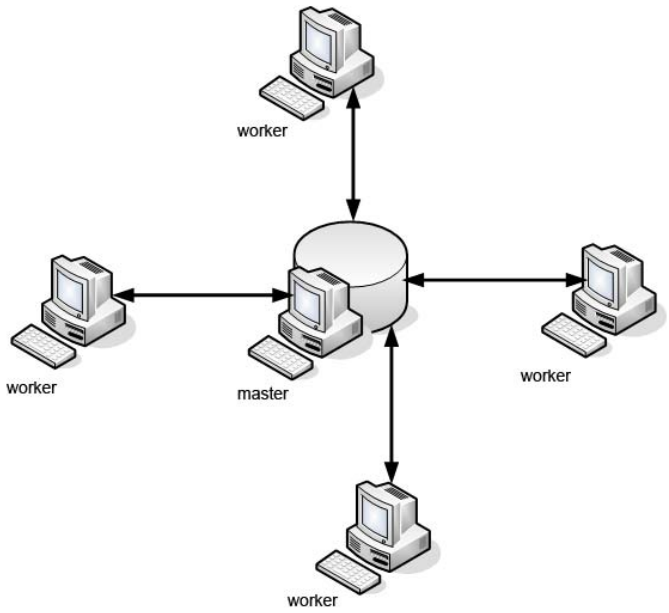
ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ & ΔΙΚΤΥΩΝ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ(Π.Μ.Σ)
«ΕΠΙΣΤΗΜΗ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑ ΥΠΟΛΟΓΙΣΤΩΝ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ»

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΑΤΡΙΒΗ

“Βελτιώσεις της μεθόδου TLA/TLC για επαλήθευση συστημάτων”

(Παραλληλοποίηση για εκτέλεση σε κατανεμημένα περιβάλλοντα)



ΓΕΩΡΓΟΥΛΗ ΚΩΝΣΤΑΝΤΙΑ

Επιβλέπων: Μούντανος Ιωάννης, Αναπληρωτής Καθηγητής

Βόλος, 2011

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ & ΔΙΚΤΥΩΝ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ(Π.Μ.Σ)
«ΕΠΙΣΤΗΜΗ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑ ΥΠΟΛΟΓΙΣΤΩΝ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ»

“Βελτιώσεις της μεθόδου TLA/TLC για επαλήθευση συστημάτων”

(Παραλληλοποίηση για εκτέλεση σε κατανεμημένα περιβάλλοντα)

Μεταπτυχιακή Εργασία η οποία υποβάλλεται
για εκπλήρωση των απαιτήσεων για το Δίπλωμα Ειδίκευσης
του Π.Μ.Σ. του τμήματος Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων
"Επιστήμη και Τεχνολογία Υπολογιστών, Τηλεπικοινωνιών και Δικτύων"

ΓΕΩΡΓΟΥΛΗ ΚΩΝΣΤΑΝΤΙΑ

Επιβλέπων: Μούντανος Ιωάννης, Αναπληρωτής Καθηγητής

Τριμελής Εξεταστική Επιτροπή :
Μούντανος Ιωάννης, Αναπληρωτής Καθηγητής
Σταμούλης Γεώργιος, Καθηγητής
Μπέλλας Νικόλαος, Αναπληρωτής Καθηγητής

Πρόλογος

Η παρούσα μεταπτυχιακή διατριβή εκπονήθηκε στα πλαίσια των προβλεπόμενων εκπαιδευτικών υποχρεώσεων του Προγράμματος Μεταπτυχιακών Σπουδών (Π.Μ.Σ) του τμήματος Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων "Επιστήμη και Τεχνολογία Υπολογιστών, Τηλεπικοινωνιών και Δικτύων".

Αντικείμενο της διατριβής είναι η παραλληλοποίηση, για εκτέλεση σε κατανεμημένα περιβάλλοντα, του αλγορίθμου που χρησιμοποιεί το TLC(Temporal Logic Checker) για την υλοποίηση της διαδικασίας του liveness model checking, που σκοπό έχει τον έλεγχο και την επαλήθευση μιας προδιαγραφής ενός συστήματος.

Σκοπός της παρούσας μεταπτυχιακής διατριβής είναι να συμβάλλει στην επιτάχυνση του χρόνου εκτέλεσης της μεθόδου του TLC για την επαλήθευση συστημάτων.

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω θερμά τον κ. Ιωάννη Μούντανο, Αναπληρωτή Καθηγητή του Τμήματος Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων για την ανάθεση της συγκεκριμένης μεταπτυχιακής διατριβής. Η συνεισφορά του στην εκπόνηση αυτής της διατριβής ήταν διττή, αφού εκτός από την πολύτιμη καθοδήγηση, προσέφερε και την αμέριστη βοήθεια του για την πραγμάτωση της κάθε στιγμή που τη χρειάστηκα.

Γεωργούλη Κωνσταντία
Φοιτήτρια Μεταπτυχιακών Σπουδών
Τμήματος Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων
Βόλος, 2011

Πίνακας περιεχομένων

Εισαγωγή

Περίληψη

Γενικά η συγγραφή μιας TLA+ προδιαγραφής μπορεί να βοηθήσει την σχεδίαση ενός συστήματος. Στην προσπάθεια να περιγράψουμε ένα σύστημα με ακρίβεια αποκαλύπτονται συχνά προβλήματα που εύκολα μπορούν να παραβλεφθούν. Αυτά τα προβλήματα είναι ευκολότερο να διορθωθούν όταν ανακαλύπτονται στην φάση της σχεδίασης και όχι αφού έχει ξεκινήσει η υλοποίηση.

Μια TLA+ προδιαγραφή μπορεί να αποτελέσει έναν σαφή και σύντομο τρόπο επικοινωνίας για την περιγραφή ενός συστήματος. Αυτή βοηθά να εξασφαλιστεί ότι οι σχεδιαστές συμφωνούν στο τι αυτοί έχουν σχεδιάσει, και παρέχει έναν πολύτιμο οδηγό για τους μηχανικούς για το ποιος υλοποιεί και ποιος τεστάρει το σύστημα. Επίσης μπορεί να βοηθήσει τους χρήστες να κατανοήσουν το σύστημα.

Μια TLA+ προδιαγραφή είναι μια μαθηματική περιγραφή στην οποία τα εργαλεία μπορούν να εφαρμοστούν για να βοηθήσουν στην εύρεση σφαλμάτων και στην απόδειξη ιδιοτήτων ενός συστήματος. Το πιο χρήσιμο εργαλείο που έχει γραφτεί έως τώρα για αυτόν τον σκοπό είναι ο TLC model checker.

Ο έλεγχος μιας προδιαγραφής στο TLC απαιτεί συνήθως αρκετό χρόνο, που εξαρτάται από τον αριθμό των καταστάσεων που παράγονται για την συγκεκριμένη προδιαγραφή και το σύστημα. Παρατηρώντας τον χρόνο που καταναλώνει το TLC για το liveness model checking στην εκτέλεση σε single workstation, βγαίνει το συμπέρασμα ότι αποτελεί το κομμάτι του αλγορίθμου στο οποίο καταναλώνεται το μεγαλύτερο ποσοστό του χρόνου εκτέλεσης.

Στόχος της διατριβής είναι η παραλληλοποίηση του τμήματος του αλγορίθμου που αφορά το liveness έλεγχο σε κατανεμημένα περιβάλλοντα έτσι ώστε να επιτυγχάνει εκμετάλλευση της αύξησης των υπολογιστικών επιδόσεων που προσφέρει ο παραλληλισμός.

