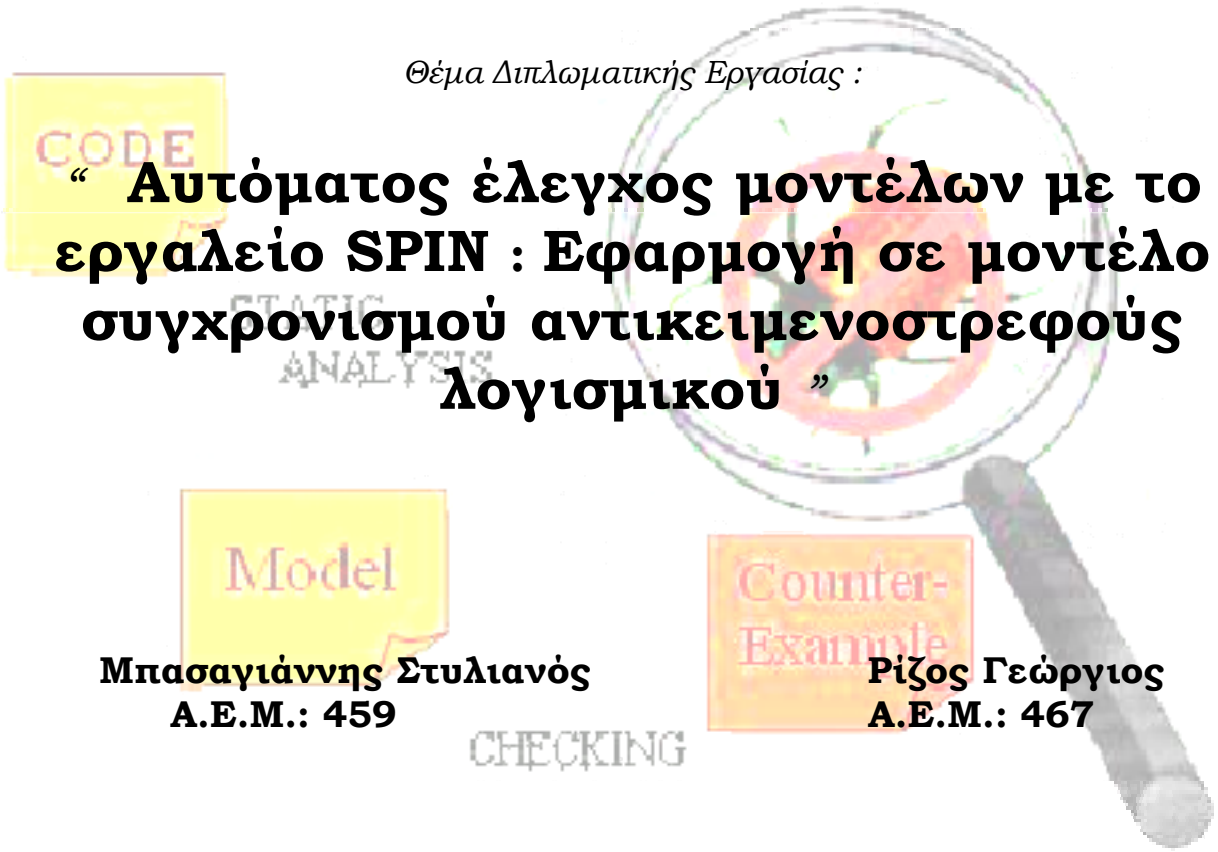
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Τμήμα Πληροφορικής ,
Ακαδημαϊκό Έτος 2003-2004

Θέμα Διπλωματικής Εργασίας :

“ Αυτόματος έλεγχος μοντέλων με το εργαλείο SPIN : Εφαρμογή σε μοντέλο συγχρονισμού αντικειμενοστρεφούς λογισμικού ”



Μπασαγιάννης Στυλιανός
Α.Ε.Μ.: 459

Ρίζος Γεώργιος
Α.Ε.Μ.: 467

Επιβλέπων καθηγητής : Dr. Παναγιώτης Κατσαρός

Θεσσαλονίκη 01/03/2004

Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Τμήμα Πληροφορικής ,
Ακαδημαϊκό Έτος 2003-2004

Θέμα Διπλωματικής Εργασίας :

" Αυτόματος έλεγχος μοντέλων με το εργαλείο SPIN : Εφαρμογή σε μοντέλο συγχρονισμού αντικειμενοστρεφούς λογισμικού "

Μπασαγιάννης Στυλιανός
Α.Ε.Μ.: 459

Ρίζος Γεώργιος
Α.Ε.Μ.: 467

Επιβλέπων καθηγητής : Dr. Παναγιώτης Κατσαρός

Θεσσαλονίκη 01/03/2004

Περιεχόμενα

- **Λίστα εικόνων**

- **Λίστα πινάκων**

- **Πρόλογος**

Στην σημερινή εποχή, ένα από τα θέματα τα οποία απασχολούν και θα συνεχίζουν να απασχολούν τους επιστήμονες της πληροφορικής είναι και αυτό της ποιότητας του λογισμικού. Η ποιότητα και η αξιοπιστία του λογισμικού αποτελούν καίρια αν μη τι άλλο ζητήματα στα οποία μια εταιρεία θέλει πάντα να εκπληρώνει με την παράδοση του προϊόντος της. Είναι γνωστό ότι το λογισμικό έχει εισβάλλει στην κοινωνία και στην ζωή μας τόσο ώστε να εξαρτιόμαστε όλο και πιο πολύ από αυτό. Παρόλο όμως της αύξησης των απαιτήσεών μας (για μεγαλύτερη ευκολία) μαζί αυξάνεται και η πολυπλοκότητα του λογισμικού, θέτοντας ένα σοβαρό ερώτημα του κατά πόσο αυτό συμφωνεί με τις προϋποθέσεις ορθότητας και λειτουργίας του. Την λύση στο συγκεκριμένο πρόβλημα έρχεται να δώσει ο συστηματικός έλεγχος του λογισμικού παίζοντας έναν σημαντικό ρόλο στην βελτίωση της ποιότητάς του. Ο έλεγχος του λογισμικού εμπεριέχει δοκιμαστικό και συνεχή έλεγχο της ορθότητας, της δομής και της λειτουργίας του. Εφαρμόζεται με την εκτέλεση πολλών δοκιμών στο υπό εξέταση σύστημα παρατηρώντας τις δοκιμές αυτές προσπαθώντας να παραχθεί το συμπέρασμα και η διαβεβαίωση για την σωστή λειτουργία του.

Η ορθότητα αυτή δίνεται στο που εξετάζεται είτε είναι γενική και περιλαμβάνει όλο το σύστημα είτε αναφέρεται σε συγκεκριμένες λειτουργίες, και άρα προδιαγραφές που έχει το σύστημα. Οι ιδιότητες αυτές περιγράφουν τι κάνει το σύστημα και τι όχι θέτοντας την βάση για οποιοδήποτε έλεγχο πάνω σε αυτές. Μια τεχνική η οποία με την σωστή εφαρμογή της μπορεί να θεωρηθεί αποτελεσματική, είναι και ο έλεγχος μοντέλων. Πρόκειται για την πραγματοποίηση όλων των αυτών που προαναφέρθηκαν ελέγχων αφού κύριος σκοπός της είναι η μοντελοποίηση ενός συστήματος και ο έλεγχος συγκεκριμένων λειτουργιών του. Στην εργασία αυτή γίνεται μια εκτενής αναφορά για τον έλεγχο μοντέλων και παρουσιάζεται το εργαλείο Spin. Πρόκειται για ένα εργαλείο αυτόματου ελεγκτή μοντέλων που βασίζεται σε αρχές και κανόνες που προϋποθέτουν την (πολυσυζητημένη) ορθότητα. Έτσι παρουσιάζεται η λειτουργία του εργαλείου και γίνεται η εφαρμογή του τόσο σε ένα αντιπροσωπευτικό παράδειγμα επικοινωνίας πρωτοκόλλων επικοινωνίας (το πρόβλημα των 4^{ων} στρατιωτών), όσο και σε ένα μοντέλο συγχρονισμού αντικειμενοστρεφούς λογισμικού (το πρόβλημα των δειπνούντων φιλοσόφων).

Παρόλο την μεγάλη του σημασία, ο έλεγχος αποτελεί μια υποβαθμισμένη έννοια κατά την φάση ανάπτυξης ενός λογισμικού αφού προϋποθέτει περισσότερους πόρους (εργασία-χρόνος-χρήμα), είναι δύσκολος στην εφαρμογή του ενώ και η επιστήμη που αφορά των έλεγχο μοντέλων βρίσκεται σε πρώιμο στάδιο. Με δεδομένη την διάθεση για βελτίωση της ποιότητας του λογισμικού, ο έλεγχος σήμερα θα πρέπει να κατέχει πρωτεύον ζήτημα για κάθε εταιρεία παραγωγής λογισμικού. Δεν είναι λίγες οι περιπτώσεις που αυτός σώζει ζωές (βλ. αεροπορικά συστήματα) αλλά και η έλλειψή του μπορεί να αποβεί μοιραία (βλ. διαστημόπλοιο Challenger).

- **Θέματα με τα οποία ασχολείται η εργασία**

Τα θέματα με τα οποία ασχολείται η εργασία είναι τα εξής

Κεφάλαιο 1ο : Εισαγωγή σε έννοιες συγχρονισμού
(Περιγράφονται βασικές έννοιες του συγχρονισμού σε σχέση με την αντικειμενοστρέφεια προετοιμάζοντας τον αναγνώστη για την εφαρμογή ελέγχου σε ένα τέτοιο μοντέλο)

Κεφάλαιο 2ο : Έλεγχος Μοντέλων
(Περιγράφεται η επιστήμη του έλεγχου μοντέλων και οι τεχνικές που χρησιμοποιεί αναλύοντας αυτές ώστε να μπορεί ο αναγνώστης να γνωρίζει βασικές έννοιες και τακτικές που εκτελεί ο έλεγχος, προετοιμάζοντάς τον για την πρακτική εφαρμογή του με το εργαλείο Spin)

Κεφάλαιο 3ο : Ο ελεγκτής μοντέλων Spin
(Περιγράφεται το εργαλείο Spin, την δομή, την λειτουργία τις δυνατότητες και τα αποτελέσματά του στο θέμα του ελέγχου. Γίνεται η εφαρμογή του εργαλείο σε ένα παράδειγμα-μοντέλο αναλύοντας τόσο το ίδιο το μοντέλο όσο και τις απαντήσεις που δίνει όσον αφορά αυτό)

Κεφάλαιο 4ο:Αυτόματος έλεγχος σε μοντέλο συγχρονισμού αντικειμενοστρεφούς λογισμικού
(Γίνεται ο έλεγχος του προβλήματος των δειπνούντων φιλοσόφων. Πρόκειται για ένα αντικειμενοστρεφές πρόβλημα συγχρονισμού το οποίο μοντελοποιείται στην γλώσσα Promela και στην συνέχεια ελέγχεται με τον αυτόματο ελεγκτή μοντέλων Spin. Το μοντέλο ελέγχεται για περίπτωση αδιέξοδου και για περίπτωση αδυναμίας τερματισμού ενώ μοντελοποιείται και με τέτοιο τρόπο ώστε να δίνεται μετά από την επιβεβαίωση του εργαλείου η σωστή λύση στο πρόβλημα)

Κεφάλαιο 5ο : Συμπεράσματα
(Ανακεφαλαίωση της εργασίας και επισήμανση των θεμάτων που κρίνονται σημαντικά)

- **Στόχος εργασίας**

Η εισαγωγή στον έλεγχο μοντέλων μέρος της θεματικής περιοχής των Τυπικών Μεθόδων (Formal Methods) , η γνωριμία και εξοικείωση με τον αυτόματο ελεγκτή μοντέλων Spin και την λειτουργία του, εφαρμόζοντας το σε μοντέλο αντιπροσωπευτικό των πρωτοκόλλων επικοινωνίας αλλά και σε μοντέλο συγχρονισμού αντικειμενοστρεφούς λογισμικού.

Εισαγωγή σε έννοιες συγχρονισμού

1.1 Εισαγωγή στον συγχρονισμό

Η βιομηχανία των υπολογιστών έχει μετακινηθεί κατά ένα μεγάλο ποσοστό στο χώρο των κατανεμημένων εφαρμογών που εκτελούνται σε ένα δίκτυο υπολογιστών. Η μετακίνηση αυτή είναι συνισταμένη πολλών παραγόντων:

- Πρώτα από όλα το γεγονός ότι η απόδοση μπορεί να διατηρηθεί ανεξάρτητα από το φόρτο εργασίας, αφού υπάρχει η δυνατότητα προσθήκης περισσότερων πόρων στην εφαρμογή.
- Επιπλέον, δεν είναι λίγες οι εφαρμογές που είναι κατανεμημένες από τη φύση τους. Ως παράδειγμα μπορεί να αναφερθεί ένα μηχάνημα ATM (Automatic Teller Machine), το οποίο συνδέεται με μία βάση δεδομένων. Επειδή ο μετρητής είναι κατανεμημένος από τη φύση του, θα είναι και ολόκληρη η εφαρμογή.

Ο κατανεμημένος υπολογισμός των εφαρμογών εισάγει καινούργια ζητήματα, τα οποία μέχρι πρότινος δεν απασχολούσαν τη βιομηχανία των υπολογιστών. Ένα από αυτά είναι το εξής: Διαφορετικά στοιχεία μιας κατανεμημένης εφαρμογής μπορούν να εκτελεστούν σε διαφορετικούς υπολογιστές, χωρίς να είναι αναγκαία η ύπαρξη ενός κοινού ρολογιού, το οποίο σημαίνει ότι η εκτέλεση είναι ασύγχρονη. Επειδή όμως στην επικοινωνία των υπολογιστών αυτών μπορεί να μεσολαβεί κάποιο δίκτυο, υπάρχει η πιθανότητα να παρουσιάζονται καθυστερήσεις, οι οποίες οφείλονται σε κάποιες άλλες εφαρμογές που εξυπηρετεί το δίκτυο. Αξίζει να σημειωθεί ότι οι καθυστερήσεις αυτές δε μπορούν να προσδιοριστούν και να προβλεφθούν. Λύση στο πρόβλημα των καθυστερήσεων που προκύπτουν, έρχεται να δώσει ο μη-ντετερμινισμός.

Επιπρόσθετα, πολλές εφαρμογές περιέχουν γεγονότα που απαιτούν την εκτέλεσή τους σε μία συγκεκριμένη σειρά. Αυτό έχει ως αποτέλεσμα ότι πρέπει να συγχρονιστούν και συντονιστούν τα διαφορετικά ασύγχρονα στοιχεία μιας εφαρμογής με σκοπό να εκτελεστούν στη συγκεκριμένη σειρά που απαιτούν. Τέλος, ο συγχρονισμός κάνει την κατασκευή κατανεμημένων εφαρμογών αρκετά περίπλοκη και ένας τρόπος υλοποίησης είναι με αφηρημένες γλώσσες υψηλού επιπέδου.

1.2 Αντικειμενοστρεφής προγραμματισμός

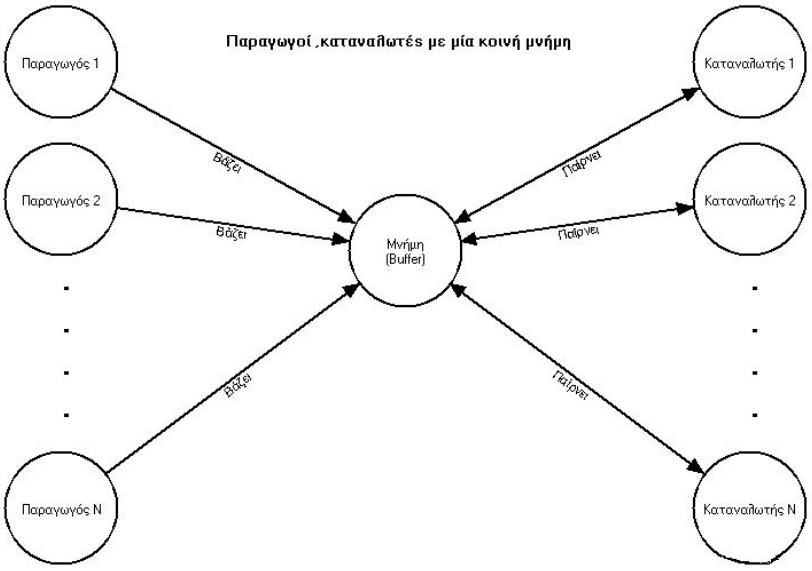
Ο αντικειμενοστρεφής προγραμματισμός προσφέρει ένα φυσικό τρόπο αποσύνθεσης ενός μεγάλου και πολύπλοκου συστήματος σε ημι-αυτόνομες οντότητες, τα αντικείμενα. Κάθε ένα από αυτά έχει μια κατάσταση και ένα σύνολο από διαδικασίες, που ονομάζονται μέθοδοι και είναι απαραίτητες για το χειρισμό της κατάστασης. Ένα αντικείμενο επικοινωνεί με ένα άλλο μόνο μέσω των μεθόδων του.

Εκτός από όμως από τα παραπάνω, ο αντικειμενοστρεφής προγραμματισμός παρέχει χρήσιμες έννοιες που βοηθούν στην έκφραση της δομής του προγράμματος. Το πιο σημαντικό από αυτά είναι η κληρονομικότητα. Μία κλάση περιγράφει τη συμπεριφορά ενός συνόλου αντικειμένων τα αντικείμενα αυτά είναι τα στιγμιότυπα της κλάσης. Τέλος, η κληρονομικότητα βοηθάει στην ιεραρχική οργάνωση των κλάσεων και των υποκλάσεων, που έχει ως αποτέλεσμα μικρότερο προγραμματιστικό κώδικα και πάνω από όλα εύκολα επεκτάσιμο.

1.3 Ορισμοί μοντέλων συγχρονισμού αντικειμενοστρεφούς λογισμικού

Actors: Το μοντέλο αυτό περιλαμβάνει έναν εσωτερικό παραλληλισμό, όπου πολλαπλές μέθοδοι «ανάγνωσης» και μια μέθοδος «εγγραφής» μπορούν να εκτελεστούν ταυτόχρονα μέσα στα πλαίσια ενός αντικειμένου. Επίσης το σημαντικό είναι ότι εγγυάται σειριοποιησιμότητα. Στο μοντέλο αυτό, μία κατανεμημένη εφαρμογή αποτελείται από μία συλλογή ασύγχρονων αντικειμένων που εκτελούνται ταυτόχρονα και επικοινωνούν με μηνύματα.
Coordination: Επειδή τα αντικείμενα εκτελούνται ασύγχρονα, όπως προαναφέραμε, η σχετική σειρά με την οποία φθάνουν τα μηνύματα σε ένα αντικείμενο είναι μη-ντετερμινιστική. Είναι απαραίτητο ένα αντικείμενο να αποστέλλει τα μηνύματα με συγκεκριμένη σειρά. Για παράδειγμα, αν έχουμε δύο μηνύματα, *στείλε* και *λάβε*, που πηγαίνουν σε ένα buffer με ένα στοιχείο, είναι προφανές ότι ο buffer δε μπορεί να δεχτεί δύο μηνύματα *στείλε*. Σε αυτό το πρόβλημα δίνει λύση ο coordination.

Ένα ακόμη παράδειγμα, όπου φαίνεται η ανάγκη και η χρησιμότητα του coordination είναι το γνωστό πρόβλημα του παραγωγού-καταναλωτή, στο οποίο ένας παραγωγός στέλνει μηνύματα στο buffer και ένας καταναλωτής τα λαμβάνει. Για τη σωστή λειτουργία της εφαρμογής απαιτείται coordination. Το πρόβλημα φαίνεται σχηματικά παρακάτω.



Εικόνα 2.3 :Παραγωγοί και καταναλωτές με κοινή μνήμη

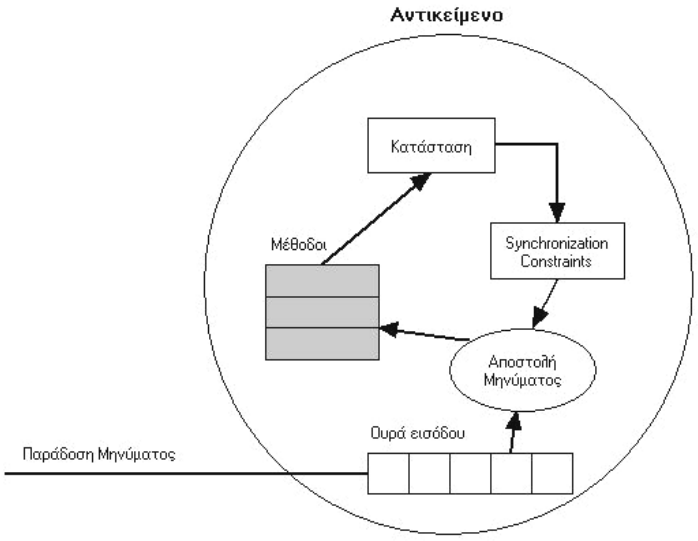
Συγχρονιστές περιορισμού: Για την καλύτερη ερμηνεία και κατανόηση αυτού του όρου θα χρησιμοποιήσουμε το παράδειγμα του παραγωγού-καταναλωτή. Πιο συγκεκριμένα, αν ο παραγωγός στείλει μήνυμα *στείλε* στο buffer και αυτός είναι γεμάτος, τότε τα synchronization constraints του buffer καθυστερούν το ληφθέν μήνυμα μέχρι να έρθει ένα μήνυμα *λάβε.* Επίσης, επιτρέπουν στο προγραμματιστή να πραγματοποιεί αλλαγές στα constraints της διάταξης μηνυμάτων ενός αντικειμένου, χωρίς να τροποποιεί τα άλλα αντικείμενα της εφαρμογής.

Συγχρονιστές: Είναι μία διακριτή οντότητα που καθορίζει τα constraints της διάταξης μηνυμάτων ενός συνόλου αντικειμένων. Ο synchronizer καθυστερεί την αποστολή μηνυμάτων από ένα σύνολο αντικειμένων σύμφωνα με κάποια, καθορισμένα από το χρήστη, κριτήρια. Μία ιδιότητα των synchronizers είναι ότι επιτρέπουν στο χρήστη να καθορίσει τα αντικείμενα που απαιτούν coordination.

1.4 Συγχρονιστές

Ένας synchronizer διατηρεί την ακεραιότητα μιας ομάδας αντικειμένων με το να εξασφαλίζει ότι η αποστολή μηνυμάτων μέσα στην ομάδα λαμβάνει χώρα την στιγμή εκείνη που ορίζει ο χρήστης. Με τους synchronizers ,ο προγραμματιστής μπορεί εύκολα να εκφράσει την ακεραιότητα των αντικειμένων σαν μια ξεχωριστή οντότητα και όχι σαν ένα μέρος ανεξάρτητης ομάδας. Οι synchronizers και τα synchronization constraints περιγράφουν την ακεραιότητα σε διαφορετικά επίπεδα. Οι μεν synchronizer περιγράφουν την ακεραιότητα ομάδας αντικειμένων, ενώ οι συγχρονιστές περιορισμού περιγράφουν την ακεραιότητα ανεξάρτητων αντικειμένων.

Αυτή η διαφορά στην περιγραφή ,δίνει μια νέα ώθηση στην δημιουργία για την κατασκευή νέων τμημάτων κώδικα, δίνοντάς του την ικανότητα να αντιμετωπίσει ακόμη περισσότερες προγραμματιστικές δυσκολίες. Οι synchronizers παρουσιάζουν παρόμοια συμπεριφορά με τα αντικείμενα όσον αφορά την περίπτωση στην οποία υπάρχει μία κατάσταση ενθυλάκωσης. Σε αντίθεση με τα αντικείμενα ,οι synchronizers δεν στέλνουν αλλά και δεν λαμβάνουν μηνύματα. Στην πραγματικότητα παρατηρεί και περιορίζει την αποστολή των μηνυμάτων των αντικειμένων.



Εικόνα 1.4: Αντικείμενο με synchronization constraints

Η λειτουργία ενός synchronizer περιγράφεται με τους όρους triggers (ωθήσεις) και constraints (περιορισμούς). Ένα trigger παρατηρεί την διαδικασία αποστολής ενός μηνύματος αντικειμένου εκτελώντας ενέργεια κάθε φορά που το αντικείμενο στέλνει μηνύματα. Η ενέργεια αυτή με την σειρά της μπορεί να αλλάξει την κατάσταση του ίδιου του synchronizer. Ένας synchronizer μπορεί να περιέχει πολλαπλούς triggers

,αλλά και η κατάστασή του μπορεί να αντικατοπτρίσει τα ιστορικά δρώμενα μιας ομάδας αντικειμένων.
Ένας synchronizer μπορεί να περιέχει δύο ειδών constraints :

- *Atomicity constraints :* πρόκειται για τους περιορισμούς εκείνους που εγγυώνται για την αδιαίρετη αποστολή πολλαπλών μηνυμάτων από πολλά αντικείμενα
- *Disabling constraints :* πρόκειται για τους περιορισμούς εκείνους που παρέχουν υπό όρους αποστολή αδιαίρετων μηνυμάτων

Τόσο τα atomicity constraints όσο και τα disabling μπορούν να χρησιμοποιηθούν για την περιγραφή διαφορετικού είδους εξαρτήσεων ανάμεσα στις ενέργειες που πραγματοποιούνται κατά την αποστολή μηνυμάτων από μία ομάδα αντικειμένων.

Τα *atomicity constraints* μπορούν να χρησιμοποιηθούν για την περιγραφή αμοιβαίων εξαρτήσεων ανάμεσα στις αποστολές μηνυμάτων. Για παράδειγμα ,εάν a και b είναι δύο μηνύματα που προορίζονται για δύο διαφορετικά αντικείμενα ,ένα atomicity constraint μπορεί να δηλώσει ότι το μήνυμα a δεν μπορεί να αποσταλεί εκτός εάν το μήνυμα b δεν έχει αποσταλεί πρώτα, και το αντίθετο. Τέτοιου είδους αμοιβαίων εξαρτήσεων περιλαμβάνουν καταστάσεις όπου ένα σετ μηνυμάτων πρέπει να αποσταλεί και δεν πρέπει ποτέ να φτάνουμε στην περίπτωση όπου ένα μέρος των μηνυμάτων αυτών έχει αποσταλεί.

Τα *disabling constraints* μπορούν να χρησιμοποιηθούν για την περιγραφή one-way (μονόδρομών) εξαρτήσεων ανάμεσα στις αποστολές μηνυμάτων. Για παράδειγμα ,εάν a και b είναι δύο μηνύματα που προορίζονται για δύο διαφορετικά αντικείμενα ,ένα disabling constraint μπορεί να δηλώσει ότι το μήνυμα a πρέπει να αποσταλεί πριν το μήνυμα b. Αυτές οι one-way εξαρτήσεις περιλαμβάνουν προσωρινές διατεταγμένες καταστάσεις αποστολής μηνυμάτων μεταξύ των αντικειμένων.

1.5 Συγχρονισμός και κληρονομικότητα

Όλα τα τμήματα κώδικα των γλωσσών προγραμματισμού που περιέχουν constrains κληρονομικότητας και συγχρονισμού ,θα πρέπει να ικανοποιούν τα δύο επόμενα κριτήρια :

- Οι σωστές ιδιότητες των synchronization constraints δεν πρέπει να εμποδίζουν μεθόδους κληρονομικότητας
- Θα πρέπει να είναι δυνατόν να κληρονομούμε και άρα να αλλάζουμε την δομή ορισμένων synchronization constraints

Το πρώτο κριτήριο είναι αρκετά σημαντικό γιατί η ανάγκη για να εκφραστεί μια μέθοδος κληρονομικότητας , υπάρχει ακόμα και κατά την μετατροπή από μεμονωμένες σε κατανεμημένες εφαρμογές. Όσον αφορά το δεύτερο κριτήριο ,αυτό τονίζει την ανάγκη για την επέκταση της κληρονομικότητας έτσι ώστε να καλύψει και τις προϋποθέσεις των synchronization constraints. Σε αντίθετη περίπτωση ,εάν η κληρονομικότητα εφαρμόζεται μόνο σε μεθόδους, τα synchronization constraints για

μεθόδους κληρονομικότητας θα πρέπει να προσδιοριστούν ξανά σε υποκλάσεις .Κάτι τέτοιο θα οδηγούσε στα ακόλουθα προβλήματα:

- *Αντίγραφα της πληροφορίας* : Παρόμοια synchronization constraints θα αντιγράφονται (2 φορές) κατά την περιγραφή πολλαπλών κλάσεων. Το πλεόνασμα της πληροφορίας εφικτά αρκετά δύσκολο στην συντήρηση και στον έλεγχο των προγραμμάτων
- *Παραβίαση ενθυλάκωσης* : Επειδή τα synchronization constraints βασίζονται στην 'ιδιωτική' κατάσταση των αντικειμένων , με τον επαναπροσδιορισμό των constraint αυτών κατά τον ορισμό των υποκλάσεων θα 'εξέθεται' κατά κάποιο τρόπο τις υπερκλάσεις

1.6 Διαφορές synchronizers και synchronization constraints

Παρά το γεγονός ότι και οι synchronizers και οι synchronization constraints, εκφράζουν constraints διάταξης μηνυμάτων, υπάρχουν σημαντικές διαφορές ανάμεσα στις δύο αυτές έννοιες.

- Οι synchronizers προσδιορίζουν constraints ως ένα κομμάτι αντικειμένου, ενώ οι synchronization constraints ως κομμάτι της εφαρμογής του αντικειμένου.
- Μία δεύτερη και τελευταία διαφορά είναι ότι οι synchronizers είναι διακριτές οντότητες, σε αντίθεση με τους synchronization constraints που εξαρτώνται από την κατάσταση του αντικειμένου.
- Τα synchronization constraints εκφράζουν την ακεραιότητα των απαιτήσεων μεμονωμένων αντικειμένων, αλλά και την ακεραιότητα ομάδας αντικειμένων. Μπορούμε να πούμε ότι η λειτουργικότητα των synchronizers υποβαθμίζουν την λειτουργικότητα των constraint συγχρονισμού γι αυτό και τα constraint συγχρονισμού θα πρέπει να περιγράφονται σαν single-object synchronizers.

Ας υποθέσουμε ότι οι synchronizers έπαιρναν την θέση των synchronization constraints και ότι τους χρησιμοποιούσαμε για να εκφράσουμε την ακεραιότητα απλών αντικειμένων. Από την στιγμή που η ακεραιότητα ενός αντικειμένου μπορεί να εξαρτάται ανάλογα της εσωτερικής του περιγραφής , και παρόμοια οι synchronizers δεν μπορούν να αναφέρονται απευθείας σε αυτή την περιγραφή, οι synchronizers θα αναγκάζονταν να αντιγράψουν για δεύτερη φορά τα αντικείμενα. Αυτός ο 'διπλασιασμός' θα εξέθεται την εσωτερική αναπαράσταση των αντικειμένων και άρα θα παραβίαζε την ιδιότητα της ενθυλάκωσης αυτών. Επίσης κάτι τέτοιο θα οδηγούσε στην επιπλέον καταγραφή της παρούσας κατάστασης των αντικειμένων.

Το συμπέρασμα που προκύπτει είναι ότι έχοντας μαζί τους synchronizers και τα constraint συγχρονισμού είναι εξαιρετικά σημαντικό έτσι ώστε να διατηρηθεί το όριο της αφαίρεσης που ορίζεται από τις διαπροσωπείες αντικειμένων.

Έλεγχος Μοντέλων

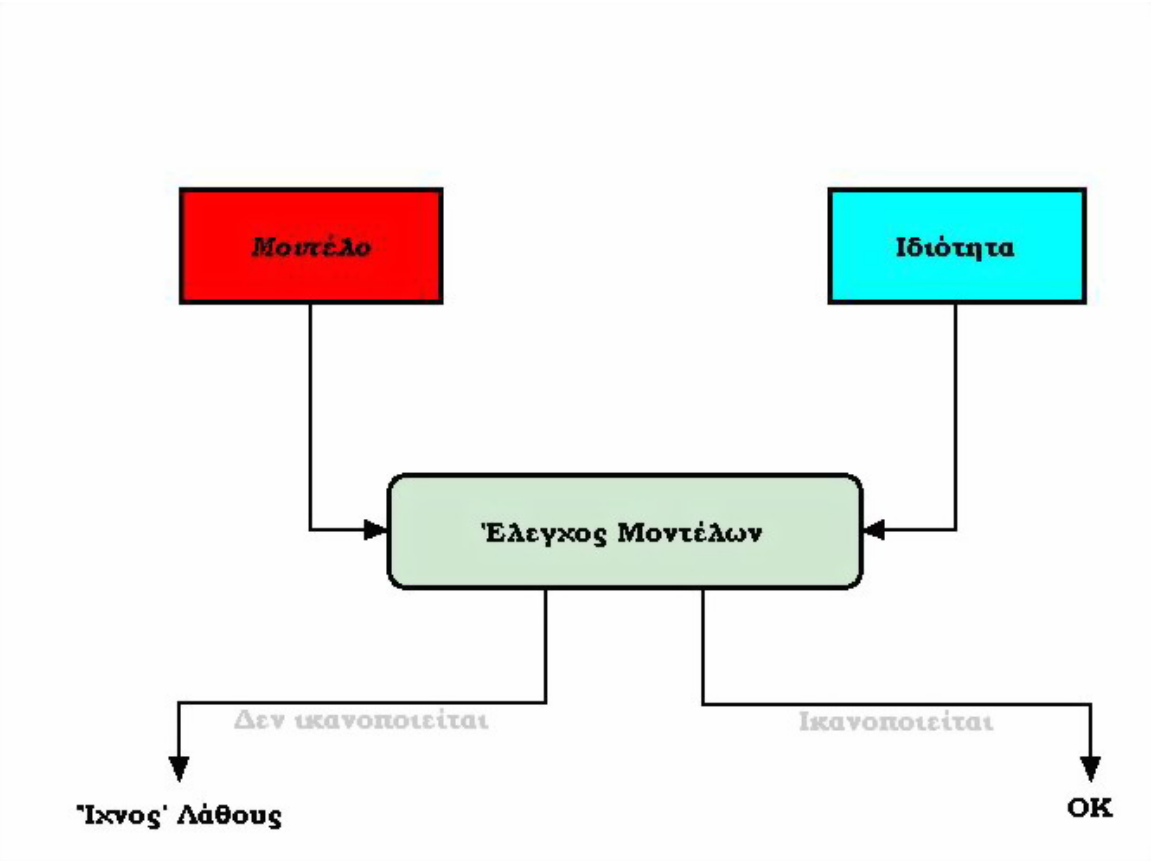
2.1 Εισαγωγή στον έλεγχο μοντέλων

Τα παράλληλα συστήματα πεπερασμένης κατάστασης διαδίδονται ολοένα και περισσότερο, ιδιαίτερα στο χώρο του σχεδιασμού ψηφιακών κυκλωμάτων και των πρωτοκόλλων επικοινωνίας. Κάποια λογικά λάθη που βρίσκονται σε ένα από τα τελευταία στάδια της σχεδίασης τέτοιων συστημάτων, αποτελούν σημαντικό πρόβλημα τόσο για τους σχεδιαστές, όσο και για τους προγραμματιστές. Τέτοιου είδους λάθη μπορούν είτε να καθυστερήσουν την παραγωγή ενός καινούργιου προϊόντος, είτε να προκαλέσουν σφάλμα σε κάποια συσκευή που είναι ήδη σε χρήση. Παράδειγμα τέτοιου λάθους είναι ο πύραυλος Ariane 5, ο οποίος εξερράγη σε λιγότερο από 40 δευτερόλεπτα από την εκτόξευσή του. Η υπεύθυνη επιτροπή έρευνας έβγαλε το πόρισμα ότι η πτώση οφειλόταν σε λάθος του λογισμικού του υπολογιστή, που ήταν υπεύθυνος για την κίνηση του πυραύλου. Πιο συγκεκριμένα, κατά τη διάρκεια της εκτόξευσης ένας 64-bit αριθμός κινητής υποδιαστολής μετατράπηκε σε 16-bit ακέραιο. Αυτή η μετατροπή δεν καλυπτόταν από το κώδικα του προγράμματος με αποτέλεσμα ο υπολογιστής να παρουσιάσει σφάλμα και κατά συνέπεια ο πύραυλος να εκραγεί. Η υπεύθυνη επιτροπή έρευνας πρότεινε τη λήψη μέτρων, κυρίως κατασκευή λογισμικού επαλήθευσης, για την αποφυγή παρόμοιων περιστατικών στο μέλλον.

Η πιο διαδεδομένη τεχνική επαλήθευσης βασίζεται σε εκτεταμένο έλεγχο ή προσομοίωση, αλλά μπορεί να μη βρει σημαντικά λάθη όταν ο αριθμός των καταστάσεων του αλγορίθμου ή του πρωτοκόλλου είναι πολύ μεγάλος. Παρά το γεγονός ότι είχαν γίνει πολλές έρευνες για την υλοποίηση της τεχνικής επαλήθευσης, τα περισσότερα αποτελέσματα δεν ήταν ικανοποιητικά, λόγω του ότι τα συστήματα που κατασκευάζονταν απαιτούσαν πολύ χρόνο για την προσομοίωση, αλλά και πολλές ενέργειες από το χρήστη. Τη δεκαετία 1980 επινοήθηκε μια εναλλακτική τεχνική επαλήθευσης από τους Clarke, Emerson, Quielle και Sifakis, που ονομάστηκε *χρονική λογική έλεγχου μοντέλων (temporal logic model checking).* Σε αυτή την προσέγγιση οι αλγόριθμοι και τα πρωτόκολλα μοντελοποιούνται ως συστήματα μετάβασης καταστάσεων.

Ο έλεγχος μοντέλων (model checking) παρουσιάζει σημαντικά *πλεονεκτήματα*:

1. Το κυριότερο είναι ότι όλη η διαδικασία είναι αυτόματη και δεν απαιτούνται πολλές ενέργειες από το χρήστη. Με απλά λόγια, ο χρήστης εισάγει μία αναπαράσταση του μοντέλου και τα χαρακτηριστικά που επιθυμεί να ελέγξει. Ο αλγόριθμος της επαλήθευσης ή θα τερματίσει το πρόγραμμα με απάντηση true, ή θα δώσει μία απαντήσει ότι η φόρμουλα είναι λάθος.
2. Επίσης η διαδικασία είναι πολύ γρήγορη και συνήθως υπάρχει απόκριση μέσα σε διάστημα λίγων λεπτών.



Εικόνα 2.1: Έλεγχος μοντέλων

Το πιο σημαντικό μειονέκτημα του ελέγχου μοντέλων είναι η έκρηξη καταστάσεων (*state explosion*), που μπορεί να συμβεί αν το σύστημα που επαληθεύεται αποτελείται από πολλά στοιχεία που κάνουν παράλληλες μετατροπές. Σε αυτή την περίπτωση οι καταστάσεις του συστήματος μπορούν να αυξάνονται εκθετικά. Εξαιτίας αυτού του προβλήματος, πολλοί ερευνητές προέβλεψαν ότι ο έλεγχος μοντέλων δε θα μπορούσε να είναι πρακτικός σε μεγάλα προβλήματα. Η έκρηξη του χώρου καταστάσεων θα περιγραφθεί σε παρακάτω παράγραφο αναλυτικά. Παρά το γεγονός αυτό, τα συστήματα που ελέγχονται αυξάνονται συνεχώς και αυτό οφείλεται κυρίως στη χρησιμοποίηση των *binary decision diagrams* που είναι μία δομή δεδομένων για αναπαράσταση Boolean συναρτήσεων. Αυτή η μέθοδος είναι εξαιρετικά χρήσιμη για σύγχρονα κυκλώματα. Σε ασύγχρονα πρωτόκολλα, είναι δυνατό να μειωθεί το μέγεθος του χώρου καταστάσεων με τη χρησιμοποίηση της τεχνικής *partial order reduction*.

Ως αποτέλεσμα αυτών των τεχνικών ο «έλεγχος μοντέλων» έχει ευρεία διάδοση ως τεχνική επαλήθευσης.

2.2 Βασικά Στάδια στον έλεγχο μοντέλων

Ο «έλεγχος μοντέλων» περιλαμβάνει τρία στάδια:

1. *Modelling:* Σε αυτό το στάδιο πρέπει να μετατραπεί ένα σχέδιο σε ένα φορμαλισμό, ο οποίος να γίνεται δεκτός από ένα εργαλείο model checking. Στις περισσότερες περιπτώσεις αρκεί μία μεταγλώττιση. Σε κάποιες άλλες όμως,

εξαιτίας του περιορισμού σε χρόνο και μνήμη, η μετατροπή του σχεδίου απαιτεί τη χρησιμοποίηση αφηρημένων εννοιών για την απομάκρυνση λιγότερο σημαντικών λεπτομερειών.

2. *Specification:* Πριν το στάδιο της επαλήθευσης, είναι απαραίτητο να καθορίσουμε τις προδιαγραφές που το σχέδιο πρέπει να ικανοποιεί. Ο καθορισμός αυτός δίνεται συνήθως σε κάποια μορφή λογικού φορμαλισμού. Για συστήματα λογισμικού χρησιμοποιούμε temporal logic(χρονική λογική), με την οποία γνωρίζουμε τη συμπεριφορά του συστήματος σε σχέση με το χρόνο. Ένα σημαντικό στοιχείο στο specification είναι η ολοκληρωτικότητα. Ο «έλεγχος μοντέλων» παρέχει τη δυνατότητα να δούμε αν ένα σχέδιο ικανοποιεί ένα συγκεκριμένο καθορισμό, αλλά είναι αδύνατο να πούμε με σιγουριά ότι ο δοθέν καθορισμός καλύπτει όλες τις ιδιότητες που πρέπει να ικανοποιεί το σύστημα.

3. *Verification:* Θεωρητικά το στάδιο της επαλήθευσης είναι αυτόματο, αλλά στην πράξη απαιτούνται και ενέργειες από το χρήστη. Μία τέτοια ενέργεια είναι η ανάλυση των αποτελεσμάτων. Σε περίπτωση αρνητικού αποτελέσματος ο χρήστης ειδοποιείται από το εργαλείο ελέγχου με ένα «ίχνος λάθους». Το τελευταίο μπορεί να αποβεί πολύ χρήσιμο στο σχεδιαστή του προγράμματος που ελέγχθηκε, ο οποίος μπορεί να δει που ακριβώς είναι το λάθος και να το διορθώσει. Μετά από αυτή τη διαδικασία το πρόγραμμα επαληθεύεται ξανά. Επιπρόσθετα, «ίχνος λάθους» μπορεί να προκύψει από λανθασμένη μοντελοποίηση ή από λανθασμένο specification(false negative). Σε αυτές τις δύο περιπτώσεις η επαλήθευση δε θα τερματίσει κανονικά και θα χρειαστεί να ξαναγίνει αφού αλλαχθεί το specification ή αφού ξαναγινεί η μοντελοποίηση, όποιο από αυτά δημιούργησε το πρόβλημα.

2.3 Η δομή Kripke (Kripke structure)

Ένα πλέον σημαντικό κομμάτι στο κεφάλαιο έλεγχος μοντέλων αποτελεί και η δομή Kripke που θα περιγραφτεί παρακάτω. Η δομή Kripke αποτελεί μια από τις κυριότερες δομές συμβολικού χαρακτήρα για την εξαγωγή τύπων που αναπαριστούν concurrent συστήματα. Ουσιαστικά αποτελεί έναν τύπο γράφου καταστάσεων μετατροπής για μπορέσει κανείς να αναπαραστήσει την συμπεριφορά αντιδραστικών συστημάτων. Αποτελείται από ένα σύνολο καταστάσεων, ένα σύνολο μετατροπών μεταξύ των καταστάσεων και μια συνάρτηση η οποία σημειώνει σε κάθε κατάσταση το σύνολο των ιδιοτήτων της που είναι αληθείς. Τα μονοπάτια σε μία δομή Kripke μοντελοποιούν τους υπολογισμούς του συστήματος. Περνώντας στον ορισμό μιας τέτοιας δομής, μία Kripke δομή θα ορίζεται ως εξής :

Έστω AP ένα σύνολο ατομικών προτάσεων.
Μια *Kripke δομή* M πάνω στο AP ορίζεται ως μιας διατεταγμένη τετράδα $M = (S, S_0, R, L)$ όπου :

1.S αποτελει πεπερασμενο συνολο καταστασεων
2.$S_0 \subseteq S$ αποτελει συνολο αρχικων καταστασεων
3.$R \subseteq S \times S$ αποτελει σχεση μετατροπης, οπου για
καθε κατασταση $s \in S$ υπαρχει κατασταση $s' \in S$ ετσι
ωστε $R(s, s')$
4.$L : S \to 2^{AP}$ αποτελει συναρτηση η οποια αναθετει σε
καθε κατασταση ενα συνολο ατομικων προτασεων, αληθεις
για καθε κατασταση

Εικόνα 2.3: Η δομή Kripke

Ένα μονοπάτι σε μια τέτοια δομή *M* για μια κατάσταση s είναι μιας άπειρη ακολουθία καταστάσεων $\pi = s_0 s_1 s_2 \ldots$ όπου $s_0 = s$ και $R(s_i, s_{i+1})$ για κάθε $i \geq 0$.

2.4 Χρονική λογική (Temporal Logic)

Ορισμός : Η χρονική λογική περιγράφει την ακολουθία μετατροπών μεταξύ καταστάσεων και συστήματος, χωρίς να εισάγει ρητά το χρόνο. Αντί για το χρόνο, χρησιμοποιεί μια φόρμουλα που δείχνει ότι *τελικά* φθάνουμε σε μία κατάσταση. Ιδιότητες όπως *τελικά* ορίζονται με χρονικούς τελεστές. Η χρονική λογική μπορεί να έχει γραμμική ή branching δομή.

2.4.1 Γραμμική Χρονική Λογική (Linear Temporal Logic)

Στη γραμμική χρονική λογική (Linear Temporal Logic) ο χρόνος είναι γραμμικά διατεταγμένος και συνήθως μετράται με φυσικούς αριθμούς. Σε ένα γραμμικό πλαίσιο (S,R), το R είναι μία συνάρτηση που αναθέτει σε κάθε χρονικό στιγμιότυπο, το αμέσως επόμενο. Η χρονική διάταξη ≤ στην LTL είναι μία συνολική διάταξη. Για παράδειγμα για κάθε δύο χρονικά στιγμιότυπα s, t ∈ S θα είναι είτε s≤t είτε t≤s. Σε αυτό το σημείο πρέπει να δοθεί η σημασιολογία αυτής της λογικής: στο πλαίσιο (N, Succ) το N είναι το σύνολο των φυσικών αριθμών και το Succ {Succ(n) =n+1} είναι η συνάρτηση αυτού του συνόλου.

Σύνταξη

Η γλώσσα της γραμμικής χρονικής λογικής αποτελείται από τους παρακάτω τέσσερις τελεστές.

Σύμβολο	***ενέργειες***
○	unary τελεστής, διάβασε «την επόμενη φορά»
□	unary τελεστής, διάβασε «πάντα»
U	binary τελεστής, διάβασε «μέχρι»
□	unary τελεστής, διάβασε «τελικά»

Πίνακας 2.4.1: Τελεστές γραμμικής χρονικής λογικής

Αν τα f και g είναι τελεστές τότε είναι και τα ○*f*, □*f*, □*f*, *f* **U** *g*.

Σημασιολογία

Στην LTL η μετάφραση I=(N, Succ,I) αναθέτει μια πραγματική τιμή σε οποιαδήποτε LTL φόρμουλα σε οποιοδήποτε στιγμιότυπο s ∈ N με τον ακόλουθο τρόπο:

→ I(s,f)=I(s,f), □□□f ∈ P

1. *Λογικοί τελεστές:*
   - I(*s*, *f*□*g*) = I(*s*, *f*) □□I(*s*, *g*)
   - I(*s*, □*f*) = □□I(*s*, *f*)

2. *Χρονικοί τελεστές:*
   - I(*s*, ○*f*) = I(*s*+1, *f*)
   - I(*s*, *f* **U** *g*) = *true* iff □□*j*□*N* . I(*s*+*j*, *g*) = *true* και □□*0* □□*i* < *j*, I(*s*+*i*, *f*) = *true*
   - I(*s*, □*f*)= *true* iff I(*s*+*i*, *f*) = *true*, □□*i* □□*0*
   - I(*s*, □*f*) = *true* iff □□*j*□*N* . I(*s*+*j*, *f*) = *true*

Επαλήθευση

Έστω M=(S,q,P,L,R) μία δομή Kripke ενός προγράμματος. Κάθε μονοπάτι p=($s_o, s_1, s_2, \ldots$) στο M προσδιορίζει μία χρονική μετάφραση i ως μια ακολουθία αναθέσεων πραγματικών τιμών σε ατομικές προτάσεις στο P, με τον ακόλουθο τρόπο: σε κάθε στιγμιότυπο n€N η πρόταση m€P είναι αληθής στο n,iff m$\in$L(s_n).

Λέμε ότι ένα μονοπάτι p=($s_o, s_1, s_2 \ldots$) ικανοποιεί μία φόρμουλα f, αν η f είναι true στο so στη μετάφραση που προσδιορίζεται από το p. Επίσης ένα πρόγραμμα M ικανοποιεί μία φόρμουλα f, αν κάθε μονοπάτι που ξεκινάει από την αρχική κατάσταση q του M ικανοποιεί την f. Κατά συνέπεια ένα πρόγραμμα ικανοποιεί μία ιδιότητα της γραμμικής χρονικής λογικής, αν όλες οι πιθανές εκτελέσεις του προγράμματος ικανοποιούν την ιδιότητα αυτή.

Οι Clarke, Vardi και Wolper πρότειναν LTL model checking αλγόριθμους που είναι εκθετικοί στο μήκος της φόρμουλας, αλλά γραμμικοί στο μέγεθος του μοντέλου. Βασιζόμενοι σε αυτό το συμπέρασμα πολλοί ερευνητές συμφώνησαν ότι η LTL μπορεί να χρησιμοποιηθεί και για μικρές φόρμουλες. Όταν λέμε μήκος φόρμουλας εννοούμε τον αριθμό των συμβόλων (τελεστές, ατομικές προτάσεις και άλλα) που χρησιμοποιούνται για την αναπαράσταση της φόρμουλας.

Οι αλγόριθμοι που προτάθηκαν βασίζονται σε Buchi αυτόματα, τα οποία είναι πεπερασμένα αυτόματα που δέχονται άπειρες λέξεις. Αυτή η προσέγγιση έχει χρησιμοποιηθεί σε ένα μεγάλο αριθμό εργαλείων που κάνουν χρήση της γραμμικής

χρονικής λογικής. Η ιδέα είναι απλή: έχει διαπιστωθεί ότι μία LTL φόρμουλα f μπορεί να μετατραπεί σε Buchi αυτόματο που δέχεται ακριβώς τις άπειρες λέξεις που ικανοποιεί η δοθείσα φόρμουλα. Η μετατροπή μπορεί να γίνει με έναν αποδοτικό αλγόριθμο, αλλά επειδή στη χειρότερη περίπτωση το μέγεθος του αυτομάτου μεγαλώνει εκθετικά με το μήκος της φόρμουλας, η μέθοδος είναι καλύτερη για μικρές φόρμουλες.

Αξίζει να σημειωθεί ότι αν θέλουμε να κάνουμε τη μετατροπή της f σε Buchi, τότε κατασκευάζεται το αυτόματο B ¬f και αυτό γίνεται γιατί το αρνητικό f μας δίνει έναν πιο αποδοτικό αλγόριθμο model checking. Αν και στη χειρότερη περίπτωση το μέγεθος του χώρου καταστάσεων είναι ίσο με το καρτεσιανό γινόμενο του συστήματος με το B, ωστόσο στην καλύτερη είναι 0. Η τελευταία είναι όταν κανένα κομμάτι της ανεπιθύμητης συμπεριφοράς που αναπαρίσταται στο B, δεν εμφανίζεται στο σύστημα και επομένως το καρτεσιανό γινόμενο, που προαναφέραμε, θα είναι 0.

2.4.2 Branching Temporal Logic

Στη γραμμική χρονική λογική, όπως είδαμε παραπάνω, κάθε χρονικό στιγμιότυπο έχει ένα αμέσως επόμενο. Στη branching λογική είναι ένα άπειρα πεπερασμένο δένδρο, δηλαδή ένα δένδρο, όπου κάθε κόμβος έχει πεπερασμένο, μη μηδενικό αριθμό διαδοχικών στιγμιοτύπων. Επομένως ο γραμμικός χρόνος είναι μία ειδική περίπτωση του branching χρόνου. Στον τελευταίο η χρονική διάταξη είναι μερική διάταξη, όπου το προηγούμενο του κάθε στιγμιότυπου είναι γραμμικά διατεταγμένο: για οποιαδήποτε χρονικά στιγμιότυπα r,s,t αν r≤t και s≤t,τότε τα r και s πρέπει να είναι γραμμικά διατεταγμένα.

