i][0], ts.D[0][j] ) ;
                ts.D[j][i] <- w_star( ts.D[j][0], ts.D[0][i] ) ;
        endfor
 endfor
end

canonicalize (ts)  { Compute shortest paths in ts.D. Destructive. }
```

```
 ts : timer_state ;
begin
i,j,k : integer ;
tl1, tl2, tl3 : list ;
 null_timer.next <- ts.active_timers ;
 for ( tl1 <- null_timer ; tl1 <> NIL ; tl1 <- tl1.next ) do
        i <- tl1.timer_id ;
        ts.D[i][i] <- 2;        { store 1's on diagonal }
 endfor
    { shortest paths algorithm }
 for ( tl1 <- null_timer ; tl1 <> NIL ; tl1 <- tl1.next ) do
    k <- tl1.timer_id ;
    for ( tl2 <- null_timer ; tl2 <> NIL ; tl2 <- tl2.next ) do
       i <- tl2->timer_id ;
       for ( tl3 <- null_timer ; tl3 <> NIL ; tl3 <- tl3.next ) do
          j <- tl3.timer_id ;
          ts.D[i][j] <- w_plus( ts.D[i][j], w_star( ts.D[i][k],
          ts.D[k][j]));
          if ( (i = j ) and ( w_le( ts.D[i][i], bound(0, LESS))) ) then
                  return( -1 );
          endif
       endfor
    endfor
 endfor
 return( 0 ) ;
end
```

Παράρτημα C

Μια περιγραφή σε COSPAN για το πρωτόκολλο ABP

Αυτή είναι μια περιγραφή σε COSPAN του πρωτόκολλου ABP με ένα παραγωγό μηνυμάτων και έναν εξωτερικό ενταμιευτή μηνυμάτων επιπλέον.

```
/*********************************************************************/
/*             General      Definitions                             */
/*********************************************************************/
macro        N        := 3
macro        BUFSIZE        := 5
macro        TIME        := 50
#ifdef        OVERFLOW
macro        PITIME        := 1
#else
macro        PITIME        := 41
#endif        OVERFLOW

        type        message : (0..N)
        type        bit        : (0..1)

#include<RT.h>

/*********************************************************************/
/* REAL TIME INFORMATION: SPECIFICATION OF LOGICAL TIMERS           */
/*********************************************************************/
```

```
/*
   Resetable logical timer of transmission protocol. Set when transmitter
   puts a message to the outgoing channel. Expires when actual ABP timer
   times out. Canceled when the message has been delivered.
 */
proc TICLK:R_CLOCK( \[,TIME, TIME , \],
      (TR.# = sent)*(TI.# = off)+(TI.$ = TO)*(AA.# = nok)*(TR.# = sent),
      (AA.# = nok) * (TI.# = to) * (TI.$ = ON), AA.#=ok )
/*
    Logical timer of outgoing channel. Set when channel is handed a
    message by the transmitter. Expires when channel delivers or
    looses the message. Channel transmission delay is chosen to be
    between 19 and 20 time units.
 */
proc CHO_TIMER:R_TIMER( \[,19,20,\], ( TR.# = sent ), M.# > 0, 0 )
/*
   Logical timer to model that message passes from transmitter to
   outgoing channel in zero time.
 */
proc ZEROTCHO:R_TIMER( \[,0,0,\],
      ( S.# ~= 0 )*(( BUF.# > 0 )+( BUF.# = 0 )*( PROD.# = newmsg )),
      ( TR.# = sent ), 0)
/*
   Logical timer of incoming channel. Set when channel is handed an
   acknowledgement by the receiving protocol. Expires when channel
   delivers or looses the acknowledgement. Channel transmission
   delay is chosen to be between 19 and 20 time units.
 */
proc CHI_TIMER:R_TIMER( \[,19,20,\],  RB.# ~= 0, ACK.# ~= pause, 0 )
/*
   Logical timer to model that acknowledgement passes from receiving
   protocol to incoming channel in zero time.
 */
proc ZEROTCHI:R_TIMER( \[,0,0,\], (ACK.# ~= pause),
                (SMON.$ = 0) * ( S.# ~= 0 ) * (INIT.$ = 1),0)
/*
```