Δομή κειμένου

Κεφάλαιο 1: TLA+

Στο πρώτο κεφάλαιο γίνεται μια ιστορική αναδρομή στην TLA+, παρουσιάζονται κάποιες εισαγωγικές έννοιες, παρουσιάζεται η δομή μιας TLA+ προδιαγραφής, γίνεται μια αναφορά στις χρονικές εκφράσεις της TLA+ και παρουσιάζονται ως έννοιες το weak fairness και το strong fairness που χρησιμοποιούνται στο προσδιορισμό των ιδιοτήτων liveness.

Κεφάλαιο 2: Ιδιότητες Liveness

Στο δεύτερο κεφάλαιο γίνεται μια εισαγωγή σχετικά με τις liveness ιδιότητες. Στην συνέχεια προτείνονται διάφοροι ορισμοί αναφορικά με την liveness ιδιότητα. Επιπρόσθετα παρουσιάζονται και κάποιες άλλες ιδιότητες που εμπεριέχουν το στοιχείο του liveness. Τέλος παρουσιάζεται το πότε μια προδιαγραφή θεωρείται machine closed.

Κεφάλαιο 3: TLC Model Checker

Το τρίτο κεφάλαιο ξεκινά με την παρουσίαση των κατηγοριών αλγορίθμων για model checking. Γίνεται μια εισαγωγή στο TLC και στην συνέχεια περιγράφεται ο τρόπος με τον οποίο το TLC εκτιμά γενικά τις εκφράσεις (χρονικές και μη), ο τρόπος με τον οποίο υπολογίζει τις καταστάσεις καθώς και το πως ελέγχει τις ιδιότητες. Στην συνέχεια παρουσιάζονται οι βασικές δομές δεδομένων που χρησιμοποιεί το TLC για την υλοποίηση των αλγορίθμων της και τέλος επεξηγείται ο αλγόριθμος που ακολουθείται στο TLC για το safety model checking αλλά και για το liveness model checking.

Κεφάλαιο 4: Liveness Model Checking σε επίπεδο δικτύου

Στο τέταρτο κεφάλαιο παρουσιάζεται η παράλληλη υλοποίηση που προϋπήρχε στο TLC για κατανεμημένα περιβάλλοντα. Αναφέρεται σύντομα η αρχιτεκτονική RMI που χρησιμοποιήθηκε για την παράλληλη υλοποίηση του liveness model checking σε κατανεμημένα περιβάλλοντα και παρουσιάζεται η παραλληλοποίηση του Liveness Model Checking (βήματα αλγορίθμου) που υλοποιήθηκε στα πλαίσια αυτής της εργασίας. Τέλος περιγράφεται ο τρόπος εκτέλεσης του TLCServer και του TLCWorker.

Κεφάλαιο 5: Μετρήσεις

Στο πέμπτο κεφάλαιο παρουσιάζονται τα αποτελέσματα των μετρήσεων που προέκυψαν κατά την εκτέλεση μιας προδιαγραφής μικρής κλίμακας (5000 καταστάσεις) και στην συνέχεια μιας προδιαγραφής μεγάλης κλίμακας (30000 καταστάσεις). Στην

τελική ενότητα αυτού του κεφαλαίου γίνεται παρουσίαση των συμπερασμάτων που έχουν εξαχθεί βάσει των μετρήσεων.

Παράρτημα Α:

Στο παράρτημα αυτό περιγράφεται αρχικά ο τρόπος εκτέλεσης του TLC. Περιγράφονται αναλυτικά οι προδιαγραφές που χρησιμοποιήθηκαν στις μετρήσεις και παρουσιάζεται η έξοδος που παράγεται από την εκτέλεση αυτών των προδιαγραφών αλλά και παραλλαγών αυτών. Τέλος ακολουθεί αναλυτική περιγραφή της εκτέλεσης αυτών των προδιαγραφών στο TLC.

Παράρτημα Β:

Στο παράρτημα αυτό παρουσιάζονται κάποιες βασικές πληροφορίες για το σημασιολογικό tableau.

Κεφάλαιο 1: TLA+

Περίληψη κεφαλαίου

Στο κεφάλαιο αυτό γίνεται μια ιστορική αναδρομή στην TLA+, παρουσιάζονται κάποιες εισαγωγικές έννοιες, παρουσιάζεται η δομή μιας TLA+ προδιαγραφής, γίνεται μια αναφορά στις χρονικές εκφράσεις της TLA+ και παρουσιάζονται ως έννοιες το weak fairness και το strong fairness που χρησιμοποιούνται στο προσδιορισμό των ιδιοτήτων liveness. Σκοπός του κεφαλαίου είναι να γίνουν σαφείς οι βασικές έννοιες της TLA+ για την μετέπειτα κατανόηση της λειτουργίας και του αλγορίθμου του TLC.

1.1 Η ιστορία της TLA+

Ο προσδιορισμός ενός συστήματος περιγράφει τις επιτρεπτές συμπεριφορές, τί πιθανόν κάνει το σύστημα στην διάρκεια μιας εκτέλεσης. Το 1977, ο Amir Pnueli εισήγαγε τη χρήση της χρονικής λογικής (Temporal Logic) για την περιγραφή των συμπεριφορών συστήματος. Η χρονική λογική του Pnueli μπορεί από την μια να ήταν ιδανική για την περιγραφή κάποιων ιδιοτήτων των συστημάτων, αλλά από την άλλη δεν ήταν και τόσο εύχρηστη για όλους. Επομένως, αυτή συνδυαζόταν συνήθως με έναν πιο παραδοσιακό τρόπο περιγραφής συστημάτων.

Στα τέλη του 1980, επινοήθηκε η TLA[1] από τον Leslie Lamport, η χρονική λογική ενεργειών (Temporal Logic of Actions), μια απλή παραλλαγή της αρχικής λογικής του Pnueli. Το μεγαλύτερο κομμάτι μιας TLA προδιαγραφής αποτελείται από «κοινά» μαθηματικά, δηλαδή μαθηματικά που δεν εμπεριέχουν χρονική λογική. Η χρονική λογική παίζει ένα σημαντικό ρόλο μόνο στην περιγραφή των ιδιοτήτων τις οποίες είναι κατάλληλη να περιγράψει. Η TLA επίσης παρέχει έναν καλό τρόπο να τυποποιηθεί ο τρόπος απόδειξης ιδιοτήτων των συστημάτων ο οποίος έχει αποδειχθεί ότι είναι περισσότερο αποδοτικός στην πράξη, ένας τρόπος γνωστός ως συλλογισμός βάσει ισχυρισμών(assertional reasoning).

Η χρονική λογική υποθέτει την ύπαρξη μιας βασικής λογικής για την έκφραση «κοινών» μαθηματικών εννοιών. Υπάρχουν πολλοί τρόποι για την διατύπωση εννοιών των «κοινών» μαθηματικών. Οι περισσότεροι επιστήμονες υπολογιστών προτιμούν έναν τρόπο που να μοιάζει με την αγαπημένη τους γλώσσα προγραμματισμού. Ο Leslie Lamport επέλεξε τελικά αντί αυτού του τρόπου που οι περισσότεροι μαθηματικοί προτιμούν, έναν που οι επιστήμονες της επιστήμης της λογικής αποκαλούν λογική πρώτης τάξης και θεωρία συνόλων.