Ένα μονοπάτι σε ένα branching-time πλαίσιο (S,R) είναι το μέγιστο, γραμμικά διατεταγμένο, σύνολο χρονικών στιγμιότυπων στο S. Η branching λογική εισάγει ένα μη ντετερμινιστικό μοντέλο, το οποίο σημαίνει ότι κάθε χρονικό στιγμιότυπο μπορεί να έχει πολλές μελλοντικές καταστάσεις. Κάθε μία από τις καταστάσεις αυτές αντιστοιχεί σε ένα μονοπάτι που ξεκινάει από το t.

Η λογική αυτή χρησιμοποιεί δύο branching τελεστές κατά αναλογία με τη γραμμική χρονική λογική. Αυτοί είναι συνήθως ο «A» (για όλες τις πιθανές μελλοντικές καταστάσεις και εκφράζει *αναγκαιότητα* και ο «E» (εκφράζει πιθανότητα).

G → γενικά	□
F → μέλλον	◇
X → επόμενο	○
U → μέχρι	u

Πίνακας 2.4.2: Αναλογία branching linear temporal logic

Οι συντακτικοί κανόνες της branching χρονικής λογικής CTL (Computation Tree Logic) διασφαλίζουν ότι οι χρονικοί τελεστές είναι πάντα ζευγάρια «A» και «E» που ακολουθούνται από γραμμικό τελεστή.

Σύνταξη

Η σύνταξη της CTL ορίζεται ως ακολούθως:

- κάθε ατομική πρόταση στο P είναι μία CTL φόρμουλα
- αν τα f και g είναι CTL φόρμουλες, τότε είναι και τα □*f*, (*f* □□*g*), A ○ *f*, E ○ *f*, A(*f* u *g*), E(*f* u *g*).

Επίσης έχουμε και τις εξής ισοδυναμίες:

- *f* □□*g* = □(□□*f* □□□*g*)
- A□*g* = A(*true* u *g*)
- E□*g* = E(*true* u *g*)
- A□*f* = □E(*true* u □*f*)
- E□*f* = □A(*true* u □*f*)

Σημασιολογία

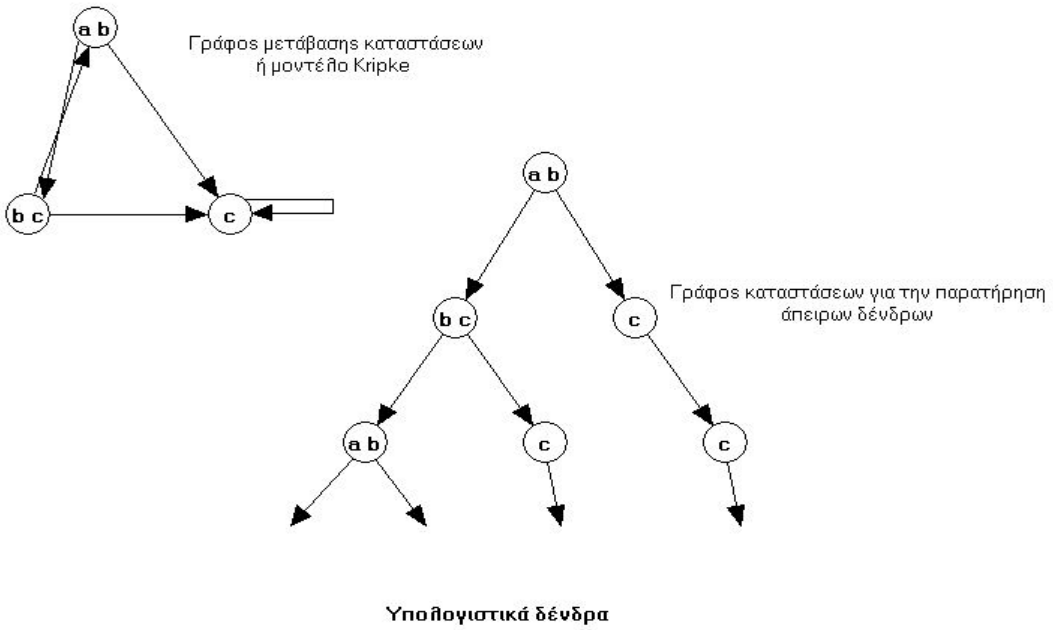
- I(*s*, *f*) = *I*(*s*, *f*), □□*f*□*P*
- I(*s*, □□*f*) = □□I(*s*, *f*)
- I(*s*, *f* □□*g*) = I (*s*, *f*) □□I (*s*, *g*)
- I(*s*0, A_*f*) = *true* iff *for all* paths (*s*0, *s*1, ...), I(*s*1, *f*) = *true*

- I(*s*0, E_*f*) = *true* iff *for some* path (*s*0, *s*1, ...), I(*s*1, *f*) = *true*
- I(*s*0, A(*f* U *g*)) = *true* iff *for all* paths (*s*0, *s*1, ...):
  □□*j*□*N* . I(*sj*, *g*) = *true* and □□*0* □□*i* < *j*, I(*si*, *f*) = *true*
- (*s*0, E(*f* U *g*)) = *true* iff *for some* path (*s*0, *s*1, ...):
  □□*j*□*N* . I(*sj*, *g*) = *true* and □□*0* □□*i* < *j*, I(*si*, *f*) = *true*

όπου τα s και s_i είναι χρονικά στιγμιότυπα στο S, $\forall$ i∈N

Επαλήθευση

Μία branching χρονική μετάφραση της μορφής I=*(T,R',I)* μπορεί να προκύψει εύκολα από μια πεπερασμένη δομή Kripke M=(S,q,P,L,R) ξεκινώντας από την αρχική κατάσταση q και ξεδιπλώνοντας το M σε ένα άπειρα πεπερασμένο δένδρο. Για κάθε χρονικό στιγμιότυπο t∈T και πρόταση m ∈ P είναι:
I(t,m)=true, iff m∈L(s) για την κατάσταση s του M για κάθε χρονική στιγμή t.
Με άλλα λόγια η συνάρτηση μετάφρασης αναθέτει σε κάθε χρονικό στιγμιότυπο αυτών των προτάσεων, οι οποίες είναι αληθείς στην κατάσταση της δομής Kripke που αντιστοιχεί στο στιγμιότυπο αυτό. Λέμε ότι το M ικανοποιεί μια CTL φόρμουλα f, αν I(q,f)=true.



Εικόνα 2.4.2: Υπολογιστικά δένδρα

Ο έλεγχος μοντέλων CTL μπορεί να πραγματοποιηθεί με έναν αλγόριθμο που είναι γραμμικός στο μήκος της φόρμουλας, αλλά και στο μέγεθος του μοντέλου Kripke του συστήματος. Στη συνέχεια θα εξετάσουμε μία προσέγγιση:

Έστω M=(S,q,P,L,R) το μοντέλο Kripke του συστήματος και Pred(S) δείχνει το πλέγμα των κατηγορημάτων του S,όπου κάθε κατηγόρημα προσδιορίζεται ως το σύνολο καταστάσεων του S που είναι true. Έτσι το μικρότερο στοιχείο του πλέγματος είναι το κενό σύνολο και δείχνεται με false και το μεγαλύτερο είναι το σύνολο όλων των καταστάσεων του S και δείχνεται με true. Μία συνάρτηση F από το Pred(S) στο Pred(S) ονομάζεται κατηγορηματικός μετασχηματισμός. Αν δούμε κάθε μία CTL φόρμουλα f ως ένα κατηγορηματικό προσδιοριστικό με τις καταστάσεις του M που ικανοποιούν την f,τότε κάθε ένας από τους βασικούς τελεστές της CTL μπορεί να χαρακτηριστεί ως ένα σταθερό σημείο ενός μονότονου κατηγορηματικού μετασχηματισμού. Για παράδειγμα, το E□p χαρακτηρίζεται ως μικρότερο σταθερό σημείο της συνάρτησης Z[p $\vee$ E□Z], όπου το Z είναι μια μεταβλητή που μπορεί να αλλάζει θέση σε μια συνάρτηση.

Για πεπερασμένα πεδία τα μικρότερα και μεγαλύτερα σταθερά σημεία μιας μονότονης συνάρτησης μπορούν να υπολογιστούν αποδοτικά. Η συνάρτηση εκτελείται σε κύκλους μέχρι να βρεθεί ένα σταθερό σημείο: στο πρώτο κύκλο γίνεται false ή true ανάλογα αν έχουμε μικρότερο ή μεγαλύτερο σταθερό σημείο και κάθε κύκλος εκτελείται πάνω στο αποτέλεσμα του προηγούμενου. Επειδή το πεδίο είναι πεπερασμένο, η διαδικασία είναι βέβαιο ότι θα τερματιστεί.

Επομένως ο αλγόριθμος model checking υπολογίζει, για μια δοθείσα Kripke δομή M και μια CTL φόρμουλα f, το σύνολο των καταστάσεων $S_i \in S$, όπου το f υπάρχει. Λέμε τότε ότι το M ικανοποιεί το f αν η αρχική κατάσταση q ανήκει στο S_i.

2.4.3 Σύγκριση CTL και LTL

Γενικά τα προγράμματα μοντελοποιούνται ως μη ντετερμινιστικά συστήματα μετατροπής. Ο μη ντετερμινισμός οφείλεται, είτε στο ότι τα προγράμματα είναι παράλληλα, είτε στην απουσία πληροφορίας για τη συμπεριφορά κάποιων στοιχείων του συστήματος ή του περιβάλλοντός του.

1. Όπως είδαμε προηγουμένως, η branching προσέγγιση στη χρονική λογική είναι ρητά μη ντετερμινιστική, αφού το χρονικό μοντέλο είναι ένα δένδρο, του οποίου κάθε κόμβος μπορεί να ενώνεται με οποιοδήποτε άλλο και δεν είναι αυστηρά καθορισμένος. Αντιθέτως, στη γραμμική λογική το πρόγραμμα αντιμετωπίζεται ως ένα σύνολο από πιθανές μελλοντικές καταστάσεις και οι φόρμουλες μεταφράζονται στις εκτελέσεις του προγράμματος, που πραγματοποιούνται γραμμικά στο χρόνο. Άρα ο με ντετερμινισμός δεν είναι ρητά καθορισμένος στην LTL.
2. Υπάρχουν πολλές ιδιότητες στην CTL που δε μπορούν να εκφραστούν άμεσα στην LTL. Για παράδειγμα, υποθέστε την ιδιότητα A□E□start στην CTL, που δείχνει ότι ανεξάρτητα σε ποια κατάσταση μπαίνει το πρόγραμμα, υπάρχει ένας υπολογισμός που το οδηγεί ξανά στην αρχική. Ούτε αυτή η ιδιότητα, αλλά ούτε και η άρνησή της μπορούν να εκφραστούν στην LTL. Από την άλλη πλευρά όμως, μια απλή και συχνά χρησιμοποιούμενη LTL φόρμουλα □□p εκφράζεται στην CTL πολύπλοκα, με τη μορφή A□A□p. Γενικά οι CTL φόρμουλες είναι μεγαλύτερες και πιο πολύπλοκες, γιατί οι branching τελεστές πρέπει να προηγούνται από τους γραμμικούς.

2.4.4 Πλεονεκτήματα των buchi αυτομάτων

Τα αυτόματα buchi έχουν αρκετά πλεονεκτήματα συγκρινόμενα με τη χρονική λογική:

1. Πρώτα από όλα είναι ικανά να εκφράζουν πιθανότητα και fairness. Αυτό έχει ως αποτέλεσμα και το σύστημα και τα χαρακτηριστικά του να εκφράζονται ομοιόμορφα. Στην LTL,αλλά και στην CTL πρέπει να εισάγονται buchi αυτόματα για το χειρισμό fairness.
2. Επιπλέον, η εκφραστική ικανότητα των buchi, όσον αφορά τις προτάσεις και τις φόρμουλες, είναι πολύ μεγαλύτερη από αυτή της χρονικής λογικής.

Οι ιδιότητες μπορούν να εκφραστούν με LTL φόρμουλες και έπειτα να μετατραπούν σε buchi, ή να εκφραστούν από την αρχή σε buchi. Οι περισσότερες ιδιότητες που ο μέσος χρήστης ενός εργαλείου model checking θέλει να ελέγξει, μπορούν να εκφραστούν και με τους τρεις προαναφερθείσες φορμαλισμούς. Επομένως η επιλογή του φορμαλισμού εξαρτάται πιο πολύ από την ευχρηστία του, παρά από την εκφραστική του δύναμη. Από τους φορμαλισμούς της χρονικής λογικής, η γραμμική προσέγγιση χρησιμοποιείται κυρίως όταν οι ιδιότητες που θα ελεγχθούν σχετίζονται με τις εκτελέσεις του προγράμματος, ενώ η branching προσέγγιση όταν οι ιδιότητες σχετίζονται με τη δομή του προγράμματος.

Συγκρινόμενη με τα αυτόματα, η χρονική λογική μπορεί να είναι πιο εύχρηστη για έκφραση απλών ιδιοτήτων, αλλά όσο γίνονται πιο πολύπλοκες τόσο πιο δύσκολα εκφράζονται. Από την άλλη πλευρά τα buchi αντιμετωπίζουν το σύστημα και τις ιδιότητες με ομοιόμορφο τρόπο. Επομένως είναι χρήσιμο για μία μέθοδο να μπορεί να χειρίζεται και τους δύο φορμαλισμούς.

2.5 Εισαγωγή στις Τυπικές Μεθόδους (Formal Methods)

Όλες οι παραδοσιακές πειθαρχίες που αφορούν τον κλάδο της μηχανολογίας βασίζονται σε μαθηματικά μοντέλα και υπολογισμούς ,έτσι ώστε να παρθούν ευκολότερα οι αποφάσεις για τον σχεδιασμό. Ο όρος «Επίσημη Μέθοδος» αναφέρεται στην ποικιλία των μαθηματικών τεχνικών μοντελοποίησης που είναι εφαρμόσιμες στον σχεδιασμό συστημάτων για υπολογιστές. Ακολουθεί ο ορισμός για την μια επίσημη μέθοδο :

«*Οι επίσημες μέθοδοι που χρησιμοποιούνται για την ανάπτυξη συστημάτων υπολογιστών βασίζονται σε μαθηματικές τεχνικές για την περιγραφή των προδιαγραφών των συστημάτων αυτών.*»

Έτσι μια μέθοδος θεωρείται επίσημη εάν αυτή στηρίζεται πάνω σε μια μαθηματική βάση η οποία και προέρχεται από μία επίσημη γλώσσα αναπαράστασης προδιαγραφών (formal specification language). Η βάση αυτή παρέχει ένα σύνολο από ακριβής ορισμένες έννοιες όπως συνέπεια, πληρότητα, προδιαγραφή, εφαρμογή και ορθότητα. Στις σημερινές επίσημες μεθόδους υπάρχει μια ταξινόμηση επιπέδων ανάλογα με τον τρόπο αναπαράστασης των προδιαγραφών του συστήματος. Έτσι θα έχουμε :

- 1ο Επίπεδο : Εφαρμογή των επίσημων μεθόδων σε όλα τα μέρη του συστήματος
- 2ο Επίπεδο : Εφαρμογή των επίσημων μεθόδων με δύο ή περισσότερα επίπεδα αφαίρεσης του συνόλου καταστάσεων χρησιμοποιώντας γραπτές αποδείξεις ότι στην συγκεκριμένη προδιαγραφή μπορούν να εφαρμοστούν τεχνικής μείωσης
- 3ο Επίπεδο : Εφαρμογή επίσημων μεθόδων στην περίπτωση προδιαγραφής η οποία ελέγχεται από ελεγκτή θεωρητικής μηχανικής

Υπάρχουν δύο τύποι ελεγκτών θεωρητικών μηχανικών :
Αυτόματοι : Προσπαθούν να αποδείξουν την συγκεκριμένη προδιαγραφή χωρίς ανθρώπινη παρέμβαση (ακριβή σε σχέση με τους μηχανικού πόρους)
Διαλογικοί : Προσπαθούν να αποδείξουν την συγκεκριμένη ιδιότητα με την καθοδήγηση του χρήστη (ακριβή σε σχέση με τους ανθρώπινους πόρους)

2.5.1 Έλεγχος με τις Επίσημες Μεθόδους (Formal Methods)

Μία ‘επίσημη ιδιότητα’ αποτελεί μια ακριβής, ολοκληρωμένη, επαρκής και μη διφορούμενη βάση τόσο για τον σχεδιασμό και την κωδικοποίηση ενός συστήματος όσο και για τον έλεγχό του. Αυτό πρόκειται για ένα μεγάλο πλεονέκτημα σε σχέση με τις παραδοσιακές διεργασίες ελέγχου. Ένα δεύτερο πλεονέκτημα της χρήσης των επίσημων

μεθόδων για έλεγχο είναι η ικανότητά τους να τον πραγματοποιούν με την βοήθεια εργαλείων. Οι αλγόριθμοι που χρησιμοποιούν τα εργαλεία αυτά, έχουν να κάνουν με τον αυτόματο έλεγχο συστημάτων που είναι αρκετά γρήγορος και με λιγότερα λάθη. Κάτι τέτοιο ανοίγει τον δρόμο για την περίπτωση ελέγχου που το μόνο που χρειάζεται είναι η επίσημη ιδιότητα του συστήματος που εξετάζεται και το ίδιο το σύστημα. Όλοι αυτοί οι έλεγχοι μπορούν αυτόματα να αποδειχθούν εάν είναι σωστοί ή όχι.

2.5.2 Επίσημος έλεγχος και επαλήθευση (Formal Check and Verification)

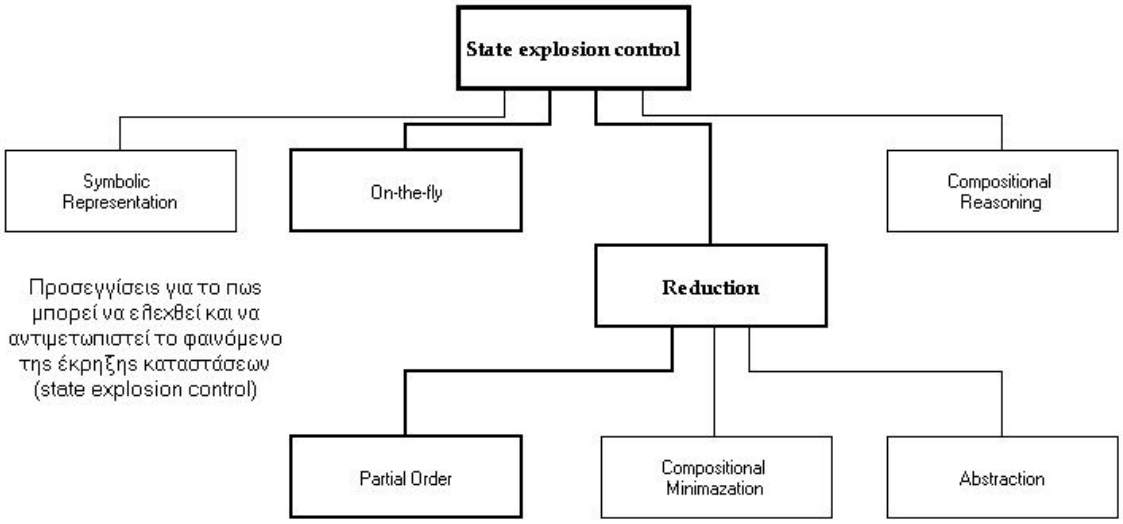
Ο επίσημος έλεγχος και η επίσημη επαλήθευσή τους αποτελούν συμπληρωματικές τεχνικές για ανάλυση και έλεγχο της ορθότητας συστημάτων. Ενώ η επαλήθευση έχει ως στόχο της την απόδειξη ιδιοτήτων συστημάτων που έχουν μοντελοποιηθεί με βάση μαθηματικούς όρους, ο έλεγχος εφαρμόζεται με το να πραγματοποιεί την πραγματική εκτέλεση της εφαρμογής ή την εκτέλεση μιας προσομοίωσης του μοντέλου. Επίσης η επαλήθευση μπορεί να δώσει απάντηση εάν μια ιδιότητα ικανοποιείται αρκεί το σύστημα να είναι σωστά μοντελοποιημένο : ισχύει ότι μια επαλήθευση είναι σωστή πάντα όταν είναι σωστό και το μοντέλο που αναπαριστά το σύστημα. Ο έλεγχος που βασίζεται μόνο με την παρατήρηση ενός μικρού συνόλου καταστάσεων ενός συστήματος ποτέ δεν είναι επαρκής. Ο έλεγχος μπορεί να δείξει μόνο την παρουσία λαθών και όχι την βεβαίωση ότι δεν πρόκειται να υπάρξουν. Αλλά από την στιγμή που ο έλεγχος μπορεί να εφαρμοστεί σε πραγματικές εφαρμογές, είναι χρήσιμο σε αυτές τις περιπτώσεις όταν ένα αξιόπιστο μοντέλο είναι δύσκολο να δημιουργηθεί για ένα σύστημα εξαιτίας της πολυπλοκότητας ή εξαιτίας του ότι το σύστημα μπορεί να αποτελείται από μέρη δύσκολα να μοντελοποιηθούν.

2.5.3 Τυπική Επαλήθευση (Formal Verification)

Η επίσημη επαλήθευση αποτελεί μια εναλλακτική οικογένεια μεθόδων που χρησιμοποιούν τεχνικές βασισμένες πάνω σε μια μαθηματική λογική με σκοπό να διασφαλίσουν την ποιότητα συστημάτων υλικού ή λογισμικού. Ο σχεδιασμός της επαλήθευσης που χρησιμοποιεί μια επίσημη λογική κατά τον σχεδιασμό των προδιαγραφών ενός συστήματος, παρέχοντας ένα δομημένο πλαίσιο για την μετάφραση υψηλού επιπέδου προδιαγραφών σε μια συλλογή διακριτών στοιχείων που περιγράφονται από αλγόριθμους, εύκολα κατανοητά από τους προγραμματιστές. Η επαλήθευση ενός προγράμματος εμπλέκει ουσιαστικά αποδείξεις ορθότητας αυτών. Πολλές εταιρείες (και όσο περνά ο καιρός ακόμα περισσότερες) εισάγουν την επαλήθευση στα συστήματά τους σαν ένα ξεχωριστό και πολύ σημαντικό μέρος του προϊόντος τους. Η αφοσίωση που δίνεται πάνω σε όλες τις μεθόδους επαλήθευσης έχει προσθέσει αρκετά επίπεδα όσο αφορά την αξιόπιστη παραγωγή του λογισμικού. Βοηθούν τον σχεδιαστή να επικεντρώσει πάνω στο θέμα του τι κάνει το λογισμικό και όχι τον τρόπο με τον οποίο το κάνει. Γενικά οι μέθοδοι αυτοί μπορούν να θεωρηθούν σαν ένα σύνολο διαφορετικών τεστ από όπου πρέπει να περάσει το προϊόν, όπως τον έλεγχο μοντέλων, έλεγχο κώδικα και αρκετές άλλες.

2.5.4 Το πρόβλημα της έκρηξης καταστάσεων (State Explosion Problem)

Το κύριο πρόβλημα της τεχνικής του έλεγχου μοντέλων είναι ότι ο χώρος καταστάσεων ενός concurrent συστήματος μπορεί να είναι αρκετά μεγάλος σε μέγεθος. Για παράδειγμα, ένα σύστημα αποτελούμενο από n διεργασίες όπου η κάθε μία μπορεί να έχει m καταστάσεις, θα έχει έναν χώρο καταστάσεων n*m. Ο αριθμός αυτός των καταστάσεων περιορίζει την εφαρμοσιμότητα της τεχνικής ελέγχου, αφού όλες οι μετατροπές του συστήματος γίνονται μεγάλες (σε μερικές περιπτώσεις άπειρες) κάνοντας αρκετά δύσκολο (ή αδύνατο) την κατασκευή των μοντέλων τους. Το φαινόμενο αυτό είναι γνωστό ως το φαινόμενο της *έκρηξης του ΄χώρου καταστάσεων (State explosion problem).*



Εικόνα 2.5.4 : Το πρόβλημα της έκρηξης του χώρου καταστάσεων

Αρκετοί ερευνητές έχουν προσπαθήσει να μετριάσουν το πρόβλημα βγάζοντας ως συμπέρασμα όμως ότι δεν υπάρχει κάποια γενική λύση για την αντιμετώπισή του. Στο σημείο αυτό θα παρουσιαστούν μερικές διαφορετικές στρατηγικές που αναφέρονται στην βιβλιογραφία και κρίνονται ως ικανές για την αντιμετώπιση της έκρηξης του χώρου καταστάσεων. Συγκεκριμένα οι στρατηγικές αυτές μπορούν να διακριθούν σε δύο κατηγορίες :

- Τις τεχνικές αυτές οι οποίες προσπαθούν να μοντελοποιήσουν μόνο τις καταστάσεις εκείνες του συστήματος που θεωρούνται αναγκαίες για την επαλήθευση της συγκεκριμένης προδιαγραφής που εξετάζεται
- Τις συμβολικές τεχνικές οι οποίες αναπαριστούν συμβολικά της καταστάσεις ενός συστήματος αντί να τις απαριθμούν με ακρίβεια

Μερικές τεχνικές οι οποίες ανήκουν στην πρώτη κατηγορία είναι οι ακόλουθες:

- Τεχνικές μερικής ταξινομημένης μείωσης (Partial Order Reduction)
- On – The – Fly τεχνικές

- Συμμετρικές τεχνικές
- Τεχνική της Αφαίρεσης (Abstraction)[1]

Η δεύτερη κατηγορία των τεχνικών που προσπαθούν να αντιμετωπίσουν το φαινόμενο της έκρηξης καταστάσεων είναι και η προσέγγιση του συμβολικού έλεγχου μοντέλων. Η ιδέα αυτής της τεχνικής είναι η αναπαράσταση όλων των καταστάσεων και μετατροπών του συστήματος που μοντελοποιεί το πρόγραμμα. Συχνά η αναπαράσταση αυτή πραγματοποιείται με τα *δυαδικά γραφήματα απόφασης (Binary Decision Diagrams, BDDs).* Πρώτος ο McMillan το 1993 στην διατριβή του ήταν αυτός ο οποίος εισήγαγε την έννοια συμβολικός έλεγχος μοντέλων. Τέλος αξίζει να σημειωθεί ότι η συμβολική αυτή τεχνική μας επιτρέπει να αντιμετωπίσουμε αρκετά πολύπλοκα συστήματα σε αντίθεση με τις άλλες τεχνικές που περιορίζονται σε αρκετά μικρό χώρο καταστάσεων.

Παράδειγμα 1 της έκρηξης του χώρου καταστάσεων

Έστω το παρακάτω κομμάτι κώδικα το οποίο χρησιμοποιεί μια εντολή επανάληψης :

```
void main (void)
{
int i = 0;
while (1) i++;
}
```

Κατά την εκτέλεση αυτού, η main θα παρουσιάσει 232 καταστάσεις σε ένα 32-bit μηχάνημα. Οι καταστάσεις αυτές είναι πάρα πολλές!! Μια προσομοίωση του κομματιού αυτού απλώς θα δημιουργούσε έναν ατέρμονα βρόχο, αλλά ένας ελεγκτής μοντέλων θα έψαχνε να βρει όλες τις πιθανές καταστάσεις του συστήματος.

Παρακάτω παρουσιάζεται ένας ενδεικτικός πίνακας ο οποίος δείχνει την σχέση διαφόρων μεθόδων ελέγχου με κάποια σημαντικά κριτήρια που αφορούν την ανάπτυξη συστημάτων υλικού και λογισμικού.

Κριτήρια Μεθόδου	Έλεγχος	Επαλήθευση	Έλεγχος Μοντέλων
Μέγεθος Συστήματος	Μικρό ώς πολύ μεγάλο	Παραδείγματα Παιχνιδιών	100 – 1000 γραμμές κώδικα
Χρόνος	Ανάλογα διάρκεια ανάπτυξης	Μέρες – Βδομάδες	Λεπτά – Ώρες

[1] Η τεχνικής της αφαίρεσης ουσιαστικά ανήκει και στις δύο κατηγορίες. Εάν η μέθοδος επαληθεύει ένα απλό μοντέλο που αναπαρίσταται με την αφαίρεση κάποιων καταστάσεών του που θεωρούνται άσχετες με την προδιαγραφή που ελέγχεται, τότε η τεχνική αυτή θα ανήκει στην πρώτη κατηγορία. Σε αντίθετη περίπτωση εάν έχει ήδη κατασκευαστεί το μοντέλο ενός συστήματος και στη συνέχεια εφαρμόζεται η τεχνικής της μείωσης για να μειωθεί ο αριθμός των μετατροπών του συστήματος, τότε θα ανήκει στην δεύτερη κατηγορία

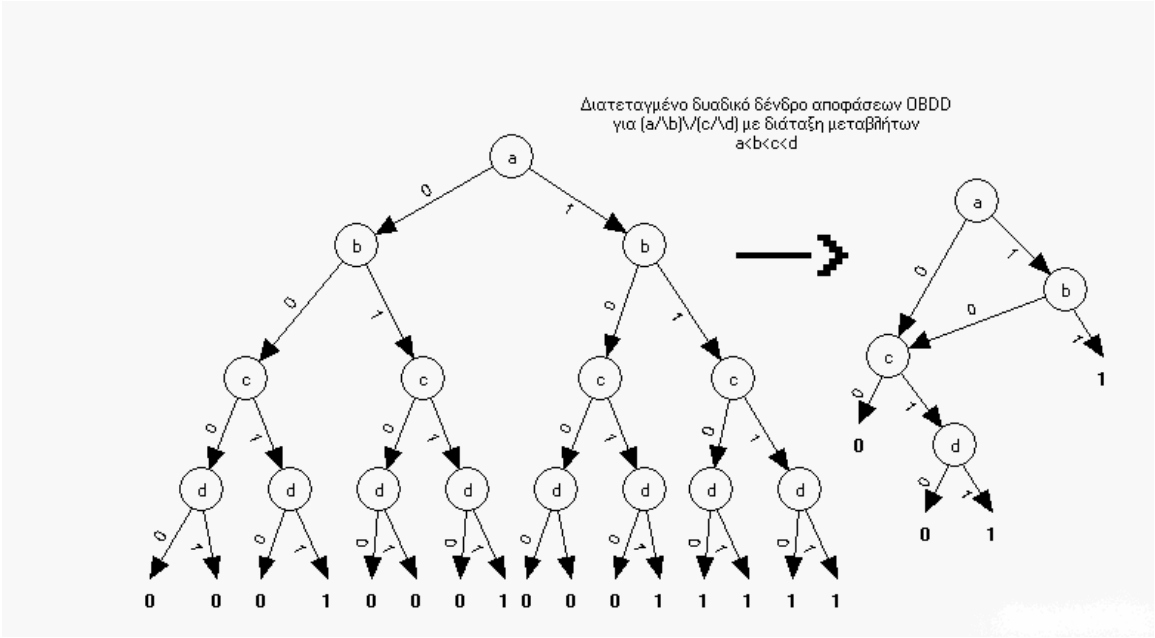
Ειδικότητα χρηστών	Έμπειροι προγραμματιστές	Μαθηματικοί, Επιστήμονες Λογικής	Επιστήμονες Λογικής
Χρήση σήμερα	Υλικό και λογισμικό	Έρευνα	Υλικό και λογισμικό
Προδιαγραφές	Ανάλογα επιθυμίας χρήστη	Λογικές ή βασισμένες σε αυτόματα	Λογικές ή βασισμένες σε αυτόματα
Μοντελοποίηση και αλλαγές	Εφαρμόζεται απευθείας	Επιβάλλεται κάτα την πραγματοποίηση	Επιβάλλεται κατά την πραγματοποίηση

Πίνακας 2.5.4: σχέση διαφόρων μεθόδων ελέγχου με κάποια σημαντικά κριτήρια που αφορούν την ανάπτυξη συστημάτων

2.6 Συμβολική Αναπαράσταση

Οι αλγόριθμοι ελέγχου μοντέλων, όπως αναφέρθηκε και παραπάνω, συναντούν δυσκολίες από το φαινόμενο της 'έκρηξης των καταστάσεων'. Παρακάτω θα συζητηθούν μερικές περιπτώσεις λύσεις οι οποίες μπορούν να μετριάσουν το φαινόμενο αυτό. Η συμβολική αναπαράσταση βασίζεται στην αναπαράσταση ενός πεπερασμένου μοντέλου που εκφράζει ένα σύστημα. Η πιο συνηθισμένη αναπαράσταση πραγματοποιείται με μία τεχνική η οποία αναφέρεται ως μια επαρκής κωδικοποίηση συναρτήσεων τύπου Boolean γνωστή ως OBDD, (**O**rdered **B**inary **D**ecision **D**iagrams). Οι αναπαραστάσεις OBDD έχουν τρία κύρια πλεονεκτήματα :

- Είναι αρκετά μικρές για την αναπαράσταση μεγάλων κλάσεων
- Είναι κανονικοποιημένες για μια δοσμένη διάταξη μεταβλητών εισόδου
- Μπορούν απευθείας να επεξεργαστούν κάτω από όλες τις βασικές Boolean συναρτήσεις



Εικόνα 2.6: Διατεταγμένα Δυαδικά δένδρα αποφάσεων

Ένα OBDD είναι παρόμοιο με ένα δυαδικό δένδρο αποφάσεων ,με την διαφορά ότι η δομή του αποτελεί έναν απευθείας μη-κυκλικό γράφο παρουσιάζοντας μία αυστηρή συμπεριφορά διάταξης πάνω στις μεταβλητές ,καθώς γίνεται προσπέλαση του γράφου από την ρίζα προς τα φύλλα. Ειδικότερα ,η αναπαράσταση OBDD μιας Boolean συνάρτησης f ,γίνεται με την μείωση μιας σχετικής δομής που ονομάζεται δυαδικό δένδρο αποφάσεων. Η απόκτηση πραγματικής τιμής μπορεί να γίνει με την προσπέλαση του δένδρου από την ρίζα ως τα φύλλα. Σε κάθε κόμβο ,η τιμή της κάθε μεταβλητής είναι αυτή η οποία θα αποφασίσει για το ποιο μονοπάτι θα ακολουθηθεί στην συνέχεια : μπορείς να κατέβεις στο δένδρο σε αριστερό/δεξιό παιδί εάν η τιμή της μεταβλητής που φέρει ο κόμβος είναι αντίστοιχα λάθος/σωστό ,(τιμή 0/1). Η μεταβλητές τοποθετούνται στο δένδρο σε αύξουσα σειρά από την ρίζα προς τα φύλλα. Το δυαδικό δένδρο αποφάσεων μπορεί να μειωθεί σε ένα OBDD με τον συνδυασμό οποιονδήποτε ισομορφικών υποδένδρων σε ένα απλούστερο, εξαλείφοντας με αυτόν τον τρόπο κόμβους όπου τα αριστερά και τα δεξιά τους παιδιά είναι όμοια (εικόνα 2.).

Το πεπερασμένο μοντέλο καταστάσεων ενός συστήματος μπορεί να εκφραστεί με την μορφή OBDD ως εξής : Κάθε κατάσταση κωδικοποιείται με μία ανάθεση τιμών τύπου Boolean σε μια ομάδα μεταβλητών καταστάσεων του συστήματος. Εάν οι μεταβλητές δεν παίρνουν τιμές 0 και 1 αλλά ορίζονται μέσα σε ένα πεδίο D τότε πρέπει να γίνει χρήση μιας συγκεκριμένης δυαδικής κωδικοποίησης πάνω στο πεδίο τιμών D. Αυτή η διεργασία μπορεί να γίνει χωρίς επενέργεια του χρήστη αλλά με εργαλεία τα οποία υποστηρίζουν συμβολική αναπαράσταση (για παράδειγμα το εργαλείο SMV). Στη συνέχεια με τη σχέση μετατροπής μπορεί τώρα να εκφραστεί σαν μια Boolean συνάρτηση δύο μεταβλητών. Μία η οποία θα αναπαριστά την κωδικοποίηση της τρέχουσας κατάστασης και μια δεύτερη η οποία θα κωδικοποιεί την καινούργια κατάσταση. Με την παραπάνω διεργασία σχηματίζεται μια OBDD αναπαράσταση.

Ο συμβολικός έλεγχος μοντέλων ,ελέγχει απευθείας temporal τύπους πάνω σε OBDD αναπαραστάσεις του μοντέλου του συστήματος. Για παράδειγμα, ο αλγόριθμος που αφορά την μέθοδο CTL ,χρησιμοποιεί OBDD αναπαραστάσεις για τις σχέσεις μετάβασης (transition relation) και τις ατομικές προτάσεις (atomic propositions), επιστρέφοντας την αναπαράσταση (OBDD) εκείνη η οποία αντιπροσωπεύει τις καταστάσεις του συστήματος ,ανάλογα με τον temporal τύπο που δόθηκε αρχικά. Όλοι οι χειρισμοί που απαιτούνται από τον αλγόριθμο ,περιλαμβάνοντας ακόμα και συγκρίσεις αναπαραστάσεων OBDDs ανάλογα με τον έλεγχο του επιθυμητού σημείου, μπορούν να εφαρμοστούν με ευκολία πάνω σε OBDDs. Η προσέγγιση αυτή έχει ήδη εφαρμοστεί με επιτυχία όσον αφορά τον έλεγχο μοντέλων με μέθοδο CTL. Από την άλλη μεριά , τόσο οι μέθοδοι ελέγχου μοντέλων LTL όσο και οι μέθοδοι που βασίζονται στην θεωρία αυτομάτων ,μπορούν να επωφεληθούν από την συμβολική αναπαράσταση του μοντέλου του συστήματος.

Συμπερασματικά ,βάση της συμβολικής αυτής προσέγγισης ,αποφεύγεται η δημιουργία γράφου καταστάσεων του concurrent συστήματος. Το θέμα μας πλέον επικεντρώνεται στο μέγεθος της OBDD αναπαράστασης και όχι στο μέγεθος του χώρου καταστάσεων. Με την προϋπόθεση ότι βρίσκεται ένας τακτικός-επαναλαμβανόμενος χώρος καταστάσεων ,είναι δυνατόν να γίνει στην συνέχεια επαλήθευση συστημάτων (ιδιαίτερα hardware) πολύ μεγαλύτερου μεγέθους από ότι τα συστήματα τα οποία θα επαληθεύονταν με ακριβή αναπαράσταση του χώρου καταστάσεων.

2.7 Επαλήθευση με την μέθοδο On-the-fly

Είναι γνωστό ότι η ανάλυση reachability αποτελεί μια τεχνικής επαλήθευσης η οποία εφαρμόζει μια εξαντλητική εξερεύνηση όλων εκείνων των καταστάσεων και μεταβάσεων που φτάνει ένα σύστημα κατά την εκτέλεσή του. Οι On-the-fly τεχνικές ,ναι μεν βασίζονται στην εφαρμογή της reachability ανάλυσης , αλλά δεν υποχρεώνονται να αποθηκεύσουν ολόκληρο τον γράφο καταστάσεων του συστήματος. Στην περίπτωση ειδικά του προβλήματος της έκρηξης καταστάσεων, θα ήταν αδύνατο να αποθηκευθεί ένας τέτοιος μεγάλος γράφος. Την λύση στο πρόβλημα θα έδινε μια προσομοίωση όλων των δυνατόν μεταβάσεων που είναι ικανό το σύστημα να παρουσιάσει. Έτσι, μια απλή αναζήτηση depth-first μπορεί να χρησιμοποιηθεί για να εξερευνηθεί το σύστημα με την μέθοδο On-the-fly, δηλαδή χωρίς την αποθήκευση των μεταβάσεων που λαμβάνουν χώρα κατά την διάρκεια της αναζήτησης.

Όταν εφαρμόζεται η αναζήτηση depth-first σε έναν γράφο, ο ελάχιστος χώρος αποθήκευσης που απαιτείται ,είναι εκείνος ο χώρος του συγκεκριμένου μονοπατιού που εξετάζεται. Κάτι τέτοιο είναι δυνατόν να εξασφαλίσει μείωση της απαιτούμενης μνήμης ενώ παράλληλα εγγυάται για μια εξαντλητική εξερεύνηση στον χώρο καταστάσεων.

Παρόλο αυτά όμως ,ο χρόνος που χρειάζεται για την επαλήθευση του συστήματος , μπορεί να μεγαλώσει δραματικά εξαιτίας της εξερεύνησης του αλγορίθμου σε καταστάσεις που ήδη έχει επισκεφθεί. Κάτι τέτοιο θα λύνονταν μόνο στην περίπτωση της αποθήκευσης όλων των επισκεφθέντων καταστάσεων μόλις αυτές προσπελάζονται. Αλλά στην περίπτωση μεγάλων reachability γράφων, θα ήταν αδύνατον να αποθηκευθούν όλες οι καταστάσεις του συστήματος. Αρκετές είναι οι τεχνικές που έχουν προταθεί και οι οποίες προσπαθούν να συνεργαστούν με τις παραπάνω στρατηγικές. Έτσι ,για την αποθήκευση του τρέχοντος μονοπατιού, μία μνήμη (cache) δημιουργείται για την αποθήκευση επιλεγμένων επισκεφθέντων καταστάσεων. Αρχικά όλες οι καταστάσεις που έχουν επισκεφθεί θα αποθηκεύονται σε αυτήν την μνήμη ,μέχρι αυτή να γεμίσει. Όταν συμβεί αυτό ,οι πιο παλιές καταστάσεις αντικαθίστανται από καινούργιες. Η αποτελεσματικότητα της μνήμης αυτής (state-space caching) ,εξαρτάται από το μέγεθός της αλλά επίσης και από την δομή του χώρου καταστάσεων. Το τελευταίο αποτελεί αρκετά δύσκολο εγχείρημα, αφού είναι αρκετά δύσκολη η πρόβλεψη της δομής αυτής. Επίσης μια τέτοια πρόβλεψη θα μπορούσε να οδηγήσει σε αδιέξοδο όσον αφορά την επιλογή της cache μνήμης, αφού μια λανθασμένη επιλογή θα μεγάλωνε παρά πολύ τον χρόνο της εκτέλεσης της αναζήτησης.

Μια τεχνική η οποία μπορεί να εφαρμοστεί στην περίπτωση όπου το μέγεθος του προβλήματος είναι τέτοιο ,που απαγορεύει εξαντλητική επαλήθευση ,είναι και αυτή με το όνομα *bit-state hashing* ή αλλιώς *supertrace* και η οποία εφαρμόζει μια μερική αναζήτηση του χώρου καταστάσεων. Στην τεχνική αυτή ,όλες οι καταστάσεις που έχουν επισκεφθεί ,αποθηκεύονται σε έναν πίνακα κατακερματισμού (hash) H, του οποίου το μέγεθος εξαρτάται από την μνήμη που είναι ελεύθερη. Για κάθε κατάσταση s, χρησιμοποιείται ένα απλό bit με διεύθυνση h(s) ,όπου το h είναι μια συνάρτηση κατακερματισμού ,η οποία επιστρέφει τις bit-διευθύνσεις στο H. Εάν το bit στην διεύθυνση h(s) έχει τιμή 1 ,τότε ο αλγόριθμος αναζήτησης υποθέτει ότι η κατάσταση s έχει ήδη επισκεφθεί. Έτσι από την στιγμή που δεν ανιχνεύεται κάποια σύγκρουση των καταστάσεων, η αναζήτηση γίνεται μερική. Επίσης αξίζει να σημειωθεί ότι τεχνική

μπορεί να επεκταθεί με την χρησιμοποίηση αυτής για αρκετές φορές, όπου σε κάθε μια από αυτές θα εφαρμόζονται διαφορετικές συναρτήσεις κατακερματισμού (μέχρι το σημείο εκείνο που το επίπεδο κάλυψης του αλγορίθμου θα είναι ικανοποιητικό). Κάτι τέτοιο δεν αποτελεί πρόβλημα ,αφού ο περιοριστικός παράγοντας στην reachability ανάλυση είναι συνήθως ο χώρος και όχι ο χρόνος.

Κατά παράδοση, η reachability ανάλυση εφαρμόζεται με επιτυχία στην ανίχνευση λαθών ,όπως για παράδειγμα σε περιπτώσεις deadlock η κώδικα που δεν εκτελείται. Επιπλέον ,η εφαρμογή των αλγόριθμων της reachability ανάλυσης ,επεκτείνεται με την ανάπτυξη του έλεγχου μοντέλων που στηρίζονται στην θεωρία αυτομάτων. Για παράδειγμα ,ο έλεγχος μοντέλων με την μέθοδο LTL, μπορεί να μειωθεί σε αυτήν της reachability ανάλυσης .Επομένως είναι δυνατόν να παρέχονται αλγόριθμοι οι οποίοι να πραγματοποιούν έλεγχο μοντέλων με την τεχνική On-the-fly. Αυτοί οι αλγόριθμοι χρησιμοποιούνται για την εφαρμογή της On-the-fly επαλήθευσης και είναι συμβατοί με τεχνικές διαχείρισης πολυπλοκότητας, όπως οι bit-state hashing και state-space caching. Αξίζει να σημειωθεί ότι έχει προταθεί αλγόριθμος (Gerth) ο οποίος μεταφράζει έναν LTL τύπο σε αυτόματο Büchi ,χρησιμοποιώντας μια πολύ απλή αναζήτηση depth-first. Με αυτόν τον τρόπο ,υπάρχουν αλγόριθμοι επαλήθευσης πρωτοκόλλου οι οποίοι ελέγχουν On-the-fly τόσο το πρωτόκολλο ,όσο και την ιδιότητα του αυτόματου κατά την διάρκεια της αναζήτησης. Πρόσφατα έχουν αναπτυχθεί αλγόριθμοι για On-the-fly έλεγχο μοντέλων με branching-time λογική.

Ένα πλεονέκτημα της On-the-fly επαλήθευσης είναι ότι αυτή προχωράει μέχρι το σημείο όπου βρεθεί λάθος, και που στην περίπτωση αυτή παράγει ένα counterexample το οποίο βοηθά τον σχεδιαστεί να κατανοήσει καλύτερα το λάθος και να το διορθώσει. Συχνά τα λάθη βρίσκονται πολύ νωρίς από το σημείο εκκίνησης της αναζήτησης ,κάτι που σημαίνει ότι μπορεί να αποφευχθεί η αναζήτηση του υπολοίπου χώρου καταστάσεων. Αντίθετα, στην περίπτωση που το σύστημα παρουσιάζεται σωστό ,η αναζήτηση καλύπτει όλο το φάσμα του χώρου καταστάσεων. Μπορούμε έτσι να συμπεράνουμε ότι η προσέγγιση αυτή ταιριάζει στις περιπτώσεις που βρισκόμαστε στα αρχικά σταδία σχεδίασης ενός συστήματος ,στα οποία είναι λογικό να παρουσιάζονται αρκετά λάθη.

2.8 Η τεχνική της Μείωσης (Reduction technique)

Η τεχνικές της μείωσης επικεντρώνονται στο να αποθηκεύουν ένα μέρος ή ένα στιγμιότυπο του χώρου καταστάσεων ενός προγράμματος, διατηρώντας όλες τις ιδιότητες του χώρου αυτού ,για να μπορεί στην συνέχεια να επαληθεύσει τις ιδιότητες που επιθυμεί. Σε αυτή την παράγραφο περιγράφονται οι κύριες προσεγγίσεις της τεχνικής μείωσης του χώρου καταστάσεων (state-space reduction).

2.8.1 Μερική Διατεταγμένη Μείωση (Partial-order reduction)

Στις περισσότερες τεχνικές ελέγχου μοντέλων ,το φαινόμενο concurrency διάφορων διεργασιών μοντελοποιείται με την παρεμβολή (interleaving) λευκών σελίδων, κάτι που αποτελεί μεγάλο παράγοντα όσον αφορά το πρόβλημα της έκρηξης καταστάσεων. Η τεχνική της μερικής μείωσης βασίζεται στην συνεχή παρατήρηση των

concurrent συστημάτων όπου η συνολική επίδραση συνόλου ενεργειών συχνά είναι ανεξάρτητη της σειράς που εκτελούνται. Σαν αποτέλεσμα ,η δημιουργία όλων των δυνατών παρεμβολών μεταξύ ενεργειών του συστήματος ,καθιστά κάποιες από αυτές άχρηστες ,γι αυτό και μπορούν να παραλειφθούν. Αρκετές μέθοδοι έχουν προταθεί βασιζόμενες στην παραπάνω ιδέα ,οι οποίες εξερευνούν έναν μειωμένο γράφο ενός συστήματος διατηρώντας παράλληλα τις ενδιαφερόμενες ιδιότητες αυτού.

Οι μέθοδοι της μερικής μείωσης εφαρμόζουν μια επιλεκτική αναζήτηση στον χώρο καταστάσεων του συστήματος. Για κάθε κατάσταση s που προσπελάσετε κατά την αναζήτηση ,υπολογίζεται ένα υποσύνολο T του συνόλου μεταβάσεων του s ,και εξετάζονται μόνο εκείνες οι μεταβάσεις που αφορούν το T. Αυτή είναι και η κύρια διαφορά τους με τις κλασσικές αναζητήσεις όπου για κάθε κατάσταση s να εξετάζονταν όλες οι μεταβάσεις που θα ήταν δυνατόν να επέλθουν ,μέσω της s. Δύο βασικές τεχνικές έχουν προταθεί για την αναγνώριση αυτού του υποσυνόλου οι οποίες βασίζονται στον υπολογισμό των persistent sets (‘επίμονων’ συνόλων) και των sleep sets (αδρανή συνόλων).

Ένα persistent set T για κάποιες καταστάσεις s, περιέχει μεταβάσεις ενεργές στο s με το παρακάτω χαρακτηριστικό : κάθε μετάβαση που είναι δυνατή να συμβεί από το s εφαρμόζοντας μεταβάσεις έξω από το υποσύνολο T ,είναι ανεξάρτητο (δεν υπάρχει αλληλεπίδραση) από τις μεταβάσεις του T. Μία από τις βασικές τεχνικές τύπου persistent set, προτάθηκαν από τον Valmari (1993) ,και βασίζονται στον προσδιορισμό των stubborn sets. Κατά την διάρκεια της μειωμένης πλέον εξερεύνησης του χώρου καταστάσεων του συστήματος ,επιλέγονται μόνο οι καταστάσεις εκείνες που ανήκουν στο stubborn set. Έχει αποδειχθεί ότι η εκτέλεση όλων των εναπομείναντα μεταβάσεων (που δεν έχουν επιλεχθεί) ,μπορεί να αναβληθεί χωρίς αυτό να επιδράσει τα αποτελέσματα της επαλήθευσης. Ο σκοπός έτσι του stubborn set είναι να παραμείνει αυτό όσο το δυνατόν πιο μικρό σε μέγεθος έτσι ώστε να επιτευχθεί μία μεγάλη μείωση του χώρου καταστάσεων. Ο αλγόριθμος που περιγράφει ο Valmari υπολογίζει τα stubborn sets κατά την διάρκεια της εξερεύνησης του χώρου καταστάσεων και μπορεί να εφαρμοστεί με την μέθοδο On-the-fly.

Η τεχνική των sleep sets, εκμεταλλεύεται το ιστορικό της αναζήτησης. Με την χρήση της ,μειώνει τον αριθμό των μεταβάσεων που εξερευνούνται αλλά όχι και τον αριθμό των καταστάσεων. Όπως προαναφέρθηκε ,κάτι τέτοιο είναι χρήσιμο όταν τα sleep sets συνδέονται με τεχνικές του χώρου καταστάσεων με μνήμη (state-space caching). Κατά την διάρκεια της αναζήτησης depth-first στον γράφο του συστήματος ,κάθε κατάσταση s σχετίζεται με ένα sleep set ,το οποίο αποτελεί ένα σύνολο μεταβάσεων που μπορούν να γίνουν από το s αλλά δεν θα εκτελούνται από αυτό. Τα sleep sets μπορούν να συνδυαστούν ακόμη και με τα persistent sets για να μειώσουν ακόμη περισσότερο ο χώρος των καταστάσεων. Επιπρόσθετα, αφού η τεχνική των persistent sets δεν μπορεί να αποφύγει την επιλογή ανεξάρτητων μεταβάσεων μιας κατάστασης ,τα sleep sets αποφεύγουν την εξερεύνηση πολλαπλών παρεμβολών αυτών των μεταβάσεων.

Οι Godefroid και Wolper (1991) προτείνουν τεχνικές μερικής μείωσης για την ανίχνευση και επαλήθευση της απουσίας deadlock και ιδιοτήτων ασφάλειας (safety properties). Σύμφωνα με την περιγραφή των παραπάνω, ο έλεγχος των safety properties γίνεται (ή καλύτερα μειώνεται) στην ανίχνευση deadlock του συστήματος. Πιο πρόσφατες τεχνικές έχουν προταθεί και οι οποίες επεκτείνουν τις αρχικές τεχνικές

μερικής μείωσης ,φέρνοντας μπροστά όλες τις δυνατότητες και των οφελών που απορρέουν από τον έλεγχο μοντέλων. Κάποιες από αυτές εφαρμόζουν έλεγχο μοντέλων σε τύπους LTL οι οποίοι δεν περιέχουν τον τελεστή "next time". Η συγκεκριμένη τεχνική ,μπορεί να χειριστεί κάθε είδους λογική LTL ,καθώς και επίσης επεκτάσεις αυτής. Αυτή η προσέγγιση χρησιμοποιεί τεχνικές που βασίζονται στην θεωρία αυτομάτων ,περιλαμβάνοντας ακόμη επεκτάσεις αυτών όπως τα w-αυτόματα.