```
   Logical timer to model that producer stays idle for PITIME time
   units after generating a message.
 */
proc PROD_TIMER:R_TIMER( \[,PITIME,PITIME,\],
               (PROD.$ = OFF) * ( PRODMON.$ = 0 ), PROD.$ = ON, 0 )
/*
   Logical timer to model that producer transition from ON state to
   OFF state takes zero time.
 */
proc ZEROTPROD:R_TIMER( \[,0,0,\], PROD.$ = ON,
                (PROD.$ = OFF) * ( PRODMON.$ = 0 ) * (INIT.$ = 1), 0 )
/*
   Logical timer to model that receiving protocol processes messages
   in zero time.
 */
proc    ZERORECDELAY:R_TIMER( \[,0,0,\], M.# > 0, RB.# ~= 0, 0 )
/*********************************************************************/
/*          UNTIMED PART OF THE PROTOCOL SPECIFICATION               */
/*********************************************************************/
proc        INIT
        stvar       $ : ( 0..1 )
        init        0
        trans

        0
                ->1         : true;
        1
                ->1         : true;
end         INIT

/* Producer process. */

proc PROD
        selvar  # : ( idle, newmsg )
        stvar   $ : ( OFF, ON )
        init        OFF
```

```
        trans

        OFF                { idle }
                ->ON         : true
                ->$         : true;

        ON                { newmsg }
                ->OFF : true;
end PROD
/*
  Needed so that set condition (PROD.$ = OFF)*( PRODMON.$ = 0 ) of
  logical timer PROD_TIMER is met only the first time PROD is in
  state OFF.
 */
monitor        PRODMON
        import PROD
        stvar   $ : ( 0..1 )
        init 0
        trans

        0
         ->1 : PROD.$ = OFF
         ->$ : else;
        1
         ->0 : PROD.$ = ON
         ->$ : else;
end        PRODMON
/*
   Models a buffer of size BUFSIZE that stores messages whenever the
   producer generates a message and the sender is busy transmitting a
   previous message (e.g. S.# = 0).
 */
proc BUF
        selvar  # : ( 0..BUFSIZE )
        stvar   $ : ( 0..BUFSIZE )
        init 0
```

```
        trans

        0 { $ }
         ->($+1) : ( PROD.# = newmsg ) * ( S.# = 0 )
         ->$ : else;

        ($>0)*($<BUFSIZE) { $ }
         ->($+1) : ( PROD.# = newmsg ) * ( S.# = 0 )
         ->($-1) : ( PROD.# = idle ) * ( S.# ~= 0 )
         ->$ : else;

        BUFSIZE { $ }
         ->($-1) : ( PROD.# = idle ) * ( S.# ~= 0 )
         ->$ : else;
end BUF

/* Sender process. UPPER PROTOCOL LAYER. (See details in ACW[90]) */

proc S
        import  TR.#
        selvar # : message
        stvar $ : ( OFF12, ON12, ON2, OFF13, ON13, ON3, OFF1, ON1 )
        cyset {OFF12, ON12, ON2, OFF13, ON13, ON3, OFF1}
        init OFF12
        trans

        OFF12 { 1..2 }
         ->ON12 : ( # = 1 ) * (( BUF.# > 0 ) +
                          ( BUF.# = 0 ) * ( PROD.# = newmsg ))
         ->ON2 : ( # = 2 ) * (( BUF.# > 0 ) +
           ( BUF.# = 0 ) * ( PROD.# = newmsg ))
         ->$ : else;

        ON12 { 0 }
         ->OFF12 : TR.# = ready
         ->$ : else;
```

```
        ON2 { 0 }
         ->OFF13 : TR.# = ready
         ->$ : else;