Η TLA παρέχει ένα μαθηματικό θεμέλιο για την περιγραφή συστημάτων. Για την συγγραφή προδιαγραφών, χρειαζόταν μια πλήρη γλώσσα χτισμένη πάνω σε αυτό το θεμέλιο. Αρχικά ο Leslie Lamport θεώρησε ότι αυτή η γλώσσα έπρεπε να είναι κάποιο είδος αφηρημένων γλωσσών προγραμματισμού η σημασιολογία των οποίων θα ήταν

βασισμένη στην TLA. Δεν γνώριζε τι είδους δομές γλώσσας προγραμματισμού θα ήταν οι καλύτερες, έτσι αποφάσισε να ξεκινήσει γράφοντας άμεσα προδιαγραφές σε TLA. Σκόπευε να εισάγει προγραμματιστικές δομές όταν θα τις χρειαζόταν. Για έκπληξη του, ανακάλυψε ότι δεν τις χρειαζόταν. Αυτό που χρειάστηκε ήταν μια εύρωστη γλώσσα για την γραφή μαθηματικών.

Η προδιαγραφή ενός πραγματικού συστήματος μπορεί να είναι δεκάδες ή ακόμη και εκατοντάδες σελίδες μεγάλη. Οι μαθηματικοί γνώριζαν πώς να γράφουν εκφράσεις 20-γραμμών, όχι εκφράσεις 20-σελίδων. Έτσι ο Leslie Lamport εισήγαγε τη σημειογραφία για την γραφή μεγάλων εκφράσεων. Αυτό που πήρε από τις γλώσσες προγραμματισμού ήταν ιδέες για την διαμόρφωση μεγάλων προδιαγραφών.

Η γλώσσα που παράχθηκε από τον Leslie Lamport αποκαλείται TLA+. Τελειοποίησε την TLA+ κατά την διάρκεια γραφής προδιαγραφών ανόμοιων συστημάτων. Βρήκε την TLA+ να είναι αρκετά καλή για τον προσδιορισμό μια ευρείας κλάσης συστημάτων, από προγραμματιστικές διεπαφές εφαρμογών (APIs) μέχρι κατανεμημένα συστήματα. Η TLA+ μπορεί να χρησιμοποιηθεί για την γραφή μιας ακριβούς, τυπικής περιγραφής σχεδόν οποιουδήποτε είδους διακριτού συστήματος. Είναι ιδιαίτερα κατάλληλη για την περιγραφή ασύγχρονων συστημάτων, δηλαδή συστημάτων με components που δεν λειτουργούν με αυστηρά συγχρονισμένο βήμα.

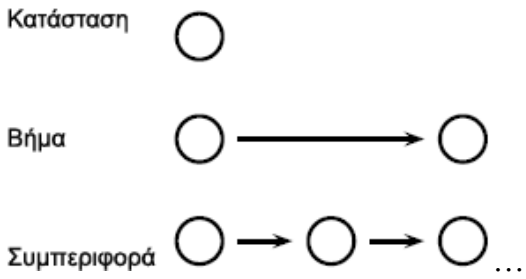
1.2 Εισαγωγικές έννοιες

Η τεχνική ελέγχου μοντέλου (model checking)[2] είναι μια αυτοματοποιημένη τεχνική για την επαλήθευση ιδιοτήτων ορθότητας σε συστήματα πεπερασμένων καταστάσεων. Το model-checking χρησιμοποιεί **μοντέλα** (**models**) τα οποία είναι απεικονίσεις της συμπεριφοράς ενός συστήματος μέσω των οποίων ελέγχεται το σύστημα. Μια **κατάσταση**[3] είναι ένα στιγμιότυπο ενός συστήματος. Ένα πραγματικό σύστημα έχει μεταβλητές οι οποίες μπορούν να λάβουν ένα συγκεκριμένο εύρος τιμών κατά την διάρκεια ζωής του προγράμματος. Μια κατάσταση είναι λοιπόν μια συγκεκριμένη ανάθεση τιμών, σε όλες τις μεταβλητές συστήματος, σε ένα στιγμιότυπο του χρόνου. Έστω x η τιμή μιας μεταβλητής στην τωρινή κατάσταση, η **x′ (x prime)** ορίζεται η τιμή της μεταβλητής x στην επόμενη κατάσταση.
Ένα **βήμα** είναι ένα ζεύγος καταστάσεων (s,t) όπου s είναι η παλιά κατάσταση και t είναι η νέα κατάσταση. Ένα βήμα το οποίο αφήνει το σύστημα στην ίδια κατάσταση ονομάζεται **stuttering** βήμα, επίσης γνωστό και ως idling βήμα.

Η εκτέλεση ενός ταυτόχρονου συστήματος αναπαρίσταται σαν μια άπειρη ακολουθία από καταστάσεις (εναλλακτικά αποκαλείται και **συμπεριφορά**):

$$\sigma = s_0 s_1 \ldots$$

Στην συνέχεια ακολουθεί μια απεικόνιση της κατάστασης, του βήματος, και της συμπεριφοράς.



Μια **συνάρτηση κατάστασης** (state function)[1] είναι μια απλή έκφραση, δηλαδή δεν περιέχει primes και □ (βλέπε ενότ. 1.4), που αποτελείται από μεταβλητές, σταθερές και constant τελεστές (δεν εμπεριέχουν χρονική λογική). Μια συνάρτηση κατάστασης αναθέτει μια constant έκφραση σε κάθε κατάσταση. Αν η συνάρτηση κατάστασης e αναθέτει στην κατάσταση s την constant έκφραση v, τότε λέμε ότι v είναι η τιμή της e στην κατάσταση s. Για παράδειγμα, αν x είναι μια δηλωθείσα μεταβλητή, T είναι μια δηλωθείσα σταθερά, και s είναι μια κατάσταση που αναθέτει στην x τη τιμή 42 τότε η τιμή της $x \in T$ στην κατάσταση s είναι η constant έκφραση $42 \in T$. Μια συνάρτηση κατάστασης που παίρνει Boolean τιμές αποκαλείται **κατηγόρημα κατάστασης** (state predicate). Ένα κατηγόρημα κατάστασης είναι έγκυρο αν και μόνο αν έχει τιμή true σε κάθε κατάσταση.

Μια **ενέργεια** είναι μια μαθηματική έκφραση στην οποία οι unprimed μεταβλητές αναφέρονται στην αρχική κατάσταση ενός βήματος και οι primed μεταβλητές αναφέρονται στην επόμενη κατάσταση του. Κάθε κατάσταση μετά την s_0 προκύπτει από την εκτέλεση μιας ενέργειας στην προηγούμενη κατάσταση. (Για μια εκτέλεση τερματισμού, μια άπειρη ακολουθία λαμβάνεται επαναλαμβάνοντας την τελική κατάσταση.)

Για μια ενέργεια A και μια συνάρτηση κατάστασης f, ορίζουμε τον τελεστή ενέργειας $[A]_f$ ως $[A]_f = A \ (\ A \vee f' = f \ )$. Ο τελεστής $[A]_f$ διαβάζεται ως τετράγωνο A με δείκτη f (square A sub f). Ένα βήμα, το οποίο είναι ένα ζεύγος καταστάσεων (s,t), ικανοποιεί την $[A]_f$ αν και μόνο αν αυτό είναι ένα A βήμα ή ένα stuttering βήμα. Αυτό γράφεται ως: $(s,t) \models \Box[A]_f$. Για παράδειγμα η $[x' = x + 3]_x$ ικανοποιείται σε ένα βήμα (s,t) αν και μόνο αν η τιμή της μεταβλητής x στην κατάσταση t είναι κατά 3 μονάδες μεγαλύτερη από την τιμή της x στην κατάσταση s ή η x παραμένει αμετάβλητη.