Με το συνδυασμό του ελέγχου μοντέλων με την τεχνική της μείωσης ,η δεύτερη ακολουθεί πάντα τους κανόνες που θέτει η υπο-επαλήθευση ιδιότητα. Πρόκειται για την περίπτωση όπου η τεχνική της μερικής μείωσης προσπαθεί να υπολογίσει ,κατά την διάρκεια της αναζήτησης, τα μέρη εκείνα του γράφου καταστάσεων που πλεονάζουν και μπορούν να παραλειφθούν.

2.9 Εργαλεία αυτόματου έλεγχου μοντέλων

είναι γνωστό πως ένας αρκετά μεγάλος αριθμός εργαλείων που προσφέρονται για έλεγχο μοντέλων ,έχουν αναπτυχθεί εδώ και αρκετό καιρό. Στην παράγραφο αυτή παρουσιάζονται και περιγράφονται συνοπτικά μερικά από τα πιο γνωστά εργαλεία :

- CADP
- Concurrency Workbench
- COSPAN
- FC2TOOLS
- FDR
- SMV
- SPIN
- Model Checkers for real-time and hybrid systems

CADP

Το πρώτο εργαλείο στο οποίο θα γίνει αναφορά ,είναι το εργαλείο CADP (CÆSAR – ALDÉBARAN Development Package) το οποίο αναπτύχθηκε από τον Fernandez. Το CADP αποτελεί ουσιαστικά μια εργαλειοθήκη επαλήθευσης για τον σχεδιασμό και την επαλήθευση πρωτοκόλλων επικοινωνίας αλλά και κατανεμημένων συστημάτων. Η γλώσσα προσομοίωσης που χρησιμοποιείται για την αναπαράσταση των μοντέλων των συστημάτων, είναι η γλώσσα LOTOS. Ολόκληρο το σημασιολογικό μοντέλο βασίζεται στην μέθοδο με τύπους LTSs. Το CADP επιτρέπει τόσο χαμηλού επιπέδου ιδιότητες εκφρασμένες σε LTL, όσο και ενδιάμεσου επιπέδου που επιτρέπει την επαλήθευση ορισμένων ιδιοτήτων πρωτοκόλλων ,προσομοιωμένες σε διαφορετικές γλώσσες (από την LOTOS), όπως αυτή της SDL. Η εργαλειοθήκη περιέχει αρκετά 'στοιχεία' λογισμικού ,τα οποία συνδέονται μεταξύ τους μέσω μιας διαπροσωπείας (με χρήση γραφικών). Η λειτουργίες που είναι ικανό το CADP να φέρει σε πέρας περιλαμβάνουν τυχαία προσομοίωση (random simulation), μερική και εξαντλητική ανίχνευση για deadlock, επαλήθευση ιδιοτήτων δίνοντας προτεραιότητα στις σχέσεις μεταξύ των καταστάσεων καθώς επίσης και επαλήθευση ιδιοτήτων branching time temporal logic με μέθοδο XTL (eXecutable Temporal Language). Οι τύποι LTSs

μπορούν να αναπαραστηθούν εύκολα με την βοήθεια των όρων των BDDs. Όσον αφορά την επαλήθευση, μπορεί να εφαρμοστεί τόσο η On-the-fly τεχνική ,όσο και αυτή της συνθετικής παραγωγής του χώρου καταστάσεων (compositional state-space generation). Τέλος ,αξίζει να σημειωθεί ότι αρκετές μελέτες έχουν γίνει με την βοήθεια του εργαλείου CADP ,ιδιαίτερα σε αρκετές βιομηχανικές εφαρμογές που χρησιμοποιούνται ευρέως σήμερα.

Concurrency Workbench

Το Concurrency Workbench αποτελεί ένα εργαλείο το οποίο είναι ικανό να χειρίζεται και να εφαρμόζει αρκετές τεχνικές ελέγχου και επαλήθευσης. Ένα σύστημα μοντελοποιείται σε μία διεργασία CCS. Οι διεργασίες με την σειρά τους ,διερμηνεύονται σε τύπους LTSs για τους σκοπούς της επαλήθευσης. Το εργαλείο μπορεί και υποστηρίζει τρεις διαφορετικές μεθόδους επαλήθευσης. Κατ' αρχήν έχει την ικανότητα να ελέγχει όλες τις συμπεριφορές ισοδυναμίας μεταξύ των LTSs του συστήματος και αυτών των ιδιοτήτων του. Υποστηρίζει αρκετές ισοδυναμίες ,μεταξύ αυτών και την (ισχυρή και παρατηρητική) ισοδυναμία του Milner που αφορά την ανίχνευση και την 'λανθασμένη ισοδυναμία'. Παρόλο που η ελαχιστοποίηση του χώρου καταστάσεων αποτελεί μια από τις μεθόδους η οποία μπορεί να εφαρμοστεί από το εργαλείο εν μέρει ,δεν δίνεται σημασία εξ' ολοκλήρου πάνω στο πρόβλημα της έκρηξης καταστάσεων. Δεύτερον, το εργαλείο υποστηρίζει τον έλεγχο μοντέλων των ιδιοτήτων ενός συστήματος, εκφρασμένο σε μια τροπική λογική (modal logic) που βασίζεται πάνω σε προτασιακή άλγεβρα (prepositional-calculus). Η άλγεβρα αυτή είναι αρκετά πιο περιεκτική σε σύγκριση με αυτήν της CTL και μπορεί να εκφράσει μια ποικιλία ιδιοτήτων μεταβάσεων ενός συστήματος, όπως εφικτές (από άποψη λειτουργίας) καταστάσεις και σχέσεις ισοδυναμίας μεταξύ των καταστάσεων . Παρόλο αυτά όμως αποτελεί μια λογική η οποία είναι αρκετά δύσκολη στην κατανόηση και γι αυτό τον λόγο το εργαλείο προσφέρει στον χρήστη την δυνατότητα να προσδιορίσει μόνος του βοηθητικά macros, που θα του χρησιμεύσουν στην αναπαράσταση του συστήματος. Ο αλγόριθμος που χρησιμοποιείται για τον έλεγχο μοντέλων ,έχει πολυπλοκότητα η οποία αυξάνεται εκθετικά σε σχέση με το μήκος του τύπου (που εκφράζει την ιδιότητα που ελέγχεται). Ένας γραμμικού χρόνου αλγόριθμος έχει προταθεί για συγκεκριμένες περιπτώσεις λογικής που ονομάζεται 'εναλλακτική προτασιακή άλγεβρα' (the alternation free modal –calculus). Τέλος αξίζει να αναφερθεί ότι έχουν γίνει αρκετές επεκτάσεις στο εργαλείο ,όπως αυτή του *NCSU Concurrency Workbench* που αναπτύχθηκε από τους Cleaveland και Sims, αλλά και εφαρμογές με γραφικό περιβάλλον που βοηθούν τον (αρχάριο) σχεδιαστή στην σχεδίαση concurrent συστημάτων.

COSPAN

Το εργαλείο COSPAN επαληθεύει μοντέλα ,αναπαριστώντας τα με βάση την θεωρία αυτομάτων. Αυτό το κάνει με τον έλεγχο της γλώσσας στην οποία είναι γραμμένο το σύστημα. Η γλώσσα η οποία χρησιμοποιεί το εργαλείο ονομάζεται S/R

(επιλογή/ανάλυση) αλλά οι διαπροσωπείες αυτού έχουν γραφτεί σε γλώσσες περιγραφής υλικού, Vérilog, VHDL και SDL (βασίζεται στο CCITT-standard protocol specification). Το σημασιολογικό μοντέλο στηρίζεται πάνω στα ω-αυτόματα (γενικά ένα σύστημα αποτελείται από συλλογές τέτοιων αυτομάτων). Για να μετατραπεί μια ιδιότητα σε ω-αυτόματο, παρέχεται μια βιβλιοθήκη με παραμετροποιημένα αυτόματα ,πολύ χρήσιμη για τις αναπαραστάσεις των μοντέλων. Όταν το εργαλείο ανιχνεύσει στο σύστημα που εξετάζει παραβίαση των ιδιοτήτων τότε επιστρέφει counter-examples στον χρήστη. Το COSPAN έχει την ικανότητα να χρησιμοποιεί είτε συμβολικούς (BDD-based) είτε state-enumeration αλγόριθμους. Οι τελευταίοι έχουν την ικανότητα να δίνουν ιδιότητες caching (μνήμης) και bit-state hashing (όπως ακριβώς και στην On-the-fly επαλήθευση). Αρκετές στρατηγικές μπορούν να χρησιμοποιηθούν ιδιαίτερα όσον αφορά την μέθοδο της μείωσης, όπου και ανάλογα με τις επιλογές του χρήστη μπορεί να εφαρμοστεί συμμετρική μείωση ή και αυτόματη μείωση ανάλογα με την χρονική στιγμή της κάθε κατάστασης. Τέλος το εργαλείο COSPAN προσφέρει στον σχεδιαστή την δυνατότητα σχεδιασμού από πάνω προς τα κάτω (top-down design) μέσω διαδοχικών καταστάσεων .

FC2TOOLS

Τα εργαλεία FC2TOOLS (the next-generation AUTO/GRAPH) ,αποτελούν μια εργαλειοθήκη επαλήθευσης ,που υποστηρίζει γραφική αναπαράσταση των ιδιοτήτων σε υπο-εξέταση concurrent συστήματα. Έχει την ικανότητα να κάνει ανάλυση των καταστάσεων, ελαχιστοποίηση ,έλεγχο ισοδυναμιών και μείωση των μοντέλων που αναπαρίστανται με την βοήθεια αυτομάτων. Η εργαλειοθήκη υποστηρίζει την αναπαράσταση των ιδιοτήτων ενός συστήματος με αυτόματα και εφαρμόζει την τεχνική της On-the-fly επαλήθευσης για να κάνει τον έλεγχο αυτών.

FDR

Το FDR αποτελεί εργαλείο το οποίο στηρίζεται στην θεωρία CSP. Το FDR ελέγχει για την ιδιότητα ενός συστήματος , μέσω των λειτουργιών του συστήματος όπως λάθη ανίχνευσης και αποτυχίες εκτέλεσης. Το μοντέλο το οποίο χρησιμοποιείται στο σύστημα αυτό ,είναι αυτό της αποτυχίας-απόκλισης (Failure-Divergence) ,από όπου βγαίνει και το όνομα του εργαλείου (Failures-Divergence Refinement). Τόσο το σύστημα όσο και η ιδιότητα αναπαρίστανται με αυτόματα έκδοσης CSP όπου στη συνέχεια μεταφράζονται σε πεπερασμένους LTSs. Πριν κάθε έλεγχο η ιδιότητα LTS κανονικοποιείται κάτι που βοηθά στον έλεγχο καταστάσεων αλλά αυξάνει εκθετικά το μέγεθος του LTS. Τέλος το FDR υποστηρίζει συνθετική ελαχιστοποίηση όπου ενδιάμεσα συστήματα μπορούν να απλοποιηθούν με ποικίλες τεχνικές συμπίεσης.

SMV

Το SMV αποτελεί εργαλείο για τον έλεγχο συστημάτων πεπερασμένων καταστάσεων σε αντίθεση με τις προδιαγραφές της χρονικής λογικής CTL. Υποστηρίζει

ευέλικτες γλώσσες για αναπαράσταση και χρησιμοποιεί έναν OBDD-based συμβολικό αλγόριθμο για επαρκή έλεγχο εάν οι προδιαγραφές (σε CTL) ικανοποιούνται από το σύστημα. Το SMV έχει χρησιμοποιηθεί για την επαλήθευση αρκετών βιομηχανικών εφαρμογών όπως τα πρωτοκολλά Futurebus+ και Gigamax.

SPIN

Το εργαλείο SPIN αναπτύχθηκε από τον G.Holzmann και αποτελεί έναν state-based ελεγκτή μοντέλων που σχεδιασμένο για την επαλήθευση κυρίως κατανεμημένων συστημάτων. Η γλώσσα προδιαγραφών στην οποία γράφονται τα μοντέλα ονομάζεται PROMELA, ενώ το σημασιολογικό μοντέλο του εργαλείου βασίζεται σε πεπερασμένα αυτόματα. Το SPIN ελέγχει ένα σύνολο από βασικές ιδιότητες όπως απουσία deadlock και unreachable code. Επίσης ελέγχει μην παραβιάζονται ιδιότητες που έχει ορίσει ο χρήστης αλλά και το σύστημα να τερματίζεται όταν αυτός το θέλει. Η γλώσσα αναπαράστασης PROMELA περιλαμβάνει δύο βασικές ετικέτες που μπορούν να περιγράψουν καταστάσεις του συστήματος. Αυτές είναι οι ετικέτες 'progress' και 'accept'. Το SPIN επίσης ελέγχει εάν κάθε κύκλος του συστήματος περιέχει μία κατάσταση progress και δεν περιέχει κατάσταση accept. Όλες οι απαιτήσεις ορθότητας του συστήματος εκφράζονται απευθείας σε LTL τύπους, οι οποίοι με την σειρά τους μεταφράζονται σε τμήματα PROMELA που ονομάζονται 'never-claims'. Τα τμήματα αυτά αναπαρίστανται σε αυτόματα Büchi που ανταποκρίνονται στην άρνηση των τύπων αυτών(που έχουν τα never-claims). Τέλος αξίζει να σημειωθεί ότι το SPIN εφαρμόζει έλεγχο με χρήση της τεχνικής On-the-fly. Χρησιμοποιεί έναν συμβατό αλγόριθμο αναζήτησης με όλες τις περιπτώσεις επαλήθευσης που υποστηρίζει το εργαλείο όπως τεχνικές εξαντλητικής αναζήτησης καταστάσεων, bit-state hashing και μερική μείωση. Με τις τεχνικές αυτές και σε συνδυασμό με την συμπίεση καταστάσεων, το εργαλείο SPIN μπορεί να αντιμετωπίσει αρκετά μεγάλο χώρο καταστάσεων ,άρα προσφέροντας επαρκή λύση στο φαινόμενο της έκρηξης των καταστάσεων.

Ελεγκτές μοντέλων για υβριδικά συστήματα και συστήματα πραγματικού χρόνου

Πρόσφατα ,ο έλεγχος μοντέλων έχει επεκταθεί σε συστήματα υβριδικά αλλά και σε συστήματα πραγματικού χρόνου (real-time and hybrid systems). Είναι γνωστό ότι τα συστήματα πραγματικού χρόνου πρέπει να εκτελέσουν ένα συγκεκριμένο στόχο μέσα σε περιορισμένο χρονικό διάστημα. Για τον λόγο αυτό μοντελοποιούνται με την βοήθεια αυτομάτων χρόνου (timed-αυτόματα) ,δηλαδή πεπερασμένων καταστάσεων μηχανές που χρησιμοποιούν μεταβλητές χρόνου (clocks) οι οποίες και εκφράζουν χρονικούς περιορισμούς μεταξύ των καθυστερήσεων των καταστάσεων. Από την άλλη μεριά τα υβριδικά συστήματα (που αποτελούν ψηφιακά συστήματα πραγματικού χρόνου σε αναλογικά περιβάλλοντα) ,μοντελοποιούνται με την βοήθεια υβριδικών αυτομάτων (hybrid-automata), δηλαδή πεπερασμένου καταστάσεων μηχανές που χρησιμοποιούν πραγματικές μεταβλητές οι οποίες και περιγράφονται σαν διαφορικές εξισώσεις. Μερικά από τα εργαλεία τα οποία έχουν αναπτυχθεί για τέτοια συστήματα είναι τα KRONOS, UPAAL και Hytech.

Ο Αυτόματος Ελεγκτής Μοντέλων Spin

3.1 Εισαγωγή στο εργαλείο Spin

Το SPIN (**S**imple **P**romela **In**terpretation) είναι ένα ευρέως διαδεδομένο λογισμικό για την επαλήθευση κατανεμημένων συστημάτων. Αναπτύχθηκε στα Bell Labs περίπου στις αρχές της δεκαετίας 1980. Είναι «γραμμένο» σε γλώσσα C και «τρέχει» σε λειτουργικό UNIX, αλλά επίσης μπορεί να μεταγλωττιστεί και να «τρέξει» και σε οποιαδήποτε άλλη πλατφόρμα, συμπεριλαμβάνοντας το λειτουργικό Linux, αλλά και τα Windows.

Τα μοντέλα επαλήθευσης του SPIN επικεντρώνονται στην απόδειξη της ορθότητας των αλληλεπιδράσεων επεξεργασίας και προσπαθούν να απεικονίσουν, όσο το δυνατό περισσότερο, τους εσωτερικούς ακολουθιακούς υπολογισμούς. Οι αλληλεπιδράσεις επεξεργασίας μπορούν να οριστούν στο SPIN με ασύγχρονο πέρασμα μηνυμάτων μέσω καναλιών buffer, ή με κοινές μεταβλητές, ή τέλος με οποιοδήποτε συνδυασμό από τις δύο προηγούμενες μεθόδους. Για να επαληθεύσουμε έναν αλγόριθμο ή πρωτόκολλο με το SPIN, πρέπει να το γράψουμε στην PROMELA, τη γλώσσα μοντελοποίησης που χρησιμοποιεί το SPIN. Η PROMELA (**PRO**cess **ME**ta **LA**nguage) είναι μια μη ντετερμινιστική γλώσσα, που χρησιμοποιεί τρεις τύπους αντικειμένων:

- *ενέργειες:* είναι καθολικά αντικείμενα
- *μεταβλητές:* τοπικές ή καθολικές μέσα σε μία ενέργεια
- *κανάλια μηνυμάτων (message channels):* τοπικά ή καθολικά σε μία ενέργεια

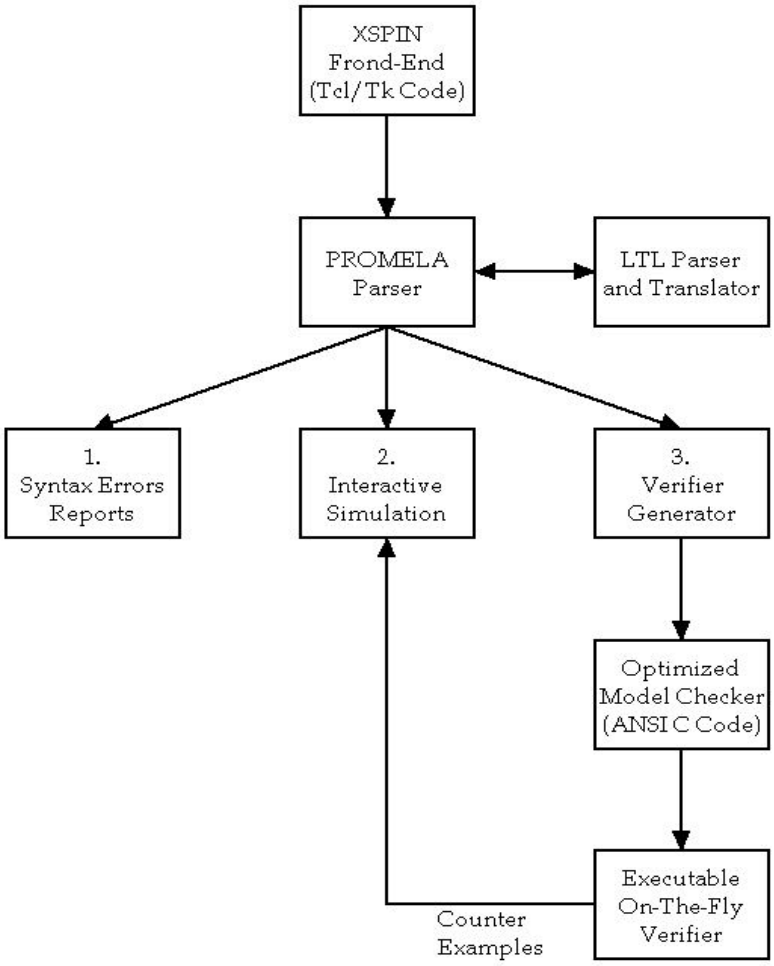
Ο ευκολότερος τρόπος να ξεκινήσει κανείς να δουλεύει με το SPIN είναι να χρησιμοποιήσει το γραφικό περιβάλλον του, το XSPIN. Το τελευταίο, «τρέχει» ανεξάρτητα από το SPIN και βοηθάει στη δημιουργία των εντολών του SPIN με απλές επιλογές από το μενού. Επίσης, το μεγάλο του πλεονέκτημα είναι ότι το αποτέλεσμα της προσομοίωσης ή της επαλήθευσης το παριστάνει γραφικά. Χρησιμοποιώντας το XSPIN, ο χρήστης δε χρειάζεται να απομνημονεύει σειρές εντολών, αλλά επικεντρώνεται περισσότερο στην ουσία που του προσφέρει το SPIN μέσω του XSPIN.

Το SPIN μπορεί να χρησιμοποιηθεί με τους ακόλουθους τρεις τρόπους:

- ως προσομοιωτής, επιτρέποντας τρία είδη προσομοίωσης, ανάλογα με τις ανάγκες του χρήστη και του αλγορίθμου ή πρωτοκόλλου που χρησιμοποιείται, την τυχαία(random), καθοδηγημένη (guided) και την διαλογική (interactive).
- ως εξαντλητικός (exhaustive) αναλυτής χώρου καταστάσεων, ικανός να αποδείξει την εγκυρότητα των χαρακτηριστικών που εισήγαγε ο χρήστης κάνοντας χρήση της θεωρίας partial order reduction,που αναλύθηκε παραπάνω.
- ως αναλυτής bit-state χώρου, ικανός να επαληθεύσει και τα μεγαλύτερα πρωτόκολλα με τη μέγιστη έκταση του χώρου καταστάσεων.

3.2 Η δομή του SPIN

Η βασική δομή του SPIN απεικονίζεται στο παρακάτω σχήμα (εικόνα 3.2.1).



Η δομή της interactive προσομοίωσης και επαλήθευσης του εργαλείου SPIN

Εικόνα 3.2: Η δομή λειτουργίας του ελεγκτή μοντέλων SPIN

Ο τυπικός τρόπος για να αρχίσει κανείς να δουλεύει με το SPIN η με το γραφικό του περιβάλλον XSPIN είναι να γράψει τα χαρακτηριστικά ενός παράλληλου συστήματος ή ενός κατανεμημένου αλγορίθμου. Μετά τη διόρθωση συντακτικών λαθών, αν βέβαια υπάρχουν, το σύστημα εισάγεται σε interactive προσομοίωση, μέχρι να βεβαιωθούμε ότι συμπεριφέρεται όπως θέλουμε και ορίζεται. Έπειτα, το SPIN χρησιμοποιείται για να παράγει μία on-the-fly επαλήθευση του προγράμματος. Η επαλήθευση εκτελείται με διαφορετικές επιλογές κάθε φόρα και αν οδηγήσει σε σφάλμα, τότε γυρίζουμε ξανά στο στάδιο της interactive προσομοίωσης για να δούμε τι προκάλεσε το σφάλμα. Αν είναι εφικτό το διορθώνουμε και κάνουμε την επαλήθευση ξανά.

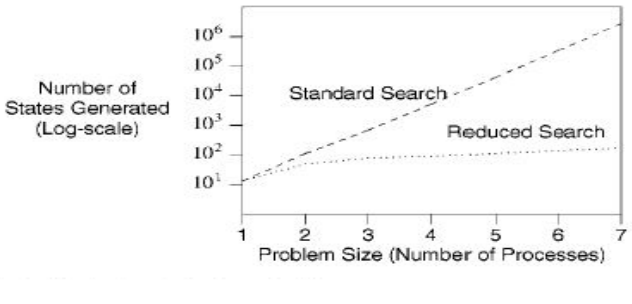
3.2.1 Προτεραιότητες του εργαλείου

Στον έλεγχο μοντέλων, σε αντίθεση με το συμβατικό προγραμματισμό, οι απαιτήσεις σε χώρο είναι πιο σημαντικές από τις απαιτήσεις σε χρόνο. Για το λόγο αυτό θα πρέπει να λαμβάνονται σοβαρά υπ᾽ όψη τα ακόλουθα:

- Αριθμός καταστάσεων: Εξαιτίας του προβλήματος της έκρηξης του χώρου καταστάσεων θα πρέπει να μειώνεται ο αριθμός των καταστάσεων.
- Μέγεθος διανύσματος κατάστασης: Αυτό δηλώνει το ποσό της μνήμης που απαιτείται και πρέπει να το ελαχιστοποιούμε για αύξηση της απόδοσης.
- Μέγεθος στοίβας αναζήτησης: Απαιτείται και εδώ ελαχιστοποιήση.
- Χρόνος επαλήθευσης: σημαντικό στοιχείο που πρέπει να μας απασχολεί.

3.2.2 Το εργαλείο SPIN κα η τεχνική της μερικής μείωσης

Το εργαλείο Spin χρησιμοποιεί την τεχνική της μερικής μείωσης για την μείωση του αριθμού των προσβάσιμων καταστάσεων που πρέπει να εξερευνηθούν για την ολοκλήρωση της επαλήθευσης. Η μείωση αυτή βασίζεται πάνω στην παρατήρηση ότι η ορθότητα της LTL πρότασης αποτελεί έχει να κάνει με την σειρά με την οποία μεταχειρίζονται οι χώροι καταστάσεων των μοντελοποιημένων συστημάτων (concurrent ή ανεξάρτητα εκτέλεσης συστήματα) στην διάρκεια της πρώτα-σε-βάθος (depth-first) αναζήτησης. Αντί το εργαλείο να παράγει (μέσω του μοντέλου) έναν εξαντλητικό χώρο αναζήτησης που να περιλαμβάνει όλες τις ακολουθίες εκτέλεσης σαν ‘μονοπάτια’, ο επαληθευτής που δημιουργείται παράγει έναν μειωμένο χώρο περιλαμβάνοντας μονάχα εκείνες τις καταστάσεις που είναι αντιπροσωπευτικές για την προδιαγραφή του συστήματος που επαληθεύεται.



Εικόνα 3.2.2: Συγκριτική μελέτη τυπικής αναζήτησης και αναζήτησης με την βοήθειας της μερικής μείωσης

Η εφαρμογή αυτής της τεχνικής χαρακτηρίζεται από μια στατική μείωση, όπως περιγράφθηκε και στο 2ο κεφάλαιο που πριν την έναρξη της πραγματικής επαλήθευσης αναγνωρίζει τις περιπτώσεις εκείνες όπου με ασφαλή τρόπο μπορεί να γίνει μείωση των καταστάσεων. Στην εικόνα 3.1.3 δίνεται ένα παράδειγμα μέτρησης του αριθμού των προσβάσιμων καταστάσεων που πρέπει να παραχθούν από το εργαλείο για το πέρας της επαλήθευσης ενός αλγόριθμου επιλογής. Παρουσιάζει την όσο το δυνατόν καλύτερη προσέγγιση του αλγορίθμου όπου οι καταστάσεις αρχικά αυξάνονται με εκθετικό τρόπο και με την τεχνική της μείωσης αυξάνονται με γραμμικό τρόπο. Τέλος ένα άλλο σημαντικό χαρακτηριστικό αυτής της στατικής μεθόδου μείωσης του Spin είναι ότι δεν χρειάζεται αρκετούς πόρους σε μνήμη σε αντίθεση με τις εξαντλητικές μεθόδους.

3.3 Process Meta Language (PROMELA)

Θα ήταν σωστό πριν περάσουμε στο παράδειγμα που ουσιαστικά δείχνει την λειτουργία του εργαλείου, να γίνει μια αναφορά στην γλώσσα και στις τεχνικές που χρησιμοποιεί αυτή για την μοντελοποίηση συστημάτων. Ειδική έμφαση δίνεται στις διεργασίες που ορίζονται και οι οποίες αποτελούν απαραίτητα στοιχεία για αναπαράσταση. Όπως αναφέρθηκε, το εργαλείο Spin είναι ικανό να ελέγχει μοντέλα τα οποία αναπαρίστανται από μια γλώσσα αρκετά διαφορετική από τις γλώσσες που χρησιμοποιούν σήμερα οι προγραμματιστές. Πρόκειται για μια γλώσσα αναπαράστασης και μοντελοποίησης με το όνομα *Promela*, που χρησιμοποιείται κυρίως για την αφηρημένη μοντελοποίηση πρωτοκόλλων επικοινωνίας (Holtzmann 1997). Η Promela ταιριάζει απόλυτα επίσης στην μοντελοποίηση πρακτόρων επικοινωνίας. Η «συζητήσεις» μεταξύ πρακτόρων αναπαρίστανται σαν διεργασίες, τα μονοπάτια αυτών των συζητήσεων μοντελοποιούνται σαν κανάλια και οι μεταβλητές που χρησιμοποιούνται ορίζονται και ελέγχονται χωρίς καμία αλλαγή. Όλες οι δηλώσεις είναι είτε εκτελέσιμες είτε μπλοκάρονται από άλλες δηλώσεις περιμένοντας με την σειρά τους να εκτελεστούν. Έτσι, προτάσεις μπορεί να μπλοκαριστούν από μία εντολή if εάν η συνθήκη αυτής είναι false. Οι προτάσεις αυτές μπορούν να εκτελεστούν την στιγμή εκείνη που η συνθήκη θα γίνει true. Κάτι τέτοιο παρέχει μηχανισμούς συγχρονισμού επικοινωνιών μεταξύ διεργασιών έχοντας ως αποτέλεσμα μια εκτελέσιμη διεργασία (responder) να περιμένει για ένα μήνυμα από μια άλλη διεργασία (initiator).

Όλες οι διεργασίες στην Promela ορίζονται με την λέξη proctype. Παρακάτω ακολουθεί ένα παράδειγμα μιας δήλωσης ενός proctype :

```
proctype ProcessA()
{
byte newVariable;
newVariable = 3
}
```

Το όνομα της διεργασίας αυτής είναι το ProcessA και μέσα στις αγκύλες περικλείεται το σώμα της δήλωσης αυτής. Παρατηρείται η δήλωση μιας τοπικής μεταβλητής newVariable τύπου byte, και έναν αρχικό ορισμό της μεταβλητής αυτής δίνοντάς της την τιμή 3. Στην Promela τα ερωτηματικά ‘ ; ’ και τα βέλη ‘ -> ’ διαχωρίζουν δηλώσεις μεταξύ τους. Στο παράδειγμα δεν χρειάζεται ερωτηματικό ή βέλος στην τελευταία δήλωση. Το βέλος χρησιμοποιείται μερικές φορές και σαν ένα τρόπο για να δειχθεί μια αιτιατή (causal) σχέση μεταξύ δύο δηλώσεων. Για παράδειγμα :

```
byte newVariable = 2;
proctype ProcessA()
{
(newVariable == 1) -> newVariable = 3
}
proctype ProcessB()
{
newVariable = newVariable – 1
}
```

Στο παράδειγμα αυτό δηλώνονται δύο διεργασίες, οι ProcessA και ProcessB. Μετά την «παγκόσμια» (global) δήλωση της μεταβλητής newVariable έξω από τις διεργασίες με την τιμή 2 παρατηρείται ότι η διεργασία Α περιέχει δύο δώσεις στο σώμα της ενώ η Β περιέχει μία δήλωση η οποία και ελαττώνει την newVariable κατά 1. Πάντα μια δήλωση είναι εκτελέσιμη και γι αυτό και η διεργασία Β δεν μπλοκάρεται αλλά εκτελείται άμεσα. Εάν η συνθήκη δεν είναι αληθής τότε η διεργασία μπλοκάρεται μέχρι αυτή να γίνει αληθής. Κάτι τέτοιο συμβαίνει στην περίπτωση της διεργασίας Α όπου δεν ισχύει η ισότητα της newVariable με 1 οπότε μπλοκάρεται η πρόταση.

Ένα proctype αποτελεί μια διεργασία. Δεν μπορεί να εκτελεστεί αυτόνομα μόνη της. Μία άλλη δήλωση πρέπει να είναι αυτή η οποία και θα θέσει την διεργασία προς εκτέλεση. Το εργαλείο Spin κάνει την χρήση μιας διεργασίας με το όνομα init για να πυροδοτήσει τις υπόλοιπες διεργασίες. Η διεργασία init είναι όμοια με την διαδικασία main στα προγράμματα Java. Ένα παράδειγμα μιας διεργασίας init είναι και το παρακάτω :

```
init
{
run ProcessA();
run ProcessB()
}
```

Σε αυτήν την περίπτωση, η λέξη κλειδί run πυροδοτεί τις δύο διεργασίες.

3.4 Έννοιες που αφορούν τον έλεγχο μοντέλων και το εργαλείο Spin

Θα ήταν σωστό στη συνέχεια να δοθούν κάποιοι βασικοί ορισμοί εννοιών που συναντώνται συχνά στην περίπτωση ενασχόλησης με το θέμα του έλεγχου μοντέλων. Ουσιαστικά θα πρόκειται για μια περιγραφή των πιο συνηθισμένων προβλημάτων που καλείται να ανίχνευση και να αντιμετωπίσει ο έλεγχος μοντέλων, γι αυτό και η περιγραφή τους θεωρείται αναγκαία.

3.4.1 Αδιέξοδο (Deadlock)

Ένα *αδιέξοδο* αποτελεί μια κατάσταση στην οποία δύο προγράμματα υπολογιστών μοιράζονται την ίδια πηγή και επιδρούν το ένα πάνω στο άλλο με το να αποτρέπουν την πρόσβαση πάνω σε αυτή την πηγή, έχοντας ως αποτέλεσμα το μπλοκάρισμα και των δύο προγραμμάτων. Όταν τα λειτουργικά συστήματα εκτελούν ένα μόνο πρόγραμμα, τότε όλες οι πηγές του συστήματος είναι ανοιχτές για το πρόγραμμα αυτό. Στην περίπτωση όμως που τα λειτουργικά συστήματα εκτελούν πολλά προγράμματα μαζί, παρεμβαίνει ανάμεσά τους έτσι ώστε τα προγράμματα να ζητούν πηγές δυναμικά. Κάτι τέτοιο μπορεί να οδηγήσει σε αδιέξοδο. Ένα παράδειγμα είναι και το παρακάτω :

- Το πρόγραμμα_1 ζητά μια πηγή Α και την λαμβάνει

- Το πρόγραμμα_2 ζητά μια πηγή Β και την λαμβάνει
- Το πρόγραμμα_1 ζητά την πηγή Β και περιμένει από το πρόγραμμα_2 να την απελευθερώσει
- Το πρόγραμμα_2 ζητά την πηγή Α και περιμένει από το πρόγραμμα_1 να την απελευθερώσει

Στην περίπτωση αυτή κανένα από τα δύο προγράμματα (ούτε το 1 αλλά ούτε και το πρόγραμμα 2) μπορεί να προχωρήσει μέχρι το άλλο να απελευθερώσει την πηγή του. Το λειτουργικό σύστημα πέφτει σε δίλημμα και δεν γνωρίζει τι ενέργεια να εκτελέσει για να αποφύγει την κατάσταση αυτή. Ως εναλλακτική λύση μπορεί να παρουσιαστεί ο τερματισμός ενός από των δύο προγραμμάτων. Έτσι μπορεί κανείς να συνειδητοποιήσει ότι η γνώση του τρόπου με τον οποίο μπορεί να διαχειριστεί τις καταστάσεις αυτές έχει μεγάλο αντίκτυπο τόσο στην ανάπτυξη λειτουργικών συστημάτων αλλά και στην ανάπτυξη συστημάτων επικοινωνίας αλλά και απλών προγραμμάτων (αρκεί να φανταστεί κανείς ότι στην θέση των προγραμμάτων βρίσκονται δύο εντολές και στην θέση των πηγών δύο βάσεις δεδομένων, έχοντας τις εντολές αυτές να ζητούν η μια πρόσβαση και ανάκτηση της βάσης της άλλης)

3.4.2 Αδυναμία Τερματισμού (Livelock)

Ο όρος *αδυναμία τερματισμού (Livelock)* αποτελεί κατάσταση στην οποία ένα κρίσιμο σημείο μιας διεργασίας αδυνατεί να τερματίσει την εκτέλεσή της. Και αυτό γιατί τα δομικά στοιχεία της συγκεκριμένης εργασίας συνεχίζουν να δημιουργούν επιπλέον ανάγκες από την εργασία μετά από την παροχή του κρίσιμου τμήματος για την αρχική ανάγκη και πριν η συγκεκριμένη εργασία «καθαρίσει» την ουρά της (από την υπάρχουσα εργασία). Η αδυναμία τερματισμού διαφέρει από ο αδιέξοδο (deadlock) στο σημείο ότι η διεργασία δεν μπλοκάρεται ή περιμένει κάτι να συμβεί, αλλά έχει μία άπειρη εικονική ποσότητα εργασίας να φέρει σε πέρας, αδυνατώντας να τερματίσει. Ένα παράδειγμα αδυναμίας τερματισμού αποτελεί μια διακοπή (interrupt) ενός λειτουργικού συστήματος. Εάν στον πυρήνα ενός λειτουργικού συστήματος φτάσουν πάρα πολλές αιτήσεις χωρίς η διαδικασία αυτή να σταματά, το λειτουργικό σύστημα δεν θα είναι σε θέση να εξυπηρετήσει κάποιες από τις αιτήσεις αυτές γιατί θα ξοδέψει όλη την υπολογιστική δύναμη του επεξεργαστή στο να επεξεργαστεί τις αιτήσεις αυτές (που συνεχίζουν να φτάνουν).

3.4.3 Η ιδιότητα Safety (Safety Property)

Μια ιδιότητα Safety βεβαιώνει ότι τίποτα κακό δεν θα συμβεί κατά την εκτέλεση μια διεργασίας [](){readySignal == 1} → (){ackSignal == 0}) Ένα παράδειγμα παραβίασης της [redacted] ποτελέσματα της εκτέλεσης. Η παρακάτω LTL πρόταση αντικατοπτρίζει την ιδιότητα Safety :

Η πρόταση αυτή διαβάζεται ως εξής «Πάντα, όταν το readySignal είναι ίσο με 1 τότε το ackSignal θα ισούται με 0 στον επόμενο κύκλο». Όπως φαίνεται γίνεται χρήση του τελεστή always, [] και του τελεστή next cycle, ( ).

Ένα δεύτερο παράδειγμα μιας ιδιότητας Safety είναι και το ακόλουθο :

```
[]({readySignal == 1} → (-){ackSignal == 0})
```

Πρόκειται για παράδειγμα ίδιο με το πρώτο με την μόνη διαφορά, αντί του τελεστή next cycle (), χρησιμοποιείται ο τελεστής previous cycle, (-). Επομένως η πρόταση θα διαβάζεται «Πάντα, όταν το readySignal είναι ίσο με 1 τότε το ackSignal θα ισούται με 0 στον προηγούμενο κύκλο».

3.4.4 Η ιδιότητα Liveness (Liveness Property)

Μια ιδιότητα Liveness βεβαιώνει ότι εντέλει κάτι καλό θα συμβεί κατά την εκτέλεση μια διεργασίας ενός προγράμματος ή ενός συνόλου καταστάσεων. Έστω η παρακάτω LTL πρόταση:

```
<>({out1==1} && ()()[]{out2 < 2} && (-){out3==0})
```

Η πρόταση αυτή διαβάζεται ως εξής «Τελικά, η out1 θα είναι ίση με 1 στη συνέχεια μετά από δύο κύκλους πάντα η out2 θα είναι μικρότερη του 2 και στον προηγούμενο κύκλο η out3 θα είναι ίση με το 0». Το παράδειγμα αυτό χρησιμοποιεί τον τελεστή eventually (με σύμβολο το διαμάντι <>), τον τελεστή always [ ] και τους τελεστές next cycle ( ) και previous cycle (-).

3.4.5 Καταστάσεις Τερματισμού (End States)

Εάν μια διεργασία δεν ολοκληρώνει την εκτέλεσή τους πριν από τον τερματισμό της εκτέλεσης του μοντέλου, το Spin σηματοδοτεί την διεργασία με την σημαία λανθασμένης κατάστασης τερματισμού (invalid end-state). Αυτή είναι μια συνηθισμένη τεχνική που χρησιμοποιείται για την ανίχνευση αδιεξόδου του μοντέλου. Εάν ο σχεδιαστής του συστήματος σχεδιάσει μια διεργασία για να σταματήσει πριν αυτή ολοκληρώσει τότε η διεργασία αυτή πρέπει να «σημαδευτεί» με την ετικέτα end label.

3.4.6 Καταστάσεις προόδου (Progress States)

Το Spin χρησιμοποιεί τις καταστάσεις προόδου για την ανίχνευση παρουσίας δηλώσεων οι οποίες εκτελούνται είτε άπειρα σε σχέση με τον χρόνο είτε όχι, προσθέτοντας σε αυτές την ετικέτα progress ακριβώς για τον λόγο που θέλει το εργαλείο να δει κάθε πόσο εκτελείται μια δήλωση. Το εργαλείο θα επιστρέψει λάθος στην περίπτωση εάν δεν μπορέσει να εκτελέσει μια διεργασία progress για άπειρες φορές. Με άλλα λόγια, οποιαδήποτε διεργασία έχει την ετικέτα progress δεν μπορεί να παραμείνει μπλοκαρισμένη χωρίς αυτή να εκτελεστεί. Για παράδειγμα :

```
proctype ProcessA(){
do
:: chanAtoB!p -> progress: chanAtoB?v
od}
```

Η παρουσία της ετικέτας progress απαιτεί εκτέλεση της δήλωσης chanAtoB?y άπειρες φορές. Ο μόνος τρόπος αυτή η δήλωση να εκτελεστεί άπειρες φορές είναι μόνο αν και η δήλωση chanAtoB!p εκτελεστεί και αυτή άπειρες φορές.

3.4.7 Καταστάσεις αποδοχής (Accept States)

Μία κατάσταση αποδοχής συμπροφέρεται ακριβώς αντίθετα από μία κατάσταση προόδου. Χρησιμοποιείται για την ανίχνευση της ορθότητας προδιαγραφών των δηλώσεων σε ένα μοντέλο. Εάν το Spin βρει μια κατάσταση αποδοχής να εκτελείται άπειρες φορές, τότε επιστρέφεται λάθος. Ο χρήστης μπορεί εύκολα να θέσει μια κατάσταση «παγίδα» προσθέτοντάς την ετικέτα accept και στην συνέχεια το εργαλείο θα ελέγξει για έναν άπειρο αριθμό εάν μπορεί να εισέλθει την κατάσταση «παγίδα». Εάν μπορέσει τότε η συνθήκη που οδηγεί στην παγίδα έχει γίνει αληθής και το Spin θα έχει καταφέρει να βρει λάθος. Για παράδειγμα έχοντας το παράδειγμα,

```
proctype ProcessA()
{ do
:: chanAtoB!p -> accept: chanAtoB?v
od
}
```

η διεργασία αυτή στο τέλος θα μπλοκαριστεί στην αρχή της δήλωσης chanAtoB!p. Εάν αυτή δεν μπλοκαριστεί τότε η διεργασία θα αδυνατούσε να τερματίσει, που σημαίνει ότι υπάρχει λάθος. Και αυτό γιατί η κατάσταση αποδοχής θα επισκέπτονταν άπειρες φορές αφού η διεργασία δεν θα μπορεί να σταματήσει κάτι που σημαίνει ότι υπάρχει λάθος.

3.4.8 Ισχυρισμοί (Never claims)

Το εργαλείο Spin χρησιμοποιεί τις δηλώσεις never-claim για να ορίσει χρονικούς τύπους. Αυτοί οι ισχυρισμοί χρησιμοποιούνται για τον έλεγχο μη επιθυμητών ή λανθασμένων καταστάσεων. Το Spin θα επιστρέψει λάθος εάν βρει κάποια ακολουθία εκτέλεσης η οποία να τερματίζει στο σημείο όπου ένα never-claim έχει τερματίσει στο

τέλος μιας διεργασίας. Συνδυάζοντας τα never-claims με τις καταστάσεις αποδοχής που προαναφέρθηκαν το εργαλείο μπορεί να ανιχνεύσει λανθασμένους άπειρες (κυκλικές) συμπεριφορές με το να σημαδεύει το σύνολο των δηλώσεων σαν ένα never-claim μαζί με μία ετικέτα accept δημιουργώντας έτσι μία κατάσταση αποδοχής. Στη συνέχεια το Spin ελέγχει τις επιλεγμένες δηλώσεις για άπειρες φορές εάν μπορεί να εισέλθει μέσα στην μπλοκαρισμένη κατάσταση αποδοχής.

3.4.9 Επιβεβαιώσεις (Assertions)

Η πρόταση σε Promela : assert(destid==a), αποτελεί μια επιβεβαίωση. Μία επιβεβαίωση είναι μία Boolean συνθήκη η οποία πρέπει να ικανοποιείται όταν μια διεργασία φτάσει σε μια συγκεκριμένη κατάσταση. Εάν η συνθήκη είναι αληθής τότε η επιβεβαίωση δεν έχει καμία επίδραση. Αντίθετα, η ορθότητα της δήλωσης παραβιάζεται εάν υπάρχει τουλάχιστον μία ακολουθία εκτέλεσης στην οποία η συνθήκη είναι λάθος την στιγμή που η επιβεβαίωση γίνεται εκτελέσιμη.

3.4.10 Non-progress Cycles

Με σκοπό να μπορεί κάποιος να ισχυριστεί την απουσία ενός non-progress cycle, θα πρέπει να βρεθεί ένας τρόπος να οριστούν καταστάσεις η οποίες αποδεικνύουν την πρόοδο της εκτέλεσης ενός μοντέλου συστήματος. Μια ετικέτα κατάστασης προόδου σημειώνει ότι μια κατάσταση πρέπει να εκτελεστεί για να δηλώσει την πρόοδο της εκτέλεσης του μοντέλου. Οι ακολουθίες εκτέλεσης οι οποίες παραβιάζουν την συγκεκριμένη ετικέτα ονομάζονται non-progress cycles.

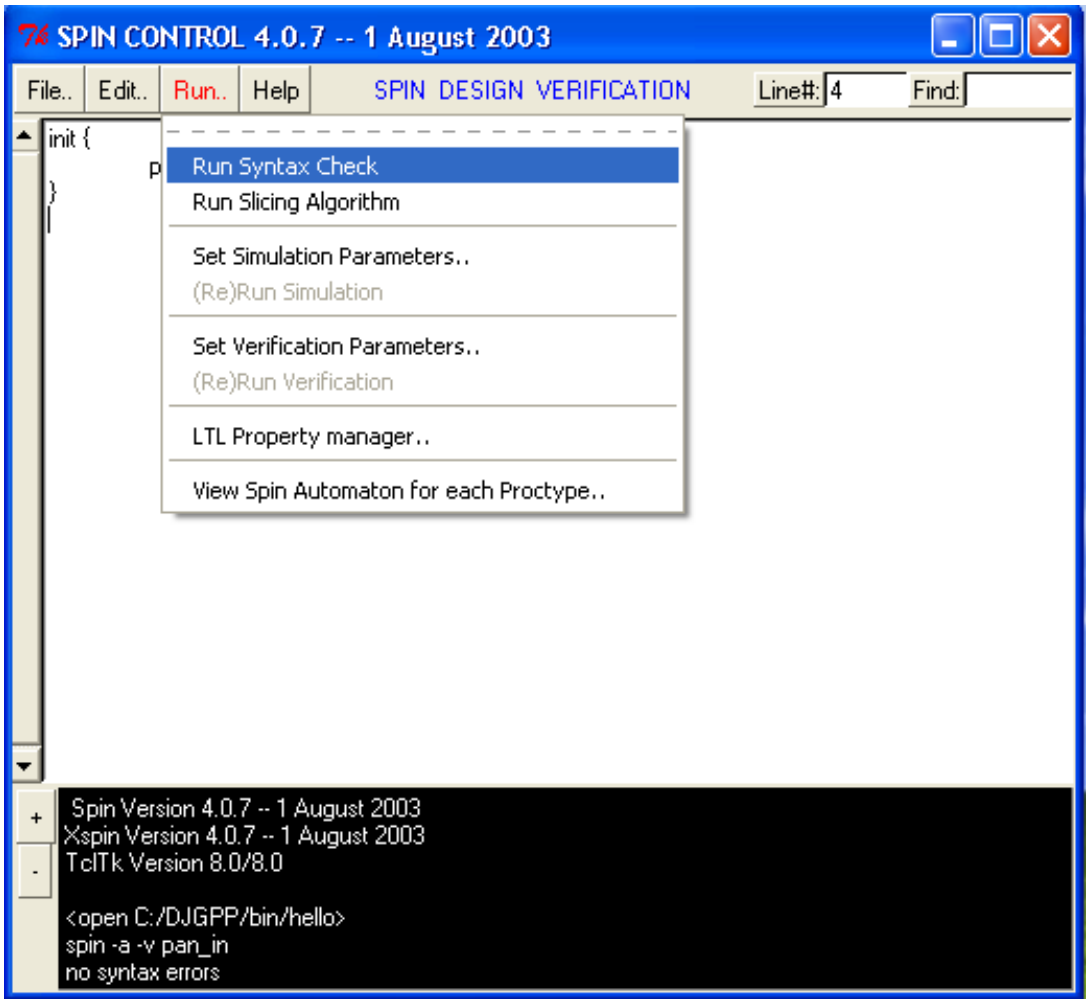
3.4.11 Χρονικοί ισχυρισμοί (Temporal Claims)

Οι χρονικοί ισχυρισμοί αποτελούν δηλώσεις ισχυρισμού οι οποίες ορίζουν την χρονική διάταξη των ιδιοτήτων για καταστάσεις του μοντέλου. Αλλά από την στιγμή που όλα τα κριτήρια ορθότητας στην γλώσσα Promela βασίζονται πάνω σε ιδιότητες που ισχυριζόμαστε ότι είναι αδύνατες, οι χρονικοί ισχυρισμοί πρέπει επίσης να εκφράζουν ιδιότητες οι οποίες είναι αδύνατες. Για παράδειγμα ένας τέτοιος ισχυρισμός ακολουθεί παρακάτω :

```
never
{
     do
          : skip
          :: condition -> break
     od;
accept: do
:: condition
}
```

3.5 Προσομοίωση με το εργαλείο XSpin

Πριν το στάδιο της προσομοίωσης είναι απαραίτητος ένας συντακτικός έλεγχος λαθών στο μοντέλο του συστήματος που έχουμε συντάξει στην γλώσσα Promela. Είναι ευνόητο ότι για να μπορέσει το εργαλείο να εφαρμόσει οποιαδήποτε διαδικασία πάνω στο μοντέλο θα πρέπει αυτό να μην περιέχει λάθη στην σύνταξή του. Κάτι τέτοιο θα γίνει με την προσεκτική σχεδίαση και ορισμό του μοντέλου σύμφωνα με τους κανόνες που διέπουν την γλώσσα αναπαράστασης Promela, και που σε αυτήν την αναφορά αναφέρονται στο παράρτημα (). Εάν το εργαλείο βρει λάθος θα ειδοποιήσει τον χρήστη με το λάθος και την γραμμή του κώδικα που υπάρχει αυτό. Ο έλεγχος αυτός πραγμα-

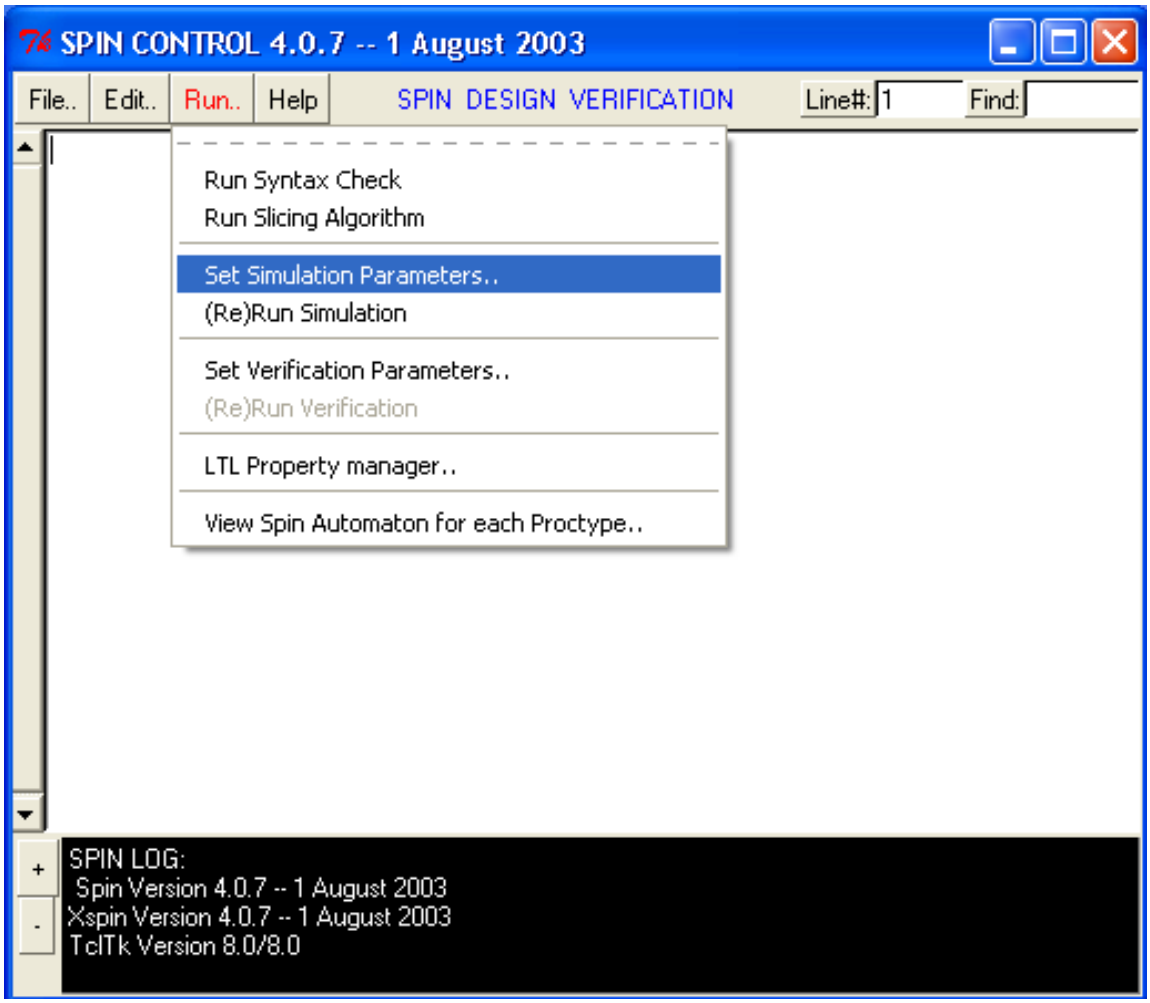


Εικόνα 3.5.1: Syntax check with XSPIN

τοποιείται με την επιλογή *syntax check* και μπορεί να μας βοηθήσει να βρούμε σημαντικά λάθη στον κώδικα του προγράμματός μας, το οποίο είναι σημαντικό για τη συνέχεια. Στο παραπάνω παράδειγμα παρατηρούμε ότι δεν υπάρχουν συντακτικά λάθη στον κώδικά μας (*no syntax errors*). Επίσης βλέπουμε και την εντολή που χρησιμοποιεί το SPIN για να πραγματοποιήσει τον έλεγχο (*spin –a –v pan_in*). Για οποιαδήποτε ενέργεια πραγματοποιήσουμε το εργαλείο κρατάει στοιχεία είτε ιστορικού περιεχομένου είτε στοιχεία που θα το βοηθήσουν περαιτέρω για την σωστό έλεγχο του

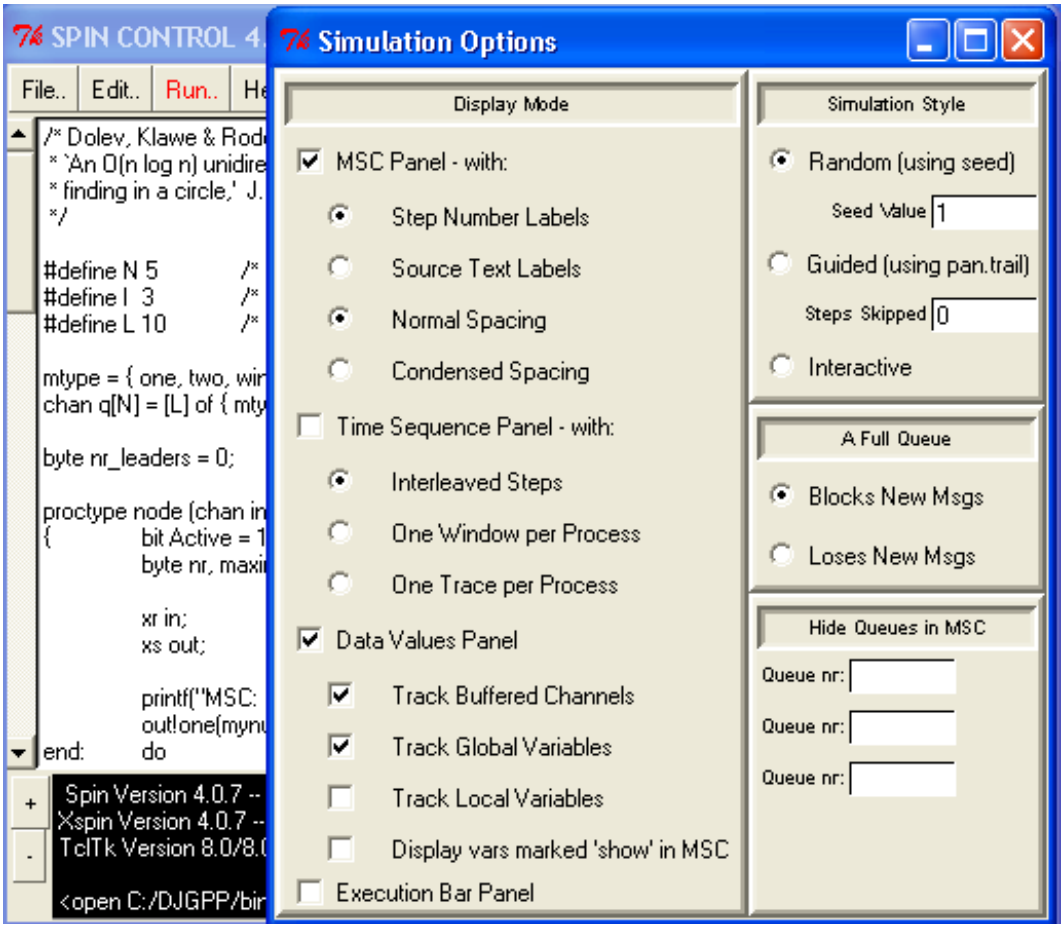
μοντέλου (στην προκειμένη περίπτωση κρατούνται στοιχεία στο αρχείο pan_in ότι για το συγκεκριμένο μοντέλο έχει ήδη γίνει συντακτικός έλεγχος και ότι αυτός ήταν σωστός).