        OFF13 { 1,3 }
         ->ON13 : ( # = 1 ) * (( BUF.# > 0 ) +
           ( BUF.# = 0 ) * ( PROD.# = newmsg ))
         ->ON3 : ( # = 3 ) * (( BUF.# > 0 ) +
           ( BUF.# = 0 ) * ( PROD.# = newmsg ))
         ->$ : else;

        ON13 { 0 }
         ->OFF13 : TR.# = ready
         ->$ : else;

        ON3 { 0 }
         ->OFF1 : TR.# = ready
         ->$ : else;

        OFF1 { 1 }
         ->ON1 : ( BUF.# > 0 ) +
           ( BUF.# = 0 ) * ( PROD.# = newmsg )
         ->$ : else;

        ON1 { 0 }
         ->OFF1 : TR.# = ready
         ->$ : else;
end S
/*
  Needed so that expire condition (SMON.$ = 0)*( S.# ~= 0 )*(INIT.$ = 1)
  of logical timer ZEROTCHI is met only the first time S selects to
  pause.
 */
monitor        SMON
        import S
```

```
        stvar   $ : ( 0..1 )
        init 0
        trans

        0
         ->1 : S.# ~= 0
         ->$ : else;
        1
         ->0 : S.# = 0
         ->$ : else;
end        SMON
/**********************************************************************/
/*                           SENDING PROTOCOL                         */
/**********************************************************************/
/*
   Sending buffer process
 */
proc SB
        import  S.#, TR.#
        selvar # : message
        stvar $ : message
        init 0
        trans

        0 { 0 }
         ->S.# : ( S.# > 0 ) * (( BUF.# > 0 ) +
           ( BUF.# = 0 ) * ( PROD.# = newmsg ))
         ->$ : else;

        $>0 { $ }
         ->0 : TR.# = ready
         ->$ : else;
end SB
/*
   Transmitter process
 */
```

```
proc TR
        import  S.#, AA.#, TI.#
        selvar  # : ( pause, ready, sent )
        stvar   $ : ( READY, TRANS, WAIT )
        cyset {TRANS}
        init READY
        trans

        READY { ready }
         ->TRANS : ( S.# > 0 ) * (( BUF.# > 0 ) +
           ( BUF.# = 0 ) * ( PROD.# = newmsg ))
         ->$ : else;

        TRANS { pause, sent }
         ->READY : AA.# = ok
         ->WAIT : else * ( # = sent )
         ->$ : else;

        WAIT { pause }
         ->READY : AA.# = ok
         ->TRANS : else * ( TI.# = to )
         ->$ : else;
end TR
/*
   Transmission timer process
 */
proc TI
        import  TR.#, AA.#
        selvar  # : ( off, on, to )
        stvar   $ : ( OFF, ON, TO )
        cyset {ON}
        init OFF
        trans

        OFF { off }
         ->ON : TR.# = sent
```

```
         ->$ : else;

        ON { on, to }
         ->OFF : AA.# = ok
         ->TO : else * ( # = to )
         ->$ : else;

        TO { to }
         ->OFF : AA.# = ok
         ->ON : else * ( TR.# = sent )
         ->$ : else;
end TI
/*
   Sending sequence number process
 */
proc SSN
        import AA.#
        selvar # : bit
        stvar $ : bit
        init 0
        trans

        $ { $ }
         ->($+1) mod 2 : AA.# = ok
         ->$ : else;
end SSN
/*
   Acknowledgement analyzer process
 */
proc AA
        import RN.#, SSN.#, ACK.#
        selvar # : ( ok, nok )
        stvar $ : ( OK, NOK )
        init NOK
        trans
```

```
        NOK { nok }
         ->OK : ( RN.# = SSN.# ) * ( ACK.# = deliver )
         ->$ : else;
        OK { ok }
         ->NOK : true;
end AA
/**********************************************************************/
/*                        OUTGOING CHANNEL                           */
/**********************************************************************/
/*
   Outgoing message process
 */
proc M
        import SB.#, TR.#, AA.#
        selvar  # : (0..(N+1))
        stvar   $ : message
        init 0
        trans