Επιπλέον για μια ενέργεια A και μια συνάρτηση κατάστασης u, ο τελεστής ενέργειας $\langle A \rangle_u$ ορίζεται να είναι ισοδύναμος με $A \wedge (v' \neq v)$. Δηλαδή, ένα βήμα, το οποίο είναι ένα ζεύγος καταστάσεων (s,t), ικανοποιεί την $\langle A \rangle_u$ αν και μόνο αν αυτό είναι ένα A βήμα που αλλάζει το u.

Ένα invariant Inv μιας προδιαγραφής Spec είναι ένα κατηγόρημα κατάστασης τέτοιο ώστε η $Spec \Rightarrow \Box Inv$ είναι ένα **θεώρημα** (theorem), δηλαδή μια χρονική φόρμουλα που ικανοποιείται από όλες τις συμπεριφορές.

Μια μεταβλητή υ είναι **τύπου** Τ σε μια προδιαγραφή Spec αν και μόνο αν υ ∈ Τ είναι ένα invariant της Spec.

Μια **ιδιότητα**[3] είναι μια έκφραση της χρονικής λογικής η οποία ορίζει ένα κατηγόρημα κατάστασης πάνω σε συμπεριφορές. Υπάρχουν δύο κύριες ταξινομήσεις ιδιοτήτων που αφορούν τα ταυτόχρονα συστήματα. Αυτές είναι οι **safety** ιδιότητες και οι **liveness** ιδιότητες. Οι safety ιδιότητες, επίσης γνωστές σαν invariance ιδιότητες, ισχυρίζονται ότι "τίποτα κακό δεν θα συμβεί ποτέ" και οι liveness ιδιότητες, ή progress ιδιότητες, ισχυρίζονται ότι, τελικά "κάτι καλό θα συμβεί". Ένα παράδειγμα μια safety ιδιότητας είναι «Η πόρτα του ασανσέρ δεν είναι ποτέ ανοιχτή ενώ το ασανσέρ κινείται». Ένα παράδειγμα liveness ιδιότητας είναι «Αν το κουμπί κλήσης πατηθεί στον πρώτο όροφο το ασανσέρ τελικά θα φτάσει σ' αυτόν τον όροφο».

1.3 Δομή μιας TLA+ προδιαγραφής

Μια TLA+ προδιαγραφή[3] αποτελείται από:

1.Τις δηλώσεις όλων των σταθερών και των μεταβλητών που εμφανίζονται στην προδιαγραφή και τους ορισμούς των τελεστών που χρησιμοποιούνται

2.Το αρχικό κατηγόρημα το οποίο ορίζει την αρχική κατάσταση του συστήματος

3.Την σχέση επόμενης κατάστασης η οποία είναι μια διάζευξη των υποενεργειών και ορίζει μόνο τις δυνατές λειτουργίες του συστήματος, ή με άλλα λόγια τις safety ιδιότητες που το σύστημα πρέπει να επιδεικνύει

4. Οποιεσδήποτε liveness ιδιότητες που το σύστημα πρέπει να επιδεικνύει

Στην συνέχεια παρουσιάζεται η μορφή μιας TLA προδιαγραφής

$$Spec \wedge Init \wedge \Box\,[Next]_{var} \rightarrow \Box\, Liveness$$

Οι συνηθισμένοι τελεστές των συνόλων επιτρέπονται, για παράδειγμα η ένωση, η τομή και η διαφορά συνόλων. Στην συνέχεια ακολουθεί η προδιαγραφή σε TLA+ για το παράδειγμα του ρολογιού που έχει γραφτεί για τον TLC model checker.

```
---------------------- MODULE HourClock ----------------------
(*****************************************************)
(* This module specifies a digital clock that displays *)
(* the current hour. It ignores real time, not *)
(* specifying when the display can change. *)
(*****************************************************)
EXTENDS Naturals
VARIABLE hr \* Variable hr represents the display.
HCini == hr \in (1 .. 12) \* Initially, hr can have any
\* value from 1 through 12.
(*********************************************)
(* The value of hr cycles from 1 through 12. *)
(*********************************************)
HCnxt ==
hr' = IF hr # 12 THEN hr + 1 ELSE 1
HC == HCini /\ [][HCnxt]_hr
(* The complete spec. It permits the clock to stop. *)
-------------------------------------------------------------
THEOREM HC => []HCini \* Type-correctness of the spec.
=============================================================
```

Οτιδήποτε μεταξύ των (* και *) είναι ένα σχόλιο. Η πρώτη γραμμή είναι η αρχή της module HourClock. Η EXTENDS Naturals δείχνει ότι το module θα χρησιμοποιήσει το προκαθορισμένο module για τους φυσικούς αριθμούς. Ακολουθεί η δήλωση της μεταβλητή hr. Η HCini ορίζει την hr σε μια ακέραια τιμή από το 1 μέχρι το 12. Η HCnxt ορίζει τις πιθανές ενέργειες του HourClock, δηλαδή η HCnxt δείχνει πως από η hr' προκύπτει από την hr. Το HC ισχυρίζεται ότι οι αρχικές συνθήκες (HCini) διατηρούνται στο αρχικό βήμα και η σχέση επόμενης κατάστασης (HCnxt) διατηρείται πάντα, δηλαδή όλα τα βήματα είναι HCnxt βήματα. Η οριζόντια γραμμή σηματοδοτεί το τέλος του module HourClock. Το THEOREM HC→□HCini ισχυρίζεται ότι η προδιαγραφή HourClock συνεπάγεται την □HCini. Αυτό σημαίνει ότι η προδιαγραφή HourClock είναι μια ειδίκευση της □HCini , όπου οι συμπεριφορές που επιτρέπονται από την προδιαγραφή HourClock είναι ένα υποσύνολο των συμπεριφορών που επιτρέπονται από την □HCini. Ο Lamport χρησιμοποιεί την λέξη κλειδί THEOREM για να δείξει μια έγκυρη έκφραση.

Η επαλήθευση ενός συστήματος σε TLA αποτελείται από τον προσδιορισμό του συστήματος, που είναι η χαμηλού επιπέδου προδιαγραφή, και από τις απαιτούμενες ιδιότητες, που συνιστούν την υψηλού επιπέδου προδιαγραφή. Η έκφραση System → Properties πρέπει τότε να αποδειχθεί. Το σύστημα προσδιορίζεται ορίζοντας την αρχική κατάσταση(ή καταστάσεις) και τις ενέργειες που είναι δυνατές. Αυτό σημαίνει ότι γενικά οι εκφράσεις που προσπαθούμε να αποδείξουμε είναι της μορφής :

$$Init \wedge [Next]_{var} \rightarrow \Box\ Properties$$

Η είσοδος στο TLC αποτελείται από ένα TLA+ module και ένα configuration αρχείο. Το TLC λαμβάνει μια προδιαγραφή που έχει μορφή σαν αυτή που περιγράψαμε πιο πάνω. Το αρχείο configuration δηλώνει στο TLC το όνομα της προδιαγραφής και τα ονόματα των ιδιοτήτων που πρέπει να ελεγχθούν μέσω της εντολής PROPERTY. Για παράδειγμα για την προδιαγραφή του HourClock έχουμε το ακόλουθο configuration αρχείο:

```
(*************************************************************************)
(* This is a TLC configuration file for testing that HCini is an   *)
(*invariant of the specification HC                                      *)
(*************************************************************************)

SPECIFICATION HC
   \* This statement tells TLC that HC is the specification that it
   \* should check.

INVARIANT HCini
   \* This statement tells TLC to check that formula HCini is an
   \* invariant of the specification--in other words, that the
   \* specification implies []HCini.
```