Μετά τον απαραίτητο συντακτικό έλεγχο, ακολουθεί η προσομοίωση του μοντέλου. Το γραφικό περιβάλλον του SPIN, δηλαδή το XSPIN εμφανίζει το επόμενο παράθυρο, από το οποίο επιλέγουμε *Set Simulation Parameters* για να εισάγουμε τις παραμέτρους με τις οποίες επιθυμούμε να προσομοιώσουμε τον αλγόριθμο ή το πρωτόκολλό μας.



Εικόνα 3.5.2: Προσομοίωση με το XSPIN

Στο κάτω μέρος του παραθύρου φαίνονται οι εκδόσεις του SPIN, XSPIN, καθώς και του Tcl/Tk, το οποίο είναι απαραίτητο για τη λειτουργία του XSPIN. Έπειτα εμφανίζεται το ακόλουθο παράθυρο με τις επιλογές που μπορούμε να κάνουμε για την προσομοίωση. Είναι καλύτερο σε περίπτωση που δε γνωρίζουμε πως συμπεριφέρεται το πρόγραμμα να πραγματοποιήσουμε την προσομοίωση με τις default επιλογές. Όταν αποκτήσουμε την απαραίτητη εξοικείωση μπορούμε να δοκιμάσουμε και τις υπόλοιπες δυνατότητες που μας προσφέρει το SPIN. Οι διάφορες επιλογές εξηγούνται αναλυτικά παρακάτω.Το παράθυρο της προσομοίωσης, όπως φαίνεται και στο σχήμα, αποτελείται από τα εξής τμήματα: Το Display Mode, Simulation Style, A Full Queue που περιγράφονται παρακάτω:



Εικόνα 3.5.3: Παράμετροι Προσομοίωσης

Display mode

Το κομμάτι αυτό χρησιμοποιείται για την εμφάνιση των αποτελεσμάτων της προσομοίωσης, ανάλογα με το τι μας ενδιαφέρει να μελετήσουμε και αποτελείται από τα:

- Γράφημα ακολουθίας μηνυμάτων (MSC , **M**essage **S**equence **C**hart)
- Ακολουθία χρόνου (Time sequence)
- Τιμές δεδομένων (Data Values)
- 'Μπάρα' Εκτέλεσης (Execution Bar)

Συνοπτικά θα δοθεί η ανάλυση του κάθε αποτελέσματος που εμφανίζεται κατά την προσομοίωση αφού θεωρείται αναγκαίο τα αποτελέσματα αυτά να αναλυθούν με την χρήση παραδείγματος. Έτσι περνώντας στο πρώτο πιο βασικό αποτέλεσμα της προσομοίωσης, το γράφημα ακολουθίας μηνυμάτων MSC, που αποτελεί μια πολύ σημαντική γραφική αναπαράσταση των μηνυμάτων μεταξύ των διεργασιών του μοντέλου που ανταλλάσσονται με το πέρας της εκτέλεσης. Εμφανίζονται έτσι την επικοινωνία αυτή παρουσιάζονται και οι τιμές των συγκεκριμένων μεταβλητών που χρησιμοποιούν οι διεργασίες δίνοντας το αποτέλεσμά τους σε κάθε βήμα ξεχωριστά. Κάτι τέτοιο φυσικά έχει σχέση και με τις επιλογές που έχουμε κάνει όπως φαίνεται και από την εικόνα 3.4.3 όπου διαλέγουμε τον τρόπο με τον οποίο θέλουμε να μας εμφανιστούν τα αποτελέσματα. Το δεύτερο αποτέλεσμα της προσομοίωσης είναι και η

ακολουθία του χρόνου όπου μας εμφανίζεται βήμα προς βήμα η προσομοίωση και η εκτέλεση κάθε εντολής του μοντέλου σε σχέση με τον χρόνο. Έτσι μπορεί κανείς να παρατηρήσει την πότε εκτελείται μια εντολή πότε αυτή δηλαδή ενεργοποιείται αλλά και πότε αυτή απενεργοποιείται. Και εδώ τα αποτελέσματα μπορούν να διαφοροποιηθούν βάσει των επιλογών αρχικά. Τέλος τα δύο τελευταία αποτελέσματα που εμφανίζονται, οι τιμές δεδομένων και η μπάρα εκτέλεσης αφορούν τις τιμές και την χρησιμοποίηση των μεταβλητών στην διάρκεια αλλά και στο τέλος της προσομοίωσης. Η χρησιμότητα όλων των αποτελεσμάτων θα φανεί αναλυτικά μέσα από το παράδειγμα των 4ων στρατιωτών, που εξηγείται παρακάτω.

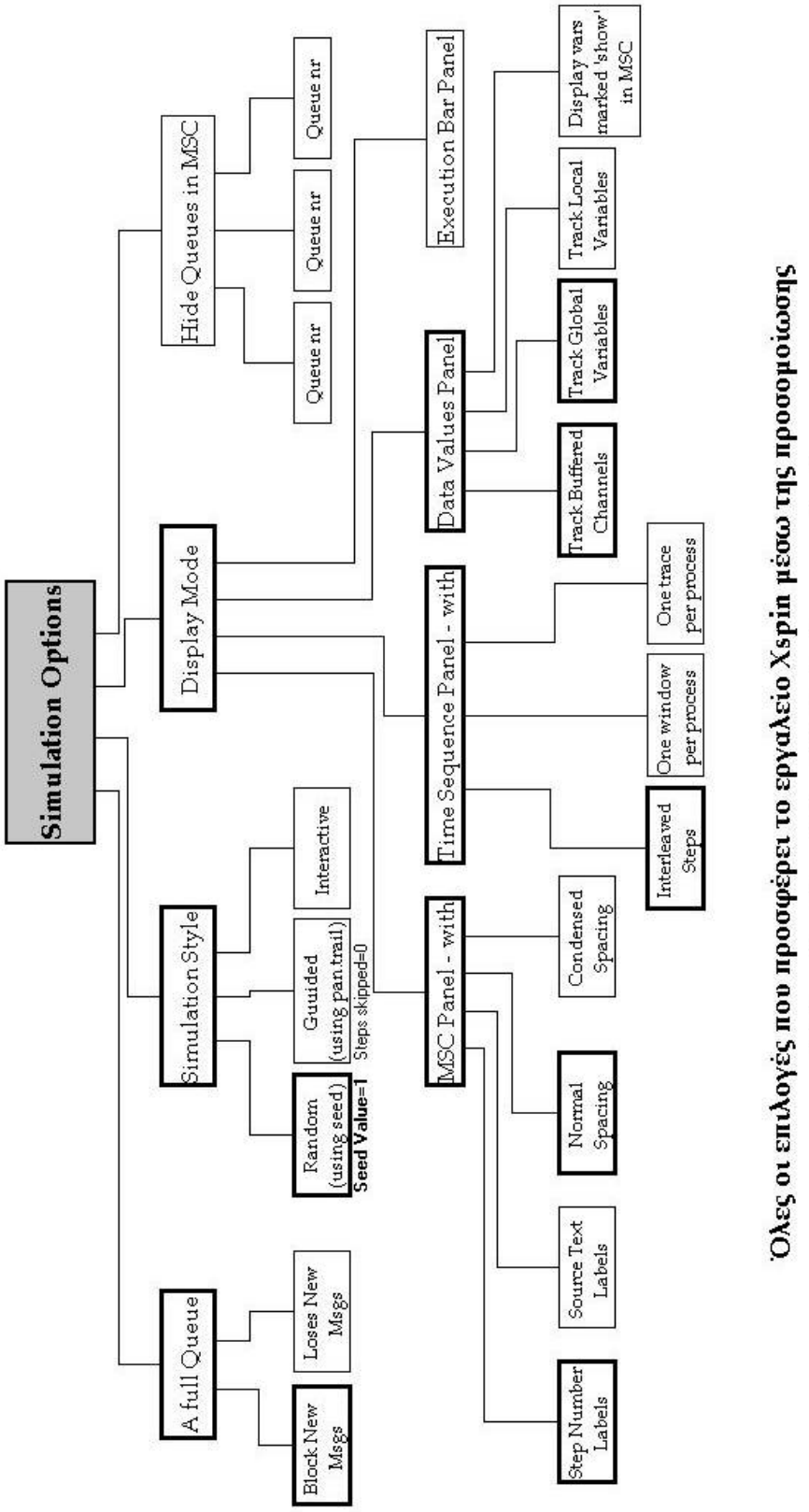
Τύπος προσομοίωσης (Simulation Style)

- *Τυχαία (random):* όλες οι μη ντετερμινιστικές αποφάσεις είναι τυχαίες και ο χρήστης δεν κάνει καμία ενέργεια όση ώρα διαρκεί η προσομοίωση. Είναι χρήσιμη όταν δε γνωρίζουμε πολλά για το πως συμπεριφέρεται ο αλγόριθμος, αλλά και όταν δεν έχουμε πολλές γνώσεις για τον «έλεγχο μοντέλων». Επίσης είναι ακριβής, όσο δεν γίνονται αλλαγές στο πεδίο seed value.
- *Διαλογική (interactive):* χρησιμοποιείται για να οδηγήσουμε την εκτέλεση σε ένα γνωστό σημείο. Σε κάθε σημείο της εκτέλεσης ο χρήστης πρέπει να παίρνει μη ντεντερμινιστικές αποφάσεις.
- *Με Καθοδήγηση (guided) :* χρησιμοποιείται για να ακολουθήσουμε ένα ίχνος λάθους που παράχθηκε κατά τη διάρκεια της επαλήθευσης. Αν το ίχνος απαιτεί να γίνουν πολλά βήματα εκτέλεσης, μπορούμε να χρησιμοποιήσουμε την επιλογή "Steps Skipped" για να προσπεράσουμε κάποια βήματα για οικονομία χρόνου.

Full Queue

Πρόκειται για την περίπτωση εκείνη όπου ο χρήστης επιθυμεί να λαμβάνει ή όχι μηνύματα της προσομοίωσης. Όλα τα μηνύματα αυτά έχουν να κάνουν με την εσωτερική χρήση των αρχείων που δημιουργούνται για την ολοκλήρωση της προσομοίωσης αλλά και με τα μηνύματα τυχόν λάθους του μοντέλου στα αποτελέσματα που παράγονται από αυτό.

3.5.1 Διάγραμμα επιλογών προσομοίωσης μέσω του XSpin

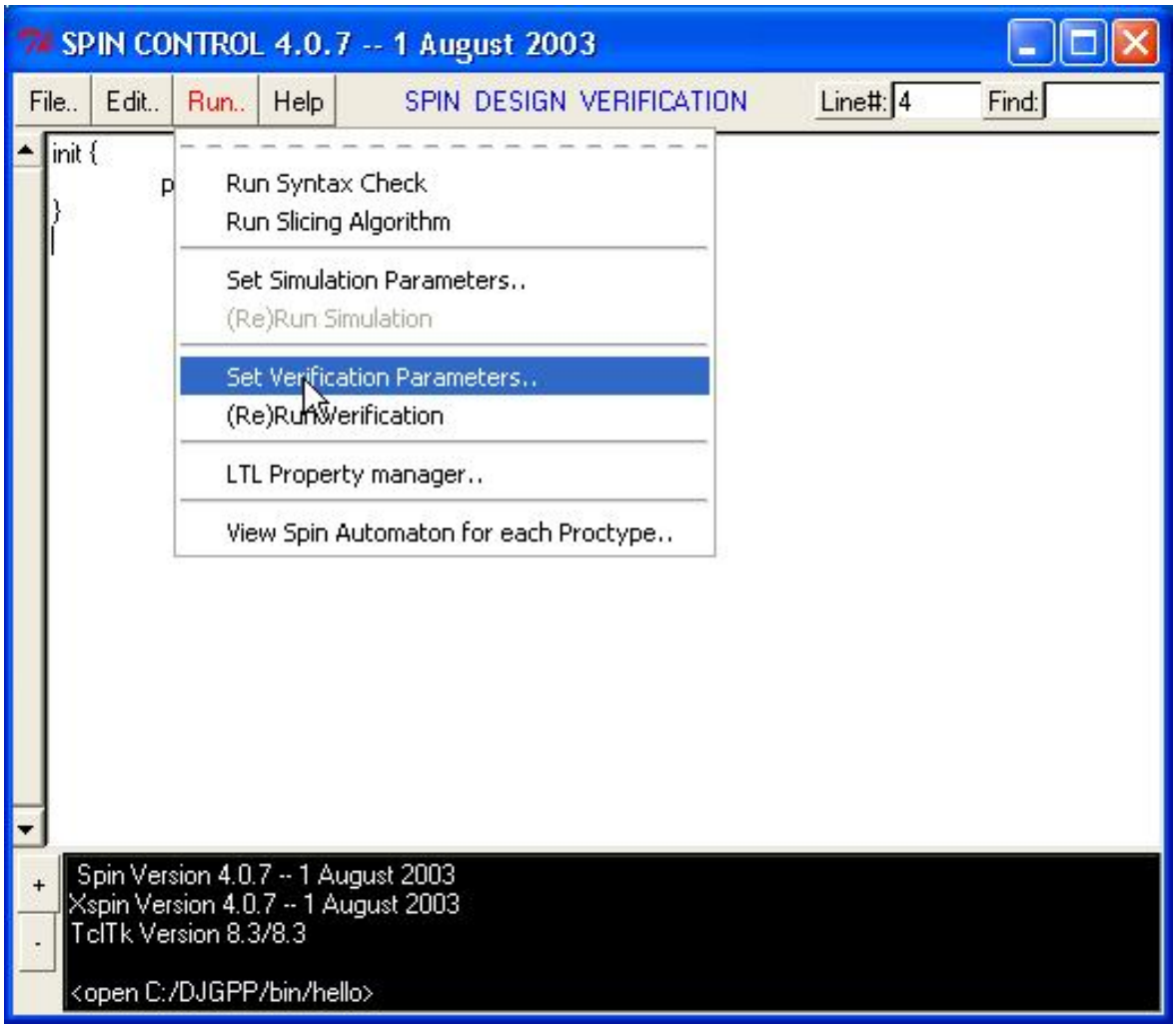


Όλες οι επιλογές που προσφέρει το εργαλείο Xspin μέσω της προσομοίωσης

(Τα έντονα 'κουτιά' αποτελούν τις default επιλογές της προσομοίωσης)

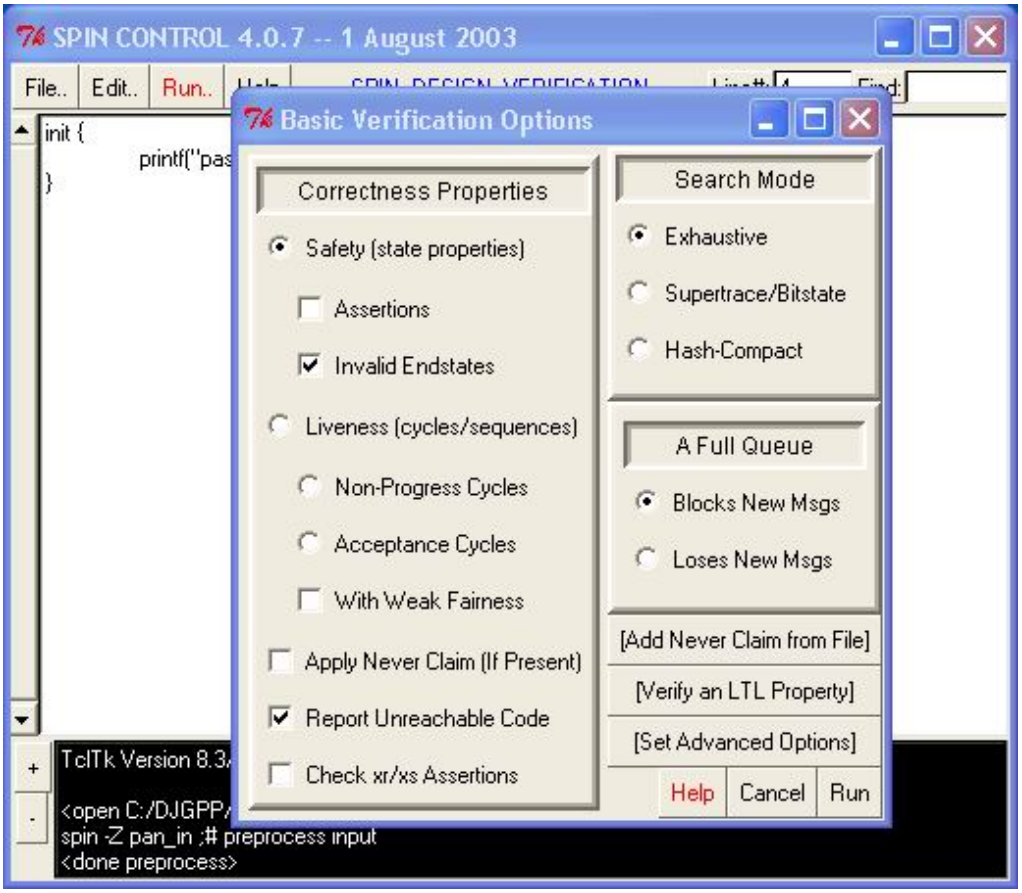
3.6 Επαλήθευση με το εργαλείο XSpin

Έχοντας τελειώσει από την απαραίτητη προσομοίωση με το εργαλείο, το επόμενο (και πιο σημαντικό βήμα) είναι η επαλήθευση του μοντέλου. Η επιλογή για αυτή την διαδικασία είναι η 'Set Verification Parameters..' και φαίνεται στην εικόνα παρακάτω :



Εικόνα 3.6.1: Επαλήθευση με το XSPIN

Πριν ξεκινήσει κανείς για την πρώτη επαλήθευση με το εργαλείο, θα ήταν ασφαλές όλες οι παράμετροι της επαλήθευσης (verification parameters) να μείνουν 'απείραχτες', τρέχοντας για μια φορά μια default επαλήθευση. Έτσι αμέσως μετά την πρώτη επιλογή στο εργαλείο θα πρέπει να είναι επιλεγμένοι οι εξής παράμετροι της επαλήθευσης, όπως φαίνεται στην εικόνα :



Εικόνα 3.6.2: Παράμετροι Επαλήθευσης

Όπως φαίνεται και από την εικόνα 3.6.2 οι προεπιλεγμένοι παράμετροι επαλήθευσης έχουν να κάνουν με την ιδιότητα Safety όπου και το μοντέλο ελέγχεται για λανθασμένες καταστάσεις τέλους και η αναφορά για unreachable code η οποία θα επιστραφεί στο τέλος της επαλήθευσης (και εάν υπάρχει). Επίσης για την πρώτη φορά επιλέγεται εξαντλητικός αλγόριθμος αναζήτησης (επιλογή exhaustive) αλλά και η επιλογή 'Blocks New Msgs' η οποία στο τέλος της επαλήθευσης μπλοκάρει καινούργια μηνύματα στην 'ουρά' του αποτελέσματος. Παίρνοντας τα πράγματα με την σειρά οι παράμετροι οι οποίοι θα πρέπει να δοθούν στο εργαλείο για την επαλήθευση έχουν σχέση με *ιδιότητες ορθότητας (Correctness Properties)*, με την *επιλογή της αναζήτησης* του χώρου καταστάσεων του μοντέλου *(Search Mode)* και με την αποθήκευση μηνυμάτων στο τέλος της επαλήθευσης *(A Full Queue)*. Επιπρόσθετα, υπάρχουν και οι επιλογές [Add Never Claim from File], [Verify an LTL property] και [Set Advanced Options] που θα περιγραφούν παρακάτω.

Όπως προαναφέρθηκε, μια Safety property έρχεται να βεβαιώσει ότι τίποτα κακό δεν θα συμβεί σε μια κατάσταση. Το Spin δίνει την δυνατότητα δύο περιπτώσεων ελέγχου Safety σε ένα μοντέλο :

- Assertions : Πρόκειται για την επιλογή η οποία θα ελέγχει στο μοντέλο εάν γίνονται παραβιάσεις 'βεβαιότητας'. Κάτι τέτοιο βέβαια θα είναι λογικό να γίνεται μόνο εάν μέσα στο μοντέλο έχουν οριστεί προτάσεις assertions ,όπου ο χρήστης θα θέλει μια συγκεκριμένη ιδιότητα πάντα να ικανοποιείται. Στην

αντίθετη περίπτωση όπου δεν έχει οριστεί assert statement το Spin θα αναφέρει λάθος με το μήνυμα 'There are no assert statements on the spec".

- Invalid Endstates : Πρόκειται για την επιλογή στην οποία ελέγχεται το μοντέλο για λανθασμένες καταστάσεις τέλους. Έτσι το Spin θα έχει την ικανότητα μετά από έλεγχο να αναφέρει πιθανές καταστάσεις deadlock ,όπου η εκτέλεση δεν είναι σωστή. Σε περίπτωση ορθότητας της επιλογής δεν γίνεται αναφορά λάθους.

Όπως φαίνεται και από το παραπάνω στιγμιότυπο, το Spin έχει την ικανότητα να ελέγχει και για ιδιότητες Liveness (ιδιότητες συχνότητας των καταστάσεων όπως άπειρες καταστάσεις). Στην συγκεκριμένη περίπτωση γίνεται έλεγχος για δύο τύπων κύκλων, acceptance cycles και progress cycles. Και εδώ ,ο χρήστης θα πρέπει μέσα στο μοντέλο του να έχει ορίσει progress ή accept cycles (με την εντολή label σε Promela) αλλιώς το Spin θα βγάλει μηνύματα λάθους (ότι δεν έχουν οριστεί μέσα στο μοντέλο progress ή accept labels). Επομένως οι επιλογές, όπως φαίνονται και από την εικόνα 3.6.2 είναι :

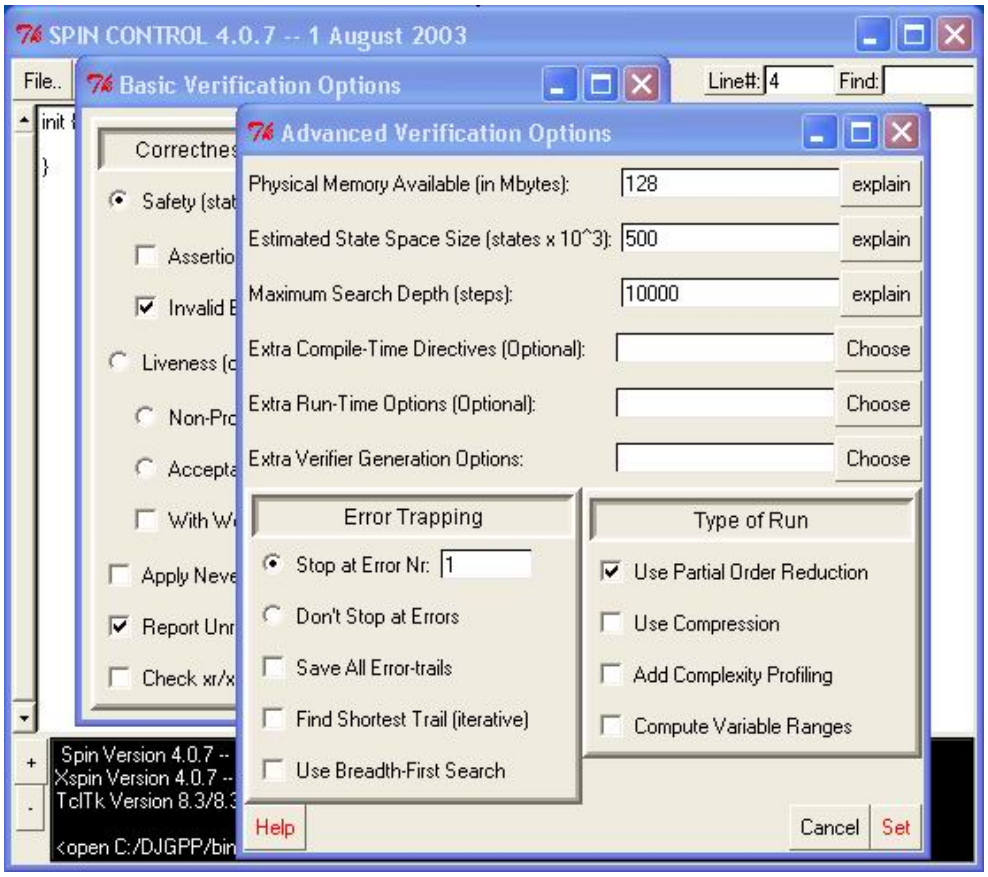
- Non-Progress Cycles : Πρόκειται για την περίπτωση όπου η εκτέλεση δεν περνά από το σημείο εκείνο του κώδικα το οποίο ο χρήστης έχει μαρκάρει με την ετικέτα 'progress'
- Acceptance Cycle : Πρόκειται για την περίπτωση όπου η εκτέλεση περνά άπειρες φορές από το σημείο εκείνο του κώδικα το οποίο ο χρήστης έχει μαρκάρει με την ετικέτα 'accept'
- *Weak Fairness* : Η συγκεκριμένη επιλογή θα πρέπει να χρησιμοποιείται μόνο όταν είναι απαραίτητο για την αποφυγή αναφορών λάθους. Μπορεί να αυξήσει σημαντικά τις απαιτήσει της CPU σε χρόνο ,περίπου στο διπλάσιο του αριθμού των διεργασιών.

Συνεχίζοντας την ανάλυση των παραμέτρων επαλήθευσης του Spin, θα έχουμε τις εξής επιλογές (όπως φαίνονται και από το παράθυρο verification options :

- *'Apply Never Claim (if present)'* : αναφέρεται στον ορισμό μιας never claim πρότασης. Πρόκειται για την περίπτωση όπου ο χρήστης έχει συντάξει μια πρόταση βάσει της οποίας δεν θέλει ποτέ να επαληθευτεί. Εάν δεν έχει οριστεί τότε το Spin θα το αναφέρει με μήνυμα λάθους. Είναι δυνατόν να εφαρμοστεί μια ένας ισχυρισμός never claim μαζί με Safety ιδιότητα, όταν θέλουμε να περιορίσουμε την αναζήτηση στις περιπτώσεις εκείνες που ταιριάζουν με το never-claim (user-guided search pruning technique).
- *'Report unreachable code'* : γίνεται αναφορά από το εργαλείο για κώδικα που δεν εκτελείται
- '*Check xr/xs assertions*' : γίνεται έλεγχος για xr/xs assertions. Πρέπει να αναφερθεί ότι στο μοντέλο που εξετάζουμε για να μπορέσουμε να κάνουμε την συγκεκριμένη επιλογή θα πρέπει να έχουμε ορίσει τέτοιου είδους assertions

Περνώντας στην επιλογή του αλγόριθμου αναζήτησης , το Spin παρέχει τρεις επιλογές :

- *Exhaustive* : Πρόκειται για εξαντλητική αναζήτηση του χώρου καταστάσεων του μοντέλου και προτείνεται κυρίως για μικρό μέγεθος χώρου
- *Supertrace/Bitstate* : Πρόκειται για αναζητήσεις προσεγγιστικές οι οποίες και χρησιμοποιούνται μόνο όταν ο χώρος των καταστάσεων δεν προσφέρεται για εξαντλητική αναζήτηση .Η επιλογή αυτή επιτρέπει την επιστροφή μιας τουλάχιστον προσεγγιστικής απάντησης για την ορθότητα μοντέλων τα οποία σε αντίθετη περίπτωση δεν μπορούσαν να επαληθευθούν από ένα πεπερασμένων καταστάσεων σύστημα
- *Hash-Compact* : Πρόκειται για αναζήτηση η οποία συμπιέζει τον χώρο των καταστάσεων και τίθεται απαραίτητη σε περιπτώσεις χαμηλών πόρων σε μνήμη



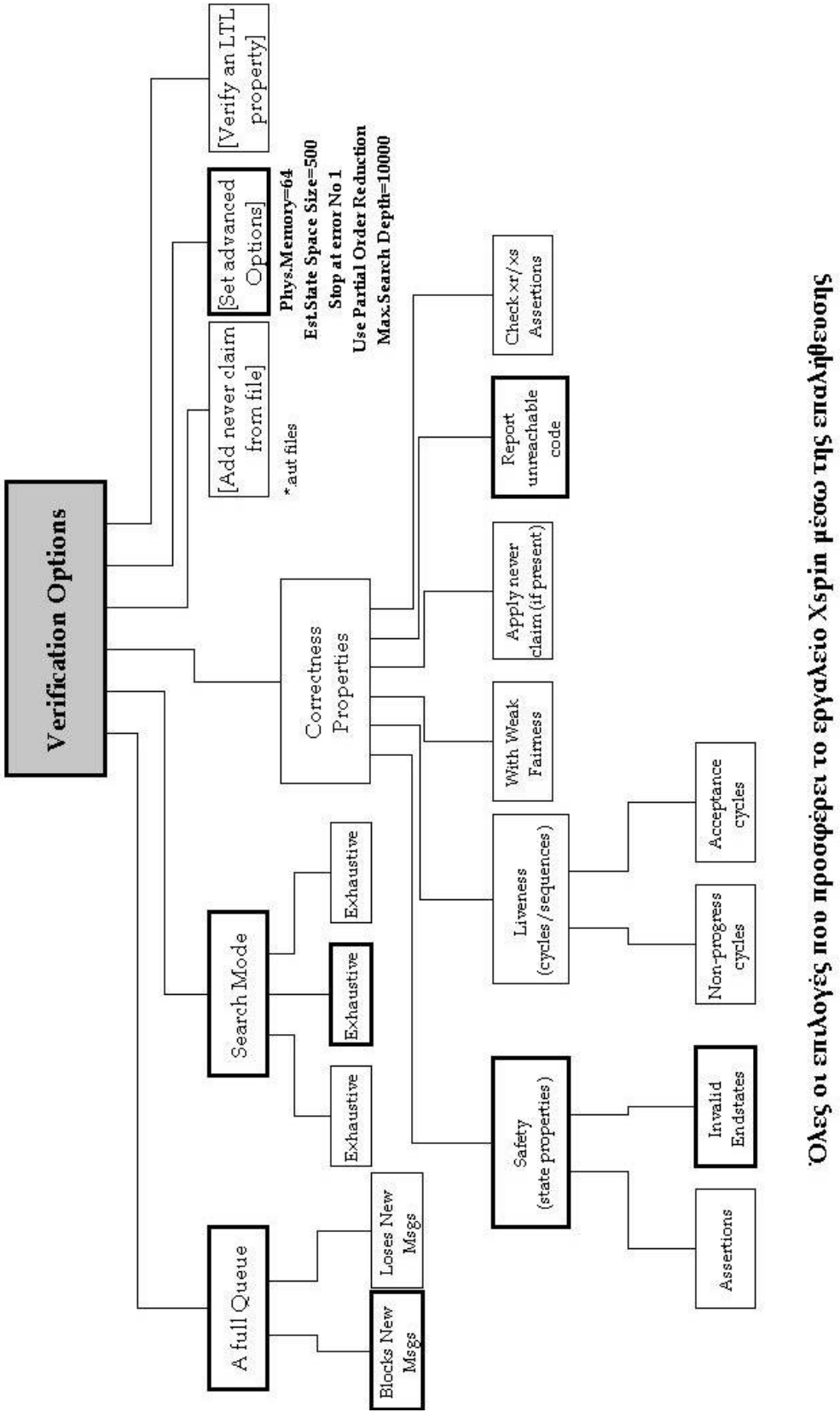
Εικόνα 3.6.3: Προχωρημένες παράμετροι επαλήθευσης

Τέλος ,το εργαλείο Spin δίνει την δυνατότητα στον σχεδιαστή να ορίσει μόνος του κάποιες προχωρημένες παραμέτρους που έχουν να κάνουν με την επαλήθευση ,μέσω της επιλογής *[Set Advanced Options]* .Σε αυτήν την περίπτωση θα εμφανίζεται το ακόλουθο παράθυρο. Σε περίπτωση που 'μείνει' κάποιος από μνήμη ,το Spin δίνει την δυνατότητα αλλαγής των ρυθμίσεών του μέσω αυτών των Advanced Options. Ένας παράγοντας είναι πολύ σημαντικός να οριστεί σωστά από την αρχή :το μέγεθος της φυσικής μνήμης του υπολογιστή που λειτουργεί το Xspin. Εάν βρεθεί λάθος κατά την default επαλήθευση ,σωστό θα ήταν να προσπαθήσετε να μειώσετε το βάθος της αναζήτησης, έτσι ώστε να βρεθεί ένα πιο σύντομο ,ισοδύναμο μονοπάτι εκτέλεσης. Μόλις βρεθεί το σωστό βάθος, προχωρήστε στην καθοδήγηση προσομοίωση (guided simulation) ,όπου και μπορεί κανείς να ελέγξει το error-trail του μοντέλου λεπτομερώς. Επίσης μπορεί να γίνει η επιλογή του 'Find Shortest Trail' ,αλλά κάτι τέτοιο θα

δημιουργούσε προβλήματα στην μνήμη και στον χρόνο εκτέλεσης. Παράλληλα με τα παραπάνω ,θα ήταν σωστό να αφήνεται πάντα ενεργή η επιλογή για partial order reduction και να απενεργοποιείται μόνο σε περιπτώσεις debugging. Σε περίπτωση που επιθυμεί ο χρήστης στο να σώσει όλα τα error-trails του Spin θα πρέπει να αντιγράψει εκείνο που τον ενδιαφέρει (έξω από τον φάκελο του Xspin) μόλις τελειώσει η επαλήθευση. Οι επιλογές που μπορεί να κάνει ο χρήστης επεξηγούνται παρακάτω :

- *Physical Memory Available:* Πρόκειται για το μέγεθος της φυσικής (όχι εικονικής) μνήμης του υπολογιστή. Όπως φαίνεται και από την εικόνα, το μέγεθος της μνήμης είναι επιλεγμένο στα 128 Mbytes(η προεπιλογή ήταν στα 64 Mbytes . Θα ήταν σωστό να οριστεί την πραγματική μνήμη του συστήματος και να μην αλλάζεται ποτέ (μόνο στην περίπτωση που αγοράζεται καινούργια μνήμη). Εάν κατά την επαλήθευση η μνήμη δεν θα είναι αρκετή τότε αυτή θα σταματήσει προς αποφυγή trashing
- *Estimated State Space Size:* Η παράμετρος αυτή χρησιμοποιείται για τον υπολογισμό του πίνακα κατακερματισμού (hash-table). Βάζοντας την παράμετρο αυτή αρκετά μεγάλη ,μπορεί να οδηγήσει σε σημαντική μείωση της μνήμης χωρίς καν να έχει αρχίζει η επαλήθευση (zero states reached). Επίσης ρυθμίζοντας την παράμετρο πολύ χαμηλά ,μπορεί να πάρουμε λανθασμένα αποτελέσματα εξαιτίας συγκρούσεων λόγω κατακερματισμού. Τέλος θα ήταν σωστό μετά το πρώτο επιτυχές τρέξιμο της επαλήθευσης με την προεπιλεγμένη τιμή της παραμέτρου, να διπλασιάζεται παρατηρώντας τον αριθμό των reached states στο output .Η μέθοδος αυτή θα πρέπει να συνεχίζεται μέχρι ο αριθμός αυτός να μείνει σταθερός.
- *Maximum Search Depth:* Πρόκειται για τον αριθμό που δείχνει το μέγεθος του depth-search σωρού που χρησιμοποιείται κατά την διάρκεια της επαλήθευσης. Ο σωρός αυτός χρησιμοποιεί μνήμη, όποτε όσο μεγαλύτερο είναι το μήκος του τόσο πιο πολύ η απαίτηση για μνήμη αυξάνεται και το αντίθετο. Σε περίπτωση που ‘πιαστεί’ το maximum search depth’ στην διάρκεια της επαλήθευσης χωρίς λάθος, ο χρήστης θα πρέπει να διπλασιάζει την παράμετρο του search depth και να επαναλαμβάνει την διαδικασία.

3.6.1 Διάγραμμα επιλογών επαλήθευσης μέσω του XSpin



Όλες οι επιλογές που προσφέρει το εργαλείο Xspin μέσω της επαλήθευσης

(Τα εντονα 'κουτιά' αποτελούν τις default επιλογές της επαλήθευσης)

3.6.2 Αποτελέσματα επαλήθευσης με την χρήση παραδείγματος

Τα αποτελέσματα τα οποία επιστρέφει το εργαλείο XSpin κάτα την επαλήθευση είναι ένας verifier με το όνομα pan.exe. Εκτελώντας το συγκεκριμένο αρχείο θα εμφανιστούν αποτελέσματα που παρουσιάζονται στο παράθυρο Verification Output του XSpin. Παρακάτω ακολουθεί μια ανάλυση των αποτελεσμάτων της επαλήθευσης μέσω του εργαλείου, χωρίς να γίνει αναφορά στο μοντέλο που χρησιμοποιείται και επαληθεύεται (επιστρέφοντας τα συγκεκριμένα αποτελέσματα). Και εδώ θα ήταν καλύτερη η κατανόηση της επαλήθευσης παραπέμποντας στο παράδειγμα των 4 στρατιωτών που ακολουθεί παρακάτω.

Ένα τυπικό παράδειγμα αποτελέσματος επαλήθευσης είναι το παρακάτω:

```
$ pan


(Spin Version 4.0.7 – 1 August 2003)
     + Partial Order Reduction


Full statespace search for:

     never-claim              – (none specified)
     assertion violations     +
     acceptance   cycles      – (not selected)
     invalid endstates   +


State-vector 32 byte, depth reached 13, errors: 0

      74 states, stored
      30 states, matched
     104 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 2 (resolved)
(max size 2^18 states)



1.533     memory usage (Mbyte)

unreached in proctype ProcA
     line 7, state 8, "G = 4"
     (1 of 13 states)
unreached in proctype :init:
     line 21, state 14, "G = 3"
     line 21, state 14, "G = 4"
     (1 of 19 states)
```

Στη συνέχεια θα εξηγήσουμε γραμμή προς γραμμή τα αποτελέσματα της επαλήθευσης.

- *(Spin Version 4.0.7 – 1 August 2003)*

δείχνει την έκδοση του SPIN που χρησιμοποιήθηκε για τη μεταγλώτιση.

- *+ Partial Order Reduction*

το σύμβολο + σημαίνει ότι χρησιμοποιήθηκε ο βασικός αλγόριθμος partial order reduction. Το σύμβολο – θα σήμαινε μεταγλώτιση με την επιλογή DNOREDUCE(εξηγείται παρακάτω).

- *Full statespace search for:*

δείχνει τον τύπο της αναζήτησης

- *never-claim – (none specified)*

το σύμβολο – σημαίνει ότι δεν υπάρχει never-claim, ούτε χρησιμοποιήθηκε LTL φόρμουλα. Αν υπήρχε θα το δηλώναμε με την ντιρεκτίβα DNOCLAIM

- *assertion violations +*

το + δείχνει ότι η αναζήτηση ελέγχει για παραβάσεις των επιλεγμένων assertions

- *acceptance cycles – (not selected)*

το – σημαίνει ότι δεν έγινε έλεγχος για acceptance ή για non-progress κύκλων. Για να γίνει αυτό χρειαζόμαστε την επιλογή DNP.

- *invalid endstates +*

το + δηλώνει ότι έγινε έλεγχος για invalid endstates (για παραδειγμα για αδιέξοδο)

- *State-vector 32 byte, depth reached 13, errors: 0*

η περιγραφή του συστήματος καταστάσεων απαίτησε 32 bytes μνήμης(για κάθε κατάσταση). Το μεγαλύτερο μονοπάτι αναζήτησης περιέχει 13 μετατροπές(από την αρχική κατάσταση). Δε βρέθηκαν λάθη.

- *74 states, stored*

ένα σύνολο από 74 καταστάσεις συστήματος αποθηκεύτηκαν στο χώρο καταστάσεων(κάθε μία αναπαρίσταται αποδοτικά με ένα διάνυσμα 32 bytes).

- *30 states, matched*

σε 30 περιπτώσεις η αναζήτηση επέστρεψε σε μια κατάσταση που είχε προηγουμένως επισκεφθεί.

- *104 transitions (= stored+matched)*

ένα σύνολο από 104 μετατροπές δημιουργήθηκαν κατα τη διάρκεια της επαλήθευσης, το οποίο μπορεί να λειτουργήσει ως στατιστικό στοιχείο.

- *1 atomic steps*

μία μόνο από τις μετατροπές είναι κομμάτι ατομικής ακολουθίας. Όλες οι υπόλοιπες δεν είναι.

- *hash conflicts: 2 (resolved)*

σε δύο περιπτώσεις το προεπιλεγμένο σχήμα κατακερματισμού οδήγησε σε σύγκρουση και τοποθέτησε τις καταστάσεις σε μια συνδεδεμένη λίστα στον πίνακα κατακερματισμού.

- *(max size 2^18 states)*

το μέγεθος του πίνακα κατακερματισμού είναι 2^18 και αντιστοιχεί στην επιλογή w18.

- *1.533 memory usage (Mbyte)*

το συνολικό ποσό της μνήμης που χρησιμοποιήθηκε είναι 1.533 Mb, συμπεριλαμβάνοντας τη στοίβα, τον πίνακα κατακερματισμού και όλες τις υπόλοιπες δομές δεδομένων.

- ```
  unreached in proctype ProcA
  line 7, state 8, "G = 4"
  (1 of 13 states)
  ```

- ```
  unreached in proctype :init:
  line 21, state 14, "G = 3"
  line 21, state 14, "G = 4"
  (1 of 19 states)
  ```

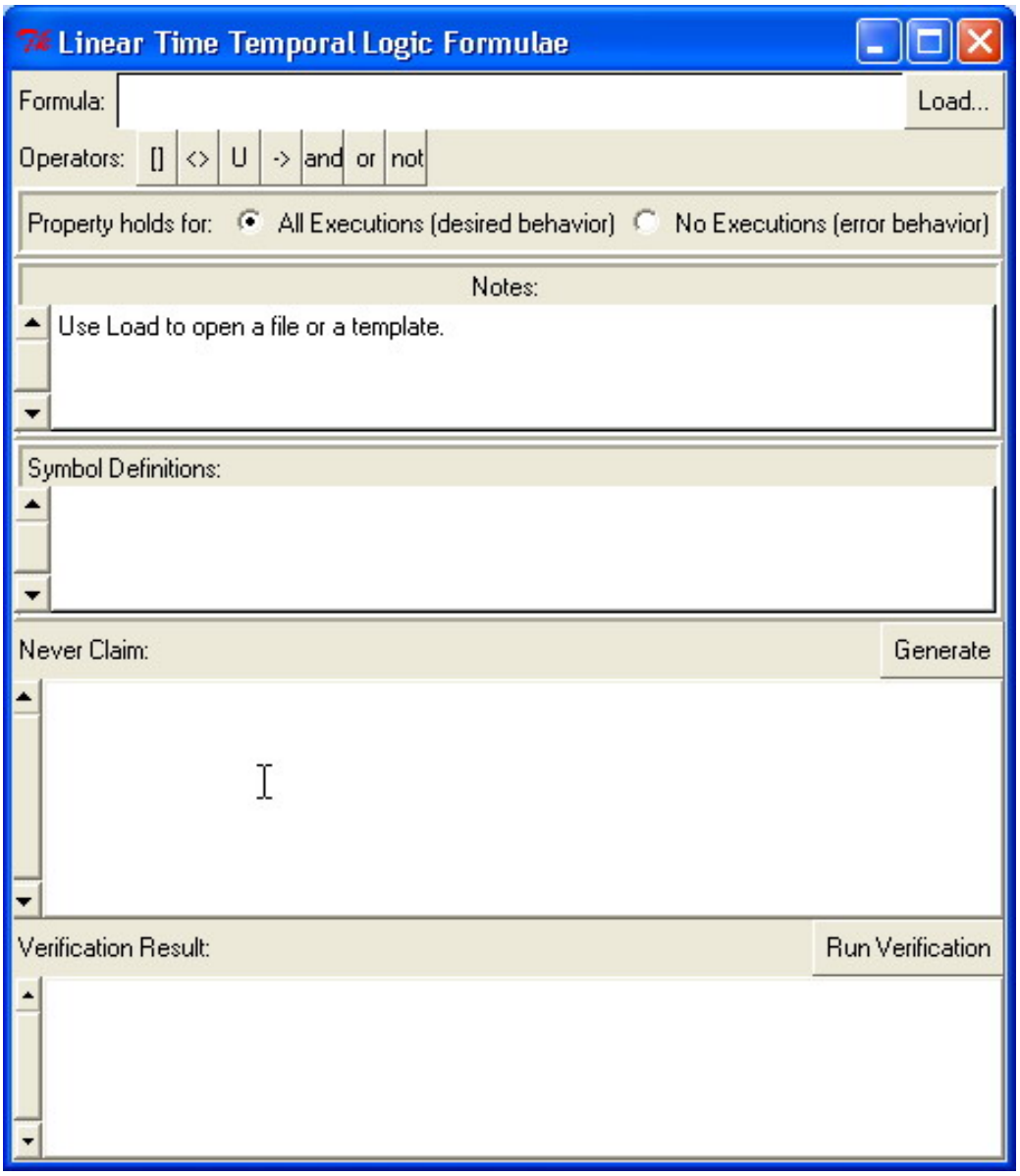
δηλώνουν τους ακριβείς αριθμούς γραμμών για τις βασικές δηλώσεις που δεν έφτασε η αναζήτηση. Αφού χρησιμοποιούμε full statespace search, τότε οι γραμμές αυτές δηλώνουν *dead code.*

3.6.3 Η επιλογή 'Set slicing algorithm..'

Πρόκειται για την επιλογή την οποία προσφέρει το εργαλείο με βάση την οποία γίνονται χρήση τεχνικών slicing. Ο χρήστης έχει την δυνατότητα να τρέξει έναν αλγόριθμο slicing που οποίος θα βοηθήσει στον 'τεμαχισμό' του μοντέλου σε ξεχωριστά κομμάτια (promela). Αυτό θα μπορούσε να συμβεί μόνο όταν ο χρήστης θα έχει ορίσει μέσα στο μοντέλο του, κριτήρια slice αλλιώς το XSpin θα επιστρέψει μηνύματα λάθους. Η επιλογή αυτή είναι ιδιαίτερα σημαντική για τον έλεγχο του μοντέλου του για πλεονασμό αλλά και όταν θέλει να κάνει αποσφαλμάτωση αυτού. Στην εργασία αυτή δεν πρόκειται να ασχοληθούμε με την επιλογή αυτή, γι αυτό και περαιτέρω ανάλυση παραλείπεται.