        0 { 0 } /* pause */
         ->SB.# : ( SB.# ~= 0 ) * ( TR.# = sent ) * ( AA.# = nok )
         ->$ : else;
        $>0 { 0, $} /*, (N+1) } /* pause, message, no loss */
         ->$ : # = 0
         ->0 : else;
end M
/*
   Outgoing channel sequence number process
 */
proc SN
        import SB.#, TR.#, AA.#, SSN.#, M.#
        selvar  # : (0..2)
        stvar   $ : (0..2)
        init 2
        trans
```

```
        2 { 2 } /* pause */
         ->SSN.# : ( SB.# ~= 0 ) * ( TR.# = sent ) * ( AA.# = nok )
         ->$ : else;
        $<2 { $ }
         ->$ : M.# = 0
         ->2 : else;
end SN
/*
   Liveness condition monitor process for outgoing channel.
 */
proc        CHOLC
        import  SB.#, TR.#, AA.#, M.#
        stvar   $ : ( NORMAL, GETM, LOSTMNOK, IDLE, ERROR )
        cyset   {GETM, LOSTMNOK, ERROR }
        init    NORMAL
        trans

        NORMAL
         ->GETM : ( SB.# ~= 0 ) * ( TR.# = sent ) * ( AA.# = nok )
         ->$     : else;
        GETM
         ->LOSTMNOK : M.# = (N+1)
         ->NORMAL : (M.# >0) * (M.# <(N+1))
         ->$ : M.# = 0;
        LOSTMNOK
         ->GETM  : ( SB.# ~= 0 ) * ( TR.# = sent ) * ( AA.# = nok )
         ->$     : ~(( SB.# ~= 0 ) * ( TR.# = sent ) * ( AA.# = nok))
         ->IDLE  : ~(( SB.# ~= 0 ) * ( TR.# = sent ) * ( AA.# = nok));
        IDLE
         ->ERROR  : ( SB.# ~= 0 ) * ( TR.# = sent ) * ( AA.# = nok )
         ->$     : else;
        ERROR
         ->ERROR  :true;
end        CHOLC
/*****************************************************************/
/*                       INCOMING CHANNEL                          */
```

```
/**********************************************************************/
/*
   Incoming acknowledgement process
 */
proc ACK
        import RB.#
        selvar  # : ( pause, loss, deliver )
        stvar   $ : bit
        init 0
        trans

        0 { pause }
         ->1 : RB.# ~= 0
         ->$ : else;
        1 { pause, deliver} /* no loss */
         ->$ : # = pause
         ->0 : else;
end ACK
/*
   Incoming channel sequence number process
 */
proc RN
        import RB.#, MA.#, E.#, ACK.#
        selvar  # : (0..3)
        stvar   $ : (0..2)
        init 2
        trans

        2 { 2 } /* pause */
            ->E.# : ( RB.# ~= 0 ) * ( MA.# = ok )
            ->(E.# + 1) mod 2 : ( RB.# ~= 0 ) * ( MA.# = nok )
            ->$ : else;
        $<2 { $ }
            ->$ : ACK.# = pause
            ->2 : else;
end RN
```

```
/*
   Liveness condition monitor process for incoming channel.
 */
proc        CHILC
        import  RB.#, MA.#, ACK.#
        stvar   $ : ( NORMAL, GETM, LOSTMNOK, LOSTMOK, ERROR )
        cyset   { GETM, LOSTMNOK, ERROR }
        init    NORMAL
        trans

        NORMAL
         ->GETM : RB.# ~= 0
         ->$     : else;
        GETM
         ->LOSTMNOK : ACK.# = loss
         ->LOSTMOK : ACK.# = loss
         ->NORMAL : ACK.# = deliver
         ->$ : ACK.# = pause;
        LOSTMNOK
         ->GETM  : RB.# ~= 0
         ->$     : else;
        LOSTMOK
         ->ERROR  : RB.# ~= 0
         ->$     : else;
        ERROR
         ->ERROR  :true;
end        CHILC
/*********************************************************************/
/*                       RECEIVING PROTOCOL                          */
/*********************************************************************/
/*
   Receiving buffer process
 */
proc RB
        import  M.#, TR.#
        selvar # : message
```