Στο αρχείο αυτό παρατηρούμε ότι χρησιμοποιείται η INVARIANT εντολή στην θέση της εντολής PROPERTY για να δηλώσει τις ιδιότητες που πρέπει να ελεγχθούν. Αυτό γίνεται για τον λόγο ότι η προσδιορισμένη ιδιότητα δεν είναι της μορφής $\Box P$. Για να ελεγχθεί το invariance με μια PROPERTY εντολή, η προσδιορισμένη ιδιότητα έπρεπε να είναι της μορφής $\Box P$. Ο προσδιορισμός ενός κατηγορήματος κατάστασης P σε μια PROPERTY εντολή λέει στο TLC να ελέγξει ότι η προδιαγραφή συνεπάγεται το P, που σημαίνει ότι το P είναι αληθές στην initial κατάσταση κάθε συμπεριφοράς που ικανοποιεί την προδιαγραφή.

1.4 Χρονικές Εκφράσεις

Ένα σύστημα περιγράφεται από ένα σύνολο μεταβλητών. Μια κατάσταση αναθέτει μια τιμή σε κάθε μεταβλητή, μια συμπεριφορά σ είναι μια μη πεπερασμένη ακολουθία καταστάσεων. Μια χρονική έκφραση F είναι αληθής ή ψευδής σε μια συμπεριφορά σ. Τυπικά, μια χρονική έκφραση F αναθέτει μια Boolean τιμή, την οποία γράφουμε $\sigma \models F$, σε μια συμπεριφορά σ. Λέμε ότι η F είναι αληθής στην σ, ή ότι η σ ικανοποιεί την F, αν και μόνο αν $\sigma \models F$ ισοδυναμεί με true. Για να ορίσουμε την έννοια μια χρονικής έκφρασης F, πρέπει να επεξηγήσουμε πως προσδιορίζουμε την τιμή της $\sigma \models F$ για κάθε συμπεριφορά σ.

Είναι εύκολο να ορίσουμε την έννοια ενός Boolean συνδυασμού από χρονικές εκφράσεις. Η έκφραση $F \wedge G$ είναι αληθής σε μια συμπεριφορά σ αν και μόνο αν F και G είναι αληθής στην σ, και $\neg F$ είναι αληθής στην σ αν και μόνο αν η F δεν είναι αληθής στην σ. Αυτοί οι ορισμοί γράφονται συμβολικά ως

$$\sigma \models (F \wedge G) \triangleq (\sigma \models F) \wedge (\sigma \models G) \qquad \sigma \models \neg F \triangleq \neg(\sigma \models F)$$

Όλες οι χρονικές εκφράσεις που έχουμε δει έχουν Boolean συνδυασμούς τριών βασικών ειδών εκφράσεων, οι οποίες έχουν ως εξής:

- Κατηγόρημα κατάστασης (state predicate), που εμφανίζεται σαν μια χρονική έκφραση, είναι αληθές σε μια συμπεριφορά αν και μόνο αν αυτό είναι αληθές στην πρώτη κατάσταση της συμπεριφοράς.

- Έκφραση $\Box P$, όπου P είναι ένα κατηγόρημα κατάστασης, που είναι αληθής σε μια συμπεριφορά αν και μόνο αν το P είναι αληθές σε κάθε κατάσταση της συμπεριφοράς.

- Έκφραση $\Box[N]_v$, όπου N είναι μια ενέργεια και υ είναι μια συνάρτηση κατάστασης, που είναι αληθής στην συμπεριφορά αν και μόνο αν κάθε διαδοχικό ζεύγος βημάτων στην συμπεριφορά είναι ένα $[N]_v$ βήμα.

Στην συνέχεια εξετάζουμε πέντε ιδιαίτερα σημαντικές κλάσεις εκφράσεων που κατασκευάζονται από αυθαίρετες χρονικές εκφράσεις F και G.

- Τελικά Αληθής, $\Diamond F$ ορίζεται να είναι ισοδύναμη με $\neg\Box\neg F$. Αυτή ισχυρίζεται ότι η F δεν είναι πάντα ψευδής, το οποίο σημαίνει ότι η F είναι αληθής κάποια στιγμή.

- $F \rightsquigarrow G$ ορίζεται να είναι ισοδύναμη με $\Box(F \Rightarrow \Diamond G)$. Η έκφραση $F \rightsquigarrow G$ ισχυρίζεται ότι οποτεδήποτε το F είναι αληθές, το G τελικά είναι αληθές δηλαδή, το G είναι αληθές τότε ή σε κάποια ύστερη χρονική στιγμή. Διαβάζουμε το $\rightsquigarrow$ ως leads to.

- $\Diamond\langle A\rangle_v$ ορίζεται να είναι ισοδύναμη με $\neg\Box[\neg A]_v$,όπου A είναι μια ενέργεια και υ είναι μια συνάρτηση κατάστασης. Ισχυρίζεται ότι κάποιο βήμα είναι ένα $\neg((\neg A) \vee (v' = v))$ βήμα που ισοδυναμεί με $A \wedge (v' \neq v)$, δηλαδή ένα βήμα A που αλλάζει υ. Έτσι η $\Diamond\langle A\rangle_v$ ισχυρίζεται ότι τελικά ένα $\langle A\rangle_v$ βήμα λαμβάνει χώρα.

- $\Box\Diamond F$ ισχυρίζεται ότι στο μέλλον η F θα είναι άπειρες φορές αληθής. Για την χρονική στιγμή 0, αυτό υπονοεί ότι η F είναι αληθής σε κάποια χρονική στιγμή $n_0 \geq 0$. Για την χρονική στιγμή $n_0 + 1$, αυτό υπονοεί ότι η F είναι αληθής σε κάποια χρονική στιγμή $n_1 \geq n_0 + 1$. Για την χρονική στιγμή $n_1 + 1$, αυτό υπονοεί ότι η F είναι αληθής σε κάποια χρονική στιγμή $n_2 \geq n_1 + 1$. Συνεχίζοντας την διαδικασία, βλέπουμε ότι η F είναι αληθής σε μια άπειρη ακολουθία από χρονικές στιγμές $n_0, n_1, n_2, \ldots$. Έτσι, η $\Box\Diamond F$ υπονοεί ότι η F είναι αληθής σε απείρως πολλές στιγμές. Αντιστρόφως, αν η F είναι αληθής σε απείρως πολλές στιγμές, τότε, σε κάθε στιγμή, η F πρέπει να είναι αληθής σε επόμενη ακόλουθη στιγμή, έτσι η $\Box\Diamond F$ είναι αληθής.

Επομένως, η $\Box\Diamond F$ ισχυρίζεται ότι η F είναι άπειρα συχνά (infinitely often) αληθής. Ειδικότερα, η $\Box\Diamond\langle A\rangle_v$ ισχυρίζεται ότι άπειρα πολλά $\langle A\rangle_v$ βήματα λαμβάνουν χώρα.