3.6.4 Η επιλογή LTL Property Manager

Όπως παρατηρούμε και από το run menu αλλά και από τις παραμέτρους της επαλήθευσης, το εργαλείο δίνει την δυνατότητα για την επαλήθευση ενός μοντέλου, ικανοποιώντας παράλληλα έναν LTL τύπο, ο οποίος και θα εκφράζει μια συγκεκριμένη ιδιότητα που θέλουμε να ικανοποιείται. Βάσει της συγκεκριμένης επιλογής το XSpin θα εμφανίζει το ακόλουθο παράθυρο :



Εικόνα 3.6.4: LTL property manager

Οι επιλογές που μπορούν να γίνουν μέσω του LTL property manager αναλύονται ως εξής :

- Όπως φαίνεται και από την εικόνα, ο χρήστης μπορεί να κάνει την επιλογή LOAD για να φορτώσει ένα LTL template ή μια LTL formulae η οποία χρησιμοποιήθηκε στο παρελθόν
- Μπορεί να οριστεί μια καινούργια LTL formulae χρησιμοποιώντας μικρά γράμματα και τελεστές, ακολουθώντας πάντα τους κανόνες της LTL λογικής (όπως αναφέρθηκε στο 2ο κεφάλαιο)
- Ο τύπος (formulae) εκφράζει είτε μια θετική (επιθυμητή) είτε μια αρνητική (ανεπιθύμητη) ιδιότητα του μοντέλου. Η θετική ιδιότητα μετατρέπεται σε αρνητική ιδιότητα από έναν μεταφραστή για να σχηματιστεί έτσι ένα never-claim ενώ η αρνητική ιδιότητα παραμένει ίδια προσθέτοντας έναν τελεστή never.

- Με την επιλογή Generate μετατρέπεται ο τύπος στο ισοδύναμο never-claim (το αποτέλεσμα αυτής της μετατροπής εμφανίζεται στο παράθυρο με τίτλο never-claim)
- Δίνεται η δυνατότητα για τον ορισμό macros τα οποία και θα χρησιμοποιούνται κατά την επαλήθευση ,για παράδειγμα #define p (a > b)
- Έχοντας ρυθμίσει όλες τις παραμέτρους για την LTL formulae δίνεται η δυνατότητα για αποθήκευση αυτής (σαν ένα αρχείο *.ltl). Αξίζει να σημειωθεί ότι αν γίνει κάτι τέτοιο ,δεν πρόκειται να χαθούν τυχών macros που έχουν οριστεί αλλά αυτόματα αυτά θα εμφανίζονται κατά την φόρτωση του LTL αρχείου (το XSpin έχει την ικανότητα να θυμάται ποιοι συμβολισμοί ορίστηκαν σε κάθε αρχείο)
- Τέλος, δίνεται και η επιλογή για απευθείας επαλήθευση μέσω του manager. Έτσι, βάσει αυτής θα εμφανιστεί ένα παράθυρο με κάποιες (από τις βασικές) επιλογές επαλήθευσης παρουσιάζοντας τα αποτελέσματα της επαλήθευσης στο παράθυρο Verification result

3.6.5 Η επιλογή 'View Spin Automaton for each Proctype..'

Η επιλογή αυτή δίνει την δυνατότητα στον χρήστη να μπορέσει να δει την δομή των αυτομάτων των μοντέλων που χρησιμοποιεί το Spin κατα την διαδικασία της επαλήθευσης. Κάθε διεργασία (proctype), που έχει οριστεί αρχικά στο μοντέλο με Promela, αναπαρίσταται με ένα μοναδικό αυτόματο. Με την προϋπόθεση της επιλογής αυτή, από το run menu, θα τοποθετήσει το Spin να παράγει έναν μηχανισμό επαλήθευσης (verifier), να τον μεταφράσει και στη συνέχεια να τον εκτελέσει (όπως και στην ιδιότητα pan –d). Έτσι το Spin θα μπορέσει να πάρει τα ονόματα και την περιγραφή της κάθε διεργασίας και να τα τοποθετήσει στα ανταποκρινόμενα αυτόματα.

Έτσι, φτάνουμε στη περίπτωση όπου το εργαλείο μας εμφανίζει ένα παράθυρο με τις διεργασίες που έχουν αναπαρασταθεί σε αυτόματα. Επιλέγοντας (με διπλό click) το proctype που επιθυμούμε, θα παραχθεί ο συγκεκριμένος γράφος. Ο γραφικός αυτός αλγόριθμος που χρησιμοποιείται εδώ, αξίζει να σημειωθεί ότι αποτελεί αυτοτελές κομμάτι του Xspin. Στον γράφο που παράγεται ,υπάρχει η δυνατότητα για συγκεκριμένες εργασίες που βοηθούν στην καλύτερη κατανόηση των διεργασιών του μοντέλου. Αυτές είναι :

- Μετακίνηση των κόμβων (moving nodes)
- Περιγραφή των κόμβων (displaying nodes labels)
- Cross-References (Μετακινώντας τον κέρσορα πάνω σε έναν κόμβο στο κύριο παράθυρο του Xspin, βλέπουμε ότι μας δείχνεται το κομμάτι του κώδικα που ανταποκρίνεται στον συγκεκριμένο κόμβο.

3.7 Τεχνικές στο XSpin για την μείωση της πολυπλοκότητας

Όταν η διαδικασία της επαλήθευσης δεν μπορεί να ολοκληρωθεί εξαιτίας προβλημάτων (υπολογιστικής) πολυπλοκότητας, υπάρχουν κάποιες στρατηγικές οι

οποίες μπορούν να εφαρμοστούν προς διευκόλυνση των υπολογισμών αλλά και του ίδιου του χρήστη. Οι τεχνικές αυτές είναι οι εξής :

- Επιλογή από το Run Menu του 'Run the Slicing Algorithm..' έτσι ώστε το XSpin βρει στο μοντέλο επιπλέον πλεονασμό όσον αφορά τις καταστάσεις του
- Να γίνει προσπάθεια διόρθωσης του μοντέλου και μετατροπή του σε ένα μοντέλο γενικότερο και πιο περιληπτικό.
- Να απορρίπτεται από το μοντέλο ότι θεωρείται ότι δεν έχει άμεση σχέση με την ιδιότητα η οποία πρόκειται να επαληθευθεί
- Να χρησιμοποιούνται συγχρονισμένα κανάλια (επικοινωνίας) μεταξύ των διεργασιών όπου είναι δυνατό. Τα ασύγχρονα κανάλια είναι σημαντικές 'πηγές' πολυπλοκότητας κατά την επαλήθευση
- Να γίνει έλεγχος διεργασιών οι οποίες μεταφέρουν μηνύματα από ένα κανάλι σε μια άλλη διεργασία και διαγράφονται
- Να συνδυάζονται οι τοπικοί υπολογισμοί σε atomic ή d_step ακολουθίες
- Αν είναι δυνατόν να γίνει :να συνδυάζεται η συμπεριφορά δύο διεργασιών σε μία (με πιο γενική συμπεριφορά)
- Να γίνει προσπάθεια εξοικείωσης με τις τεχνικές της μερικής μείωσης. Έτσι αν είναι δυνατόν να γίνεται χρήση των xr και xs assertions. Επίσης θα ήταν χρήσιμο να αποφεύγεται ο διαμοιρασμός των καναλιών ανάμεσα σε πολλαπλούς πομπούς και δέκτες μηνυμάτων (διεργασίες)

3.8 Ντιρεκτίβες που χρησιμοποιούνται από τον μεταφραστή για την παραγωγή του επαληθευτή

Είναι αρκετά σημαντικό σε αυτό το σημείο να γίνει αναφορά στο περιβάλλον στο οποίο μπορεί το εργαλείο Spin αλλά και η γραφική του μορφή XSpin να εκτελεστεί (κάτι το οποίο είναι και εμφανές από την εγκατάσταση του εργαλείου ,Παράρτημα __). Επειδή όπως αναφέρθηκε κατά την επαλήθευση ενός μοντέλου δημιουργείται ένας μηχανισμός επαλήθευσης (pan.exe), το εκτελέσιμο αρχείο αυτό προέρχεται από την μεταγλώττιση ενός προγράμματος σε γλώσσα C. Το πρόγραμμα αυτό pan.c μέσω του μεταγλωττιστή (που έχει τοποθετηθεί στο σύστημα C:\DJGPP\bin) gcc.exe, μετατρέπεται στο εκτελέσιμο pan.exe το οποίο και επαληθεύει το μοντέλο επιστρέφοντας τα αποτελέσματα της διαδικασίας αυτής.

Κάτα την διάρκεια όμως της μεταγλώττισης, ο χρήστης μπορεί να ορίσει τις παραμέτρους εκείνες οι οποίες και θα έχουν σχέση με το είδος της επαλήθευσης που ο ίδιος επιθυμεί να γίνει πάνω στο μοντέλο του, αλλά και να ρυθμίσει θέματα που έχουν να κάνουν με την ταχύτητα, την απαίτηση μνήμης και άλλων χαρακτηριστικών που αφορούν το σύστημα. Έτσι υπάρχει η δυνατότητα ορισμού ντιρεκτίβων (directives) οι οποίες ποικίλουν ανάλογα με τον σκοπό της επαλήθευσης :

- Directives που υποστηρίζονται από το XSpin
- Directives που σχετίζονται με την μέθοδο Partial Order Reduction
- Directives για την αύξηση της ταχύτητας
- Directives για την μείωση χρήσης της μνήμης

- Directives που χρησιμοποιούνται όταν απαιτείται από το Pan
- Directives για το Debugging των επαληθευτικών μηχανισμών του Pan
- Directives για πειραματική χρήση

Παρακάτω γίνεται η ανάλυση όλων των ντιρεκτίβων των παραπάνω κατηγοριών μαζί με τις εντολές αυτών.

Εντολές	Directives που υποστηρίζονται από το XSpin
BITSTATE	Χρησιμοποιεί supertrace/bitstate αντί για εξαντλητική εξερεύνηση
MEMCNT=N	Θέτει ένα άνω όριο στο μέγεθος της μνήμης που θα χρησιμοποιηθεί ,π.χ. -DMEMLIM=20 για το πολύ 2^20 bytes
MEMLIM=N	Θέτει ένα κάτω όριο για τον πραγματικό αριθμό των Megabytes που μπορούν να χρησιμοποιηθούν, π.χ. -DMEMLIM=200 για το πολύ 200 Megabytes (εναλλακτική της εντολής MEMCNT)
NOCLAIM	Αποκλείει το never-claim από την επαλήθευση ,εάν αυτό υπάρχει
NOFAIR	Απενεργοποιεί τον κώδικα για weak-fairness (πιο γρήγορο)
NOREDUCE	Απενεργοποιεί τον αλγόριθμο partial order reduction
NP	Ενεργοποιεί την ανίχνευση κύκλων τύπου non-progress (ιδιότητα –l), αντικαθιστώντας την ιδιότητα –a για ανίχνευση κύκλων τύπου acceptance
PEG	Προσθέτει complexity profiling (μετρά μεταβάσεις)
SAFETY	Χρησιμοποιείται στην περίπτωση όπου δεν χρειάζεται ανίχνευση κύκλων (πιο γρήγορη επαλήθευση, χρησιμοποιεί λιγότερη μνήμη, απενεργοποιεί μαζί τις ιδιότητες –l, -a)
VAR_RANGES	Υπολογίζει την κλίμακα τιμών των μεταβλητών (από 0 ως 255)
CHECK	Παράγει πληροφορίες όσον αφορά το debugging (βλέπε Debug)

Πίνακας 3.8.1: Directives που υποστηρίζονται από το XSpin

Εντολές	Directives που σχετίζονται με την μέθοδο Partial Order Reduction
CTL	Επιτρέπει εκείνες τις μειώσεις που είναι συνεπής με την branching time λογική CTL (π.χ. το persistent set περιέχει είτε μία είτε όλες τις μεταβάσεις)
GLOB_ALPHA	Θεωρεί την διεργασία death σαν global ενέργεια (χρήσιμη για συμβατότητα με τις εκδόσεις Spin 2.8.5 και 2.9.7) consider process death a global action (for compatibility with versions of Spin between 2.8.5 and 2.9.7)
NIBIS	Εφαρμόζει μια βελτιστοποίηση της μερικής μείωσης (κάποιες φορές γρήγορη κάποιες όχι)
NOREDUCE	Απενεργοποιεί τον αλγόριθμο της μερικής μείωσης
XUSAFE	Απενεργοποιεί τον έλεγχο ορθότητας των x[rs] assertions (πιο γρήγορη και κάποιες φορές χρήσιμη εάν ο έλεγχος είναι αυστηρός π.χ. όταν τα κανάλια παρίστανται σαν παράμετροι διεργασιών)

Πίνακας 3.8.2: Directives που σχετίζονται με την μέθοδο Partial Order Reduction

Εντολές	Directives για την αύξηση της ταχύτητας
NOBOUNDCHECK	Δεν ελέγχει παραβιάσεις συνδεδεμένων πινάκων (πιο γρήγορη)
NOCOMP	Δεν συμπιέζει καταστάσεις με αποθήκευση fullstate (πιο γρήγορη αλλά δεν είναι συμβατή με το φαινόμενο liveness εκτός εάν υπάρχει η -DBITSTATE)
NOFAIR	Απενεργοποιεί τον κώδικα για weak-fairness (πιο γρήγορο)
NOSTUTTER	Απενεργοποιεί κανόνες stuttering (προειδοποίηση :αλλάζει την σημασιολογία). Οι κανόνες stuttering αποτελούν τον βασικό τρόπο για την επέκταση μιας πεπερασμένης συχνότητας εκτέλεσης σε μια μη-πεπερασμένη ,επιτρέποντας την συνεπή μετάφραση των acceptance κανόνων Buchi
SAFETY	Χρησιμοποιείται στην περίπτωση όπου δεν χρειάζεται ανίχνευση κύκλων (πιο γρήγορη επαλήθευση, χρησιμοποιεί λιγότερη μνήμη, απενεργοποιεί μαζί τις ιδιότητες –l, -a)

Πίνακας 3.8.3: Directives για την αύξηση της ταχύτητας

Εντολές	Directives για την μείωση χρήσης της μνήμης
BITSTATE	Χρησιμοποιεί supertrace/bitstate αντί για εξαντλητική εξερεύνηση
HC	Μέθοδος συμπίεσης state-vector. Μειώνει το μέγεθος του state-vector στα 32+16 bits και τα αποθηκεύει σε έναν συμβατικό πίνακα κατακερματισμού (Συμβολισμοί : HC0, HC1, HC2, HC3 για 32, 40, 48, και 56. HC2 : προεπιλογή
COLLAPSE	Μέθοδος συμπίεσης state-vector.Μειώνει το μέγεθος του state-vector μέχρι 90% (-DSEPQS ή –DJOINPROCS)
MA=N	Χρησιμοποιεί μια DFA κωδικοποίηση για τον χώρο καταστάσεων, που είναι της BDD, έχοντας σαν δεδομένο έναν μέγιστο αριθμό N bytes του state-vector (μπορεί να συνδυαστεί με την εντολή –DCOLLAPSE για μεγαλύτερη επίδραση σε περιπτώσεις που ο state-vector είναι μεγάλος)
MEMCNT=N	Θέτει ένα άνω όριο στο μέγεθος της μνήμης που θα χρησιμοποιηθεί ,π.χ. -DMEMLIM=20 για το πολύ 2^20 bytes
MEMLIM=N	Θέτει ένα κάτω όριο για τον πραγματικό αριθμό των Megabytes που μπορούν να χρησιμοποιηθούν, π.χ. -DMEMLIM=200 για το πολύ 200 Megabytes (εναλλακτική της εντολής MEMCNT)
SC	Ενεργοποιεί το σωρό cycling. Αυτό θα αφαιρέσει κομμάτια μιας μεγάλης αναζήτησης σωρού σε αρχείο κατά την διάρκεια της επαλήθευσης. Η σημαία –m που θέτει το μέγεθος του σωρού παραμένει ,μόνο που τώρα θέτει το μέγεθος εκείνου του σωρού που βρίσκεται στον πυρήνα. Χρησιμοποιείται για σπάνιες εφαρμογές όπου ο σωρός αναζήτησης έχει ‘βάθος’ αρκετών εκατομμυρίων καταστάσεων και καταναλώνοντας την μεγαλύτερη μνήμη

Πίνακας 3.8.4: Directives για την μείωση χρήσης της μνήμης

Εντολές	Directives που χρησιμοποιούνται όταν απαιτείται από το Pan
NFAIR=N	Διαθέτει μνήμη για χρησιμοποίηση του φαινομένου weak-fairness π.χ. -DNFAIR=3 (Προεπιλογή είναι 2)
VECTORSZ=N	Διαθέτει μνήμη (σε Bytes) για χρησιμοποίηση του state-vector π.χ. –DVECTORSZ=2048 (Προεπιλογή είναι 1024

Πίνακας 3.8.5: Directives που χρησιμοποιούνται όταν απαιτείται από το Pan

Εντολές	Directives για το Debugging των επαληθευτικών μηχανισμών του Pan
VERBOSE	Προσθέτει λεπτομερείς αναφορές debugging
CHECK	Προσθέτει επιπλέον πληροφορίες για debugging
SVDUMP	Εάν οριστεί, προσθέτει την ιδιότητα –pN στους μηχανισμούς επαλήθευσης ώστε να παραχθεί ένα αρχείο sv_dump στο τέλος της εκτέλεσης έχοντας μια δυαδική αναπαράσταση όλων των καταστάσεων χρησιμοποιώντας N bytes για κάθε κατάσταση (βλέπε SDUMP)
SDUMP	Εάν χρησιμοποιηθεί μαζί με την εντολή CHECK : Προσθέτει ASCII dumps των state-vectors στην έξοδο (π.χ. μια ASCII έκδοση της εντολή SVDUMP)

Πίνακας 3.8.6: Directives για το Debugging των επαληθευτικών μηχανισμών του Pan

Εντολές	Directives για πειραματική χρήση
BCOMP	Όταν βρισκόμαστε σε BITSTATE mode, αυτή η εντολή υπολογίζει τις συναρτήσεις κατακερματισμού πάνω από συμπιεσμένο state-vector (συμπίεση με byte-masking) σε μερικές περιπτώσεις, βελτιώνοντας την επικάλυψη
COVEST	Μεταγλωττίζει σε extra κώδικα δοσμένης επικάλυψης κατά το τέλος της εκτέλεσης της εντολής BITSTATE .Πρέπει να μεταγλωττίσεις το αρχείο pan.c με την ιδιότητα –lm
LC	Χρησιμοποιείται σε συνδυασμό με την εντολή BITSTATE. Ενεργοποιείται αυτόματα όταν χρησιμοποιείται το –DSC στην BITSTATE mode. Η LC ενισχύει την χρησιμοποίηση συμπίεσης hashcompact για stackstates. Κάνει την αναζήτηση πιο αργή ,αλλά μπορεί να σώσει μνήμη. Χρησιμοποιεί 4 bytes για κάθε κατάσταση
NOVSZ	Ριψοκίνδυνη εντολή – αφαιρεί 4 bytes από τον state-vector -. Στις περισσότερες περιπτώσεις αυτό είναι ικανοποιητικό για αυτό όταν μειώνεται η μνήμη να χρησιμοποιείτε. Όταν ο αριθμός των καταστάσεων που αποθηκεύονται μειώνεται όταν η εντολή –DNOVSZ χρησιμοποιείται (ο έλεγχος για safety είναι σωστός αλλά ο έλεγχος για liveness μπορεί να είναι λάθος). Η NOVSZ δεν μπορεί να συνδυαστεί με την εντολή COLLAPSE
PRINTF	Ενεργοποιεί την εντολή printf κατά την διάρκεια της επαλήθευσης

RANDSTORE	Όταν βρισκόμαστε σε BITSTATE mode,χρησιμοποιούμε για παράδειγμα DRANDSTORE=33 για να μειωθεί η πιθανότητα αποθήκευσης των bits στον πίνακα κατακερματισμού στο 33%. Η τιμή που ορίζεται πρέπει να είναι μεταξύ 0 και 99. Η μικρές τιμές αυξάνουν τον φόρτο εργασίας (πολυπλοκότητα χρόνου) και αυξάνεται η επικαλυψιμότητα για μεγάλους χώρους καταστάσεων
REACH	Εξασφαλίζει απουσία λαθών μέχρι το όριο βάθους –m
W_XPT=N	Σε συνδυασμό με την εντολή MA γράφει checkpoint αρχεία για κάθε N καταστάσεις που αποθηκεύονται
R_XPT	Σε συνδυασμό με την εντολή MS, επανεκινεί την επαλήθευση από το τελευταίο checkpoint αρχείο που έχει γραφτεί. Μπορεί να συνδυαστεί και με την εντολή W_XPT

Πίνακας 3.8.7: Directives για πειραματική χρήση

3.9 Το πρόβλημα των 4ων στρατιωτών

Στις παρακάτω παραγράφους ακολουθεί η πρώτη εφαρμογή του αυτόματου ελεγκτή μοντέλου SPIN πάνω σε ένα επιλεγμένο πρόβλημα; το λεγόμενο πρόβλημα των 4 στρατιωτών. Αφού κατασκευαστεί το αντιπροσωπευτικό μοντέλο, πραγματοποιούμε την προσομοίωση αυτού μέσω του SPIN και στην συνέχεια προχωρούμε στην επαλήθευση του μοντέλου, προσπαθώντας να εκμαιεύσουμε την λύση στο συγκεκριμένο πρόβλημα.

3.9.1 Πρόλογος του προβλήματος

Για την καλύτερη κατανόηση του εργαλείου Spin και του τρόπου με τον οποίο το εργαλείο αυτό πραγματοποιεί προσομοίωση και επαλήθευση του μοντέλου που εξετάζει, θα ήταν σωστό να γίνει η χρήση ενός προβλήματος και η επίλυσή του. Γι αυτό τον λόγο επιλέχθηκε το πρόβλημα των 4 στρατιωτών, ένα πρόβλημα το οποίο αποτελεί ουσιαστικά παράδειγμα *λειτουργίας πρωτοκόλλων επικοινωνίας*. Έτσι στην παράγραφο αυτή παρουσιάζεται αρχικά η ανάλυση του προβλήματος, η κατασκευή του μοντέλου του προβλήματος στην γλώσσα PROMELA αναλύοντας κάθε βήμα, η προσομοίωση του προβλήματος και τα αποτελέσματα αυτής και η επαλήθευση του προβλήματος όμοια με τα αποτελέσματά της. Η χρήση του εργαλείου θα γίνει με το γραφικό περιβάλλον του Spin ,το XSpin για την καλύτερη παρουσίαση της προσομοίωσης και της επαλήθευσης. Τέλος ο αναγνώστης μπορεί να δει στο παράρτημα του εγγράφου τα συνολικά αποτελέσματα τόσο του μοντέλου συνολικά όσο και των αποτελεσμάτων του εργαλείου.

3.9.2 Εκφώνηση του προβλήματος

Τέσσερις στρατιώτες οι οποίοι είναι βαριά τραυματισμένοι θέλουν να φτάσουν στην πατρίδα τους. Ο εχθρός τους κυνηγά μέσα στην νύχτα ,ώσπου αυτοί συναντάνε μία γέφυρα η οποία χωρίζει την πατρίδα τους από τη εχθρική χώρα (στην οποία και βρίσκονται την παρούσα στιγμή). Η γέφυρα είναι κατεστραμμένη και μπορεί 'κρατήσει' μέχρι δύο στρατιώτες πάνω της. Επίσης αρκετές νάρκες έχουν τοποθετηθεί πάνω σ' αυτή γι αυτό και για να διασχίσουν την γέφυρα οι στρατιώτες πρέπει να κρατάνε μαζί τους ένα φακό. Ο εχθρός βρίσκεται ακριβώς από πίσω τους ,γι αυτό και οι στρατιώτες έχουν στην διάθεσή τους ένα λεπτό (60 min) για να διασχίσουν την γέφυρα. Οι στρατιώτες έχουν μόνο έναν φακό (όλοι μαζί) και δεν είναι τραυματισμένοι το ίδιο βαριά ο καθένας. Έτσι κάθε στρατιώτης χρειάζεται έναν ξεχωριστό χρόνο για να διασχίσει την γέφυρα.

Ακολουθεί ο πίνακας με τους χρόνους του κάθε στρατιώτη:

Στρατιώτης	Χρόνος
S0	5 min
S1	10 min
S2	20 min
S3	25 min

Πίνακας 3.9.2: Χρόνοι επιμέρους στρατιωτών (Πρόβλημα 1)

→Υπάρχει κάποια διαδικασία σύμφωνα με την οποία και οι τέσσερις στρατιώτες θα περάσουν την γέφυρα (safe side) μέσα σε εξήντα λεπτά ;

3.9.3 Λύση προβλήματος

Ανάλυση της εργασίας σε επιμέρους βήματα:

- Παρόλο που στον ορισμό του προβλήματος έχουμε τον χρονικό περιορισμό των 60 λεπτών για να περάσουν οι στρατιώτες στην 'ασφαλή πλευρά', κατά την διάρκεια της μοντελοποίησης δεν θα γίνει χρήση ρολογιού ή κάποιον ελεγκτή μοντέλου πραγματικού χρόνου (real-time model checker)
- Κατά συνέπεια πρόκειται για ένα πρόβλημα σχεδιασμού/ προγραμματισμού .Κάτι τέτοιο μας δίνει την ευκαιρία να εκμεταλλευτούμε το γεγονός ότι ένας ελεγκτής μοντέλου ,εξερευνά όλες τις πιθανές περιπτώσεις στο δένδρο που κατασκευάζεται για το πρόβλημα αυτό.
- Έτσι μπορούμε εύκολα να χρησιμοποιήσουμε τον ελεγκτή μοντέλου να 'κατασκευάσει' όλες τις διαφορετικές περιπτώσεις κινήσεων των στρατιωτών από την μια πλευρά στην άλλη
- Αξίζει να σημειωθεί ότι στην επαλήθευση του προβλήματος που ακολουθεί παρακάτω, αυτή θα γίνει τόσο στην περίπτωση των στρατιωτών χωρίς χρονικό περιορισμό (για την κατανόηση των αποτελεσμάτων του Spin) όσο και στην

περίπτωση με τον χρονικό περιορισμό (προφανώς για την λύση του προβλήματος με το παράδειγμα που θα παράγει το Spin).

3.9.4 Βασικές έννοιες μοντελοποίησης του προβλήματος των στρατιωτών

⇒ Πρέπει να υπάρχει επικοινωνία μεταξύ των δύο πλευρών : Της μη-ασφαλής πλευράς (unsafe side) και της ασφαλής πλευράς (safe side). Έτσι δημιουργούμε δύο διεργασίες (proctypes),

proctype Unsafe() και *proctype Safe()*

⇒ Και στις δύο πλευρές θα πρέπει να έχουμε από 4 'θέσεις' (μνήμης) τις οποίες και θα καταλαμβάνουν οι στρατιώτες

⇒ Στην διάρκεια του προβλήματος κατά την μετάβαση προς την Safe πλευρά , 2 στρατιώτες θα πρέπει πάντα να πηγαίνουν προς αυτή (ο ένας για να κρατάει τον φακό και ο άλλος (!!!) για την ..σωτηρία του). Ο χρόνος που θα κάνουν για να περάσουν οι στρατιώτες απέναντι θα είναι ίσος με τον χρόνο που θα έκανε ο πιο αργός στρατιώτης (ή αλλιώς ο πιο βαριά τραυματισμένος). Μόλις ο 2ος στρατιώτης μεταβεί στην Safe πλευρά τότε ο 1ος θα πρέπει να γυρίσει πίσω για να 'σώσει' τους άλλους, κοκ. Έτσι θα έχουμε δύο συναρτήσεις για το κανάλι μετάβασης από την unsafe στη safe πλευρά (που θα παίρνει σαν ορίσματα 2 στρατιώτες) και από την safe στην unsafe (που θα παίρνει σαν όρισμα 1 στρατιώτη). Αυτές θα είναι οι:

```
Chan unsafe_to_safe=[0] of {soldier,soldier};
```
και
```
Chan safe_to_unsafe=[0] of {soldier};
```

Αξίζει να σημειωθεί ότι δεν υπάρχει ανάγκη για χρήση μνήμης (buffer) αφού ουσιαστικά η κατάσταση των στρατιωτών θα είναι είτε ασφαλής ή μη ασφαλής. Η γέφυρα μοντελοποιείται από δύο κανάλια. Συμπερασματικά: το κανάλι unsafe_to_safe μοντελοποιεί την πλευρά unsafe και το κανάλι safe_to_unsafe μοντελοποιεί την πλευρά safe. Κάθε φορά που δύο από τους στρατιώτες πρέπει να περάσουν στην safe πλευρά ,ο ένας από αυτούς πρέπει να γυρίσει πίσω στην unsafe πλευρά με τον φακό. Είναι φανερό ότι δεν μπορούν να γυρίσουν και οι δύο οι στρατιώτες. Για τον λόγο αυτό το κανάλι safe_to_unsafe περνά έναν μόνο στρατιώτη προς την unsafe πλευρά.

3.9.5 Δημιουργία του αρχείου soldiers.prom βήμα προς βήμα

Οι 4 στρατιώτες συμβολίζονται με ακέραιους από το 0..N-1, όπου το N ορίζεται ως εξής:

```
#define N          4
```

κάτι που σημαίνει ότι κάθε ένας στρατιώτης αντιπροσωπεύεται από 1 Byte ,δηλαδή

```
#define soldier          byte
```

Το πέρασμα από την γέφυρα μοντελοποιείται από στρατιώτες που περνούν από δύο διεργασίες (processes): Την Unsafe και Safe. Στην αρχή όλοι οι στρατιώτες βρίσκονται στην Unsafe πλευρά και ο στόχος είναι οι στρατιώτες να περάσουν στην Safe πλευρά. Γίνεται χρήση μιας διεργασίας ,της Timer που μοντελοποιεί το πέρας του χρόνου. Αυτή η διεργασία είναι υπεύθυνη για την 'ανανέωση' της global μεταβλητής time ,που δείχνει τα λεπτά τα οποία έχουν περάσει από τότε που ξεκίνησαν οι στρατιώτες να κινούνται. Έτσι η global μεταβλητή time ορίζεται ως:

```
Byte time;
```

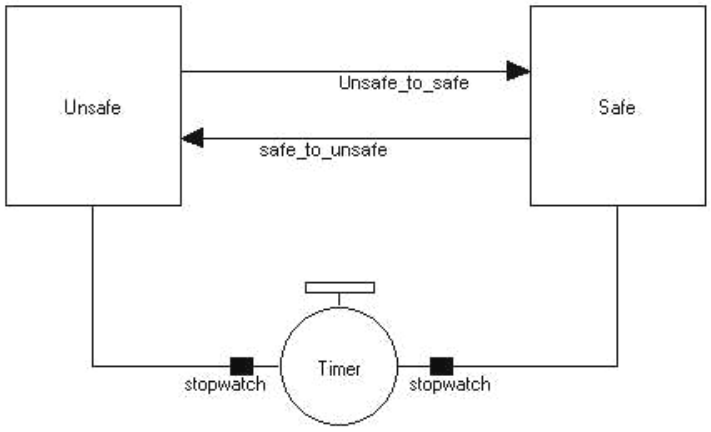
Κάθε στιγμή που ένας στρατιώτης περνά την γέφυρα ,οι διεργασίες Unsafe και Safe 'ειδοποιούν' την διεργασία Timer στέλνοντας την ταυτότητα (id) του στρατιώτη (έναν αριθμό δηλαδή ανάμεσα στο 0 ως 3) που κάνει την περισσότερη ώρα για να περάσει την γέφυρα (του πιο αργού). Γι αυτόν τον λόγο εισέρχεται στο πρόγραμμα η έννοια του καναλιού stopwatch, και ορίζεται ως εξής:

```
Chan stopwatch      =    [0]     of    {soldier} ;
```

Αφού ληφθεί το id του στρατιώτη ,η διεργασία Timer θα ανανεώσει την μεταβλητή time. Ο κώδικας της διεργασίας Timer είναι ο παρακάτω :

```
active proctype Timer()
{
end:
do
:: stopwatch ? 0 -> atomic { time=time+5 ; MSCTIME }
:: stopwatch ? 1 -> atomic { time=time+10 ; MSCTIME }
:: stopwatch ? 2 -> atomic { time=time+20 ; MSCTIME }
:: stopwatch ? 3 -> atomic { time=time+25 ; MSCTIME }
od
}
```

Παρατηρούμε από τον κώδικα ότι η διεργασία Timer ανανεώνει την μεταβλητή time κάθε φορά που ανάλογα με το εάν κάποιος στρατιώτης 'παρουσιάσει stopwatch' (εικόνα 3.9.5). Η ετικέτα end μπροστά από το do-loop είναι αναγκαία για την περίπτωση που όλοι οι στρατιώτες περάσουν στην Safe πλευρά. Στη συνέχεια (του κώδικα) η Timer θα περιμένει μέχρι την στιγμή που ο επόμενος στρατιώτης θα παρουσιάσει stopwatch.



Εικόνα 3.9.5: Unsafe to safe diagram

Το macro MSCTIME χρησιμοποιείται για την έξοδο του προγράμματος αφού αυτό είναι που γράφει τον χρόνο κάθε στιγμής στο 'Message Sequence Chart' κατά την προσομοίωση με το SPIN ή το XSPIN. Ορίζεται ως εξής:

```
#define MSCTIME          printf("MSC: %d\n", time)
```

3.9.5.1 *Οι δύο πλευρές σαν διεργασίες*

Μοντελοποιούμε την τοποθεσία των στρατιωτών χρησιμοποιώντας τον πίνακα here τύπου bit. Εάν το here[i] έχει την τιμή 1, τότε ο στρατιώτης i βρίσκεται στην Unsafe πλευρά της γέφυρας. Ο ορισμός στην promela είναι :

```
Bit  here[N];
```

Αρχικά όλοι οι στρατιώτες βρίσκονται στην Unsafe πλευρά οπότε η διεργασία Unsafe θα ξεκινά με την αρχικοποίηση του πίνακα των στρατιωτών ως εξής:

```
here[0]=1 ; here[1]=1 ; here[2]=1 ; here[3]=1 ;
```

Το υπόλοιπο μέρος της διεργασίας είναι υπεύθυνο για το πέρασμα της γέφυρας (όλη η επανάληψη στο do loop). Σε κάθε επανάληψη δύο στρατιώτες στέλνονται από την unsafe στη safe πλευρά και ένας στρατιώτης γυρνάει στην unsafe πλευρά για να γυρίσει τον φακό. Οι δύο πρώτοι στρατιώτες που επιλέγονται από την unsafe πλευρά ,επιλέγονται με έναν μη ντετερμινιστικό τρόπο .Γι αυτόν τον λόγο δημιουργούμε το macro select_soldier(x) :

```
#define select_soldier(x)     \
if                            \
::  here[0] → x=0             \
```

```
::  here[1] → x=1            \
::  here[2] → x=2            \
::  here[3] → x=3            \
fi  ;
here[x] = 0;
```

Θα εκτελείτε μόνο εκείνος ο στρατιώτης που η τιμή του here[i] θα είναι 1(όπου στο x ανατίθεται ο κάθε αριθμός του στρατιώτη) . Εάν δεν υπάρχουν στρατιώτες στην unsafe πλευρά ,το do-loop της διεργασίας αυτής θα τερματιστεί. Η εντολή break χρειάζεται εδώ ,γιατί στην περίπτωση που όλοι οι στρατιώτες θα έχουν φύγει ,κανένας από αυτούς δεν θα χρειαστεί να γυρίσει πίσω(έτσι εξηγείται και ο τερματισμός).

Στη συνέχεια ορίζονται κάποια χρήσιμα macros τα οποία είναι απαραίτητα για την λειτουργία του κώδικα :

```
#define IF          if  ::
#define FI          ::  else fi
#define all_gone    (!here[0]   &&   !here[1]   &&   !here[2]   &&
!here[3])
```

Ο στρατιώτης s1 είναι ο στρατιώτης ο οποίος γυρίζει πίσω στην unsafe πλευρά με τον φακό. Το μήνυμα s1 στέλνεται πάνω από το κανάλι stopwatch για να δώσει την ένδειξη ότι ο s1 όντως έχει 'πατήσει το stopwatch' .

Από την άλλη μεριά παρόμοιος είναι και ο κώδικας στην περίπτωση της διεργασίας της safe πλευράς, με μόνη διαφοροποίηση τα macros all_here και max που ορίζονται ως εξής:

```
#define  max(x,y)   ((x>y)  →  x : y )
#define all_here         (here[0]  &&  here[1]  &&  here[2]  &&
here[3])
```

Βάση της παραπάνω ανάλυσης του μοντέλου αξίζει να ειπωθεί ότι το μοντέλο συντάχθηκε βάση της λειτουργίας που επιβάλλει το πρόβλημα να έχει και όχι ανάλογα με τους κανόνες που θέτει η γλώσσα της Promela.

3.9.6 Βήματα απαραίτητα για την λύση του προβλήματος

⇒ Προσομοίωση του προβλήματος με το XSPIN : Με την βοήθεια αυτής θα μπορέσουμε να δούμε πως θα εκτελούνταν επακριβώς η εκτέλεση του μοντέλου, με ποια σειρά θα εκτελούνταν οι εντολές που έχουμε ορίσει, πότε εκτελούνται και ποια είναι η συχνότητα εμφάνισης των μεταβλητών σε σχέση και με ολόκληρο το μοντέλο αλλά και σε σχέση με τον χρόνο. Κάτι τέτοιο θα έδινε στον χρήστη την ικανότητα να κατανοήσει και να εμπεδώσει την λειτουργία του μοντέλου παρέχοντας την πληροφορία απαριθμητά αλλά και με γραφικό τρόπο.

⇒ Εξαντλητική επαλήθευση με το XSPIN : Πώς μπορούμε να βάλουμε το SPIN να ψάξει και να μας παρουσιάσει ένα 'πρόγραμμα' ,(μία λύση) , σύμφωνα με το

οποίο επιτρέπει στους στρατιώτες να μεταβούν στην safe πλευρά σε λιγότερο ή ίσο από 60 λεπτά. Αρχικά θα γίνει η επαλήθευση του προβλήματος ελέγχοντας την λειτουργία του μοντέλου χωρίς να έχουμε τον χρονικό περιορισμό. Κάτι τέτοιο γίνεται για να επιβεβαιωθεί ο χρήστης για την ορθότητα του μοντέλου όσον αφορά την λειτουργία του ,δηλαδή να μην έχουμε σημεία νεκρού κώδικα (που δεν εκτελείτε ποτέ) αλλά και να μην υπάρχουν λανθασμένα σημεία τερματισμού

⇒ Το SPIN είτε θα κάνει την επαλήθευση εμφανίζοντας στην έξοδο μήνυμα ότι στο μοντέλο δεν υπάρχει λάθος είτε θα εμφανίσει μήνυμα λάθους δίνοντας όμως στον χρήστη την ικανότητα να προσπεράσει το λάθος με ένα αριθμητό παράδειγμα (μετά από καθοδηγημένη προσομοίωση)

⇒ Το SPIN μπορεί να μας δώσει δύο ειδών αποτελέσματα λάθους :

Αντίθετο παράδειγμα (counter-example)
και
Παραβίαση ιδιότητας (property-violation)

⇒ Γι αυτό είναι αναγκαίο να ρωτήσουμε το SPIN να προσπαθήσει να αποδείξει ότι δεν υπάρχει διαδικασία που να επαληθεύει το πρόβλημα. Εάν βρει ένα counterexample θα μας παρουσιάσει την διαδικασία που αποτελεί την λύση του προβλήματος (Message Sequence Chart).

⇒ Με χρήση της Linear Temporal Logic μπορούμε να τυποποιήσουμε άμεσα την επιθυμητή ιδιότητα του προβλήματος. Έτσι στην LTL η επιθυμητή ιδιότητα (που στο παρών πρόβλημα είναι τα 60 min) θα γράφεται ως εξής:

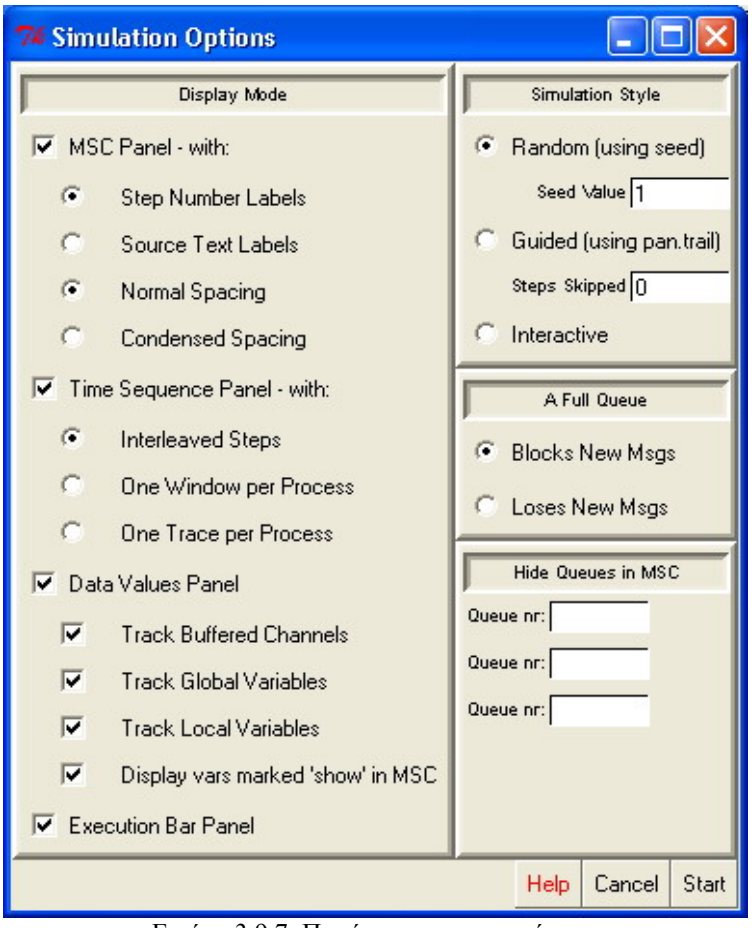
```
<>(time>60)
```

Και αυτό για να μπορέσει το εργαλείο να δώσει την σωστή διαδικασία με την οποία πρέπει να περάσουν οι στρατιώτες στην ασφαλή πλευρά (δηλαδή με ποια σειρά και ποιοι πρέπει να επιλέγονται κάθε φορά).

3.9.7 Προσομοίωση του προβλήματος με το XSpin

Ξεκινώντας τον αυτόματο έλεγχο του προβλήματος με την βοήθεια του εργαλείου Xspin θα ήταν σωστό να αναφερθεί ότι από το εγχειρίδιο χρήσης του Spin συνίσταται πριν από κάθε επαλήθευση που γίνεται σε ένα μοντέλο για την καλύτερη κατανόηση αυτού ,θα πρέπει να πραγματοποιείται και η προσομοίωσή του. Στην περίπτωση της προσομοίωσης ,το εργαλείο μας επιτρέπει να επιλέξουμε εκείνες τις ιδιότητες που μας ενδιαφέρει να μας δοθούν από το μοντέλο και έχουν να κάνουν με τον χρόνο, τις μεταβλητές και το είδος της προσομοίωσης που θα πραγματοποιηθεί (τυχαία, καθοδηγούμενη). Αξίζει να σημειωθεί ότι ο χρήστης πριν την εκκίνηση χρήσης των επιλογών του εργαλείου θα πρέπει να κάνει πάντα έναν έλεγχο για λάθη στο μοντέλο που έχει ορίσει, και στη συνέχεια να προχωρήσει παρακάτω.

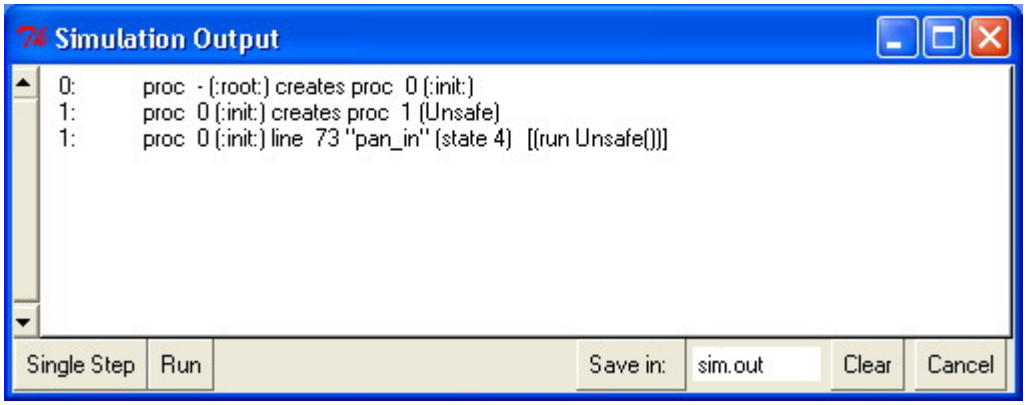
Οι επιλογές με τις οποίες θα προσομοιώσουμε το πρόβλημα των στρατιωτών φαίνονται στην παρακάτω εικόνα :



Εικόνα 3.9.7: Παράμετροι προσομοίωσης

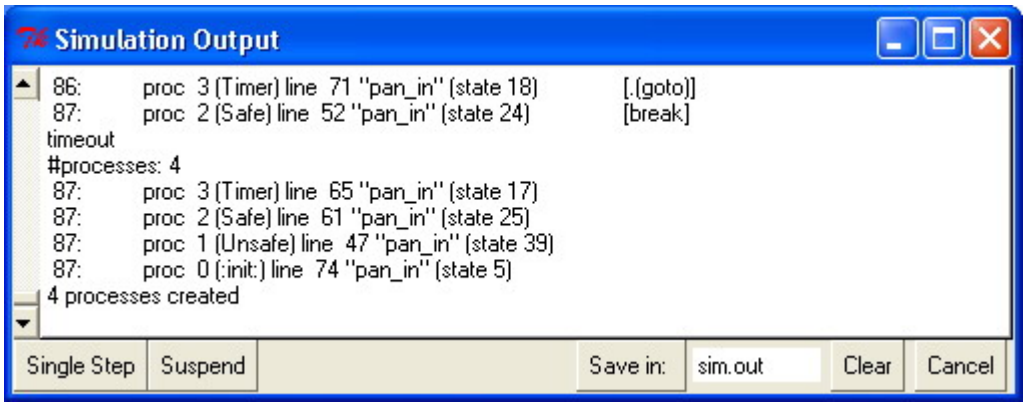
Όπως μπορεί να παρατηρήσει κανείς πραγματοποιούμε *random* προσομοίωση με *seed value* 1. Όσον αφορά τα αποτελέσματα που θέλουμε να μας εμφανίζει η προσομοίωση αυτά φαίνονται στο display mode της παραπάνω εικόνας. Έτσι έχουμε επιλέξει για το για Message Sequence Chart το οποίο παράγει το XSpin να εμφανίζεται με Step Number Labels και Normal Spacing. Θα εξηγήσουμε αναλυτικά την έξοδο της προσομοίωσης για κάθε επιλογή που έχουμε κάνει.

Πατάμε το πλήκτρο start (βρίσκεται στο κάτω δεξιά μέρος του παραθύρου) και το XSpin εμφανίζει τα πέντε παράθυρα των αποτελεσμάτων τα οποία είναι κενά. Το πρώτο από αυτά είναι το Simulation Output (εικόνα 3.9.8).



Εικόνα 3.9.8: Αποτελέσματα Προσομοίωσης (αρχικά)

Βλέπουμε ότι στο βήμα 0 έχει δημιουργηθεί η διεργασία εκκίνησης του μοντέλου init (όπως έχουμε ορίσει στο μοντέλο μας), ενώ στο βήμα 1 δημιουργούνται οι διεργασίες Safe και Unsafe. Το εργαλείο μας αναφέρει επίσης ότι η δημιουργία αυτή των διεργασιών αναφέρεται στο προσωρινό αρχείο C, pan_in στην σειρά 74. Στο κάτω μέρος του παραθύρου δίνονται δύο επιλογές προσομοίωσης. Η μια αναφέρεται στην προσομοίωση βήμα προς βήμα του μοντέλου και η άλλη αναφέρεται στην προσομοίωση του μοντέλου μέχρι αυτή να ολοκληρωθεί. Πατάμε Run για να ξεκινήσει η μοντελοποίηση του προβλήματος.



Εικόνα 3.9.9: Αποτελέσματα Προσομοίωσης (τελικά)

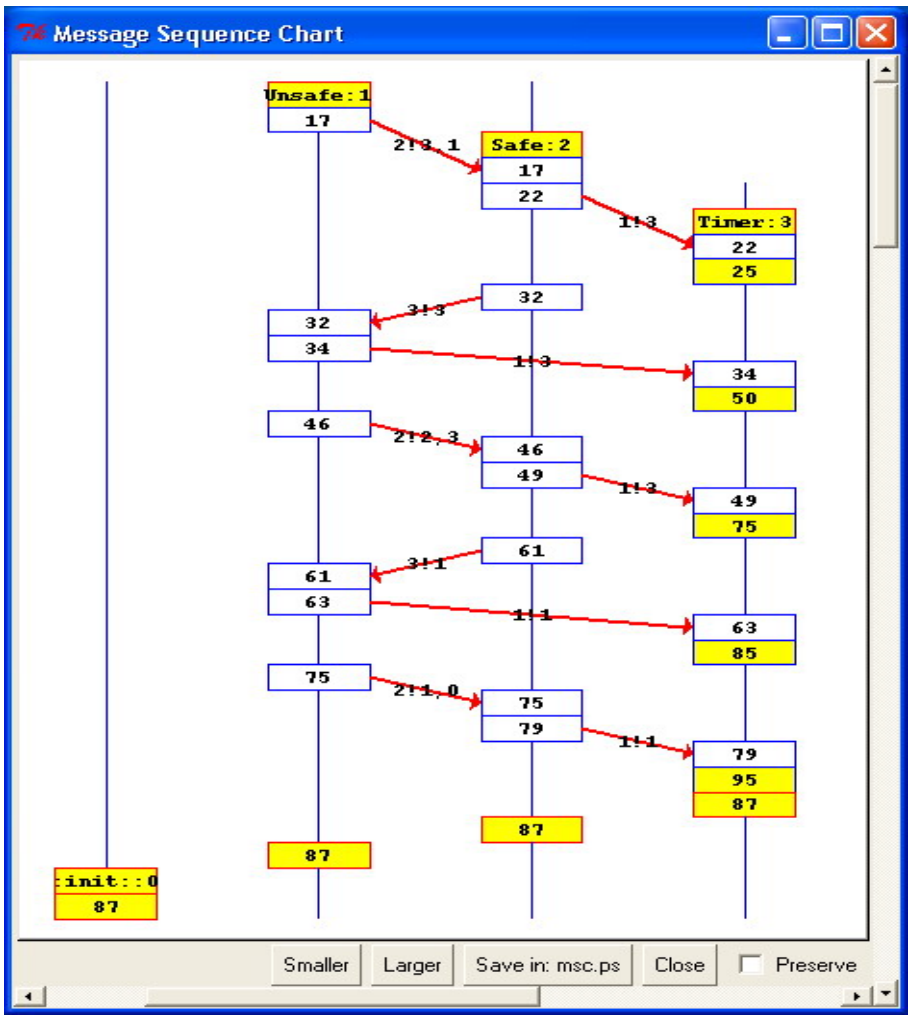
Στο σημείο αυτό το XSpin έχει τελειώσει την προσομοίωση και μπορούμε να ερμηνεύσουμε τα αποτελέσματα. Θα ήταν σωστό να ειπωθεί ότι κατά την έναρξη της προσομοίωσης ο χρήστης μπορεί να παρατηρήσει ότι στον φάκελο που βρίσκεται το αρχείο prom ,δημιουργούνται προσωρινά αρχεία με όνομα pan.* τα οποία και περιέχουν όλες τις απαραίτητες πληροφορίες για την προσομοίωση του μοντέλου. Ανοίγοντας ένα από τα αρχεία αυτά μπορεί κανείς να δει όλες τις πληροφορίες που συλλέγονται και αποθηκεύονται στα αρχεία αυτά τα οποία και χρησιμεύουν στο εργαλείο για την πιο γρήγορη προσομοίωση σε περίπτωση επανάληψης αυτής. Ξεκινώντας από το Simulation Output, παρατηρούμε ότι έχουν δημιουργηθεί τέσσερις διεργασίες (που ορίστηκαν στο μοντέλο αρχικά) οι οποίες είναι οι :

- Init
- Unsafe
- Safe

- Timer

Τα βήματα εκτέλεσης όπως φαίνεται και από την εικόνα είναι 87 και μπορούμε να δούμε σε πιο βήμα δημιουργείται κάθε διεργασία (εικόνα 3.12.8.3). Επίσης παρατηρούμε σε κάθε βήμα και το μέρος του κώδικα της Promela (όλες οι εντολές που αναφέρονται με την λέξη state και τον αριθμό αυτών) που εκτελείται αναφέροντας παράλληλα την κατάσταση στην οποία αποθηκεύεται το βήμα από τον αλγόριθμό που χρησιμοποιούμε στο αρχείο pan_in .Όπως φαίνεται και από την εικόνα 3.12.8.3 μπορούμε να σώσουμε τα αποτελέσματα του Simulation σε ένα αρχείο της αρεσκείας μας (π.χ. sim.out). Το αρχείο sim.out εμφανίζεται στο παράρτημα Γ.