```
        stvar $ : message
        init 0
        trans

        0 { 0 }
         ->$ : ( M.# = 0 ) + ( M.# = N + 1 )
         ->M.# : else;

        $>0 { $ }
         ->0 : true;
end RB
/*
   Expected sequence number process
 */
proc E
        import MA.#, RB.#
        selvar # : bit
        stvar $ : bit
        init 0
        trans

        $ { $ }
         ->($+1) mod 2 : ( MA.# = ok ) * ( RB.# ~= 0 )
         ->$ : else;
end E
/*
   Message analyzer process
 */
proc MA
        import M.#, SN.#, E.#
        selvar # : ( ok, nok )
        stvar $ : ( OK, NOK )
        init NOK
        trans

        NOK { nok }
```

```
         ->OK : ( E.# = SN.# ) * ( M.# ~= 0 ) * ( M.# ~= N + 1 )
         ->$ : else;
        OK { ok }
         ->NOK : true;
end MA
/*
   Receiver process. UPPER PROTOCOL LAYER
 */
proc R
        import  MA.#, RB.#
        selvar # : message
        stvar $ : message
        init 0
        trans

        0 { 0 }
         ->RB.# : ( MA.# = ok ) * ( RB.# ~= 0 )
         ->$ : else;

        $>0 { $ }
         ->0 : true;
end R

#ifdef        UTILIZATION
/*
  Task complement to check whether buffer utilization exceeds a
  certain point (BUFSIZE-2 in this case), depending on the
  producer message generation rate.
*/
monitor BUFTHRESHOLD
        import BUF
        stvar   $ : ( 0..1 )
        cyset   { 0 }
        init 0
        trans
```

```
        0
         ->1 : BUF.$ = BUFSIZE - 2
         ->$ : else;

        1
         ->$ : true;
end BUFTHRESHOLD
#endif        UTILIZATION

#ifdef  CORRECTNESS
/* Task complement to check in-order message delivery */

monitor TC
        import R
        stvar $ : (A, B, C, D, DEAD, ERROR)
        cyset {C, DEAD}
        init A
        trans

        A
         ->B : (R: 2)
         ->ERROR : (R: 3)
         ->$ : else;

        B
         ->C : (R: 3)
         ->ERROR : (R: 2)
         ->$ : else;

        C
         ->ERROR : (R.# = 2)+(R.# = 3)
         ->$ : (R.# = 1)+(R.# = 0)
         ->D : (R.# = 0);

        D
         ->$ : (R.# = 0)
```

```
         ->DEAD : else;

        DEAD
         ->$ : true;

        ERROR
         ->$ : true;
end TC
#endif  CORRECTNESS
/********************** END OF SPECIFICATION **********************/
```

Παρουσιάζουμε τώρα τις S/R διεργασίες ελεγκτές και τους λογικούς μετρητές που χρειάζονται για την επαλήθευση της παράδοσης μηνυμάτων μέσα σε συγκεκριμένα χρονικά όρια, και για την ανίχνευση της παρουσίας διπλών μηνυμάτων. Στην πρώτη περίπτωση είναι αναγκαία η τροποποίηση των καναλιών επικοινωνίας ώστε να μην χάνονται μηνύματα ή βεβαιώσεις μηνυμάτων, ενώ στη δεύτερη περίπτωση χρειάζεται να μεταβάλουμε ανάλογα μόνο το εισερχόμενο κανάλι. Παραλείπουμε την υπόλοιπη περιγραφή αφού είναι η ίδια όπως προηγουμένως.