- $\Diamond\Box F$ ισχυρίζεται ότι τελικά (eventually) (κάποια χρονική στιγμή, η F γίνεται αληθής και παραμένει αληθής από εκεί και έπειτα. Με άλλα λόγια, η $\Diamond\Box F$ ισχυρίζεται ότι η F είναι τελικά πάντα (eventually always) αληθής. Ειδικότερα, η $\Diamond\Box[N]_v$ ισχυρίζεται ότι, τελικά, κάθε βήμα είναι ένα $[N]_v$ βήμα.

1.5 Fairness

Μια από τις πιο σημαντικές έννοιες στην ταυτόχρονη επεξεργασία είναι το fairness[4]. Fairness σημαίνει ότι κάθε διεργασία έχει την ευκαιρία να σημειώσει πρόοδο, ανεξάρτητα από τις άλλες διεργασίες. Το fairness είναι εγγυημένο σε ένα πραγματικά ταυτόχρονο σύστημα στο οποίο κάθε διεργασία τρέχει στον δικό της επεξεργαστή: μια διεργασία δεν μπορεί να σταματήσει την πραγματική εκτέλεση μιας διεργασίας που τρέχει σε έναν διαφορετικό επεξεργαστή. Τα συστήματα πολυπρογραμματισμού, στα οποία ένας single επεξεργαστής μοιράζεται σε διάφορες διεργασίες, μπορεί να παρέχει ή να μην παρέχει fairness, γεγονός που εξαρτάται από τον scheduling αλγόριθμο που χρησιμοποιείται για την διανομή (allocation) του επεξεργαστή.

Στην απόδειξη των safety ιδιοτήτων, δεν έχει νόημα το κατά πόσον υπάρχει fairness – οποιαδήποτε safety ιδιότητα που διατηρείται υπό δίκαιο scheduling θα διατηρείται επίσης και υπό μη δίκαιου scheduling. Ωστόσο, πολλά προγράμματα ικανοποιούν τις liveness απαιτήσεις τους μόνο αν η βασική υλοποίηση εγγυάται fairness. Μια μεθοδολογία που δεν μπορεί να αποδείξει αυτές τις ιδιότητες θα είναι ανεπαρκής για την αντιμετώπιση πραγματικού ταυτοχρονισμού, επομένως το fairness πρέπει να είναι μέρος μιας γενικής μεθόδου για την αντιμετώπιση ταυτόχρονων προγραμμάτων. Η προσέγγιση της χρονική λογικής καθιστά εύκολο το να εκφράσουμε τις fairness ιδιότητες και να τις χρησιμοποιήσουμε στις αποδείξεις μας.

1.6 Weak Fairness

Είναι εύκολο να προσδιορίσουμε liveness ιδιότητες με τους χρονικούς τελεστές $\Box$ και $\Diamond$. Για παράδειγμα, μελετώντας την hour-clock προδιαγραφή της module HourClock. Μπορούμε να απαιτήσουμε ότι το ρολόι ποτέ δεν σταματά (πράγμα που γίνεται στην LiveHourClock.tla) (για περισσότερες πληροφορίες σχετικά με αυτά τα module μεταβείτε στο Παράρτημα Α) υποστηρίζοντας ότι πρέπει να υπάρχουν άπειρα πολλά HCnxt βήματα. Ο προφανής τρόπος να γράψουμε αυτόν τον ισχυρισμό είναι η $\Box\Diamond HCnxt$, αλλά αυτή δεν είναι μια νόμιμη TLA έκφραση επειδή η HCnxt είναι μια ενέργεια, όχι μια χρονική έκφραση. Ωστόσο, ένα HCnxt βήμα προχωρά την τιμή hr του ρολογιού, έτσι αυτό αλλάζει την hr. Επομένως, ένα HCnxt βήμα είναι ένα βήμα που αλλάζει την hr, δηλαδή είναι ένα $\langle HCnxt\rangle_{hr}$ βήμα. Μπορούμε να γράψουμε την

liveness ιδιότητα ότι το ρολόι ποτέ δεν σταματά ως $\Box\Diamond\langle HCnxt\rangle_{hr}$. Έτσι, μπορούμε να λάβουμε την $HC \wedge \Box\Diamond\langle HCnxt\rangle_{hr}$ να είναι η προδιαγραφή ενός ρολογιού που ποτέ δεν σταματά.

Η TLA+ ορίζει την **ENABLED** A να είναι το κατηγόρημα που είναι αληθές αν και μόνο αν η ενέργεια A είναι ενεργοποιημένη σε μια κατάσταση, δηλαδή είναι δυνατό στην κατάσταση αυτή να λάβουμε μια ενέργεια A.

Αν οποιαδήποτε συμπεριφορά ικανοποιεί την safety προδιαγραφή HC (HourClock), είναι πάντα δυνατό να λάβουμε ένα HCnxt βήμα που αλλάζει την hr. Η ενέργεια $\langle HCnxt\rangle_{hr}$ είναι επομένως πάντα ενεργοποιημένη, επομένως έχουμε $\text{ENABLED}\,\langle HCnxt\rangle_{hr}$ που είναι αληθής σε όλη αυτήν την συμπεριφορά. Αφού η $\text{TRUE} \Rightarrow \Diamond\langle HCnxt\rangle_{hr}$ είναι ισοδύναμη με την $\Diamond\langle HCnxt\rangle_{hr}$, μπορούμε να αντικαταστήσουμε την liveness συνθήκη $\Box\Diamond\langle HCnxt\rangle_{hr}$ για το hour clock με

$$\Box(\text{ENABLED}\,\langle HCnxt\rangle_{hr} \Rightarrow \Diamond\langle HCnxt\rangle_{hr})$$

Αυτό υποδηλώνει την ακόλουθη γενική liveness συνθήκη για μια ενέργεια A:

$$\Box(\text{ENABLED}\,\langle A\rangle_{v} \Rightarrow \Diamond\langle A\rangle_{v})$$

Αυτή η συνθήκη ισχυρίζεται ότι, όποτε η A είναι ενεργοποιημένη, τότε ένα A βήμα τελικά θα συμβεί, ακόμη και αν το A παραμένει ενεργοποιημένο για μόνο ένα κλάσμα ενός νανοδευτερολέπτου και ποτέ ξανά δεν ενεργοποιείται. Η προφανής πρακτική δυσκολία της υλοποίησης μιας τέτοιας συνθήκης δείχνει ότι αυτή είναι πολύ ισχυρή (strong). Επομένως, αντικαθιστούμε αυτή με την ασθενέστερη (weaker) έκφραση $\text{WF}_{v}(A)$, ορίζοντας την να είναι ίση

1) $$\Box(\Box\text{ENABLED}\,\langle A\rangle_{v} \Rightarrow \Diamond\langle A\rangle_{v})$$

Αυτή η έκφραση ισχυρίζεται ότι, όποτε η A γίνεται για πάντα ενεργοποιημένη , τότε ένα A βήμα πρέπει τελικά να συμβεί. Η WF αντιπροσωπεύει την Weak Fairness, και η συνθήκη $\text{WF}_{v}(A)$ αποκαλείται weak fairness στην A.