Το δεύτερο παράθυρο το οποίο εμφανίζεται και στο οποίο επιλέξαμε από τις αρχικές επιλογές της προσομοίωσης να εμφανιστεί, είναι το παράθυρο Message Sequence Chart (εικόνα 3.9.10) .

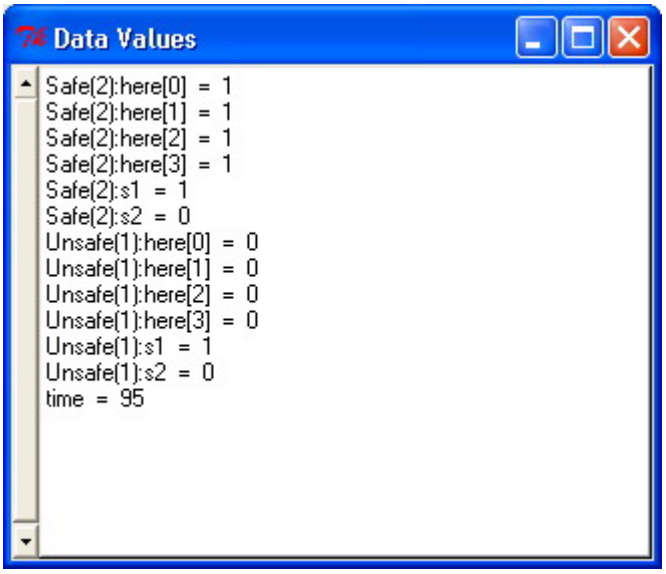


Εικόνα 3.9.10: Διάγραμμα ακολουθιακών μηνυμάτων (MSC)

Πρόκειται για το παράθυρο στο οποίο ουσιαστικά μας παρουσιάζεται η εκτέλεση του μοντέλου με μηνύματα ανάμεσα στις διεργασίες (proctypes) τις οποίες έχουμε ορίσει στο μοντέλο. Για την καλύτερη κατανόηση τόσο το διαγράμματος αλλά και γενικώς της προσομοίωσης του προβλήματος ο αναγνώστης καλείτε να έχει μπροστά του τα αποτελέσματα των παραθύρων του Xspin, Simulation Output και Message

Sequence Chart (εικόνα 3.9.10). Όπως μπορεί να παρατηρήσει κανείς, στο MSC φαίνεται καθαρά πότε εκτελείται κάθε διεργασία σε σχέση με τον αριθμό που της δίνεται κατά σειρά, στην διάρκεια της προσομοίωσης (εκτέλεσης) του μοντέλου. Επίσης μπορεί να παρατηρηθεί ποιος στρατιώτης μεταφέρεται από την μία πλευρά στην άλλη (δηλαδή 'περνάει' από την μία διεργασία στην άλλη), ανανεώνοντας σε κάθε τέτοια μετακίνηση τον χρόνο που έχει περάσει. Κάτα την τυχαία επιλογή των στρατιωτών βλέπουμε ποιοι από αυτούς επιλέγονται και σε ποιους από αυτούς επενεργεί η εντολή stopwatch! που ορίστηκε στο μοντέλο παραπάνω. Στο τέλος της προσομοίωσης φαίνεται καθαρά ότι ο χρόνος που έχει περάσει για την μετακίνηση και των 4ων στρατιωτών από την Unsafe στην Safe πλευρά ισούται με 95, σημειώνοντας παράλληλα ότι η εκτέλεση σταματάει στο βήμα 87 (στο σημείο αυτό τερματίζει η διεργασία init). Είναι σωστό να γίνει η υπενθύμιση ότι εκτελούμαι την προσομοίωση και άρα κατά κάποιο τρόπο την εκτέλεση του μοντέλου, χωρίς να έχουμε πάρει υπ' όψιν τον χρονικό περιορισμό που τίθεται στο ερώτημα του προβλήματος. Κάτι τέτοιο θα δειχθεί στην συνέχεια με τα αποτελέσματα της επαλήθευσης όπως θα φανεί παρακάτω.

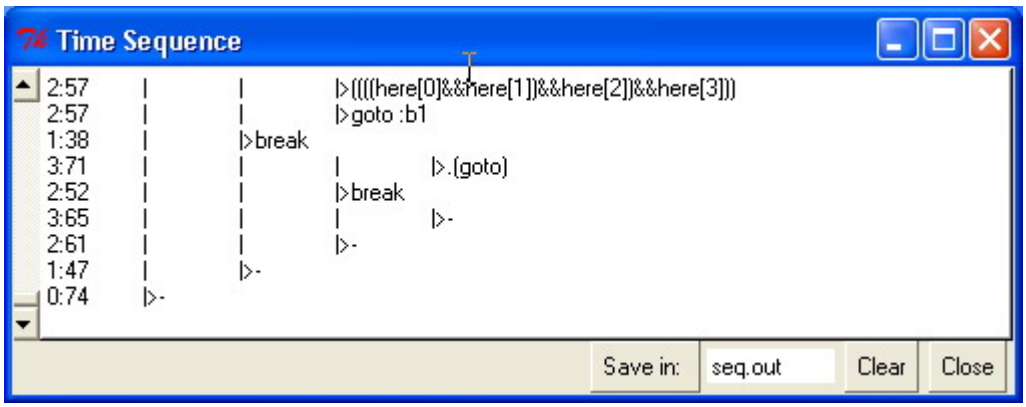
Στην διάρκεια της προσομοίωσης, ένα παράθυρο το οποίο εμφανίζεται είναι και αυτό των τιμών των δεδομένων, Data Values. Στο συγκεκριμένο



Εικόνα 3.9.11: Τιμές Παραμέτρων

παράθυρο εμφανίζονται οι πληροφορίες εκείνες οι οποίες εμείς αρχικά επιλέξαμε να εμφανίζονται από τον καθορισμό των παραμέτρων της προσομοίωσης. Έτσι, όπως φαίνεται και από την εικόνα 3.12.8.5, μπορεί κανείς να παρατηρήσει όλες τις τιμές εκείνες που έχουν πάρει οι τοπικές ή 'παγκόσμιες' μεταβλητές (για παράδειγμα here[0]=1, time=95) αλλά επίσης καταγράφονται και οι μεταβλητές που παρουσιάζονται στο MSC μαζί με τα κανάλια (Buffered Channels). Στην διάρκεια της προσομοίωσης ο χρήστης μπορεί να δει πως αλλάζουν οι τιμές αυτές με το πέρασμα του χρόνου (και της προσομοίωσης), καταλήγοντας στην τελική κατάσταση των μεταβλητών. Έτσι φαίνεται καθαρά από το παράθυρο ότι όλοι οι στρατιώτες έχουν περάσει από την Unsafe πλευρά όπου τους ορίσαμε αρχικά στο μοντέλο, (here[0]=here[1]=here[2] =here[3]=0) στην Safe πλευρά(here[0]=here[1]=here[2]= here[3]=1), με την μεταβλητή time να έχει πάρει την τελική τιμή 95 .

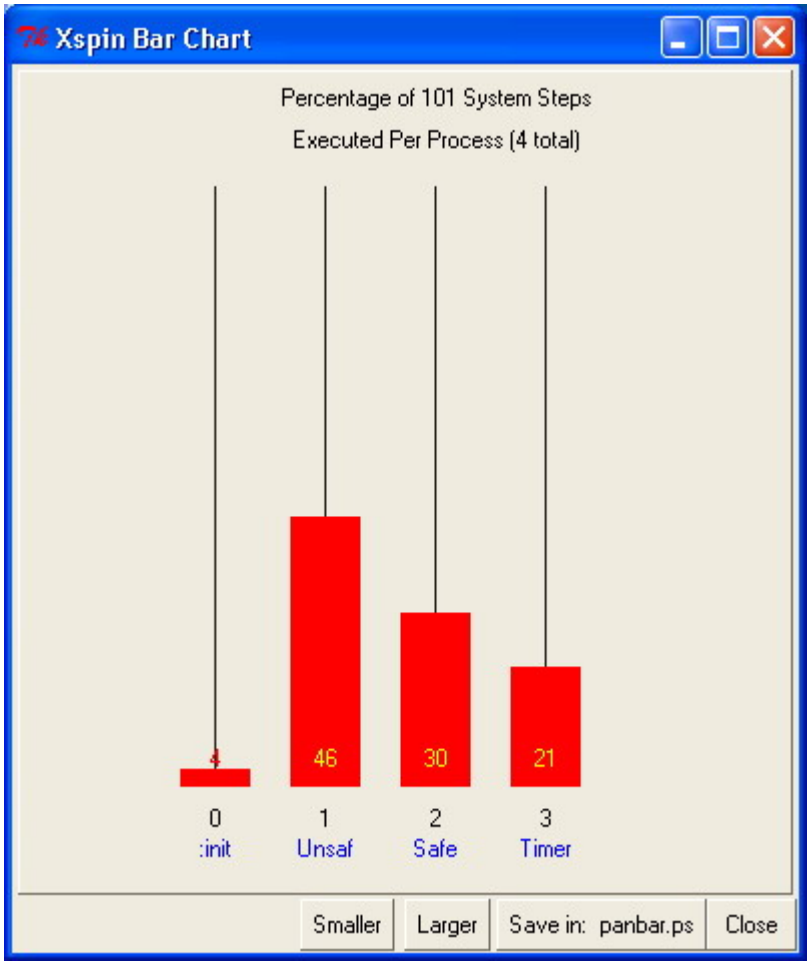
Ένα άλλο αποτέλεσμα που ζητήθηκε από το εργαλείο να μας παράγει είναι και η ακολουθία του χρόνου σε σχέση με την εκτέλεση του μοντέλου (Time Sequence). Έτσι με την προϋπόθεση της επιλογής αυτής θα εμφανιστεί το παράθυρο της εικόνας 3.9.12 ,στο οποίο και παρατηρείται το ακριβές σημείο του κώδικα του μοντέλου που εκτελείται στην συγκεκριμένη χρονική στιγμή, δίνοντας παράλληλα και την ακολουθία εκτέλεσης αυτών. Παράλληλα



Εικόνα 3.9.12: Χρονικό ακολουθιακό Διάγραμμα

'ανοίγοντας' το παράθυρο, ο χρήστης θα έχει την ικανότητα να δει ολόκληρο το εύρος των εντολών που εκτελούνται στο μοντέλο, πότε αυτές ενεργοποιούνται και πότε απενεργοποιούνται. Έτσι δίνεται η ικανότητα στον χρήστη με την σωστή παρακολούθηση τόσο του αποτελέσματος της προσομοίωσης όσο και αυτού της ακολουθίας του χρόνου να γνωρίζουν και με το παραπάνω κάθε τι που συμβαίνει στο σύστημα το οποίο ελέγχεται, μέσω του μοντέλου που ορίστηκε στην γλώσσα της Promela.

Τέλος, ένα από τα αποτελέσματα το οποίο απαιτήσαμε από την προσομοίωση του εργαλείου να μας δώσει είναι και το Xspin Bar Chart εικόνα 3.9.13. Πρόκειται για το παράθυρο εκείνο το οποίο μας παρουσιάζει γραφικά όλες τις διεργασίες που υπάρχουν στο μοντέλο και την χρήση αυτών από τις μεταβλητές του μοντέλου, δίνοντας τον συνολικό αριθμό των βημάτων στα οποία αυτές είναι παρούσες. Έτσι παρατηρούμε ότι σε μεγαλύτερη χρήση βρίσκεται η διεργασία Unsafe, πράγμα που είναι απόλυτα λογικό αφού ένας από τους στρατιώτες που μόλις έχουν μεταφερθεί στην ασφαλή πλευρά, πρέπει να γυρίσει πίσω στην μη ασφαλή για να φέρει πίσω τον φακό έτσι ώστε να περάσουν απέναντι και οι υπόλοιποι στρατιώτες (μην ξεχνάτε ότι είναι νύχτα και στον δρόμο υπάρχουν νάρκες!!). Κάτι τέτοιο μπορεί να αποδειχθεί και ανατρέχοντας στο Simulation Output όπου και ο συνολικός αριθμός παρουσίας (άρα και εκτέλεσης) των διεργασιών που είναι όσο μας δείχνει το συγκεκριμένο παράθυρο, δηλαδή 101 βήματα.



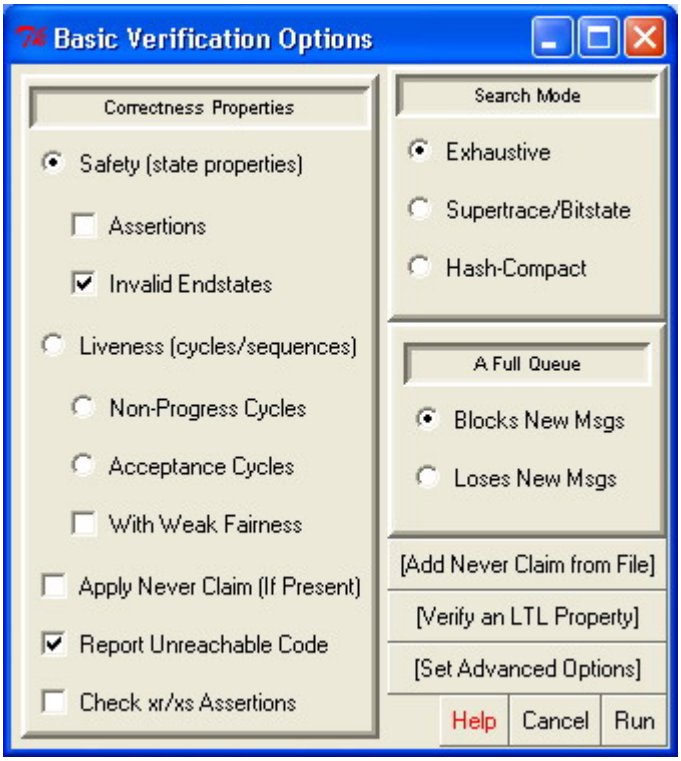
Εικόνα 3.9.13: XSPIN Bar Chart

Με την ολοκλήρωση μιας πρώτης εικόνας της ενέργειας της προσομοίωσης που μας προσφέρει το εργαλείο XSpin, θα ήταν σωστό να αναφερθούν κάποια αποτελέσματα και συμπεράσματα τα οποία αναμφισβήτητα παράγονται από την παραπάνω διαδικασία. Έτσι με την προϋπόθεση της κατανόησης των όσων προαναφέρθηκαν, μπορεί κανείς να δει ότι το μοντέλο κατά την εκτέλεσή του δεν συναντά κανένα πρόβλημα deadlock αφού κάποτε τερματίζει. Επίσης παρατηρώντας τα αποτελέσματα της προσομοίωσης από το Simulation Output αλλά και από το Data Values μπορεί κανείς να δει ολόκληρη την λειτουργία του μοντέλου, την χρήση των διεργασιών από τις μεταβλητές αλλά επίσης και τις τιμές αυτών και κατά την διάρκεια της προσομοίωσης αλλά και στο τέλος αυτής (βλέποντας έτσι ότι όλοι οι στρατιώτες περνάνε εν τέλει απέναντι). Έτσι με μια τυχαία επιλογή των παραμέτρων, όλοι οι στρατιώτες περνάνε από την ασφαλή μεριά μέσα σε 95 λεπτά (υπενθυμίζεται ότι δεν έχουμε πάρει τον χρονικό περιορισμό που μας ζητά το πρόβλημα των 60 λεπτών). Αλλά κάτι τέτοιο θα αποδειχθεί και από το εργαλείο και από την επαλήθευση του μοντέλου , με την επιβεβαίωση της απουσίας λάθους από τα αποτελέσματά της τα οποία εμφανίζονται παρακάτω.

3.9.8 Επαλήθευση του προβλήματος με το XSpin χωρίς τον περιορισμό των 60 λεπτών

Έχοντας τελειώσει την απαραίτητη προσομοίωση του μοντέλου περνάμε στην επαλήθευση αυτού, για την επιβεβαίωση του εργαλείου ότι το μοντέλο και άρα το σύστημα που μοντελοποιήθηκε δουλεύει σωστά. Όσον αφορά το Spin πρέπει να καθορίσουμε :τις ιδιότητες (options) του Spin που είναι υπεύθυνες για την παραγωγή των αρχείων C του pan analyzer τις ντιρεκτίβες για τον C compiler για την δημιουργία του επαληθευτή pan και τις run-time ιδιότητες (options) αυτού. Για κάθε μοντέλο της Promela αποθηκεύουμε αυτές τις συγκεκριμένες ιδιότητες και ντιρεκτίβες του spin σε ένα ειδικό αρχείο δεδομένων options.dat.

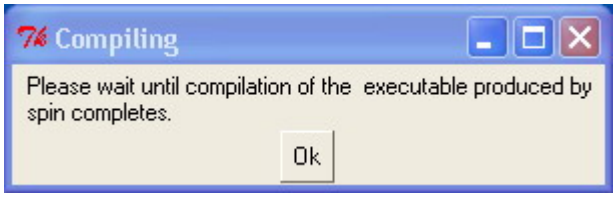
Όπως και με την περίπτωση της προσομοίωσης έτσι και στην επαλήθευση θα πρέπει να επιλεχθούν οι παράμετροι με τους οποίους αυτή θα πραγματοποιηθεί και φυσικά ανάλογα με τις ανάγκες του χρήστη. Περνώντας έτσι στις παραμέτρους της επαλήθευσης όπως φαίνεται και από την εικόνα 3.9.14 γίνονται οι εξής επιλογές :



Εικόνα 3.9.14: Παράμετροι Επαλήθευσης

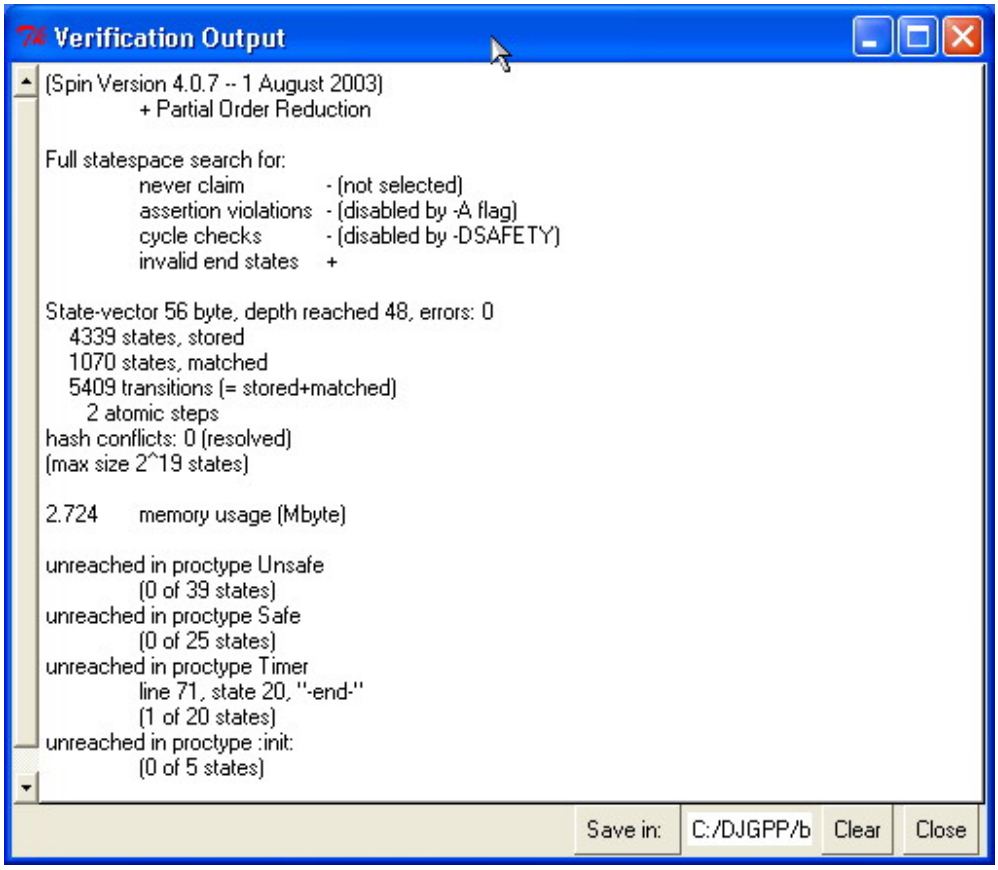
Όπως φαίνεται και στην εικόνα στις παραμέτρους που αφορούν την ορθότητα του μοντέλου (Correctness Properties) ,το ίδιο το μοντέλο θα επαληθευθεί ειδικά για ιδιότητες ασφάλειας. Έτσι επιλέγεται να ελεγχθεί εάν το μοντέλο παρουσιάσει λανθασμένες καταστάσεις τερματισμού. Η επιλογή για βεβαιώσεις (assertions) δεν επιλέγεται αφού δεν έχουν οριστεί τέτοιες προτάσεις μέσα στο μοντέλο όπως δεν επιλέγεται και ο έλεγχος για βεβαιώσεις xr/xs. Κάτι τέτοιο θα μας προειδοποιήσει και το ίδιο το εργαλείο μόλις κάνουμε την συγκεκριμένη επιλογή. Ομοίως δεν επιλέγεται ο έλεγχος για ιδιότητες liveness αφού δεν έχουμε ορίσει acceptance, non-progress cycles ή weak fairness στο μοντέλο. Το ίδιο ισχύει και για την επιλογή εφαρμογής never claim αφού αυτό γίνεται μόνο όταν υπάρχει τέτοιος ισχυρισμός. Στην επιλογή της αναζήτησης Search mode, επιλέγεται ο τύπος αυτής άρα ο αλγόριθμος που θα χρησιμοποιηθεί για να πραγματοποιηθεί. Όπως ειπώθηκε αρχικά για μικρά μοντέλα σωστό είναι να γίνεται αναζήτηση των καταστάσεων με χρήση εξαντλητικού

αλγορίθμου, κάτι το οποίο και επιλέγεται. Οι άλλες δυο περιπτώσεις Supertrace/Bitstate και Hash-Compact πρέπει να επιλέγονται μόνο σε περιπτώσεις που το μοντέλο είναι αρκετά μεγάλο. Τέλος επιλέγεται να μπλοκάρονται όλα τα νέα μηνύματα που δημιουργούνται με την επαλήθευση αποθηκεύοντάς τα στο trail αρχείο που δημιουργείται στο σημείο όπου βρίσκεται το αρχείο prom.



Εικόνα 3.9.15: Παράθυρο μεταγλώττισης

Αρχίζοντας την επαλήθευση του μοντέλου πατώντας το πλήκτρο Run ,το εργαλείο θα αρχίσει την μεταγλώττιση του αρχείου pan.c που έχει παραχθεί στο ίδιο σημείο με το αρχείο prom με σκοπό την δημιουργία του επαληθευτή. Έτσι μετά από την εμφάνιση του παραθύρου της εικόνας 3.9.15 και την παραγωγή του εκτελέσιμου pan.exe (όπως φαίνεται στον ίδιο φάκελο) θα εμφανιστεί το παράθυρο που θα παρουσιάζει τα αποτελέσματα της επαλήθευσης ,Verification Output που φαίνεται στην εικόνα 3.9.16.



Εικόνα 3.9.16: Αποτελέσματα επαλήθευσης

Αναλύοντας τα αποτελέσματα της προσομοίωσης γραμμή προς γραμμή(όπως έγινε ακριβώς και στην παράγραφο 3.6.2), θα έχουμε :

- *(Spin Version 4.0.7 – 1 August 2003)*

δείχνει την έκδοση του SPIN που χρησιμοποιήθηκε για τη μεταγλώττιση

- *+ Partial Order Reduction*

το σύμβολο + σημαίνει ότι χρησιμοποιήθηκε ο βασικός αλγόριθμος partial order reduction. Το σύμβολο – θα σήμαινε μεταγλώττιση με την επιλογή DNOREDUCE (εξηγείται παρακάτω)

- *Full statespace search for:*

δείχνει τον τύπο της αναζήτησης

- *never-claim - (not selected)*

το σύμβολο – σημαίνει ότι δεν υπάρχει never-claim, ούτε χρησιμοποιήθηκε LTL φόρμουλα. Αν υπήρχε θα το δηλώναμε με την ντιρεκτίβα DNOCLAIM όπως θα φανεί και στην περίπτωση όπου θα πάρουμε τον χρονικό περιορισμό των 60 λεπτών

- *assertion violations -(disabled by A-flag)*

το - δείχνει ότι η αναζήτηση δεν ελέγχει για παραβάσεις των επιλεγμένων επιβεβαιώσεων (assertions) και ότι η επιλογή αυτή έχει απενεργοποιηθεί από την εσωτερική εντολή του εργαλείο A-flag

- *acceptance cycles - (disabled by DSAFETY)*

το – σημαίνει ότι δεν έγινε έλεγχος για acceptance ή για non-progress κύκλων.
Κάτι τέτοιο έχει απενεργοποιηθεί με την επιλογή της ντιρεκτίβας DSAFETY όπου με την χρήση της γίνεται η μεταγλώττιση. Για να γίνει αυτό χρειαζόμαστε την επιλογή DNP

- *invalid end states +*

το + δηλώνει ότι έγινε έλεγχος για invalid endstates (για παράδειγμα για αδιέξοδο) όπως επιλέξαμε στις παραμέτρους της επαλήθευσης

- *State-vector 56 byte, depth reached 48, errors: 0*

η περιγραφή του συστήματος καταστάσεων απαίτησε 56 bytes μνήμης(για κάθε κατάσταση). Το μεγαλύτερο μονοπάτι αναζήτησης περιέχει 48 μετατροπές (από την αρχική κατάσταση). Στην διάρκεια της επαλήθευσης δεν βρέθηκαν λάθη

- *4339 states, stored*

ένα σύνολο από 4339 καταστάσεις συστήματος αποθηκεύτηκαν στο χώρο καταστάσεων (κάθε μία αναπαρίσταται αποδοτικά με ένα διάνυσμα 56 bytes)

- *1070 states, matched*

σε 1070 περιπτώσεις η αναζήτηση επέστρεψε σε μια κατάσταση που είχε προηγουμένως επισκεφθεί

- *5409 transitions (= stored+matched)*

ένα σύνολο από 5409 μετατροπές δημιουργήθηκαν κατα τη διάρκεια της επαλήθευσης, το οποίο μπορεί να λειτουργήσει ως στατιστικό στοιχείο

- *2 atomic steps*

δύο μόνο από τις μετατροπές είναι κομμάτι ατομικής ακολουθίας. Όλες οι υπόλοιπες δεν είναι.

- *hash conflicts: 0 (resolved)*

δεν υπήρχε καμία περίπτωση όπου το προεπιλεγμένο σχήμα κατακερματισμού οδήγησε σε σύγκρουση και τοποθέτησε τις καταστάσεις σε μια συνδεδεμένη λίστα στον πίνακα κατακερματισμού.

- *(max size 2^19 states)*

το μέγεθος του πίνακα κατακερματισμού είναι 2^19 και αντιστοιχεί στην επιλογή w19 (η τιμή αυτή είναι δυνατόν να αλλαχθεί από την αλλαγή του εργαλείου [Set Advanced Options]→[Estimated State Space Size (states x 10^3]

- *2.724 memory usage (Mbyte)*

το συνολικό ποσό της μνήμης που χρησιμοποιήθηκε είναι 2.724 Mb, συμπεριλαμβάνοντας τη στοίβα, τον πίνακα κατακερματισμού και όλες τις υπόλοιπες δομές δεδομένων.

- *unreached in proctype Unsafe (0 of 39 states)*
- *unreached in proctype Safe (0 of 25)*
- *unreached in proctype Timer*
  line 71, state 20, "-end-" (0 of 20 states)
- *unreached in proctype Init (0 of 5 states)*

δηλώνουν τους ακριβείς αριθμούς γραμμών για τις βασικές δηλώσεις που δεν έφτασε η αναζήτηση. Αφού χρησιμοποιούμε full statespace search, τότε οι γραμμές αυτές δηλώνουν *νεκρό κώδικα.*

Οι ντιρεκτίβες που χρησιμοποιήθηκαν για την επαλήθευση του μοντέλου χωρίς τον χρονικό περιορισμό των 60 λεπτών, και γενικά η εντολή μεταγλώττισης του επαληθευτή περιγράφονται από την παρακάτω εντολή (όπως φαίνεται και από την εκτέλεση της επαλήθευσης με το XSpin) :

```
gcc -w -DPC -o pan -D_POSIX_SOURCE -DMEMLIM=128 -DSAFETY -
DNOCLAIM -DXUSAFE -DNOFAIR pan.c
```

Παρατηρείται ότι χρησιμοποιούνται οι ντιρεκτίβες :

- -DPC : Απαραίτητη ντιρεκτίβα για την δημιουργία του επαληθευτή
- -D_POSIX_SOURCE : Απαραίτητη ντιρεκτίβα για την δημιουργία του επαληθευτή
- -DMEMLIM=128 :Θέτει ως κατώτατο όριο χρήσης μνήμης τα 128 MB

- -DSAFETY :Χρησιμοποιείται επειδή δεν έχουμε επιλέξει την παράμετρο για ανίχνευση κύκλων (απενεργοποιεί μαζί τις ιδιότητες του Spin –a –l)
- -DNOCLAIM : Αποκλείει από την επαλήθευση τον ισχυρισμό never-claim, εάν αυτός υπάρχει
- -DXUSAFE : Απενεργοποιεί τον έλεγχο ορθότητας των x[rs] και xr[a] βεβαιώσεων (assertions). Η χρήση της γίνεται γιατί είναι πιο γρήγορη και κάποιες φορές η επαλήθευση και χρήσιμη εάν ο έλεγχος είναι αυστηρός π.χ. όταν τα κανάλια παρίστανται σαν παράμετροι διεργασιών όπως στο παράδειγμά μας
- -DNOFAIR : Απενεργοποιεί τον κώδικα για weak-fairness αφού δεν έχει γίνει τέτοια επιλογή στις παραμέτρους επαλήθευσης. Κάτι τέτοιο κάνει την επαλήθευση πιο γρήγορη

Έχοντας τελειώσει την πρώτη επαλήθευση του μοντέλου με το XSpin, το συμπέρασμα που προκύπτει από τα αποτελέσματα αυτής είναι ότι στο μοντέλο που συντάχθηκε δεν παρουσιάζει λάθη. Έτσι βεβαιώνεται ότι μετά από μια full αναζήτηση όλου του χώρου καταστάσεων που δημιουργούνται κατά την εκτέλεση, δεν συναντάται καμία κατάσταση όπου να τερματίζει λανθασμένα όπως επίσης έχουμε και απουσία νεκρού κώδικα. Μετά από τα συμπεράσματα αυτά παρατηρείται ότι δεν υπάρχει κανέναν πρόβλημα όσον αφορά τις ιδιότητες ασφάλειας (safety properties) κάτι που σημαίνει ότι το εργαλείο είναι έτοιμο να μας δώσει απάντηση με την προσθήκη ενός never-claim ισχυρισμού. Κάτι το οποίο θα γίνει στην επόμενη παράγραφο και που ουσιαστικά θα δώσει την τελική λύση του προβλήματος.

3.9.9 Επαλήθευση του προβλήματος με το XSpin με τον χρονικό περιορισμό των 60 λεπτών

Μετά από την επαλήθευση χωρίς τον χρονικό περιορισμό, θα πρέπει να πραγματοποιηθεί η επαλήθευση και με βάση το όριο των 60 λεπτών, έτσι ώστε να σωθούν οι στρατιώτες του προβλήματος. Παρατηρώντας τα αποτελέσματα της προσομοίωσης που πραγματοποιήθηκε στην προηγούμενη παράγραφο, μπορεί κανείς να δει ότι και οι τέσσερις στρατιώτες περνούν (κάποτε) από την ασφαλή πλευρά σε συνολικό χρόνο 95 λεπτών. Το μόνο που απομένει είναι να θέσουμε στο εργαλείο εντολή έτσι ώστε να μπορέσει να βρει μία λύση-διαδικασία με τη οποία οι στρατιώτες θα περάσουν απέναντι μέσα σε 60 λεπτά.

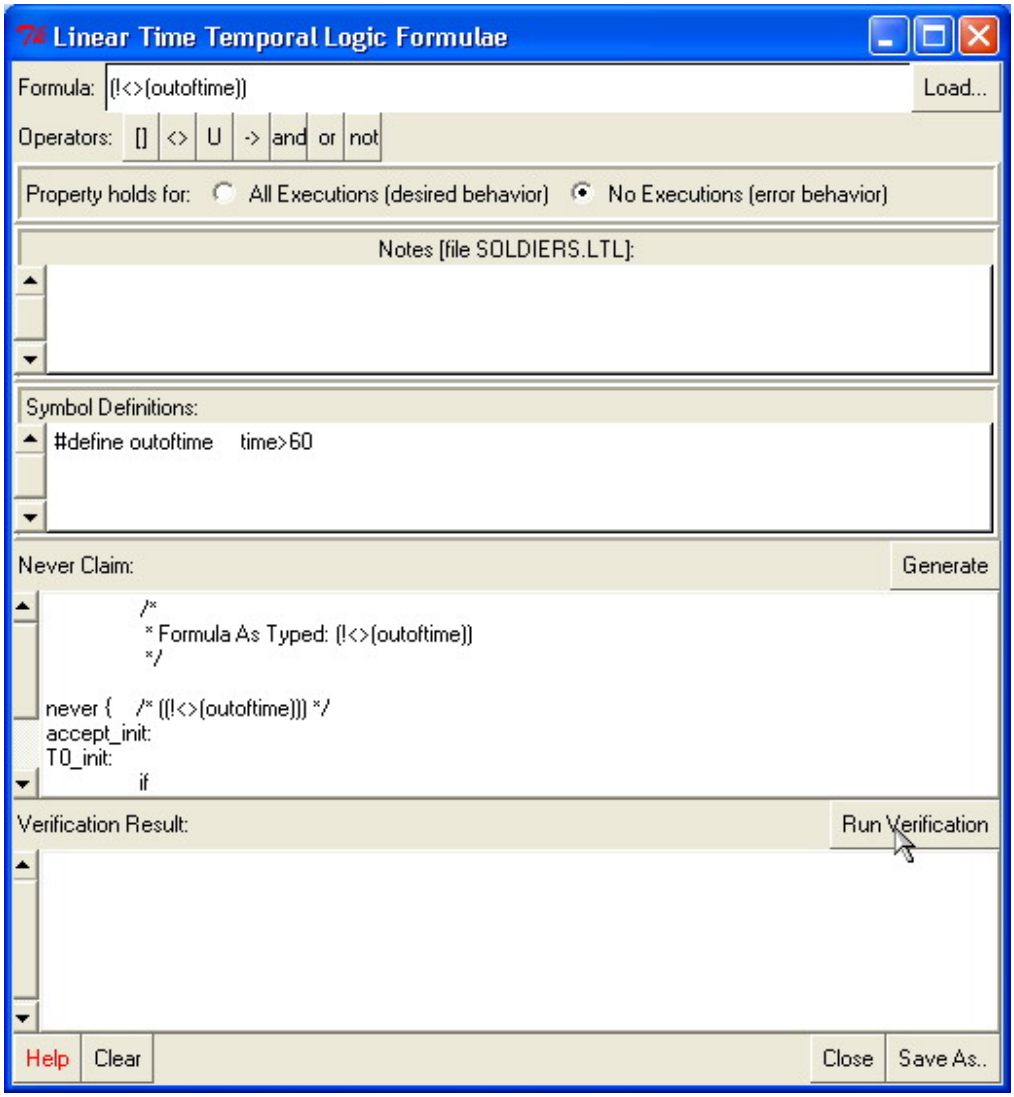
Κάτι τέτοιο γίνεται με τον ορισμό μιας πρότασης LTL η οποία μετά την μετατροπή της σε never-claim από το ίδιο το εργαλείο, θα θέσει το εργαλείο να ψάξει να βρει μια διαδικασία η οποία δεν θα επαληθεύει τον ισχυρισμό never-claim. Άρα η πρόταση LTL όπως προαναφέρθηκε και στα τελικά βήματα για την λύση του προβλήματος θα είναι :

```
<>(time>60)
```

Έτσι με βάση την σύνταξη της γλώσσας LTL που παρουσιάστηκε στο 2ο κεφάλαιο, δημιουργείται (με έναν επεξεργαστή κειμένου) ένα αρχείο soldiers.ltl το οποίο θα περιέχει μόνο την πρόταση :

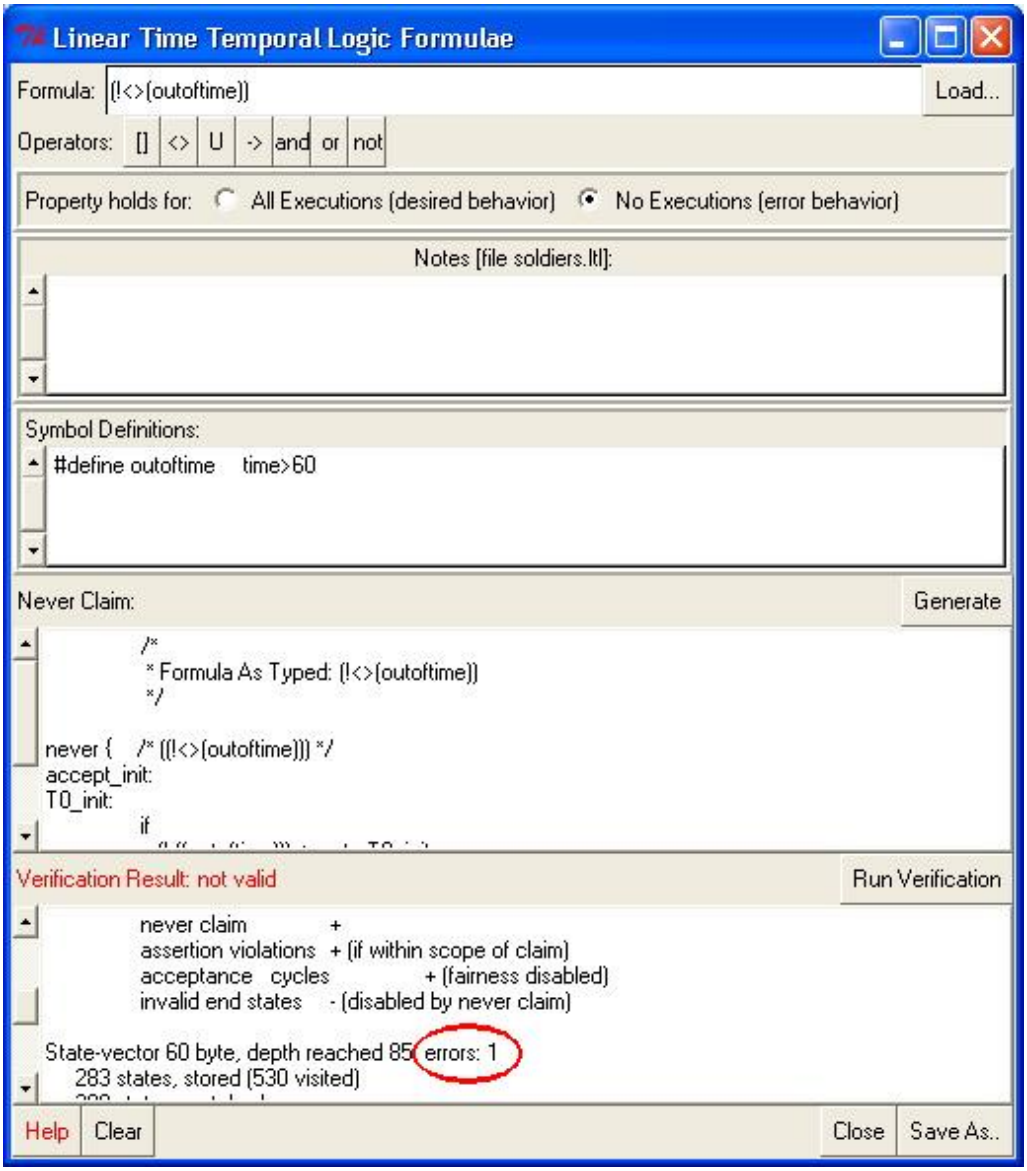
```
(!<>(outoftime))
```

όπου η μεταβλητή outoftime έχει οριστεί αρχικά στο αρχείο soldiers.prom με χρήση της εντολής #define ως time>60. Στην συνέχεια στην επιλογή του εργαλείου Run επιλέγεται το LTL Property Manager. Εμφανίζεται έτσι το παράθυρο της εικόνας () .Στην συνέχεια πατώντας το πλήκτρο Load 'δείχνουμε' στο εργαλείο το σημείο εκείνο στο οποίο βρίσκεται το αρχείο soldiers.ltl και το επιλέγουμε. Αξίζει να σημειωθεί ότι η ίδια ακριβώς διαδικασία μπορεί να γίνει απευθείας χωρίς την δημιουργία του αρχείου ltl αλλά απλώς συντάσσοντας τον τύπο LTL (εάν αυτός είναι μικρός σε μέγεθος) στο σημείο του παραθύρου formula. Το μόνο που απομένει τώρα είναι το εργαλείο να παράγει τον ισχυρισμό never-claim. Αυτό θα γίνει με την επιλογή του πλήκτρου generate το οποίο θα δώσει τον ισχυρισμό αλλά επίσης ζητήσει από τον χρήστη να ορίσει ξανά την μεταβλητή outoftime στο πλαίσιο Symbol Definitions του παραθύρου. Επίσης για μεγαλύτερη ευκολία σε μελλοντικές επαληθεύσεις του ίδιου του μοντέλου το εργαλείο δίνει την ικανότητα στον χρήστη να αποφύγει εντελώς τον LTL Property Manager ζητώντας του να δημιουργήσει ένα αρχείο *.nvr το οποίο θα περιέχει τον παραγόμενο ισχυρισμό μετά την επιλογή generate. Το αρχείο αυτό μπορεί να δοθεί στο εργαλείο με την επιλογή από το παράθυρο των παραμέτρων επαλήθευσης, [Add Never Claim From File]. Συνεχίζοντας με τον αρχικό τρόπο που αναφέρθηκε για την προσθήκη του ισχυρισμού γράφουμε την εντολή time>60 στο σημείο Symbol Definitions φτάνοντας έτσι στο αποτέλεσμα της εικόνας 3.9.17 όπου και αποτελεί το τελικό σημείο πριν την επαλήθευση.



Εικόνα 3.9.17: Παράθυρο διαλόγου του LTL property Manager

Τρέχοντας την επαλήθευση με την επιλογή Run Verification το εργαλείο θα ζητήσει από τον χρήστη τις παραμέτρους σύμφωνα με τις οποίες θα πραγματοποιηθεί αυτή. Έτσι επιλέγοντας ως τύπο αναζήτησης την εξαντλητική και αφήνοντας όλες τις υπόλοιπες επιλογές ως έχουν, πατάμε το πλήκτρο Run για την έναρξη της επαλήθευσης. Μετά από λίγο το εργαλείο αφού συντάξει ξανά το αρχείο pan.c για την δημιουργία του επαληθευτή και το μεταγλωττίσει βάση των επιλογών που έχουν γίνει με στις παραμέτρους επιστρέφει τα αποτελέσματα της επαλήθευσης, με βασικό μήνυμα ότι χαρακτηρίζει την επαλήθευση ως Not Valid όπως φαίνεται και από την εικόνα 3.9.18. Αξίζει να σημειωθεί ότι το μήνυμα αυτό δεν αποτελεί λάθος εκτέλεση του εργαλείου αλλά αντίθετα όπως θα φανεί και από τα αποτελέσματα της επαλήθευσης, το εργαλείο έχει βρει λάθος στο μοντέλο, κάτι το οποίο και περιμέναμε μετά την προσθήκη του never-claim. Στα αποτελέσματα της επαλήθευσης και συγκρίνοντάς τα με εκείνα τα αποτελέσματα που παράχθηκαν χωρίς τον χρονικό περιορισμό παρατηρείται ότι το εργαλείο έχει βρεθεί ένα λάθος στο μοντέλο το οποίο σημειώνεται στον κόκκινο κύκλο. Επίσης υπάρχουν αρκετές διαφορές όσον



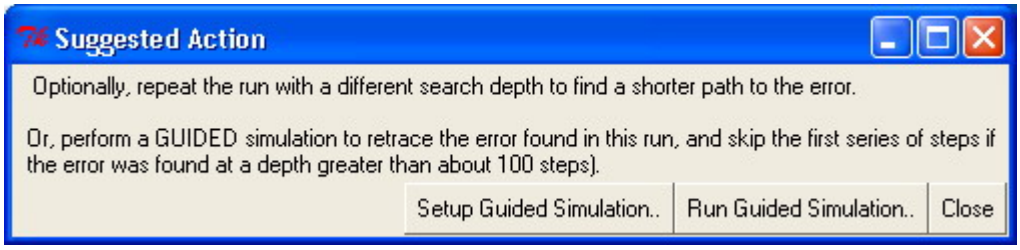
Εικόνα 3.9.18: Αποτελέσματα επαλήθευσης μέσω της δηλωθείσας LTL formulae

αφορά τις επιλογές με τις οποίες έγινε η επαλήθευση (το never-claim είναι πλέον ενεργοποιημένο +), αλλά και διαφορές που έχουν να κάνουν με την μνήμη που χρειάστηκε για την αποθήκευση των καταστάσεων του μοντέλου (χρειάστηκαν 60 bytes μνήμης πραγματοποιώντας 85 μετατροπές από την αρχική κατάσταση. Επίσης αποθηκεύθηκαν 283 καταστάσεις, 530 επισκέφθηκαν από τον αλγόριθμο, 299 επισκέφθηκαν ξανά, έγιναν 829 μετατροπές ενώ η συνολική μνήμη που απαιτήθηκε για όλα τα δεδομένα της επαλήθευσης ανέρχεται στα 2.622 MB. Υπενθυμίζεται ότι ο αναγνώστης μπορεί να δει όλα τα αποτελέσματα του εργαλείου στο παράρτημα Γ.

Αμέσως το XSpin 'βλέποντάς' το λάθος το οποίο βρήκε η επαλήθευση δίνει στον χρήστη την δυνατότητα να τρέξει μια προς καθοδήγηση προσομοίωση, δηλαδή επιστρέφοντάς του ένα παράδειγμα (counter-example) με το οποίο θα προτείνει την διαδικασία-εκτέλεση εκείνη στην οποία δεν θα υπάρχει λάθος. Με άλλα λόγια θα δώσει στον χρήστη την λύση του προβλήματος.