```
proc DELIV_TIME :R_TIMER( \[,1,DEL_TIME,\],
                        (S.# = 2)*(DELIVERY_TIME.$ = 1),
(R.# = 2)*(DELIVERY_TIME.$ = 2), 0)


/*
   Task complement to check whether a message (e.g. 2)
   is delivered within DELIV_TIME time units.
*/
monitor DELIVERY_TIME
      import S,R
      stvar   $ : (1..3)
      cyset   {1, 2}
      init    1
      trans

      1
             ->2             : S.# = 2
             ->$             : else;
```

```
      2
              ->3             : R.# = 2
              ->$             : else;

      3
              ->$             : true;


end DELIVERY_TIME

/*
 Assume that receiving protocol processes messages in zero time.
*
proc    ZERORECDELAY:R_TIMER( \[,0,0,\], M.# > 0, RB.# ~= 0, 0 )
/*
   Task complement to check whether duplicate
   messages are ever send, supposing that no
   acknowledgements are lost.
*/
monitor DUPLICATES
        import RB
        stvar   $ : (OK1, OK2, NOK)
        cyset   {OK1, OK2}
        init    OK1
        trans

        OK1
                -> OK2  : ( RB.$ = 2 )
                -> $    : else;
        OK2
                -> NOK  : ( RB.$ = 2 )
                -> $    : else;
        NOK
                -> $    : true;
```