Η τελευταία έκφραση αποδεικνύεται ότι είναι ισοδύναμη με την ακόλουθη έκφραση[1]:

2) $$\Diamond\Box(\text{ENABLED}\,\langle A\rangle_{v}) \;\Rightarrow\; \Box\Diamond\langle A\rangle_{v}$$

Η οποία ισχυρίζεται ότι όποτε η A είναι τελικά ενεργοποιημένη για πάντα, τότε απείρως πολλά A βήματα συμβαίνουν.

Οι liveness συνθήκες για το hour clock μπορούν να γραφτούν ως weak fairness συνθήκες. Αν οποιαδήποτε συμπεριφορά ικανοποιεί την HC, ένα $\langle HCnxt\rangle_{hr}$ βήμα είναι πάντα ενεργοποιημένο, έτσι η $\Diamond\Box(\text{ENABLED}\,\langle HCnxt\rangle_{hr})$ ισούται με true.

Επομένως, η HC συνεπάγεται ότι η $\mathrm{WF}_{hr}(HCnxt)$, η οποία ισοδυναμεί με την δεύτερη από τις παραπάνω εκφράσεις, είναι ισοδύναμη με την έκφραση $\Box\Diamond\langle HCnxt\rangle_{hr}$, η liveness συνθήκη για το hour clock.

Επομένως weak fairness μιας ενέργειας Α, σημαίνει ότι αν η Α είναι πάντα ενεργοποιημένη από κάποια στιγμή, τότε αυτή πρέπει τελικά να εκτελεστεί. Χρησιμοποιώντας το weak fairness σε μια ενέργεια Α, προσδιορίζεται ότι αυτή δεν θα λιμοκτονήσει από μια ενέργεια Β η οποία είναι πάντα ενεργοποιημένη την ίδια χρονική στιγμή. Μπορούμε ακόμη να έχουμε την περίπτωση όπου μια ενέργεια Β εκτελείται 10 φορές περισσότερες από την ενέργεια Α, αλλά όταν και οι δύο είναι weak fair, καμία από αυτές δεν θα μπορεί να περιμένει για πάντα.

1.7 Strong Fairness

Strong fairness της Α σημαίνει ότι αν η Α είναι απείρως συχνά ενεργοποιημένη από κάποια στιγμή, αυτή τελικά πρέπει να εκτελεστεί. Αυτή είναι μια χρήσιμη έννοια για την περίπτωση που δύο ή περισσότερες ενέργειες είναι ενεργοποιημένες στην ίδια κατάσταση, αλλά η εκτέλεση μιας αυτών κάνει την άλλη να μην είναι ενεργοποιημένη πλέον. Ένα καλό παράδειγμα από αυτό είναι ένα σενάριο αμοιβαίου αποκλεισμού όπου διάφορες οντότητες περιμένουν για ένα token επομένως αυτές μπορούν να εκτελέσουν μια κρίσιμη εργασία που περιλαμβάνει διαμοιραζόμενες μεταβλητές. Το strong fairness προσδιορίζεται από το:

$$\mathrm{SF}_{vars}(A)$$

Ορίζουμε την $\mathrm{SF}_v(A)$, strong fairness της ενέργειας Α, να είναι μια από τις ακόλουθες δύο ισοδύναμες εκφράσεις:

1) $$\Diamond\Box(\neg\,\mathrm{ENABLED}\,\langle A\rangle_v)\ \vee\ \Box\Diamond\langle A\rangle_v$$

Η οποία ισχυρίζεται ότι η Α τελικά απενεργοποιείται για πάντα, ή άπειρα πολλά Α βήματα συμβαίνουν.

2) $$\Box\Diamond\mathrm{ENABLED}\,\langle A\rangle_v\ \Rightarrow\ \Box\Diamond\langle A\rangle_v$$

Η οποία ισχυρίζεται ότι αν η Α είναι απείρως συχνά (infinitely often) ενεργοποιημένη, τότε άπειρα πολλά Α βήματα συμβαίνουν.

Αφού η $\Diamond\Box F$(τελικά πάντα/eventually always F) είναι ισχυρότερη δηλαδή συνεπάγεται την $\Box\Diamond F$ (απείρως συχνά/infinitely often F) για οποιαδήποτε έκφραση F, η strong fairness είναι ισχυρότερη από την weak fairness. Μπορούμε να εκφράσουμε την weak και την strong fairness ως ακολούθως:

- **Weak fairness** της Α ισχυρίζεται ότι ένα Α βήμα πρέπει τελικά να συμβεί αν η Α είναι continuously ενεργοποιημένη.

- **Strong fairness** της Α ισχυρίζεται ότι ένα Α βήμα πρέπει τελικά να συμβεί αν η Α είναι continually ενεργοποιημένη.

Continuously σημαίνει χωρίς διακοπή. Continually σημαίνει επανειλημμένα, πιθανώς με διακοπές.

Βασικοί κανόνες

WF Κανόνας Σύζευξης (WF Conjunction Rule)[1] Αν $A_1, \ldots, A_n$ είναι ενέργειες τέτοιες ώστε, για οποιαδήποτε διακριτό i και j , οποτεδήποτε το $\langle A_i \rangle_v$ είναι εφικτό, το $\langle A_j \rangle_v$ δεν μπορεί να γίνει εφικτό εκτός αν ένα $\langle A_i \rangle_v$ βήμα συμβεί, τότε η $\mathrm{WF}_v(A_1) \wedge \ldots \wedge \mathrm{WF}_v(A_n)$ είναι ισοδύναμη με την $\mathrm{WF}_v(A_1 \vee \ldots \vee A_n)$.

SF Κανόνας Σύζευξης (SF Conjunction Rule) Αν $A_1, \ldots, A_n$ είναι ενέργειες τέτοιες ώστε, για οποιαδήποτε διακριτά i και j , οποτεδήποτε η ενέργεια A_i είναι εφικτή, η ενέργεια A_j δεν μπορεί να γίνει εφικτή μέχρι ένα A_i βήμα να συμβεί, τότε η $\mathrm{SF}_v(A_1) \wedge \ldots \wedge \mathrm{SF}_v(A_n)$ είναι ισοδύναμη της $\mathrm{SF}_v(A_1 \vee \ldots \vee A_n)$.

Βασικές ταυτολογίες

Ένα σημαντικό ζευγάρι διπλών ταυτολογιών βεβαιώνει ότι το $\Box\Diamond$ επιμερίζεται στον τελεστή $\vee$ και ο $\Diamond\Box$ επιμερίζεται στον τελεστή $\wedge$:

$$\Box\Diamond(F \vee G) \equiv (\Box\Diamond F) \vee (\Box\Diamond G) \qquad \Diamond\Box(F \wedge G) \equiv (\Diamond\Box F) \wedge (\Diamond\Box G)$$

Οι fairness συνθήκες είναι ένας τρόπος για να εκφραστούν οι liveness ιδιότητες[5]. Η ερώτηση του κατά πόσον ένα πραγματικό σύστημα ικανοποιεί μια liveness ιδιότητα μπορεί να απαντηθεί μόνο παρατηρώντας το σύστημα για ένα άπειρου μήκους χρονικό διάστημα, τα πραγματικά συστήματα όμως δεν μπορούν να τρέχουν για πάντα. Το liveness είναι πάντα μια προσέγγιση για την ιδιότητα που πραγματικά μας ενδιαφέρει. Θέλουμε ένα σύστημα να τερματίζει μέσα σε 100 χρόνια, αλλά αυτό αποδεικνύει ότι θα απαιτούσε την προσθήκη κάποιων χρονικών παραδοχών. Επομένως, αποδεικνύουμε την ασθενέστερη συνθήκη όπου το σύστημα τελικά τερματίζει. Αυτό δεν αποδεικνύει ότι το σύστημα θα τερματίσει κατά την διάρκεια της ζωής μας, αλλά αυτό αποδεικνύει την απουσία άπειρων βρόγχων. Το liveness προσδιορίζεται σαν την σύζευξη των weak και/ή strong fairness ιδιοτήτων οποτεδήποτε είναι δυνατό και συνήθως αυτό είναι σχεδόν πάντα δυνατό. Έχοντας έναν ενιαίο τρόπο για να εκφράσουμε το liveness κάνουμε τις προδιαγραφές ευκολότερες στην κατανόηση.