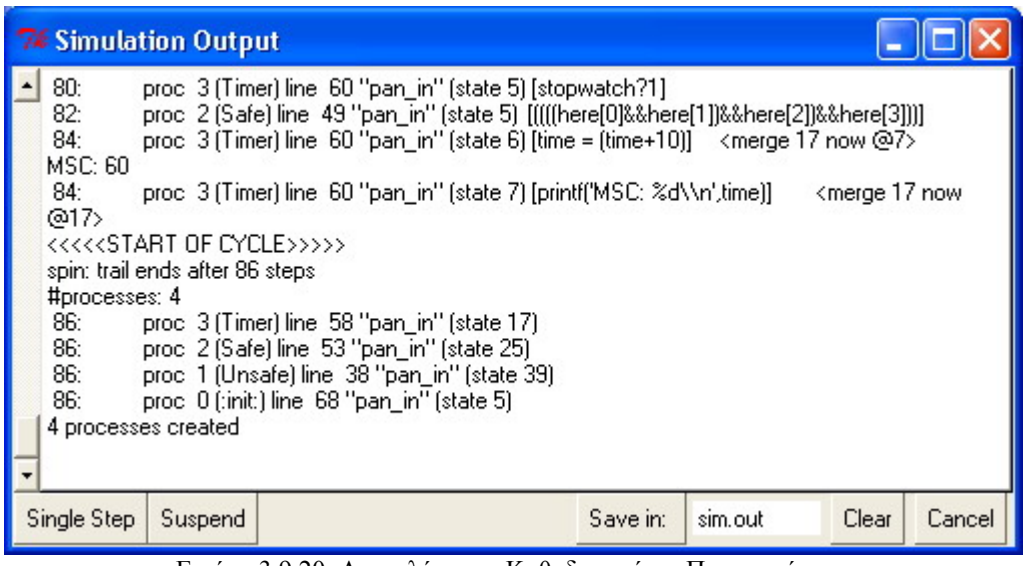
3.9.10 Προσομοίωση με καθοδήγηση του προβλήματος και εύρεση της λύσης

Στο ίδιο σημείο της εμφάνισης των αποτελεσμάτων της επαλήθευσης το εργαλείο θα προτείνει στον χρήστη να διεξάγει μια νέα προσομοίωση καθοδηγώντας τον να βρει την διαδικασία εκείνη που θα του δώσει την επιθυμητή λύση. Έτσι εμφανίζεται το παράθυρο της εικόνας 3.9.19 ζητώντας από την χρήστη είτε να ορίσει αυτός τις παραμέτρους της προσομοίωση είτε να αφήσει το εργαλείο να κάνει την δουλειά αυτή. Παρα-

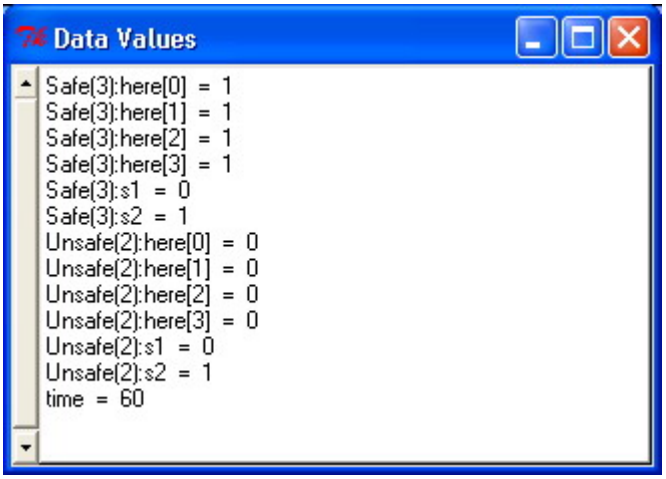


Εικόνα 3.9.19: Παράθυρο διαλόγου καθοδηγημένης προσομοίωσης

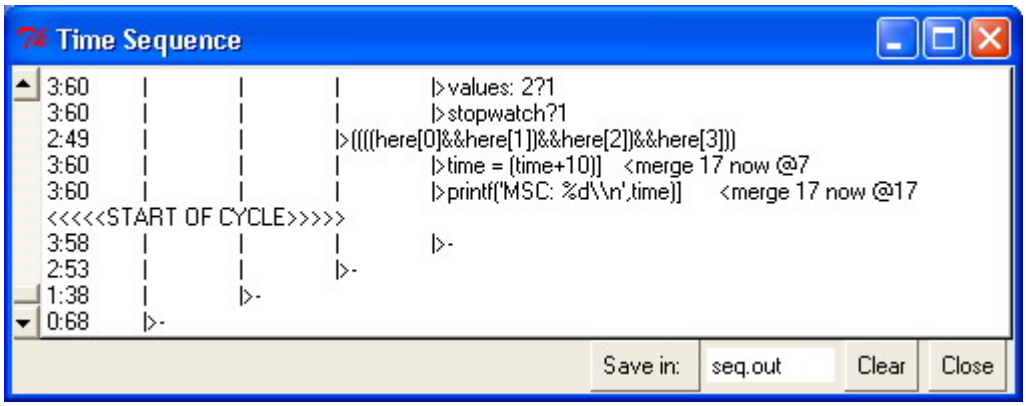
τηρείται ότι το εργαλείο μας προτείνει να επαναλάβουμε την επαλήθευση αυξάνοντας το βάθος της αναζήτησης με στόχο την εύρεση ενός πιο σύντομου μονοπατιού για την αποφυγή του λάθους. Επιμένοντας στο παράδειγμα που θα μας παράγει το XSpin παραλείπεται η περίπτωση αυτή και γίνεται η επιλογή του ορισμού των παραμέτρων μιας προσομοίωσης με καθοδήγηση. Στη συνέχεια επιλέγοντας τις παραμέτρους εκείνες (δηλαδή τα αποτελέσματα) που θέλουμε να εμφανίζει το εργαλείο στην προσομοίωση όπως ακριβώς και στην εικόνα 3.9.20 με την μόνη διαφορά ότι η αναζήτηση που γίνεται είναι με καθοδήγηση (guided seed_value=0), εμφανίζοντας τα παρακάτω αποτελέσματα (με την προϋπόθεση της επιλογής του πλήκτρου Run στο παράθυρο Simulation Output):



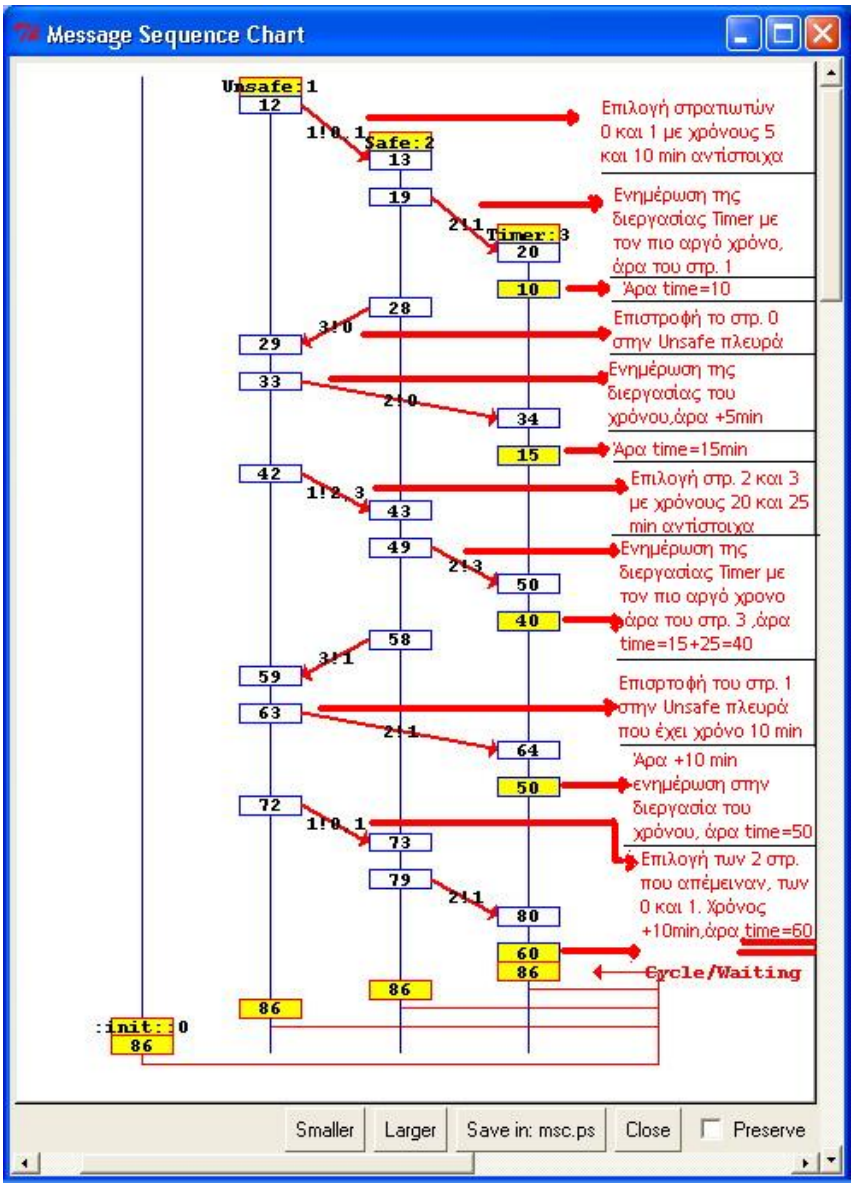
Εικόνα 3.9.20: Αποτελέσματα Καθοδηγημένης Προσομοίωσης



Εικόνα 3.9.21: Αποτελέσματα των τιμών των μεταβλητών του μοντέλου



Εικόνα 3.9.22: Αποτελέσματα Προσομοίωσης σε ακολουθία χρόνου

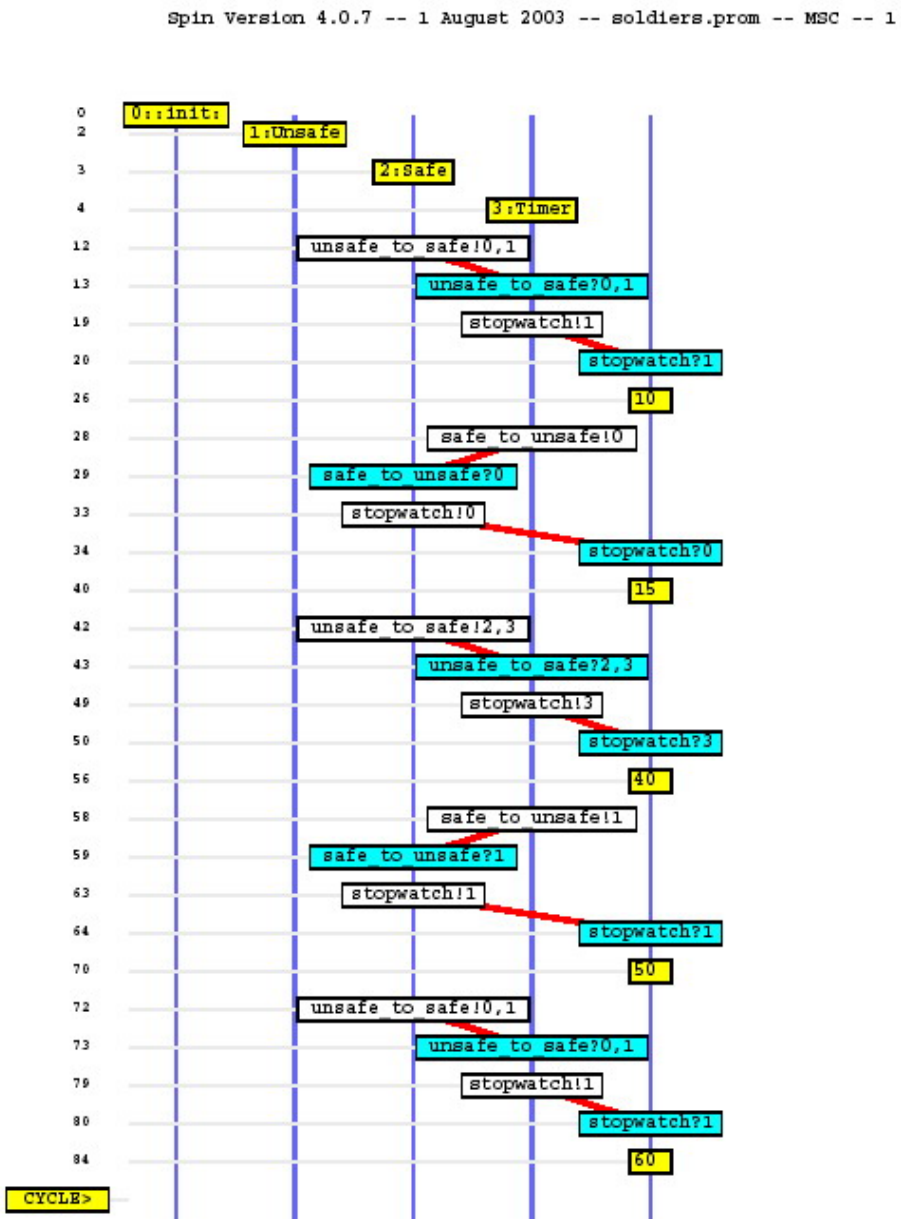


Εικόνα 3.9.23: Αποτελέσματα ακολουθίας των μηνυμάτων μεταξύ των διεργασιών MSC

Αμέσως μετά την προσομοίωση και τα αποτελέσματα αυτής μπορεί κανείς να δει ότι το εργαλείο έχει επιστρέψει την διαδικασία (επιλογής) εκείνη των στρατιωτών που θα τους φέρει σώους στην ασφαλή πλευρά μέσα σε 60 λεπτά. Από την εικόνα ακόμα η οποία και παρουσιάζει τα τελικά αποτελέσματα της προσομοίωσης όσον αφορά όλες τις τιμές των μεταβλητών του μοντέλου βλέπει κανείς ότι όλοι οι στρατιώτες βρίσκονται στην ασφαλή πλευρά (here[0]=here[1]=here[2]=here[3]=1) και ότι αυτό έχει γίνει μέσα σον χρόνο των 60 λεπτών (time=60). Αλλά κάτι τέτοιο δεν είναι αρκετό αφού αναζητάμε με ποια σειρά και ποιοι στρατιώτες θα πρέπει να περάσουν έτσι ώστε να φτάσουμε σε αυτή την λύση. Κάτι τέτοιο μπορούμε να το δούμε τόσο ανατρέχοντας στα αποτελέσματα της προσομοίωσης για να δούμε ποιοι επιλέγονται βήμα-βήμα όσο φυσικά και στο παράθυρο με την ακολουθία των μηνυμάτων που ανταλλάσσουν οι διεργασίες μεταξύ τους. Έτσι από τα αποτελέσματα της εικόνας 3.9.23 βλέπουμε ότι :
Αρχίζοντας την εκτέλεση της προσομοίωσης του μοντέλου, παρατηρείται από MSC ότι στο βήμα 12 επιλέγονται οι στρατιώτες 0 και 1 με αντιστοίχους χρόνους προσπέλασης της γέφυρας 5 και 10 λεπτά. Στο βήμα 13 βλέπουμε ότι οι στρατιώτες βρίσκονται στην

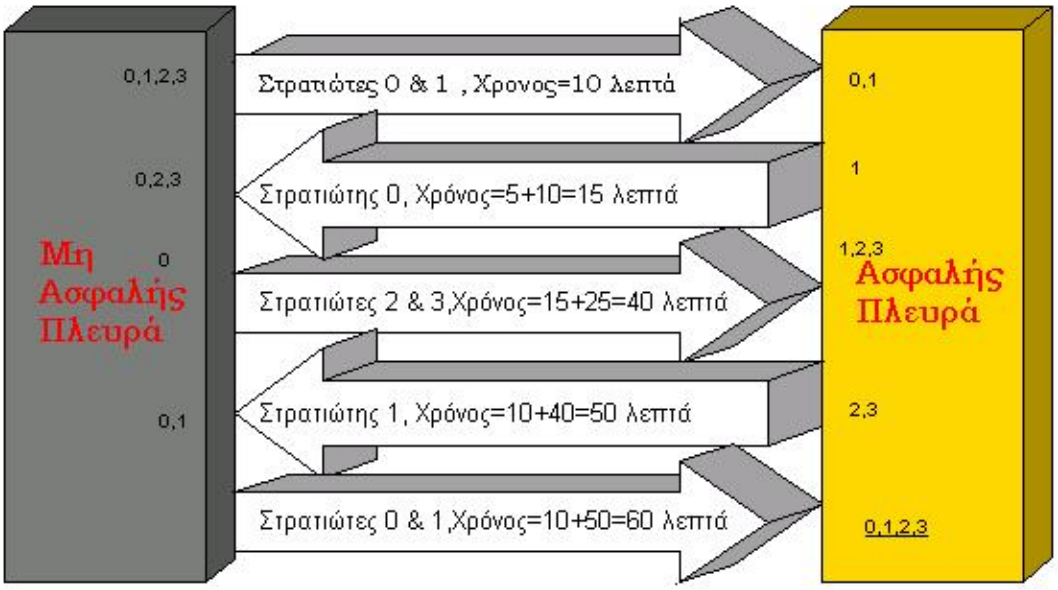
ασφαλή πλευρά της γέφυρας. Στη συνέχεια βλέπουμε ότι στο βήμα 19 στέλνεται ‘μήνυμα’ στη διεργασία του χρόνου και άρα στη μεταβλητή της time, πληροφορώντας την με τον συνολικό χρόνο που έχει περάσει για την πρώτη μετάβαση αυτή. Ο χρόνος αυτός θα είναι ίσος με τον χρόνο του πιο αργού στρατιώτη, οπότε θα είναι ο χρόνος του στρατιώτη 1 που ισούται με 10 λεπτά. Επομένως στο βήμα 20 ενημερώνεται η μεταβλητή time η οποία όπως φαίνεται και από την εικόνα παίρνει την τιμή 10 min. Στο βήμα 28 ο στρατιώτης 0 επιστρέφει την Unsafe πλευρά (για να γυρίσει πίσω τον φακό!). Στο βήμα 29 βρίσκεται εκεί. Αλλά έχουν περάσει 5 λεπτά για την μετάβαση αυτή. Οπότε στο βήμα 33 ενημερώνεται η Timer, οπότε στο βήμα 34 η time έχει τιμή 15 min. Γυρνώντας στην μη ασφαλή πλευρά βλέπουμε ότι επιλέγονται οι στρατιώτες 2 και 3 στο βήμα 42 με αντίστοιχους χρόνους προσπέλασης της γέφυρας 20 και 25 λεπτά. Ομοίως στο επόμενο βήμα (43) βρίσκονται στην ασφαλή πλευρά, ενημερώνοντας στην συνέχεια στο βήμα 49 την διεργασία χρόνου με τον πιο αργό χρόνο της νέας μετάβασης, που είναι 25min. Άρα στο βήμα 50 η time θα έχει πάρει τιμή 40 (time=15+25=40min). Στο βήμα 58 βλέπουμε ότι επιλέγεται ο στρατιώτης 1 που μεταφέρθηκε στην πρώτη μετάβαση να γυρίσει πίσω στην μη ασφαλή πλευρά για να πάρει και τον εναπομείναντα στρατιώτη. Στο βήμα 59 βρίσκεται στην μη ασφαλή πλευρά έχοντας κάνει έναν επιπλέον χρόνο τον 10 λεπτών (βήμα 63). Ενημέρωση της Timer στο βήμα 64, οπότε time=50. Έτσι φτάνουμε στο βήμα 72 όπου επιλέγονται φυσικά οι στρατιώτες που απέμειναν (οι αρχικοί που επιλέχθηκαν), οι στρατιώτες 0 και 1 με χρόνους 5 και 10 λεπτά για να μεταφερθούν απέναντι. Στο βήμα 73 βρίσκονται στην ασφαλή πλευρά ενημερώνοντας στο βήμα 79 την μεταβλητή time με τον χρόνο του πιο αργού στρατιώτη (του 1 με χρόνο 10), άρα time=50+10=60!!!!. Κάτι τέτοιο θα γίνει στο βήμα 80. Μετά από 6 βήματα, στο βήμα 86 της προσομοίωσης βλέπουμε ότι τίθεται σε ισχύ η εντολή που έχουμε ορίσει στο μοντέλο, δηλαδή εάν όλοι οι στρατιώτες έχουν περάσει από την ασφαλή πλευρά, τότε το να σταματήσει η εκτέλεση. Βλέπουμε έτσι ότι στο βήμα 86 έχουμε την παύση της εκτέλεσης και αναμονή (από το εργαλείο) για τον επόμενο κύκλο της προσομοίωσης. Θα ήταν επίσης σωστό να αναφερθεί ότι τα συμπεράσματα αυτά μπορούνε να βγούνε και από το Simulation Output ανατρέχοντας στα παραπάνω βήματα που αναφέρθηκαν. Σε κάθε βήμα αντιστοιχεί ακριβώς η εντολή του μοντέλου που εκτελείται και στο MSC φαίνεται το αποτέλεσμά της. Για την καλύτερη γραφική αναπαράσταση της λύσης του προβλήματος γίνεται και η εμφάνιση των αποτελεσμάτων του XSpin και με διαφορετικές παραμέτρους, όσον αφορά το διάγραμμα ακολουθίας μηνυμάτων μεταξύ των διεργασιών.

Επομένως παρακάτω παρουσιάζεται το ίδιο διάγραμμα, αλλά μόνο με τους στρατιώτες που επιλέγονται και των διεργασιών που χρησιμοποιούν (Το συγκεκριμένο MSC αποτελεί και προεπιλεγμένο αποτέλεσμα του εργαλείου Spin stand-alone , εάν εφαρμόσουμε το πρόβλημα).



Εικόνα 3.9.24: Προτεινόμενη λύση προβλήματος από το SPIN

Η λύση του προβλήματος έτσι ώστε να σωθούν οι στρατιώτες (από τα χέρια του εχθρού!!) στον χρόνο των 60 λεπτών θα είναι αυτή της παρακάτω εικόνας (εικόνα 3.9.24)



Εικόνα 3.9.25: Τελική λύση του προβλήματος των 4^{ων} στρατιωτών

Για το συγκεκριμένο παράδειγμα δεν παίζει σημαντικό ρόλο οι ντιρεκτίβες του Spin που θα χρησιμοποιήσουμε για την δημιουργία του επαληθευτή. Αλλά σε μια πραγματική αναφορά επαλήθευσης για ένα σύστημα επιβάλλεται μια ανάλυση και εξήγηση των ντιρεκτίβων που θα χρησιμοποιηθούν. Οι ντιρεκτίβες που χρησιμοποιήθηκαν για την επαλήθευση του μοντέλου με τον χρονικό περιορισμό των 60 λεπτών, και γενικά η εντολή μεταγλώττισης του επαληθευτή περιγράφονται από την παρακάτω εντολή (όπως φαίνεται και από την εκτέλεση της επαλήθευσης με το XSpin) :

```
gcc -w -DPC -o pan -D_POSIX_SOURCE -DMEMLIM=128 -DXUSAFE -
DNOFAIR pan.c
```

Παρατηρείται ότι χρησιμοποιούνται οι ντιρεκτίβες

- -DPC : Απαραίτητη ντιρεκτίβα για την δημιουργία του επαληθευτή
- -D_POSIX_SOURCE : Απαραίτητη ντιρεκτίβα για την δημιουργία του επαληθευτή
- -DMEMLIM=128 :Θέτει ως κατώτατο όριο χρήσης μνήμης τα 128 MB
- -DXUSAFE : Απενεργοποιεί τον έλεγχο ορθότητας των x[rs] και xr[a] βεβαιώσεων (assertions). Η χρήση της γίνεται γιατί είναι πιο γρήγορη και κάποιες φορές η επαλήθευση και χρήσιμη εάν ο έλεγχος είναι αυστηρός π.χ. όταν τα κανάλια παρίστανται σαν παράμετροι διεργασιών όπως στο παράδειγμά μας
- -DNOFAIR : Απενεργοποιεί τον κώδικα για weak-fairness αφού δεν έχει γίνει τέτοια επιλογή στις παραμέτρους επαλήθευσης. Κάτι τέτοιο κάνει την επαλήθευση πιο γρήγορη

Στην περίπτωση του χρονικού περιορισμού των 60 λεπτών η βασική ιδέα είναι να χρησιμοποιήσουμε το XSpin να βρει μια διαδικασία να περάσει όλους τους στρατιώτες στην ασφαλή πλευρά μέσα σε 60 λεπτά. Η προσπάθεια έτσι της επαλήθευσης

‘μετακινείται’ στην τεχνική ότι στην περίπτωση που όλοι οι στρατιώτες θα βρίσκονται στην ασφαλή πλευρά ο χρόνος που θα έχει γίνει αυτό να είναι πάντα μεγαλύτερος των 60 λεπτών. Με άλλα λόγια προσπαθούμε να επαληθεύσουμε ότι ‘τελικά ο χρόνος θα είναι μεγαλύτερος του 60’. Η ιδιότητα αυτή μπορεί να γραφτεί πολύ απλά με χρήση της Γραμμικής Χρονικής Λογικής (LTL), ως <>(time>60) . Έτσι η LTL ιδιότητα αυτή παραβιάζεται στην περίπτωση που οι στρατιώτες φτάσουν στην ασφαλή πλευρά σε χρόνο λιγότερο ή ίσο των 60 λεπτών. Για να επιτρέψουμε στο εργαλείο να βρει ένα παράδειγμα (counter-example) που παραβιάζει την ιδιότητα, η ιδιότητα αυτή μετατρέπεται σε έναν ‘never-claim’ ισχυρισμό σε γλώσσα Promela . Μετά από την εύρεση του λάθους από το εργαλείο και την προσομοίωση με καθοδήγηση που μας προσφέρεται εύκολα μπορεί κανείς την λύση που μας δίνεται στο πρόβλημα από το παράδειγμα που το ίδιο το Xspin μας δείχνει ότι η ιδιότητα που δώσαμε παραβιάζεται Πράγμα που σημαίνει ότι αυτή η παραβίαση είναι και η λύση που ψάχνουμε.

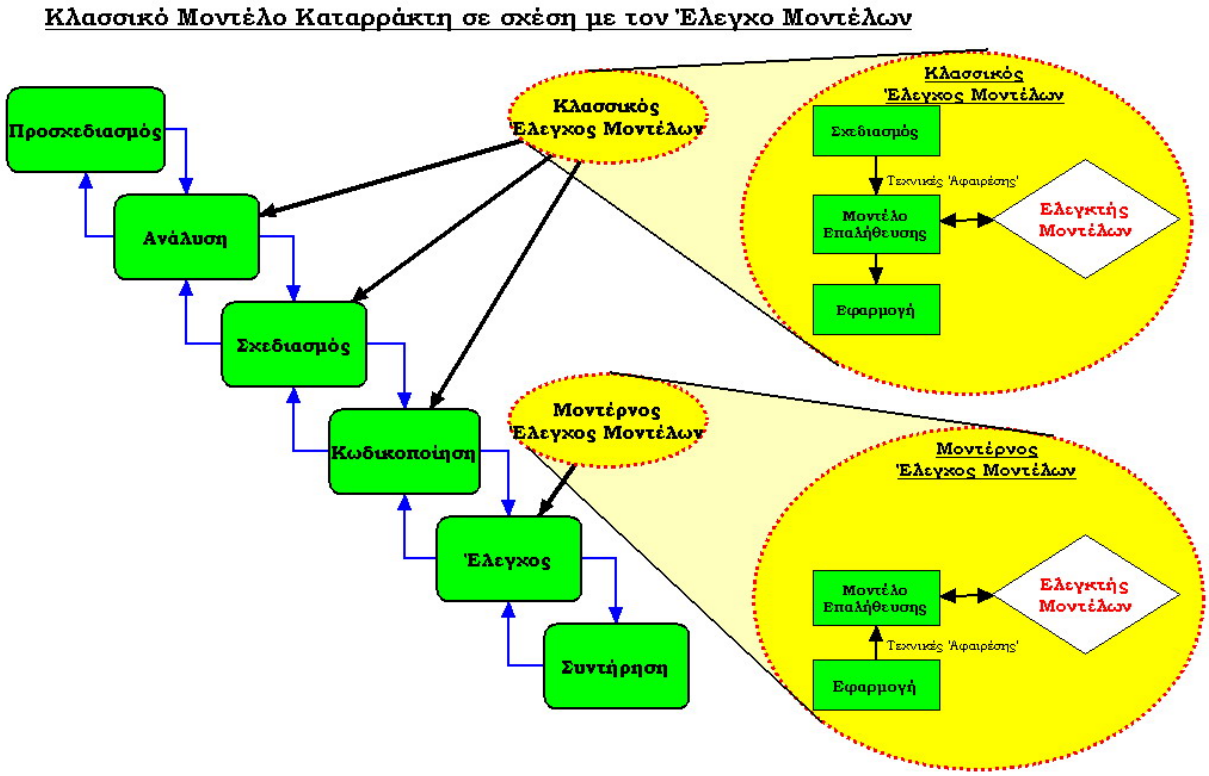
Αξίζει να σημειωθεί τέλος ότι ο αναγνώστης μπορεί να ανατρέξει στο παράρτημα του εγγράφου αυτού για τα αποτελέσματα του εργαλείου σε κάθε βήμα εκτέλεσης που αναφέρθηκε παραπάνω για καλύτερη κατανόηση του προβλήματος.

Αυτόματος έλεγχος σε μοντέλο συγχρονισμού αντικειμενοστρεφούς λογισμικού

4.1 Αντικειμενοστρέφεια και έλεγχος μοντέλων

Θα ήταν σωστό πριν την αναφορά στον έλεγχο και στην διαδικασία μοντελοποίησης του προβλήματος με το οποίο ασχολούμαστε να γίνει αναφορά στο σημείο εκείνο στο οποίο επενεργείται ο έλεγχος μοντέλων κατά την κατασκευή ενός αντικειμενοστρεφούς συστήματος. Έτσι επιλέγεται το κατ' εξοχήν μοντέλο κατασκευής το οποίο και παρουσιάζεται παρακάτω.

Ένα από τα κλασσικά μοντέλα τα οποία καθοδηγούν έναν προγραμματιστή στην δημιουργία και ανάπτυξη ενός αντικειμενοστρεφούς μοντέλου είναι και αυτό του μοντέλου του καταρράκτη. Στην περίπτωση του ελέγχου του μοντέλου αυτού και με την προϋπόθεση ότι επιθυμούμε να μοντελοποιήσουμε και να ελέγξουμε το μοντέλο με ένα αυτόματο εργαλείο, θα πρέπει να δειχθεί σε ποια στάδια ακριβώς θα γίνει ο έλεγχος αυτός είτε πρόκειται για τον κλασσικό είτε για τον μοντέρνο έλεγχο.



Εικόνα 4.1: Σχέση μοντέλου καταρράκτη με τον Έλεγχο μοντέλων

Από το σχήμα της εικόνας 4.1 στην οποία φαίνονται τα στάδια στα οποία επενεργεί ο έλεγχος μοντέλων μπορούμε να παρατηρήσουμε ότι ο κλασσικός έλεγχος μοντέλων είναι 'αναγκασμένος' να πραγματοποιείται σε τρία διαδοχικά στάδια, της ανάλυσης του σχεδιασμού και της κωδικοποίησης. Από την άλλη μεριά ο μοντέρνος έλεγχος μοντέλων πραγματοποιείται αποκλειστικά στο στάδιο του ελέγχου. Κάτι τέτοιο θα πρόσφερε σημαντική εξοικονόμηση χρόνου όσον αφορά την τελική κατασκευή του συστήματος χρησιμοποιώντας λιγότερους πόρους για να πραγματοποιηθεί.

Στην συγκεκριμένη περίπτωση θα ασχοληθούμε με το πρόβλημα των συνδαιτυμόνων φιλοσόφων που αποτελεί χαρακτηριστικό παράδειγμα αντικειμενοστρεφούς προγράμματος. Το πρόβλημα των συνδαιτυμόνων φιλοσόφων

αποτελεί ένα γνωστό παράδειγμα μιας κατάστασης η οποία παρουσιάζει αδιέξοδο. Συχνά αυτό χρησιμοποιείται σαν ένα πρότυπο για την κατανόηση των μεθόδων που αφορά τον concurrent προγραμματισμού. Υποθέτουμε ότι ο αναγνώστης έχει κάποια εμπειρία με την λειτουργία του προβλήματος γι αυτό και θα παρουσιαστεί ενδεικτικά η κλάση (ενός φιλόσοφου) η οποία δείχνει την συμπεριφορά που έχει ένας φιλόσοφος, σε αντικειμενοστρεφή γλώσσα όπως η Java. Η κλάση αυτή παρουσιάζεται ως εξής :

```
class Philosopher extends Thread

{
Object left, right;
Philosopher(Object l, Object r) {
left = l;
right = r;
}

public void run() {
while (true) {
synchronized (left) {
synchronized (right) { /* eating */ }
}
}
}
}
```

Κάθε πιρούνι αναπαρίσταται σαν ένα αντικείμενο (object left, right). Μόλις ένας φιλόσοφος πάρει ένα πιρούνι τότε σε αυτό εισέρχεται ένα κλείδωμα lock, έτσι ώστε να αποκλείσει τον διπλανό φιλόσοφο να ζητήσει πρόσβαση σε αυτό. Κάτι τέτοιο όπως θα φανεί και παρακάτω αποτελεί έναν περιορισμό ατομικότητας η οποία βοηθά στον συγχρονισμό των αντικειμένων. Ο συγχρονισμός που απαιτείται για την λύση του προβλήματος επεξηγείται αναλυτικότερα στις παρακάτω παραγράφους. Αρχικά δίνεται η εκφώνηση του προβλήματος των συνδαιτυμόνων φιλοσόφων και στη συνέχεια παραουσιάζονται δύο διαφορετικές υλοποιήσεις: η πρώτη παρουσιάζοντας αδιέξοδο και η δεύτερη λύνοντας το πρόβλημα αυτό.

4.2 Το πρόβλημα των συνδαιτυμόνων φιλοσόφων

Υπάρχουν κάποιοι φιλόσοφοι που κάθονται σε ένα στρογγυλό τραπέζι και μοναδικός τους σκοπός είναι να συζητάνε και να τρώνε. Κάθε φιλόσοφος για να φάει χρειάζεται να χρησιμοποιήσει τα δύο πιρούνια που βρίσκονται δεξιά και αριστερά του. Αν ένας φιλόσοφος πάρει ένα πιρούνι, τότε δεν το αφήνει μέχρι να πάρει και το άλλο, έτσι ώστε να είναι ικανός να φάει. Έπειτα αφήνει κάτω και τα δύο πιρουνιά και περιμένει μέχρι να πεινάσει ξανά. Η μοντελοποίηση αυτού του προβλήματος μπορεί να φαντάζει εύκολη. Για παράδειγμα: μπορεί κάθε φιλόσοφος να παίρνει πρώτα το πιρούνι που βρίσκεται στα αριστερά του και έπειτα να περιμένει να πάρει αυτό στα δεξιά του. Τι θα γίνει όμως αν κάθε φιλόσοφος πάρει από ένα πιρούνι; Πολύ απλά το σύστημα οδηγείται σε κατάσταση αδιεξόδου, λόγω παρατεταμένης στέρησης. Το πρόβλημα των συνδαιτυμόνων φιλοσόφων, μπορεί να θεωρηθεί ως αντιπρόσωπος των

προβλημάτων που σχετίζονται με τον συντονισμό των διαμοιραζόμενων πόρων που απαιτείται όταν μια εφαρμογή περιέχει ταυτόχρονα νήματα εκτέλεσης (threads).

4.3 Επεξήγηση του συγχρονισμού του προβλήματος

Με την προϋπόθεση ότι ο φιλόσοφος θέλει να φάει, πρέπει να ανακτήσει και το δεύτερο πιρούνι. Έτσι πρέπει να εκτελέσει δύο συγχρονισμένες προτάσεις (την μία για το αριστερό και την άλλη για το δεξί πιρούνι). Όπως δηλώνεται και από τον τίτλο του κεφαλαίου, θα γίνει η προσπάθεια αυτόματου ελέγχου ενός μοντέλου συγχρονισμού αντικειμενοστρεφούς λογισμικού από τον αυτόματο ελεγκτή μοντέλων εργαλείο Spin. Το πρόβλημα όπως παρουσιάστηκε και αρχικά στο οποίο θα εφαρμοστεί ο έλεγχος αναφέρεται στην περίπτωση των συνδαιτυμόνων φιλοσόφων οι οποίοι αποτελούν ένα κατεξοχήν παράδειγμα συγχρονισμού. Μία ομάδα από φιλόσοφους (διεργασίες) χρειάζεται να συντονιστεί έτσι ώστε να έχουν πρόσβαση σε έναν αριθμό από πιρούνια (πόρους). Ο αριθμός των φιλοσόφων είναι ο ίδιος με τον αριθμό των πιρουνιών και κάθε φιλόσοφος χρειάζεται δύο πιρούνια για να δειπνήσει. Η πρόσβαση στα πιρούνια θα πρέπει να οργανωθεί έτσι ώστε να αποφευχθούν οι καταστάσεις αδιεξόδου και starvation. Καταστάσεις οι οποίες και θα ανιχνευθούν κατά τον έλεγχο από το εργαλείο και θα επιβεβαιωθούν σε περίπτωση απουσία τους. Μπορούμε να αναπαραστήσουμε τους φιλόσοφους και τα πιρούνια σαν κατανεμημένα αντικείμενα. Η δομή της κλάσης ενός πιρουνιού παρουσιάζεται παρακάτω. Κάθε πιρούνι έχει και μια μέθοδο η οποία δείχνει ότι ένα πιρούνι έχει ανακτηθεί από έναν φιλόσοφο (pickup) και άλλη μια η οποία να δείχνει ότι ο φιλόσοφος έχει αφήσει το πιρούνι (drop).

```
Class Chopstick
     isPicked := false;
disable pickup if isPicked;

method pickup(phil)
     isPicked :=true;
     …
     end pickup;

method drop()
     isPicked :=false;
     …
     end drop;
end Chopsick;
```

Όπως αναφέρθηκε κάθε πιρούνι πρέπει να χρησιμοποιείται κάθε φορά από έναν μόνο φιλόσοφο. Έτσι μοντελοποιείται η απαίτηση αυτή σαν μια ιδιότητα του κάθε πιρουνιού και αναπαρίσταται σαν ένας περιορισμός συγχρονισμού (synchronization constraint κεφ.1). Η πρόταση isPicked αναφέρεται στην κατάσταση εκείνη του πιρουνιού όπου ανάλογα με την τιμή true ή false ένα πιρούνι θα έχει καταληφθεί από έναν φιλόσοφο ή θα είναι ελεύθερο αντίστοιχα. Εάν κάθε ένας φιλόσοφος σηκώσει από ένα πιρούνι τότε έχουμε αδιέξοδο. Προς αποφυγή της κατάστασης αυτής θα πρέπει ο

κάθε φιλόσοφος να σηκώνει και τα δύο τα πιρούνια ατομικά. Η ατομικότητα αυτή πραγματοποιείται με την εφαρμογή ενός περιορισμού ατομικότητας στις pickup μεθόδους του κάθε πιρουνιού. Έτσι ένας συγχρονιστής που θα δείχνει την ακολουθία όπου ο κάθε φιλόσοφος θα πρέπει να σηκώνει ένα πιρούνι παρουσιάζεται παρακάτω :

```
Synchronizer indivisiblePickup (c1,c2,phill)
     Atomic
     (c1.pickup(p1) if p1 = phill,
     c2.pickup(p2) if p2 = phill);
end indivisiblePickup;
```

Ο παραπάνω συγχρονιστής συντονίζει την πρόσβαση στα πιρούνια για κάθε έναν φιλόσοφο. Για την πλήρη λύση του προβλήματος θα πρέπει να υπάρχει ένας τέτοιος συγχρονιστής για κάθε φιλόσοφο. Με την λειτουργία αυτών των μηχανισμών συγχρονισμού θα λυθεί το πρόβλημα και θα αποφευχθεί η περίπτωση του αδιέξοδου.

4.4 Απαιτήσεις των φιλοσόφων για την λύση του προβλήματος

Οι απαιτήσεις του προβλήματος για τον συγχρονισμό του κάθε φιλόσοφου και την τελική λύση του προβλήματος, θα είναι οι παρακάτω :

- Για να δειπνήσει ένας φιλόσοφος θα πρέπει να αποκτήσει το δεξί και το αριστερό πιρούνι
- Τα πιρούνια δεν μπορούν να μοιράζονται ταυτόχρονα σε δύο φιλόσοφους
- Δεν πρέπει το πρόβλημα να παρουσιάζει κατάσταση αδιέξοδου
- Δεν πρέπει να υπάρχει φιλόσοφος ο οποίος να μην τρώει ποτέ (κατάσταση παρατεταμένης στέρησης)

Βάσει των παραπάνω απαιτήσεων με τις οποίες θα μοντελοποιήσουμε το πρόβλημα των συνδαιτυμόνων φιλοσόφων αξίζει να σημειωθεί ότι θα κατασκευαστούν δύο μοντέλα στην γλώσσα promela. Ένα το οποίο θα περιέχει κατάσταση αδιέξοδου όπου κάθε φιλόσοφος θα έχει από ένα πιρούνι (και θα περιμένει μάταια να απελευθερωθεί το επόμενο) και το εργαλείο Spin θα βρίσκει το λάθος αυτό, και ένα δεύτερο μοντέλο το οποίο θα μοντελοποιεί το πρόβλημα έτσι ώστε να πληρούνται όλες οι παραπάνω προϋποθέσεις. Αξίζει να σημειωθεί ότι στην δεύτερη περίπτωση το εργαλείο δεν θα βρει λάθος αλλά επίσης θα προσομοιώσει το πρόβλημα δείχνοντας ότι όλοι οι φιλόσοφοι τρώνε χωρίς να έρχονται ποτέ σε αδιέξοδο.

Αξίζει να σημειωθεί ότι οι ελεγκτές μοντέλων έχουν την ικανότητα να κάνουν αποτελεσματικό έλεγχο σε μοντέλα (αναφερόμενοι στο συγκεκριμένο παράδειγμα) που μοντελοποιούνται σαν πεπερασμένα μοντέλα καταστάσεων (δηλαδή ο αριθμός των φιλοσόφων να είναι πεπερασμένος). Σε αντίθετη περίπτωση ο ελεγκτής μοντέλων δεν μπορεί να ελέγξει μια περίπτωση Ν φιλοσόφων.

Παρακάτω θα γίνει η παρουσίαση του πρώτου μοντέλου του προβλήματος και η εξήγηση βήμα προς βήμα του κώδικά αυτού έτσι ώστε να γίνει κατανοητό από τον

αναγνώστη τόσο ο μηχανισμός συγχρονισμού των φιλοσόφων όσο και το αποτέλεσμα του αδιέξοδου που παρουσιάζει το μοντέλο.

4.5 Υλοποίηση του 1ου μοντέλου του προβλήματος

Ακολουθεί παρακάτω η δημιουργία του μοντέλου του προβλήματος στην γλώσσα promela όπου και θα αναλυθεί κάθε πρόταση που συντάσσεται. Το ολοκληρωμένο μοντέλο φαίνεται στο παράρτημα Γ αρχείο plilldeadlock.pr .

Αρχικά ορίζονται οι πόροι του προβλήματος, τα πιρούνια, σαν καταστάσεις bit. Στο συγκεκριμένο πρόβλημα επιλέγεται η περίπτωση όπου έχουμε τρεις φιλόσοφους άρα και τρία πιρούνια.

bit fork[3];

Οι καταστάσεις στις οποίες μπορεί ένας φιλόσοφος μπορεί να βρεθεί είναι τρεις : κατάσταση στην οποία δεν έχει κανένα πιρούνι στην κατοχή του η οποία αποτελεί την κατάσταση thinking, κατάσταση στην οποία ο φιλόσοφος έχει ανακτήσει ένα πιρούνι η οποία αποτελεί την κατάσταση waiting, και η κατάσταση στην η οποία ο φιλόσοφος έχει και τα δύο πιρούνια (και άρα μπορεί να φάει) η οποία αποτελεί την κατάσταση eating. Έτσι ορίζονται διαδοχικά οι τρεις καταστάσεις :

```
mtype = {thinking, waiting, eating};
```

Στη συνέχεια ορίζεται η διεργασία του φιλόσοφου οι οποία και όπως φαίνεται παρακάτω αποτελείται από παράμετρους

```
proctype philosopher(byte left, right) {
```

καταστάσεις τύπου byte και αναπαριστούν τα αντίστοιχα το δεξί και το αριστερό πιρούνι του φιλόσοφου. Οι μέθοδοι pickup και drop που αναφέρθηκαν παραπάνω αναπαρίστανται σαν καταστάσεις 1 και 0 αντίστοιχα.

```
mtype state = thinking;
```

Αρχική κατάσταση όλων των φιλοσόφων χωρίς να έχουν στην κατοχή τους πιρούνια, άρα κατάσταση thinking

```
   do
   :: ((state==thinking) && (fork[left]==0)) ->
atomic{fork[left]=1; state=waiting;}
```

Μετάβαση από κατάσταση thinking χωρίς κανένα πιρούνι σε κατάσταση waiting όπου ο φιλόσοφος έχει ένα πιρούνι

```
       :: ((state==waiting) && (fork[right]==0)) ->
atomic{fork[right]=1; state=eating;}
```

Μετάβαση από κατάσταση waiting με ένα πιρούνι σε κατάσταση eating όπου ο φιλόσοφος έχει και τα δύο πιρούνια και μπορεί να φάει

```
       :: (state==eating) ->
```

Αφού ο φιλόσοφος φάει μεταβαίνει σε κατάσταση thinking και αφήνει τόσο το δεξί όσο και το αριστερό πιρούνι

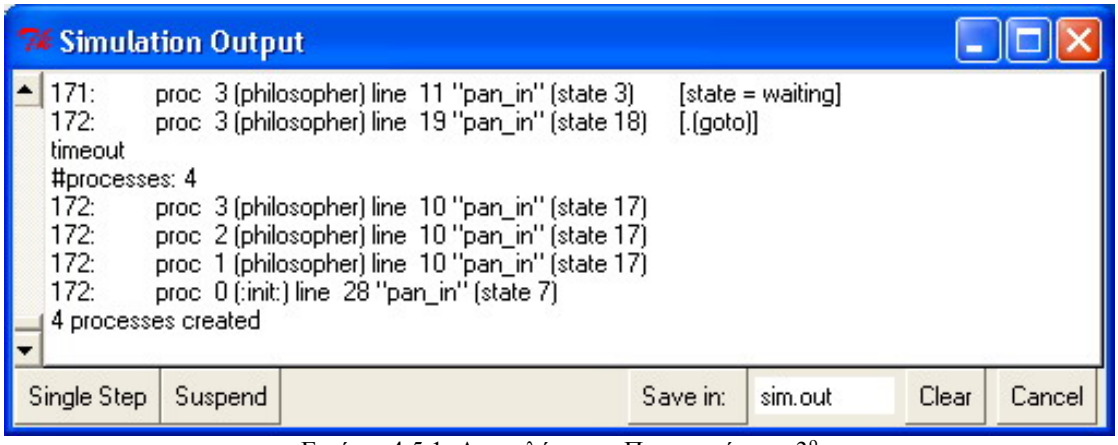
```
        if
          :: skip;
          :: atomic{state=thinking; fork[right]=0;
fork[left]=0;}
          fi
       od;
     }
```

Διεργασία έναρξης init με την οποία αρχίζει η 'εκτέλεση' του προβλήματος. Κάθε πιρούνι βρίσκεται άθικτο, κατάσταση 0 και έπειτα εκτελούνται εναλλακτικά κάθε φιλόσοφος δείχνοντας την θέση των πιρουνιών στα οποία αυτός έχει πρόσβαση.

```
     init {
       fork[0]=0;
       fork[1]=0;
       fork[2]=0;
       run philosopher(0, 1);
       run philosopher(1, 2);
       run philosopher(2, 0);
     }
```

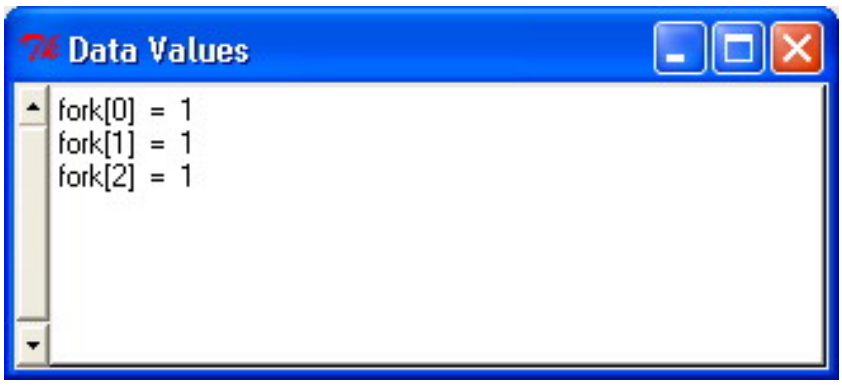
4.5.1 Προσομοίωση του προβλήματος με το εργαλείο Spin

Δημιουργώντας το αρχείο philldeadlock.prom ξεκινάμε την προσομοίωση του συγκεκριμένου μοντέλου. Έτσι μετά από έναν απαραίτητο συντακτικό έλεγχο για επιβεβαίωση της ορθότητας του μοντέλου θα ξεκινήσουμε την προσομοίωση αυτού επιλέγοντας τις default επιλογές προσομοίωσης. Τα αποτελέσματα ακολουθούν παρακάτω :



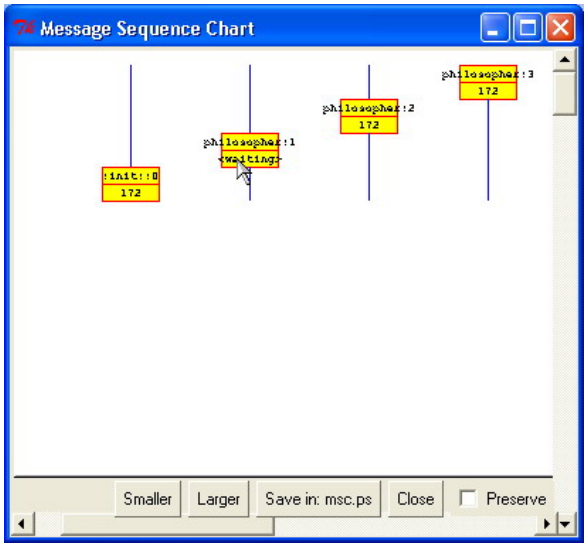
Εικόνα 4.5.1: Αποτελέσματα Προσομοίωσης 3ο

Παρατηρείται ότι στο βήμα 172 ολοκληρώνεται η προσομοίωση του προβλήματος έχοντας δημιουργήσει 4 διεργασίες (τρεις φιλόσοφους και μια διεργασία έναρξης). Στο παράρτημα Γ όπου παρουσιάζονται τα αποτελέσματα της προσομοίωσης ο αναγνώστης μπορεί να δει λεπτομερώς την λειτουργία του προβλήματος, πότε ο κάθε φιλόσοφος βρίσκεται σε κατάσταση thinking waiting και eating. Επίσης μπορεί να δει ότι όλοι οι φιλόσοφοι τρώνε από μια φορά τουλάχιστον, πράγμα που σημαίνει ότι αποφεύγεται η κατάσταση starvation.



Εικόνα 4.5.2: Αποτελέσματα τιμών των μεταβλητών

Από τις τελικές τιμές των μεταβλητών του μοντέλου παρατηρείται ότι κάθε φιλόσοφος έχει και από ένα πιρούνι, πράγμα που μας προμηνύει ότι έχουμε φτάσει σε κατάσταση αδιεξόδου.

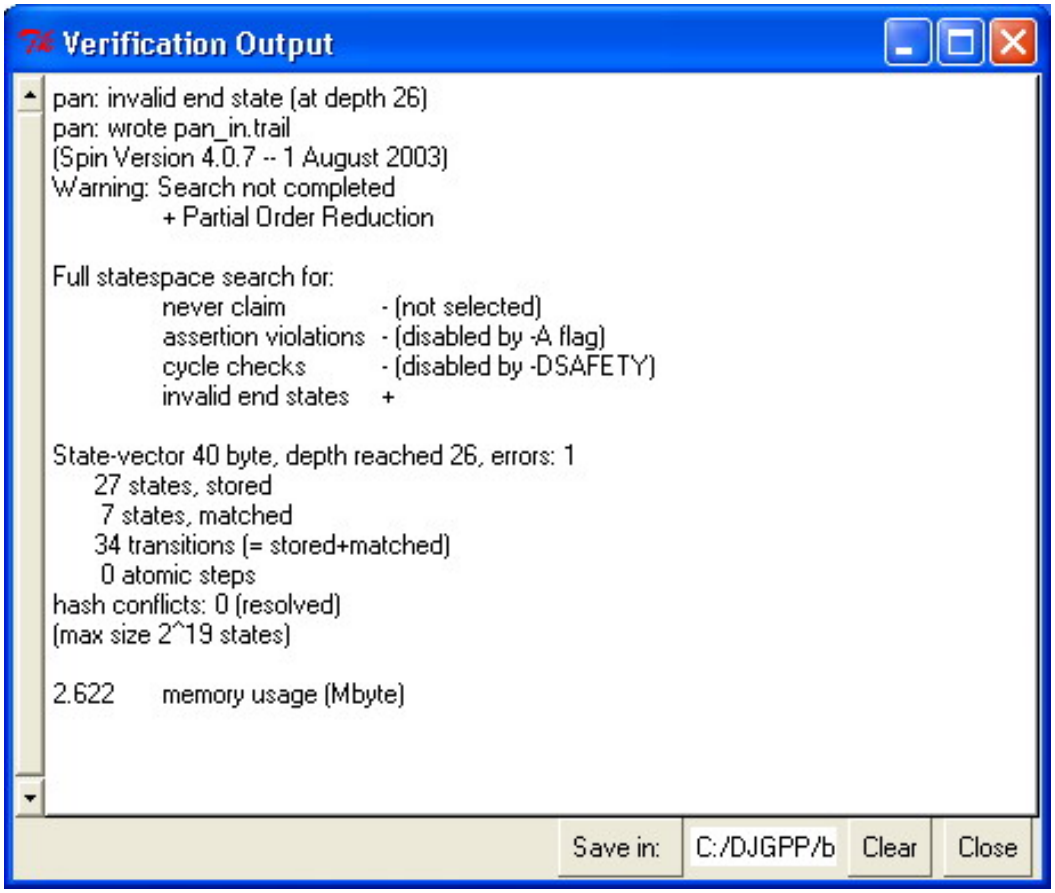


Εικόνα 4.5.3: Αποτελέσματα διαγράμματος ακολουθίας

Φαίνεται από τα αποτελέσματα του MSC ότι έχουν ολοκληρωθεί όλες οι διεργασίες του μοντέλου και επιπλέον (που είναι και το βασικό) οι διεργασίες των φιλοσόφων βρίσκονται και οι τρεις στην κατάσταση waiting. Κάτι που σημαίνει ότι όλοι οι φιλόσοφοι έχουν από ένα πιρούνι και άρα περιμένουν το δεύτερο. Επομένως έχουμε αδιέξοδο. Κάτι τέτοιο θα φανεί παρακάτω και από την επαλήθευση του μοντέλου από το εργαλείο από όπου περιμένει κανείς το Spin να του εμφανίσει λάθος.

4.5.2 Επαλήθευση του προβλήματος με το εργαλείο Spin

Περνώντας στην επαλήθευση του προβλήματος οι επιλογές οι οποίες θα κάνουμε για αυτήν την περίπτωση θα έχουν να κάνουν με τις ιδιότητες ασφάλειας του μοντέλου που έχουμε συντάξει (Safety properties). Έτσι επιλέγοντας από τις ιδιότητες επαλήθευσης του εργαλείου για λανθασμένες καταστάσεις τερματισμού (αδιέξοδο), και πραγματοποιώντας την επαλήθευση θα έχουμε το παρακάτω αποτέλεσμα :



Εικόνα 4.5.4: Αποτελέσματα Επαλήθευσης

Από τα αποτελέσματα της επαλήθευσης του προβλήματος μπορούμε να δούμε ότι το εργαλείο έχει βρει λάθος (όπου πρόκειται για λανθασμένη κατάσταση τερματισμού) αδιεξόδου προτείνοντάς μας να ξανακάνουμε προσομοίωση με καθοδήγηση ή όχι για την αποφυγή της ανεπιθύμητης κατάστασης. Κάτι τέτοιο δεν θα γίνει (σε αντίθετη περίπτωση με την λύση που μας δίνει το εργαλείο στην περίπτωση των $4^{ων}$ στρατιωτών) υπενθυμίζεται ότι θέλουμε απλώς το εργαλείο να εξετάσει το μοντέλο που δημιουργήθηκε για λάθος και όχι να μας δώσει την λύση. Την λύση του προβλήματος θα την δώσει μια καινούργια, πιο αποτελεσματική μοντελοποίησή του η οποία και θα υπακούει στους κανόνες του συγχρονισμού των φιλοσόφων και των πιρουνιών που αυτοί χρησιμοποιούν. Το νέο αυτό μοντέλο δεν θα έρχεται σε κατάσταση αδιεξόδου όπου και αυτό θα επιβεβαιώνεται και από τα αποτελέσματα του εργαλείου αλλά επίσης θα υπακούει σε όλες τις απαιτήσεις της λύσης του προβλήματος που αναφέρθηκαν σε πιο παραπάνω παράγραφο.

4.6 Υλοποίηση του $2^{ου}$ μοντέλου του προβλήματος χωρίς αδιέξοδο

Το πρόγραμμα που ακολουθεί είναι η δεύτερη υλοποίηση των συνδαιτυμόνων φιλοσόφων. Στην υλοποίηση αυτή, αποτρέπεται το πρόβλημα του αδιεξόδου με αποτέλεσμα οι φιλόσοφοι να τρώνε συνέχεια χωρίς ποτέ το σύστημα να φτάνει σε κατάσταση αδιεξόδου. Το πρόβλημα επιλύθηκε με συγχρονισμό των διαδικασιών.

Παρακάτω θα εξηγήσουμε γραμμή προς γραμμή τον κώδικα του αρχείου που υλοποιήσαμε σε *promela*.

```
#define Think1 printf( "MSC: Think \n")
#define Eat1   printf( "MSC: Eat \n")
#define Wait1  printf( "MSC: Wait \n" )
```

Στις τρεις αυτές γραμμές ορίζουμε κάποιες σταθερές και συγκεκριμένα τις:
Think1, Eat1, Wait1. Όλες τυπώνουν στο Message Sequence Chart τα μηνύματα Think,Eat,Wait. Δηλαδή η σταθερά Think1 τυπώνει το Think και ομοίως και οι υπόλοιπες. Η παραπάνω διαδικασία δεν είναι απαραίτητη αλλά διευκολύνει τον χρήστη να παρακολουθήσει την εξέλιξη της προσομοίωσης στο MSC.

```
mtype={Think,Wait,Eat}
```

Το mtype είναι μια συμβολική σταθερά.

```
byte S1=Think;
byte S2=Think;
byte S3=Think;
```

Στην αρχή ορίζουμε κάθε φιλόσοφος να είναι στην κατάσταση THINK, δηλαδή να μην έχει κανένα πιρούνι. Τα S1,S2,S3 είναι μεταβλητές και αντιστοιχούν στους τρεις φιλοσόφους.