```
end DUPLICATES
```

Παράρτημα D

Αλλαγές στη δεύτερη έκδοση του RTCOSPAN

Σε αυτό το ευρετήριο περιγράφουμε με συντομία τις αλλαγές που έγιναν τον Αύγουστο του 1992 κατά την κατασκευή της δεύτερης έκδοσης του RTCOSPAN. Το RTCOSPAN πλέον είναι πιο ανεξάρτητο από τη συγκεκριμένη αρχιτεκτονική της μηχανής, αλλά και από το ίδιο το COSPAN. Εχει δοκιμαστεί σε μηχανές sun4 kai SGI με λειτουργικό σύστημα SunOS 4.1.2 kai IRIX αντίστοιχα, αλλά δεν περιμένουμε να έχει προβλήματα σε οποιοδήποτε μηχάνημα και έκδοση του UNIX. Το ίδιο το RTCOSPAN αλλά και ο `C` κώδικας που παράγεται μπορεί να μεταφραστεί και απο ANSI `C` μεταφραστές εκτός από τους παραδοσιακούς R&K `C` μεταφραστές. Η έκδοση του COSPAN που χρησιμοποιείται είναι η 7.9.1 (rl) 7/27/92 και δεν αναμένεται να υπάρχουν πρβλήματα συμβατότητας ούτε με μελλοντικές εκδόσεις του COSPAN.

Επίσης έχουν αφαιρεθεί το αρχείο εντολών `awk` από το στοιχειώδη μεταφραστή, και η παράμετρος `tid` που αναφέρονται στο κεφάλαιο 5, χωρίς όμως να αλλάξει καθόλου η διασύνδεση χρήστη-εργαλείου, με αποτέλεσμα η νέα έκδοση του RTCOSPAN να είναι πλήρως συμβατή με την προηγούμενη. Λεπτομερείς περιγραφές των αλλαγών αναμένεται να εμφανιστούν σε μελλοντική τεχνική αναφορά.

Βιβλιογραφία

[AC85] S. Aggarwal and C. Courcoubetis. Distributed implementation of a model of communication and computation. In *Proceedings of the 18th Hawaii Intl. Conference on System Sciences*, pages 206–218, jan 1985.

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of the 5th Symposium on Logic in Computer Science*, pages 414–425, Philadelphia, June 1990.

[ACD91a] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for probabilistic real-time systems. In *Proceedings of the 18th ICALP*, pages 115–126, Madrid, July 1991.

[ACD91b] R. Alur, C. Courcoubetis, and D. Dill. Verifying automata specifications of probabilistic real-time systems. In *Proceedings of the REX Workshop*, Plasmolen, June 1991.

[ACW90] S. Aggarwal, C. Courcoubetis, and P. Wolper. Adding liveness properties to coupled finite-state machines. *ACM Transactions on Programming Languages and Systems*, 12(2):303–339, 1990.

[AD90] Rajeev Alur and David Dill. Automata for Modeling Real-Time Systems. In *Automata, Languages and Programming : 17th Annual Colloquium*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335, 1990. Warwick University, July 16-20.

[AH90] R. Alur and T. Henzinger. Real-time logics: complexity and expressiveness. In *Proceedings of the 5th Symposium on Logic in Computer Science*, Philadelphia, June 1990.

[AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison Wesley, 1974.

[AIKY92] R. Alur, A. Itai, R. Kurshan, and M. Yannakakis. Timing verification by successive approximation. In *CAV 92*, Montreal, Canada, July 1992.

[AK83] S. Aggarwal and R.P. Kurshan. Modelling elapsed time in protocol specification. In H. Rudin and C.H. West, editors, *Protocol Specification, Testing and Verification, III*, pages 51–62. Elsevier Science Publisers B.V., 1983.

[Alu91] Rajeev Alur. Techniques for automatic verification of real-time systems. Technical Report STAN-CS-91-1378, Department of Computer Science, Stanford University, August 1991. Ph.D. Thesis.

[BG80] B. Kurshan B. Gopinath. The selection/resolution model for coordinating concurrent processes. In *AT&T Bell Laboratories Technical Report*, 1980.

[BM83] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time petri nets. In *Information Processing*, pages 41–46. Elsevier Scinece Publishers B.V. (North-Holland), 1983.

[Buc62] R. Buchi. On a decision method in restricted second order arithmetic. In *Proc. Int. Congr. Logic, Method and Philos. Sciences*. Stanford U. Press, 1962.

[Bur89] J. R. Burch. Combining CTL, Trace Theory, and Timing Models. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 334–348. Springer–Verlag, 1989.

[CDT92] C. Courcoubetis, D. Dill, and P. Tzounakis. Adding dense time properties to finite–state machines: The tool cospan. 1992.

[Dil89] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. Workshop on Computer Aided Verification*, Grenoble, June 1989. Lecture Notes in Computer Science, Springer-Verlag.

[HK89] Z. Har'El and R. Kurshan. Automatic verification of coordinating systems. In *Proceedings of Workshop on Automatic Verification Methods for Finite-State Systems*, Grenoble, June 1989. to appear.

[HPOG89] N. Halbwachs, D. Pilaud, F. Ouabodessalam, and A-C. Glory. Specifying, programming and verifying real-time systems using a synchronous declarative language. In J. Sifakis, editor, *Automatic Verification Methods for Finite*

State Systems, volume 407 of *Lecture Notes in Computer Science*. Springer–Verlag, 1989.

[JK86] B. Kurshan J. Katzenelson. S/r: A language for specifying protocols and other coordinating processes. In *Proc. 5th Ann. Int'l Phoenix Conf. Comput. Commun., IEEE*, 1986.

[JM86] Farnam Jahanian and Aloysius Ka-Lau Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, 12(9), September 1986.

[Kur90] R. Kurshan. Analysis of discrete event coordination. *Lecture Notes in Computer Science*, 480, 1990.

[McN66] R. McNaughton. Testing and generating infinite sequences by a finite automata. *Information and Control*, 9(2):521–530, 1966.

[Ost90] J. Ostroff. *Temporal Logic of Real-Time Systems*. Research Studies Press, 1990.

[Rab72] M. O. Rabin. Automata on infinite objects and church's problem. In *American Mathematical Society*, 1972.

[SA83] K. K. Sabnani S. Aggarwal, R. P. Kurshan. A calculus for protocol specification and validation. In *Protocol Specification, Testing and Verification, III*. North-Holland, 1983.

[SA86] K. Z. Meth S. Aggarwal, D. Barbara. Spanner - a tool for the specification, analysis, and evaluation of protocols. *IEEE Trans. on Software Engineering*, 1986.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99, 1983.

[ZH90] R. P Kurshan Z. Har'El. Software for analytical development of communication protocols. *AT&T Technical Journal*, 1990.