Σε αυτό το σημείο είναι καλό να γίνει μια διευκρίνιση όσον αφορά τις μεταβλητές. Όταν ο φιλόσοφος είναι σε κατάσταση Think τότε S=3, σε κατάσταση Wait, S=2 και σε κατάσταση Eat S=1. Στην αρχή είναι S1=S2=S3=3

Παρακάτω ορίζουμε τις τρεις διεργασίες με τις οποίες υλοποιείται η μοντελοποίηση. Θα εξηγήσουμε μόνο την πρώτη γιατί και οι υπόλοιπες δύο είναι παρόμοιες και δεν χρειάζεται περαιτέρω διευκρίνιση.

```
active proctype P1 () {
do
     ::S3=Think ->S1=Wait;Wait1;
                  do
                      :: S2=Think->progress: S1=Eat;Eat1;
printf("P1 eating\n");
S1=Think;Think1;
break
     od;
od
}

active proctype P2 () {
do
```

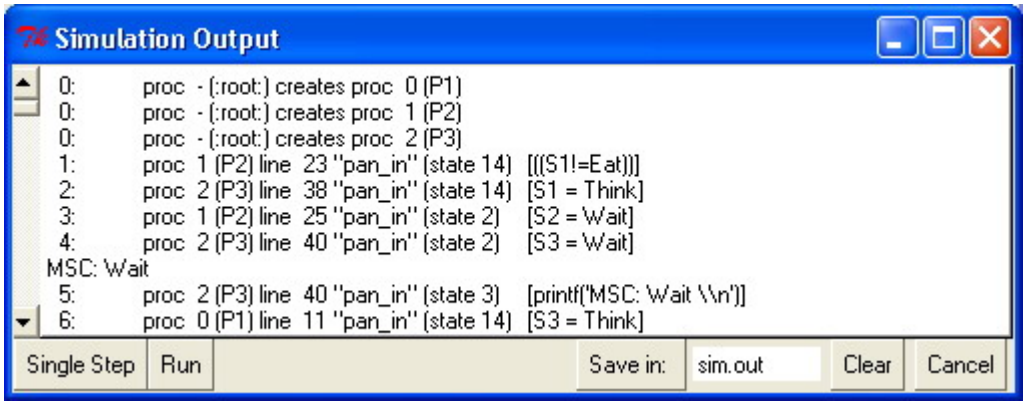
```
:: S1!=Eat->
      S2=Wait;Wait1;
     do
     :: S3!=Eat->
     progress: S2=Eat;Eat1;
printf("P2 eating\n");
S2=Think;Think1;
break
     od;
od
}

active proctype P3 () {
do
:: S1=Think->
      S3=Wait;Wait1;
     do
     :: S2!=Eat->
     progress:S3=Eat;Eat1;
printf("P3 eating\n");
S3=Think;Think1;
break
     od;
od
}
```

Με απλά λόγια η διεργασία που αναφέρεται στον πρώτο φιλόσοφο λειτουργεί ως εξής: Ελέγχει αν ο τρίτος φιλόσοφος είναι στην κατάσταση Think, δηλαδή αν δεν έχει κανένα από τα δύο πιρούνια. Αν συμβαίνει αυτό τότε ο πρώτος φιλόσοφος παίρνει το πιρούνι που βρίσκεται στα αριστερά του και μεταβαίνει στην κατάσταση Wait. Έπειτα ελέγχει για το δεύτερο φιλόσοφο αν είναι σε κατάσταση Think και αν είναι, τότε ο πρώτος φιλόσοφος παίρνει το πιρούνι που βρίσκεται στα δεξιά του και μεταβαίνει στην κατάσταση Eat. Αφού φάει, αφήνει και τα δύο πιρούνια και μεταβαίνει στην κατάσταση Think.

Με παρόμοιο τρόπο λειτουργούν και οι δύο επόμενες διεργασίες που αναφέρονται αντίστοιχα στο δεύτερο και τρίτο φιλόσοφο. Έτσι επιτυγχάνεται ο μηχανισμός συγχρονισμού ανάμεσα στις τρεις αυτές διεργασίες. Με τον τρόπο αυτό δεν γίνεται ποτέ το σύστημα να φτάσει σε αδιέξοδο (deadlock). Εκτελείται συνέχεια και οι φιλόσοφοι τρώνε για άπειρο χρονικό διάστημα.

4.6.1 Προσομοίωση του προβλήματος

Πρέπει να σημειωθεί ότι στη διαδικασία της προσομοίωσης χρησιμοποιήσαμε τις εξ ορισμού επιλογές που μας προσφέρει το γραφικό περιβάλλον XSpin. Δεν θα επεκταθούμε σε λεπτομέρειες, όσον αφορά τις επιλογές αυτές, μιας και το θέμα αυτό αναλύθηκε εκτενώς στο προηγούμενο κεφάλαιο. Μετά τον απαραίτητο συντακτικό έλεγχο κάνουμε την προσομοίωση (επιλογή Random). Τα αποτελέσματα εξηγούνται παρακάτω.



Εικόνα 4.6.1: Αποτελέσματα της προσομοίωσης

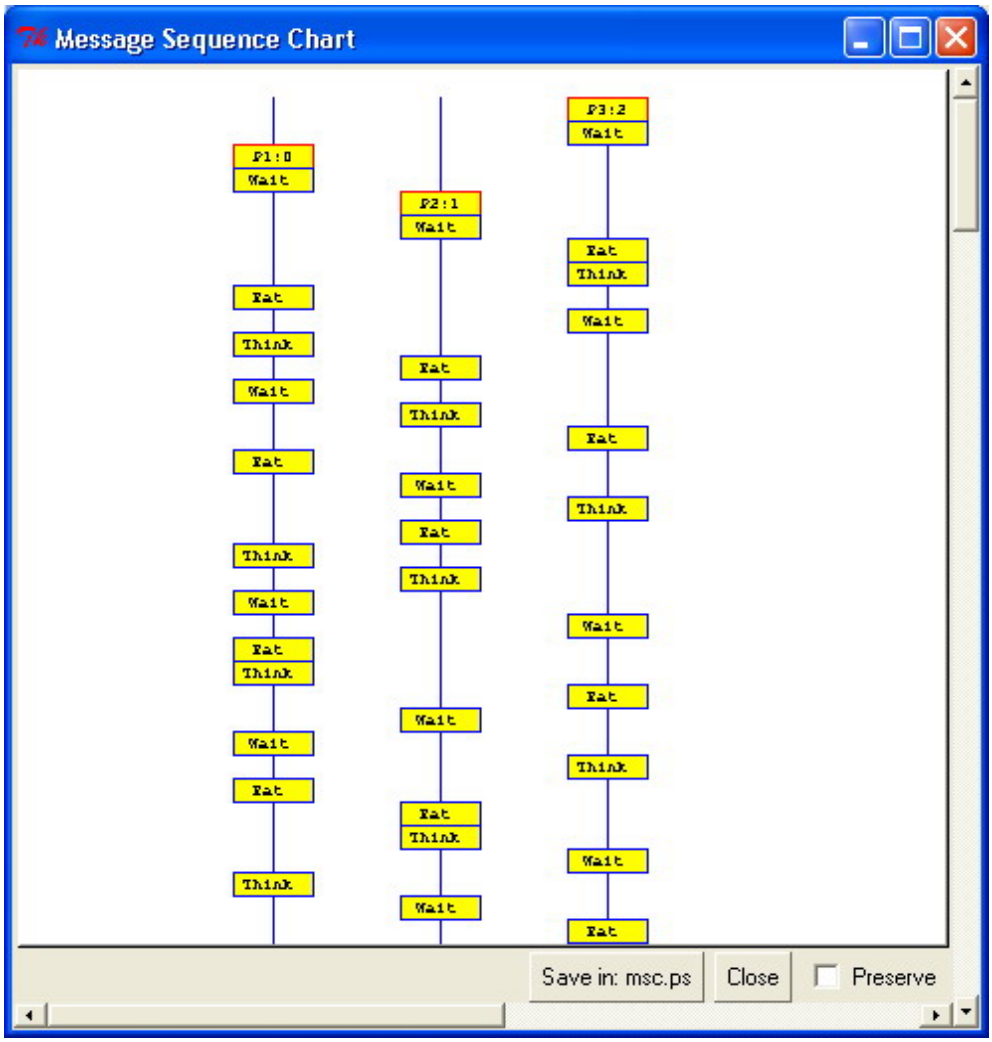
Στο παράθυρου της εξόδου μπορούμε να παρατηρήσουμε τη δημιουργία των διεργασιών και να δούμε τι τυπώνει κάθε χρονική στιγμή στο MSC το SPIN.

Εικόνα 4.6.2: Παράθυρο τιμών μεταβλητών

Το παραπάνω παράθυρο εμφανίζει τις τιμές των μεταβλητών S1,S2,S3. Παρατηρούμε ότι την χρονική στιγμή που ξεκινάει η προσομοίωση, έχουν τιμή 3. Το γεγονός αυτό εξηγήθηκε στην ανάλυση του κώδικα και σημαίνει ότι στην αρχή και οι τρεις φιλόσοφοι είναι στην κατάσταση Think. Όταν κάποιος, για παράδειγμα ο πρώτος φιλόσοφος είναι σε κατάσταση Eat το παράθυρο έχει ως εξής:

Εικόνα 4.6.3: Τιμές Μεταβλητών

Οι τιμές των μεταβλητών αυτών αλλάζουν συνεχώς, μεταξύ των τιμών 1-3, όσο διαρκεί η προσομοίωση του προγράμματός μας, που στη συγκεκριμένη περίπτωση δεν τερματίζει ποτέ.Αυτό είναι ένα κομμάτι από το παράθυρο MSC, μιας και όπως έχουμε προαναφέρει το πρόγραμμα δεν τερματίζει, οπότε δεν μπορούμε να το έχουμε όλο. Από το ενδεικτικό αυτό κομμάτι μπορεί να φανεί καθαρά ο συγχρονισμός μεταξύ των φιλοσόφων, δηλαδή αποφεύγεται η περίπτωση όλοι οι φιλόσοφοι να βρίσκονται σε κατάσταση wait.

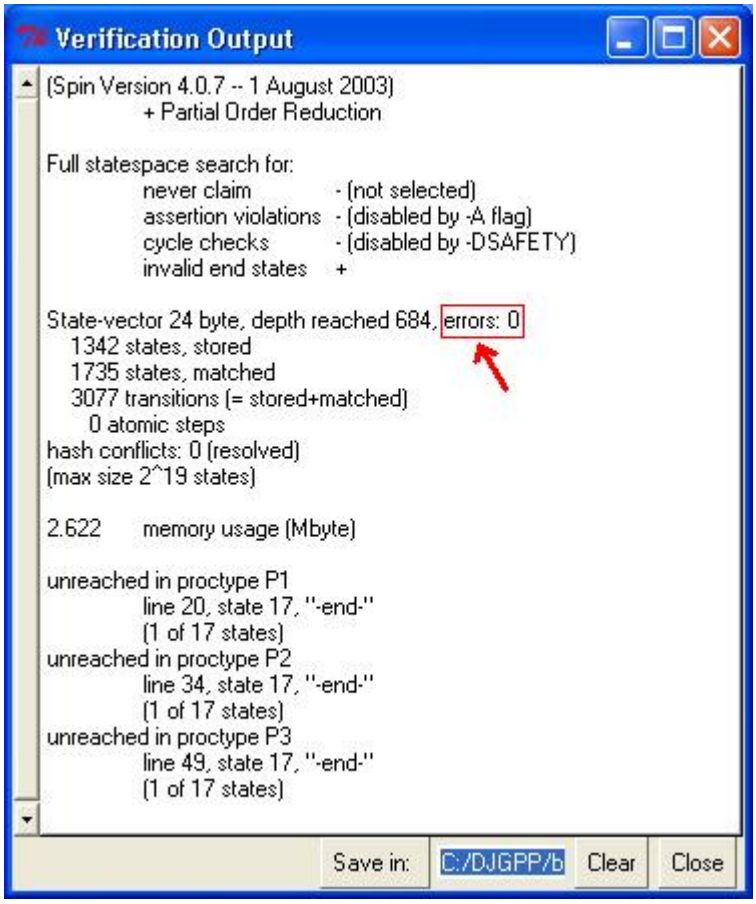


Εικόνα 4.6.4: Αποτελέσματα διαγράμματος ακολουθίας

Μετά από το βήμα της προσομοίωσης σειρά έχει η επαλήθευση του μοντέλου από τα αποτελέσματα της οποίας μπορούμε να βγάλουμε χρήσιμα συμπεράσματα, όσον αφορά την υλοποίηση του προγράμματός μας καθώς και για το αν έγινε χωρίς προβλήματα η προσομοίωση.

4.6.2 Επαλήθευση του προβλήματος

Όπως και στην προσομοίωση, έτσι και στην επαλήθευση χρησιμοποιούμε τις εξ ορισμού επιλογές. Το αποτέλεσμα φαίνεται παρακάτω :



Εικόνα 4.6.5: Αποτελέσματα Επαλήθευσης (Β)

Τα αποτελέσματα της επαλήθευσης έχουν εξηγηθεί σε προηγούμενο κεφάλαιο, οπότε αυτό που μας ενδιαφέρει είναι αν βγάζει ή όχι λάθος το SPIN. Βλέπουμε ότι σε αυτή την περίπτωση το SPIN δεν βγάζει λάθος (errors 0). Το γεγονός αυτό σημαίνει ότι δεν υπάρχει αδιέξοδο (deadlock) και επομένως η υλοποίηση του προγράμματος μας δεν έχει κάποιο πρόβλημα.

Συμπεράσματα-Επίλογος της εργασίας

5.1 Γενική ιδέα και αποτελέσματα της εργασίας

Η γενική ιδέα της εργασίας αυτής είναι η εξοικείωση στον μεγάλο θέμα που λέγεται έλεγχος λογισμικού. Ο έλεγχος αναμφισβήτητα αποτελεί και πρέπει να αποτελεί αναπόσπαστο κομμάτι κάθε διαδικασίας κατασκευής λογισμικού. Ο έλεγχος μοντέλων είναι ακριβώς συνώνυμο του έλεγχου του λογισμικού. Το λογισμικό μοντελοποιείται με τεχνικές που αναφέρθηκαν στην εργασία αυτή και στη συνέχεια επεξεργάζεται προσπαθώντας ο προγραμματιστής να μπορέσει να βρει τυχόν λάθη ή ακόμη καλύτερα να επιβεβαιωθεί ότι δεν υπάρχουν λάθη. Έτσι στην συγκεκριμένη περίπτωση, έχοντας επιλέξει το εργαλείο με το οποίο θα γίνει ο έλεγχος περιγράφονται οι δυνατότητες αυτού του εργαλείου εφαρμόζοντάς το σε ενδεικτικά παραδείγματα.

Περνώντας στα κεφάλαια που παρουσιάζονται στην εργασία αυτή, αρχικά γίνεται μια εισαγωγή στον συγχρονισμό, εισαγωγή η οποία και θα βοηθήσει τον αναγνώστη να κατανοήσει την περαιτέρω εφαρμογή του εργαλείου σε ένα πρόβλημα συγχρονισμού αντικειμενοστρεφούς λογισμικού. Στη συνέχεια ουσιαστικά μπαίνουμε στην θεματική ενότητα που καλείται έλεγχος μοντέλων. Εδώ παρουσιάζονται βασικές έννοιες που αφορούν τον έλεγχο μοντέλων, γλώσσες αναπαραστάσεις των μοντέλων, γνωστά προβλήματα αλλά και εργαλεία που δίνουν αυτόματη λύση σε αυτά. Στο 3ο κεφάλαιο γίνεται η παρουσίαση του αυτόματου ελεγκτή μοντέλων Spin και η εφαρμογή αυτού σε ενδεικτικό πρόβλημα πρωτοκόλλων επικοινωνίας (το πρόβλημα των 4ων στρατιωτών). Τέλος έχουμε την εφαρμογή του εργαλείου στο γνωστό πρόβλημα των συνδαιτυμόνων φιλοσόφων πρόβλημα συγχρονισμού αντικειμενοστρεφούς λογισμικού παρουσιάζοντας τα αποτελέσματα του εργαλείου πάνω στα μοντέλα του συγκεκριμένου προβλήματος.

5.2 Συμπεράσματα - Επίλογος εργασίας

Όπως αναφέρθηκε και στην εισαγωγή της αναφοράς αυτής ο έλεγχος του λογισμικού αποτελεί βασικό κομμάτι κάθε εταιρείας προγραμματισμού η οποία θα θέλει να παράγει αξιόπιστα προϊόντα. Με την βοήθεια του ελέγχου μπορούν να αποφευχθούν μοιραίες καταστάσεις όπου τα λάθη μπορούν να κοστίσουν ακόμη και ανθρώπινες ζωές. Για να γίνουμε πιο συγκεκριμένοι και κάνοντας αναφορά στις ενότητες με τις οποίες ασχοληθήκαμε, για την πραγματοποίηση έλεγχου μοντέλων ο κάθε αναλυτής θα πρέπει να έχει γνώση των βασικών εννοιών που αφορούν την ενότητα αυτή έχοντας επίσης εξοικείωση με μαθηματικούς όρους που αφορούν την μοντελοποίηση και γενικά την αναπαράσταση προτάσεων. Στη συνέχεια και αφού πρέπει να επιλεχθεί ο τρόπος με τον οποίο θα γίνει ο έλεγχος (για παράδειγμα το εργαλείο στην περίπτωσή μας) θα πρέπει να αποκτηθεί η γνώση θεωρητική και πρακτική της λειτουργίας αυτού. Η επιλογή του εργαλείου κρίνεται ιδιαίτερα σημαντική αφού σήμερα υπάρχουν στην αγορά αρκετά εργαλεία τα οποία προσφέρουν έλεγχο σε συγκεκριμένες περιπτώσεις λογισμικού. Προχωρώντας θα πρέπει χρήστης να μοντελοποιήσει το λογισμικό που επιθυμεί με τον τρόπο όπου το εργαλείο θα απαιτεί. Κάτι τέτοιο απαιτεί επιπλέον κόπο αφού πρέπει να γίνει γνωστή η σύνταξη της γλώσσας μοντελοποίησης με την οποία θα δεχθεί ως είσοδο το εργαλείο. Μετά από την δημιουργία του μοντέλου το εργαλείο θα δώσει

αποτελέσματα από τα οποία και πρέπει να βγαίνουν τα σωστά συμπεράσματα για το μοντέλο και κατ' επέκταση για το αρχικό λογισμικό. Στην αναφορά αυτή είδαμε πως ένα εργαλείο μπορεί να δώσει λύση σε ένα μοντελοποιημένο πρόβλημα αλλά και πως αυτό μπορεί να ανιχνεύσει εάν το μοντέλο είναι σωστό ή όχι. Εξετάστηκε και η περίπτωση στην οποία έχουμε ένα πρόβλημα συγχρονισμού αντικειμενοστρεφούς λογισμικού, όπου και εδώ το Spin επιστρέφει την αντίστοιχη απάντηση στο μοντέλο που του ανατέθηκε. Επειδή η γλώσσα promela δεν υποστηρίζει αντικειμενοστρεφή μοντελοποίηση κατασκευάστηκε το μοντέλο με όσο το δυνατόν καλύτερη προσέγγιση στην πραγματική του αντικειμενοστρεφής κατασκευή εμπεριέχοντας τον μηχανισμό συγχρονισμού που χρειάζεται για την λύση του. Τέλος στο παράρτημα της αναφοράς ο αναγνώστης μπορεί να δει ένα βασικό λεξικό εννοιών που έχουν να κάνουν με την εργασία, την εγκατάσταση του εργαλείου στον υπολογιστή, όλα τα αποτελέσματα του εργαλείου μετά την εφαρμογή αυτού σε συγκεκριμένα προβλήματα, εγχειρίδιο χρήσης της promela και του εργαλείου Spin και φυσικά την βιβλιογραφία και τις πηγές από τις οποίες αντλήθηκαν όλες οι γνώσεις για την πραγματοποίηση αυτής της ερευνητικής εργασίας.

****

Οι εταιρείες λογισμικού αντιμετωπίζουν σοβαρές προκλήσεις στο να ελέγξουν τα προϊόντα τους. Οι προκλήσεις αυτές γίνονται μεγαλύτερες ανάλογα με την αύξηση της πολυπλοκότητας του λογισμικού. Το πρώτο και πιο σημαντικό πράγμα που πρέπει κανείς να κάνει (κατά την προσωπική μας άποψη) είναι να μπορέσει να αναγνωρίσει την φύση του ελέγχου και να την πάρει αρκετά σοβαρά αν θέλει να προχωρήσει παραπέρα. Προσλαμβάνοντας ικανά άτομα από τα οποία θα γίνει ο έλεγχος, βοηθώντας τα στην εκπαίδευσή τους με αυτόματα εργαλεία ελέγχου θα πρέπει να δοθεί στην συνέχεια σε αυτούς η μεγαλύτερη εμπιστοσύνη και ο τελευταίος λόγος πριν ένα προϊόν βγει στην αγορά. Η αγνόησή τους μπορεί να αποτελέσει το σημαντικότερο (και πιο ακριβό) λάθος που θα έχει κάνει μια εταιρεία. Είναι γνωστό πως σήμερα αρκετές εταιρείες βρίσκονται σε μια ανυπόμονη κατάσταση να χρηματοδοτούν καλές ερευνητικές ιδέες που θα τους δώσουν το μονοπώλιο σε σχέση με τους ανταγωνιστές της. Αλλά πάντα, ακόμη και μια ιδέα θεωρείται σωστή όταν αυτή έχει πρώτα περάσει από έναν έλεγχο. Πόσο μάλλον τα προϊόντα της!!

Παράρτημα Α

Εγκατάσταση του εργαλείο Spin

Υπάρχουν αρκετοί διαφορετικοί τρόποι με τους οποίους μπορείτε να εγκαταστησετε το SPIN και το XSpin. Το βέβαιο είναι ότι χρειάζεστε τα παρακάτω αρχεία.

- spin(εκτελέσιμο)
- xspin(wish command){προαιρετικό}
- tcl/tk{απαιτείται για το xspin}
- μεταγλωττιστή C(π.χ. gcc)
- προεπεξεργαστή cpp(π.χ. cpp.exe)

Παρακάτω παραθέτουμε δύο τρόπους. Ο πρώτος είναι χρησιμοποιώντας τις οδηγίες του επίσημου δικτυακού τόπου του spin και ο δεύτερος μέσω του περιβάλλοντος DJGPP. Ας τα δούμε όμως αναλυτικά:

Α΄ τρόπος: Μέσω του επίσημου δικτυακού τόπου του spin

1. Απόκτηση Spin

Το Spin τρέχει στα εξής λειτουργικά:Unix, Solaris, Linux, Macs και σε Windows

- Συστήματα Unix: Κατεβάστε το πιο πρόσφατο αρχείο .tar, το γραφικό περιβάλλον Xspin, το έγγραφο τεκμηρίωσης και παραδείγματα από το http://spinroot.com/spin/Src/index.html Συνεχίστε στο 2ο βήμα.
- Προσωπικοί υπολογιστές (PCs) (Windows95/98/2000/NT/XP): Κατεβάστε το πιο πρόσφατο αρχείο pc_spin*.zip ,το γραφικό περιβάλλον Xspin, και μερικά παραδείγματα από το http://spinroot.com/spin/Src/index.html Συνεχίστε στο 2ο βήμα.

2. Εγκατάσταση του Spin

- Συστήματα Unix
- Προσωπικοί υπολογιστές (PCs) (Windows95/98/2000/NT/XP):

Εγκατάσταση σε unix

Τοποθετήστε το αρχείο *.tar.gz σε έναν κενό φάκελο. Αν έχετε τη στανταρ έκδοση του UNIX, αποσυμπιέστε το αρχείο, και έπειτα μεταγλωττίστε το εκτελέσιμο.Για παράδειγμα:

```
gunzip *.tar.gz
tar -xf *.tar
cd Src*
make -f make_unix
```

Σε συστήματα Unix το Spin υποθέτει ότι ο C προεπεξεργαστής cpp είναι αποθηκευμένος στο αρχείο "/lib/cpp". Σε μερικά συστήματα όμως αυτό είναι διαφορετικό.

Ενσωμάτωση του Xspin (Unix)

Το Xspin είναι προαιρετικό αλλά συνιστώμενο. Μπορείτε να το αποκτήσετε από τη διεύθυνση http://spinroot.com/spin/Src/index.html.

Το Xspin είναι συμβατό με τις εξής εκδόσεις:

```
Tk version 4.2 - Tcl version 7.6
Tk version 8.3 - Tcl version 8.3
Tk version 8.4 - Tcl version 8.4
```

Τυπώνει τον αριθμό έκδοσης του Spin και του Tcl/Tk όταν αρχίζει.

Εγκατάσταση του Xspin στο Unix:

- Μετονομάστε το xspin*.tcl σε ένα πιο βολικό όνομα(όπως xspin) και ακολουθήστε τις οδηγίες που βρίσκονται στο πάνω μέρος του αρχείου xspin.tcl file.
- Αντιγράψτε το αρχείο στον υποκατάλογο που βρίσκεται στο μονοπάτι αναζήτησης (search path) και μετατρέψτε το σε εκτελέσιμο, για παράδειγμα:

  - cp xspin.tcl /usr/local/bin/xspin
  - chmod +x /usr/local/bin/xspin

Εγκατάσταση σε λειτουργικό windows

Κατεβάστε το πιο πρόσφατο pc_spin*.zip file από http://spinroot.com/spin/Src/index.html. Αποσυμπιέστε τα αρχεία και τοποθετήστε το spin*.exe στον υποκατάλογο που είναι στο μονοπάτι αναζήτησης (π.χ. c:/cygwin/bin/, or c:\apps\spin\). Μπορείτε να βρείτε το μονοπάτι αυτό, πληκτρολογώντας 'set' στο κέλυφος εντολών του MS-DOS. Επίσης χρειάζεστε έναν μεταγλωττιστή C. Μπορείτε να τον αποκτήσετε από: http://spinroot.com/spin/Man/README.html#S3

Ενσωμάτωση Xspin (PC)

Μπορείτε να το αποκτήσετε από τη διεύθυνση: http://spinroot.com/spin/Src/index.html. Αντιγράψτε το αρχείο xspin*.tcl στον ίδιο φάκελο στον οποίο σχεδιάζετε να εργαστείτε, ή δημιουργήστε μια συντόμευση στην επιφάνεια εργασίας.

3. GCC(μεταγλωττιστής C)

Στο Unix έχετε το gcc, ή κάποιο παρόμοιο εγκατεστημένο. Σε windows χρειάζεστε ή ένα αντίγραφο του Visual Studio (για την εντολή cl), ή απλά τα εκτελέσιμα αρχεία: gcc.exe και cpp.exe μέσα στο μονοπάτι αναζήτησης ,μαζί με όλες τις στανταρ βιβλιοθήκες και αρχεία κεφαλίδας της C. Μπορείτε να αποκτήσετε εκδόσεις αυτών των αρχείων με το εργαλείο cygwin, το οποίο είναι διαθέσιμο από τη διεύθυνση http://www.cygwin.com/. Επίσης μπορείτε να συμβουλευτείτε και το http://spinroot.com/spin/Bin/index.html.

- Tcl/Tk Wish

Για να «τρέξει» το xspin χρειάζεστε μια έκδοση του Tcl/Tk. Το τελευταίο είναι γραμένο από τον John Ousterhout (john.ousterhout@eng.sun.com). Χρειάζεστε μια έκδοση του Tcl/Tk. Διαθέσιμη από τη διεύθυνση:
http://www.tcl.tk.

Β΄ τρόπος: Μέσω του περιβάλλοντος DJGPP

Το περιβάλλον αυτό χρησιμοποιήσαμε στην εργασία μας, μιας και κατα την προσωπική μας άποψη είναι πιο εύχρηστο. Η διαφορά με τον προηγούμενο τρόπο, έγκειται μόνο στην απόκτηση του μεταγλωττιστή και προεπεξεργαστή. Το DJGGP ειναι ένα περιβάλλον για τη δημιουργία 32-bit λογισμικού σε C/C++. Η επίσημη σελίδα στο διαδίκτυο είναι http://www.delorie.com/djgpp από όπου μπορεί κανείς να αποκτήσει το περιβάλλον αυτό.

Απαραιτητα αρχεία

- djdev203.zip
- gcc*b.zip
- bnu*b.zip
- gpp*b.zip(για c++)

Εγκατάσταση

1. Δημιουργήστε το φάκελο C:\DJGPP.

2. Αποσυμπιέστε όλα τα αρχεία *.zip, για παράδειγμα:

pkunzip -d djdev203

ή

unzip32 djdev203

3. Μετά την αποσυμπίεση, πρέπει να τοποθετήσετε τη μεταβλήτη περιβάλλοντος του DJGPP να δείχνει στο αρχείο DJGPP.ENV. Αυτό μπορείτε να το κάνετε με τον εξής απλό τρόπο. Επεξεργαστείτε το αρχείο autoexec.bat και προσθέστε τις δύο επόμενες γραμμές.

   set DJGPP=C:\DJGPP\DJGPP.ENV
   set PATH=C:\DJGPP\BIN;%PATH%

4. Κάντε επανεκκίνηση.

Μετά από αυτά τα σύντομα βήματα το spin είναι έτοιμο να δουλέψει. Δημιουργήστε μια συντόμευση στην επιφάνεια εργασίας για το XSpin και ξεκινήστε την προσομοίωση.

Παράρτημα Β

Αποτελέσματα των 4ων στρατιωτών και των συνδαιτυμόνων φιλοσόφων πριν και μετά τον έλεγχο με το εργαλείο

Στο παράρτημα αυτό περιγράφονται τα αρχεία εισόδου στο XSpin και τα αρχεία εξόδου που παράγονται από το εργαλείο όσον αφορά για το πρόβλημα των 4ων στρατιωτών ,που παρουσιάστηκε στο 3ο κεφάλαιο, αλλά και όλα τα αρχεία που χρησιμοποιούνται και παράγονται στην περίπτωση των δειπνούντων φιλοσόφων ,που παρουσιάστηκε στο 4ο κεφάλαιο.

Αρχεία «εισόδου» για το παράδειγμα των 4ων στρατιωτών

Soldiers.prom

(Το αρχείο-μοντέλο που δημιουργήθηκε στην γλώσσα Promela)

```
#define outoftime time > 60
#define N 4
#define MSCTIME     printf("MSC: %d\n", time)
#define select_soldier(x) \
if                        \
:: here[0] -> x=0         \
:: here[1] -> x=1         \
:: here[2] -> x=2         \
:: here[3] -> x=3         \
fi ;                      \
here[x] = 0

#define IF          if ::
#define FI          :: else fi
#define all_gone    (!here[0] && !here[1] && !here[2] && !here[3])
#define max(x,y)    ((x>y) -> x : y)
#define all_here    (here[0] && here[1] && here[2] && here[3])
#define soldier     byte

chan unsafe_to_safe = [0] of {soldier, soldier} ;
chan safe_to_unsafe = [0] of {soldier} ;
chan stopwatch      = [0] of {soldier} ;
byte time ;

proctype Unsafe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  here[0] = 1 ; here[1] = 1 ; here[2] = 1 ; here[3] = 1 ;
  do
  ::  select_soldier(s1) ;
      select_soldier(s2) ;
      unsafe_to_safe ! s1, s2 ;
      IF all_gone -> break FI ;
```

```
      safe_to_unsafe ? s1 ;
      here[s1] = 1 ;
      stopwatch ! s1 ;
  od
}

proctype Safe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  do
  ::  unsafe_to_safe ? s1, s2 ;
      here[s1] = 1 ;
      here[s2] = 1 ;
      stopwatch ! max(s1, s2) ;
      IF all_here -> break FI ;
      select_soldier(s1) ;
      safe_to_unsafe ! s1
  od;
}

proctype Timer()
{
end:
  do
  ::  stopwatch ? 0 -> atomic { time=time+5  ; MSCTIME }
  ::  stopwatch ? 1 -> atomic { time=time+10 ; MSCTIME }
  ::  stopwatch ? 2 -> atomic { time=time+20 ; MSCTIME }
  ::  stopwatch ? 3 -> atomic { time=time+25 ; MSCTIME }
  od;
}

init {
 atomic { run Unsafe() ; run Safe() ; run Timer() ; }
}
```

Soldiers.ltl

(Το αρχείο το οποίο δημιουργήθηκε για τον never-claim ισχυρισμό σε σύνταξη LTL)

```
(!<>(outoftime))
```

Never_claim.nvr

(Το αρχείο αυτό δημιουργήθηκε για την περίπτωση όπου θα απαιτήσουμε το εργαλείο να μας βρει μια λύση λαμβάνοντας υπ'όψιν τον χρονικό περιορισμό time=60. Το αρχείο αυτό παράγεται με την επιλογή generate στην επιλογή LTL Manager αφού πρώτα έχουμε ορίσει τον τύπο του LTL. Με την δημιουργία του αρχείου αυτού, μπορούμε να κάνουμε απευθείας την επαλήθευση επιλέγοντας στις παραμέτρους της επαλήθευσης την επιλογή [Add Never Claim File])

```
/*
       * Formula As Typed: (!<>(outoftime))
       */
```

```
never {    /* ((!<>(outoftime))) */
accept_init:
T0_init:
      if
      :: (! ((outoftime))) -> goto T0_init
      fi;
}
```

Αρχεία «εξόδου» για το παράδειγμα των 4ων στρατιωτών

Var_soldiers.out

(Αποτελεσματα των τιμών μεταβλητών που χρησιμοποιούνται στο πρόβλημα χωρίς τον χρονικό περιορισμό, Data Values)

```
Safe(2):here[0]  =  1
Safe(2):here[1]  =  1
Safe(2):here[2]  =  1
Safe(2):here[3]  =  1
Safe(2):s1  =  1
Safe(2):s2  =  0
Unsafe(1):here[0]  =  0
Unsafe(1):here[1]  =  0
Unsafe(1):here[2]  =  0
Unsafe(1):here[3]  =  0
Unsafe(1):s1  =  1
Unsafe(1):s2  =  0
time  =  95
```

soldiers.PROM.out

(Αποτελέσματα της επαλήθευσης των 4ων στρατιωτών χωρίς τον χρονικό περιορισμό , Verification Output)

```
warning: never claim + accept labels requires -a flag to fully verify
hint: this search is more efficient if pan.c is compiled -DSAFETY
warning: for p.o. reduction to be valid the never claim must be stutter-
invariant
(never claims generated from LTL formulae are stutter-invariant)
(Spin Version 4.0.7 -- 1 August 2003)
      + Partial Order Reduction

Full statespace search for:
      never claim             +
      assertion violations    - (disabled by -A flag)
      acceptance   cycles     - (not selected)
      invalid end states      - (disabled by never claim)

State-vector 60 byte, depth reached 85, errors: 0
    3379 states, stored
     662 states, matched
    4041 transitions (= stored+matched)
```

```
       2 atomic steps
hash conflicts: 2 (resolved)
(max size 2^19 states)

2.724       memory usage (Mbyte)

unreached in proctype Unsafe
      (0 of 39 states)
unreached in proctype Safe
      (0 of 25 states)
unreached in proctype Timer
      line 71, state 20, "-end-"
      (1 of 20 states)
unreached in proctype :init:
      (0 of 5 states)
```

soldiers-60.PROM.out

(Αποτελέσματα της επαλήθευσης των 4ων στρατιωτών με τον χρονικό περιορισμό , Verification Output)

```
warning: for p.o. reduction to be valid the never claim must be stutter-
invariant
(never claims generated from LTL formulae are stutter-invariant)
pan: acceptance cycle (at depth 85)
pan: wrote pan_in.trail
(Spin Version 4.0.7 -- 1 August 2003)
Warning: Search not completed
      + Partial Order Reduction

Full statespace search for:
      never claim             +
      assertion violations    + (if within scope of claim)
      acceptance   cycles     + (fairness disabled)
      invalid end states      - (disabled by never claim)

State-vector 60 byte, depth reached 85, errors: 1
     283 states, stored (530 visited)
     299 states, matched
     829 transitions (= visited+matched)
       2 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.622       memory usage (Mbyte)
```

soldiers.PROM.ltl

(Αποτελέσματα του ισχυρισμού μετά την επαλήθευση με την LTL formulae με τον χρονικό περιορισμό, Linear Temporal Logic)

```
#define outoftime time>60

      /*
```

```
        * Formula As Typed: (!<>(outoftime))
        */

never {    /* ((!<>(outoftime))) */
accept_init:
T0_init:
      if
      :: (! ((outoftime))) -> goto T0_init
      fi;
}

#ifdef NOTES

#endif
#ifdef RESULT
warning: for p.o. reduction to be valid the never claim must be stutter-
invariant
(never claims generated from LTL formulae are stutter-invariant)
pan: acceptance cycle (at depth 85)
pan: wrote pan_in.trail
(Spin Version 4.0.7 -- 1 August 2003)
Warning: Search not completed
      + Partial Order Reduction

Full statespace search for:
      never claim             +
      assertion violations    + (if within scope of claim)
      acceptance   cycles     + (fairness disabled)
      invalid end states      - (disabled by never claim)

State-vector 60 byte, depth reached 85, errors: 1
     283 states, stored (530 visited)
     299 states, matched
     829 transitions (= visited+matched)
       2 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.622       memory usage (Mbyte)


#endif
```

(Αποτελέσματα των τιμών των μεταβλητών μετά την καθοδηγημένη προσομοίωση με τον χρονικό περιορισμό, →time=60 , Data Values)

varguided.out

```
Safe(3):here[0]  =  1
Safe(3):here[1]  =  1
Safe(3):here[2]  =  1
Safe(3):here[3]  =  1
Safe(3):s1  =  0
Safe(3):s2  =  1
Unsafe(2):here[0]  =  0
```

```
Unsafe(2):here[1]  =  0
Unsafe(2):here[2]  =  0
Unsafe(2):here[3]  =  0
Unsafe(2):s1  =  0
Unsafe(2):s2  =  1
time  =  60
```

Αρχεία «εισόδου» για το παράδειγμα των Συνδαιτυμόνων Φιλόσοφων

(Παρακάτω παρουσιάζεται το πρώτο αρχείο μοντελοποίησης του προβλήματος το οποίο και παρουσιάζει αδιέξοδο Philldeadlock.prom)

```
bit fork[3];

     mtype = {thinking, waiting, eating};
     proctype philosopher(byte left, right) {
       mtype state = thinking;
       do
       ::  ((state==thinking)  &&  (fork[left]==0))  ->  atomic{fork[left]=1;
state=waiting;}
       ::  ((state==waiting)  &&  (fork[right]==0))  ->  atomic{fork[right]=1;
state=eating;}
       :: (state==eating) ->
          if
          :: skip;
          :: atomic{state=thinking; fork[right]=0; fork[left]=0;}
          fi
       od;
     }

     init {
       fork[0]=0;
       fork[1]=0;
       fork[2]=0;
       run philosopher(0, 1);
       run philosopher(1, 2);
       run philosopher(2, 0);
     }
```

(Παρακάτω παρουσιάζεται το δεύτερο αρχείο μοντελοποίησης του προβλήματος το οποίο δεν παρουσιάζει αδιέξοδο Philldeadlock.prom)

```
nodeadlock.prom

#define Think1    printf( "MSC: Think \n")
#define Eat1      printf( "MSC: Eat \n")
#define Wait1     printf( "MSC: Wait \n" )
mtype={Think,Wait,Eat}

byte S1=Think;
byte S2=Think;
byte S3=Think;

active proctype P1 () {
do
     ::S3=Think ->S1=Wait;Wait1;
```

```
                    do
                        :: S2=Think->progress: S1=Eat;Eat1;
printf("P1 eating\n");
S1=Think;Think1;
break
      od;
od
}

active proctype P2 () {
do
:: S1!=Eat->
      S2=Wait;Wait1;
      do
      :: S3!=Eat->
      progress: S2=Eat;Eat1;
printf("P2 eating\n");
S2=Think;Think1;
break
      od;
od
}


active proctype P3 () {
do
:: S1=Think->
      S3=Wait;Wait1;
      do
      :: S2!=Eat->
      progress:S3=Eat;Eat1;
printf("P3 eating\n");
S3=Think;Think1;
break
      od;
od
}
```

sim.out (Στην περίπτωση αυτού του αποτελέσματος της προσομοίωσης βάζουμε ένα ενδεικτικό κομμάτι αυτής μιας και δεν έχουμε εισάγει συνθήκη τερματισμού του προβλήματος)

```
0:      proc  - (:root:) creates proc  0 (P1)
 0:     proc  - (:root:) creates proc  1 (P2)
 0:     proc  - (:root:) creates proc  2 (P3)
 1:     proc  1 (P2) line  23 "pan_in" (state 14)    [((S1!=Eat))]
 2:     proc  2 (P3) line  38 "pan_in" (state 14)    [S1 = Think]
 3:     proc  1 (P2) line  25 "pan_in" (state 2)     [S2 = Wait]
 4:     proc  2 (P3) line  40 "pan_in" (state 2)     [S3 = Wait]
MSC: Wait
 5:     proc  2 (P3) line  40 "pan_in" (state 3)     [printf('MSC: Wait \\n')]
 6:     proc  0 (P1) line  11 "pan_in" (state 14)    [S3 = Think]
 7:     proc  2 (P3) line  48 "pan_in" (state 12)    [.(goto)]
 8:     proc  2 (P3) line  41 "pan_in" (state 11)    [((S2!=Eat))]
 9:     proc  2 (P3) line  43 "pan_in" (state 5)     [S3 = Eat]
10:     proc  0 (P1) line  12 "pan_in" (state 2)     [S1 = Wait]
MSC: Wait
11:     proc  0 (P1) line  12 "pan_in" (state 3)     [printf('MSC: Wait \\n')]
MSC: Wait
12:     proc  1 (P2) line  25 "pan_in" (state 3)     [printf('MSC: Wait \\n')]
```

```
 13:     proc  0 (P1) line  19 "pan_in" (state 12)     [.(goto)]
MSC: Eat
 14:     proc  2 (P3) line  43 "pan_in" (state 6)      [printf('MSC: Eat \\n')]
 15:     proc  1 (P2) line  33 "pan_in" (state 12)     [.(goto)]
 16:     proc  0 (P1) line  13 "pan_in" (state 11)     [S2 = Think]
P3 eating
 17:     proc  2 (P3) line  44 "pan_in" (state 7)      [printf('P3 eating\\n')]
 18:     proc  2 (P3) line  45 "pan_in" (state 8)      [S3 = Think]
 19:     proc  0 (P1) line  14 "pan_in" (state 5)      [S1 = Eat]
MSC: Think
 20:     proc  2 (P3) line  45 "pan_in" (state 9)      [printf('MSC: Think \\n')]
 21:     proc  2 (P3) line  46 "pan_in" (state 10)     [goto :b5]
 22:     proc  2 (P3) line  41 "pan_in" (state 13)     [break]
 23:     proc  2 (P3) line  49 "pan_in" (state 15)     [.(goto)]
MSC: Eat
 24:     proc  0 (P1) line  14 "pan_in" (state 6)      [printf('MSC: Eat \\n')]
 25:     proc  2 (P3) line  38 "pan_in" (state 14)     [S1 = Think]
P1 eating
 26:     proc  0 (P1) line  15 "pan_in" (state 7)      [printf('P1 eating\\n')]
 27:     proc  1 (P2) line  26 "pan_in" (state 11)     [((S3!=Eat))]
 28:     proc  2 (P3) line  40 "pan_in" (state 2)      [S3 = Wait]
MSC: Wait
 29:     proc  2 (P3) line  40 "pan_in" (state 3)      [printf('MSC: Wait \\n')]
 30:     proc  1 (P2) line  28 "pan_in" (state 5)      [S2 = Eat]
 31:     proc  0 (P1) line  16 "pan_in" (state 8)      [S1 = Think]
MSC: Think
 32:     proc  0 (P1) line  16 "pan_in" (state 9)      [printf('MSC: Think \\n')]
 33:     proc  0 (P1) line  17 "pan_in" (state 10)     [goto :b1]
 34:     proc  2 (P3) line  48 "pan_in" (state 12)     [.(goto)]
 35:     proc  0 (P1) line  13 "pan_in" (state 13)     [break]
 36:     proc  0 (P1) line  20 "pan_in" (state 15)     [.(goto)]
MSC: Eat
 37:     proc  1 (P2) line  28 "pan_in" (state 6)      [printf('MSC: Eat \\n')]
 38:     proc  0 (P1) line  11 "pan_in" (state 14)     [S3 = Think]
P2 eating
 39:     proc  1 (P2) line  29 "pan_in" (state 7)      [printf('P2 eating\\n')]
 40:     proc  0 (P1) line  12 "pan_in" (state 2)      [S1 = Wait]
MSC: Wait
 41:     proc  0 (P1) line  12 "pan_in" (state 3)      [printf('MSC: Wait \\n')]
 42:     proc  1 (P2) line  30 "pan_in" (state 8)      [S2 = Think]
MSC: Think
 43:     proc  1 (P2) line  30 "pan_in" (state 9)      [printf('MSC: Think \\n')]
 44:     proc  2 (P3) line  41 "pan_in" (state 11)     [((S2!=Eat))]
 45:     proc  1 (P2) line  31 "pan_in" (state 10)     [goto :b3]
 46:     proc  1 (P2) line  26 "pan_in" (state 13)     [break]
 47:     proc  1 (P2) line  34 "pan_in" (state 15)     [.(goto)]
 48:     proc  1 (P2) line  23 "pan_in" (state 14)     [((S1!=Eat))]
 49:     proc  0 (P1) line  19 "pan_in" (state 12)     [.(goto)]
 50:     proc  2 (P3) line  43 "pan_in" (state 5)      [S3 = Eat]
 51:     proc  1 (P2) line  25 "pan_in" (state 2)      [S2 = Wait]
 52:     proc  0 (P1) line  13 "pan_in" (state 11)     [S2 = Think]
MSC: Eat
 53:     proc  2 (P3) line  43 "pan_in" state 6)       [printf('MSC: Eat \\n')]
```

Αναφορές

Για την ολοκλήρωση της έρευνας χρησιμοποιήθηκαν πηγές τόσο σε έντυπη μορφή, όσο και σε ηλεκτρονική.

Βιβλιογραφία

1. Model checking: *Edmund M. Clarke,Jr., Orna Grumberg and Doron A. Peled*
2. Concurrent programming: *Tom Axford*
3. Distributed coordination systems

Πηγές και άρθρα στο διαδίκτυο

G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, Englewood Cli.s, New Jersey, USA, 1991.

G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*

Gul A. Agha. *Actors – A Model of Concurrent Computation is Distributed Systems*. The MIT Press, Cambridge, Massachusetts, 1986.

B. Boigelot and P. Godefroid, "Model Checking in Practice: An Analysis of the ACCESS Bus Protocol Using SPIN,'' *Proc. Formal Methods Europe (FME96*), Oxford, England, *Lecture Notes in Computer Science 1,051*, pp. 465-478. Springer-Verlag, Mar. 1996.

L. Bouvin, "Design of Validation Models Access Protocol of the PTM Project," Report Royal Inst. Of Technology, Stockholm, Sweden, Aug. 1991.

T. Cattel, "Modeling and Verification of a Multiprocessor Real- Time OS Kernel," *Proc. Seventh Int'l Conf. Formal Description Techniques*, pp. 35-50, Berne, Switzerland, Oct. 1994.

M. Griffioen, "Specification and Verification of a Wireless LAN Controller Chip Using PROMELA and SPIN," Technical Report, AT&T Network Wireless Systems, The Netherlands, 1996.

J. Hajek, "Automatically Verified Data Transfer Protocols," *Proc. Fourth ICCC*, pp. 749-756, Kyoto, Aug. 1978.

G.J. Holzmann, "PAN: A Protocol Specification Analyzer," Technical Report TM81-11271-5, AT&T Bell Laboratories, Mar. 1981.

C-T. Chou, and D. Peled, "Verifying a Model-Checking Algorithm,"*Proc. Tools and Algorithms for the Construction and Analysisof Systems (TACAS96),* Passau, Germany, *Lecture Notes in Computer Science 1,055*, pp. 241-257. Springer-Verlag, Mar. 1996.

A. Cimatti, A. Giunchiglia, G. Mongardi, D. Romano, F. Torielli,and P. Traverso, "Model Checking Safety Critical Software with SPIN: An Application to a Railway Interlocking System," *Proc. Third SPIN Workshop*, R. Langerak, ed., Twente Univ., The Netherlands, Apr. 1997.

J.C. Corbett, "Evaluating Deadlock Detection Methods for Concurrent Software," *IEEE Trans. Software Eng.*, vol. 22, no. 3, pp. 161-180, Mar. 1996.

C. Courcoubetis, M.Y. Vardi, P. Wolper, M. Yannakakis, "Memory Efficient Algorithms for the Verification of Temporal Properties," *Formal Methods in Systems Design*, vol. I, pp. 275-288, 1992. Englewood Cliffs, N.J.: Prentice Hall, 1991.

G.J. Holzmann, "Protocol Design: Redefining The State of the Art," *IEEE Software*, pp. 17-22, Jan. 1992.

G.J. Holzmann, P. Godefroid, and D. Pirottin, "Coverage Preserving Reduction Strategies for Reachabily Analysis," *Proc. IFIP/WG6.1 Symp. Protocol Specification, Testing, and Verification (PSTV92),* pp. 349-364, Orlando, Fla., North-Holland, June 1992.

G.J. Holzmann, "Design and Validation of Protocols: A Tutorial," *Computer Networks and ISDN Systems*, vol. 25, no. 9, pp. 981-1,017, 1993.

G.J. Holzmann, "The Theory and Practice of a Formal Method: NewCoRe," *Proc. 13th IFIP World Computer Congress*, pp. 35-44, Hamburg, Germany, North-Holland, Aug. 1994.

G.J. Holzmann and D. Peled, "An Improvement in Formal Verification," *Proc. Seventh FORTE Conf. Formal Description Techniques,* pp. 177-194, Bern, Switzerland, Oct. 1994.

G.J. Holzmann, "An Analysis of Bit-State Hashing," *Proc. IFIP/WG6.1 Symp. Protocol Specification, Testing, and Verification (PSTV95),* pp. 301-314, Warsaw, Poland, Chapman & Hall, June 1995.

G.J. Holzmann, D. Peled, and M. Yannakakis, "On Nested Depth- First Search," *Proc. Second SPIN Workshop*, Rutgers Univ., New Brunswick, N.J., DIMACS/32, Am. Math. Soc., Aug. 1996.

F. Gagnon, "Boulier, un validateur pour la language de spécification Gaston," PhD thesis, Univ. de Quebec, Canada, July 1995.

R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper, "Simple On-The- Fly Automatic Verification of Linear Temporal Logic," *Proc. IFIP/WG6.1 Symp. Protocol Specification, Testing, and Verification (PSTV95),* pp. 3-18, Warsaw, Poland, Chapman & Hall, June 1995.

P. Godefroid, and G.J. Holzmann, "On the Verification of Temporal Properties," *Proc. IFIP/WG6.1 Symp. Protocol Specification, Testing, and Verification (PSTV93),* pp. 109-124, Liege, Belgium, North-Holland, June 1993.

P. Godefroid, "Partial Order Methods for the Verification of Concurrent Systems," *Lecture Notes in Computer Science 1,032*. Springer-Verlag, 1996.

P. Godefroid, "Symbolic Protocol Verification with Queue BDDs," *Proc. Logic in Computer Science*, pp. 198-206, Rutgers Univ., New Brunswick, N.J., July 1996.

J.-C. Gregoire, "State Space Compression in SPIN with GETSs," *Proc. Second SPIN Workshop*, Rutgers Univ., New Brunswick, N.J., DIMACS/32, Am. Math. Soc., Aug. 1996.

Workshop, J.-Ch. Gregoire, ed., INRS Quebec, Canada, Oct. 1995.

S. Loeffler and A. Serhrouchni, "Protocol Design: From Specification to Implementation," *Proc. Fifth Open Workshop for High Speed Networks*, Paris, Mar. 1996.

IEEE Std. 802-2-1985, ISO DIS 8802/2," IEEE Standards for Local Area Networks: Logical Link Control, Published by the IEEE Standards Board, 345 E. 47th Street, New York, NY 10017, USA, 111 pp., ISBN 471-82748-7, 1984. Revised as 802-2-1989 in Aug. 1989.

K.L. McMillan, *Symbolic Model Checking*. Boston: Kluwer Academic, 1993.

E. Najm and F. Olsen, "Reactive EFSMs, Reactive PROMELA/ RSPIN," *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS96),* pp. 349-368, Passau, Germany, *Lecture Notes In Computer Science 1,055*, Springer-Verlag, Mar. 1996.

T. Nakatani, "Verification of a Group Address Registration Protocol using PROMELA and SPIN," *Proc. Third SPIN Workshop*, R. Langerak, ed., Twente Univ., The Netherlands, Apr. 1997.

R.M. Needham and A.J. Herbert, *The Cambridge Distributed Computing*