Κεφάλαιο 2: Ιδιότητες Liveness

Περίληψη κεφαλαίου

Στο κεφάλαιο αυτό γίνεται μια εισαγωγή σχετικά με τις liveness ιδιότητες. Στην συνέχεια προτείνονται διάφοροι ορισμοί αναφορικά με την liveness ιδιότητα. Επιπρόσθετα παρουσιάζονται και κάποιες άλλες ιδιότητες που εμπεριέχουν το στοιχείο του liveness. Τέλος παρουσιάζεται το πότε μια προδιαγραφή θεωρείται machine closed. Σκοπός του κεφαλαίου είναι να γίνει κατανοητή η liveness ιδιότητα ως συστατικό στοιχείο του αλγορίθμου του liveness model checking που επεξηγείται στα επόμενα κεφάλαια.

2.1 Εισαγωγή

Τα σύγχρονα λειτουργικά συστήματα κάνουν εκτεταμένη χρήση ταυτόχρονων και κατανεμημένων αλγορίθμων. Αυτοί οι αλγόριθμοι είναι περίπλοκοι και κατά συνέπεια είναι εύκολο να υπάρχει κάποιο λάθος οδηγώντας μας σε μια λανθασμένη υψηλού επιπέδου σχεδίαση.

Ο μόνος τρόπος με τον οποίο μπορούμε να είμαστε σίγουροι ότι ένα ταυτόχρονο σύστημα κάνει ότι πιστεύουμε ότι πρέπει να κάνει, είναι να αποδείξουμε αυστηρά ότι όντως αυτό συμβαίνει. Υπάρχουν δυο είδη ιδιοτήτων που ζητούνται από ένα ταυτόχρονο σύστημα να ικανοποιεί[4]:

- Ιδιότητες Safety, που δηλώνουν ότι κάτι "κακό" δεν πρόκειται ποτέ να συμβεί, δηλαδή, ότι το πρόγραμμα ποτέ δεν θα εισαχθεί σε μια μη αποδεκτή κατάσταση. Εκφράζουν ότι η συμπεριφορά ενός συστήματος ποτέ δεν παραβιάζει συγκεκριμένες συνθήκες.

- Ιδιότητες Liveness, που δηλώνουν ότι κάτι "καλό" θα συμβεί τελικά, δηλαδή, ότι το πρόγραμμα εισάγεται τελικά σε μια αποδεκτή κατάσταση. Εκφράζουν ότι το σύστημα θα εμφανίσει μια συγκεκριμένη συμπεριφορά μέσα σε μια πεπερασμένη περίοδο χρόνου.

Κάποιες συνηθισμένες ιδιότητες safety είναι:

- **Μερική Ορθότητα (Partial correctness)**: αν το πρόγραμμα ξεκινά με αληθή προσυνθήκη (precondition), τότε αυτό ποτέ δεν μπορεί να τερματίσει με την ψευδή μετασυνθήκη (postcondition).

- **Απουσία Αδιεξόδου (Absence of deadlock)**: το πρόγραμμα ποτέ δεν εισάγεται σε μια κατάσταση στην οποία καμία περαιτέρω πρόοδος δεν είναι δυνατή.

- **Αμοιβαίος αποκλεισμός (Mutual exclusion)**: δυο διαφορετικές διεργασίες δεν βρίσκονται ποτέ ταυτόχρονα στα κοινά κρίσιμα τμήματα εκτέλεσης τους.

- **Χρονική προτεραιότητα (First-come-first-serve)**: μια αίτηση που πραγματοποιήθηκε μετά από μια άλλη αίτηση δεν μπορεί ποτέ να εξυπηρετηθεί πριν από αυτήν.

Παραδείγματα liveness ιδιοτήτων περιλαμβάνουν:

- **Έλλειψη λιμοκτονίας (Starvation freedom)**: η οποία ορίζει ότι μια διεργασία εμφανίζει πρόοδο άπειρα συχνά, το ”κάτι καλό ” είναι ότι υπάρχει πρόοδος.

- **Τερματισμός (Termination)**: ο οποίος ισχυρίζεται ότι ένα σύστημα δεν θα εκτελείται για πάντα, το “κάτι καλό” είναι η ολοκλήρωση της τελευταίας εντολής.

- **Εγγυημένη υπηρεσία (Guaranteed service)**: η οποία ορίζει ότι κάθε αίτηση για υπηρεσία τελικά ικανοποιείται, το “κάτι καλό” είναι η λαμβανόμενη υπηρεσία.

Το TLC μπορεί να χρησιμοποιηθεί για τον έλεγχο των safety και liveness[6] ιδιοτήτων σε μια προδιαγραφή γραμμένη στην μορφή $Init \wedge \Box Next \wedge Liveness$.

Init είναι το αρχικό κατηγόρημα κατάστασης, μια έκφραση που περιγράφει όλες τις έγκυρες αρχικές καταστάσεις.

Next είναι η συσχέτιση επόμενης κατάστασης, η οποία προσδιορίζει όλα τα δυνατά βήματα (ζεύγη από διαδοχικές καταστάσεις) σε μια συμπεριφορά συστήματος. Αυτή είναι μια διάζευξη ενεργειών που περιγράφει τις διαφορετικές λειτουργίες του συστήματος.

Liveness είναι μια χρονική έκφραση που προσδιορίζει τις ιδιότητες του συστήματος ως την σύζευξη των fairness συνθηκών στις ενέργειες (WFvars (A) και/ή SFvars (A), για τις ενέργειες A).

Για τις safety ιδιότητες, το TLC εξερευνά όλες τις προσβάσιμες καταστάσεις στο μοντέλο συστήματος, ψάχνοντας μια κατάσταση στην οποία μια invariant δεν ικανοποιείται ή αν ένα αδιέξοδο υφίσταται (δεν υπάρχει δυνατή επόμενη κατάσταση). Όσον αφορά τις liveness ιδιότητες, το TLC χρησιμοποιεί την τυποποιημένη tableau μέθοδο (για περισσότερες πληροφορίες σχετικά με το σημασιολογικό tableau μεταβείτε στο Παράρτημα Β). Μια tableau κατασκευή είναι ένας αλγόριθμος που μεταφράζει μια λογικά χρονική έκφραση σε ένα αυτόματο πεπερασμένων καταστάσεων που αποδέχεται ακριβώς όλες τις απεικονίσεις (models) της έκφρασης. Το TLC δεν χρησιμοποιεί την κατασκευή tableau της χρονικής έκφρασης αν αυτή περιλαμβάνει μόνο χρονικές υποεκφράσεις $\Box\Diamond A$ (απείρως συχνά) και $\Diamond\Box A$ (σχεδόν πάντα) όπου A είναι μια κατάσταση ή ένα κατηγόρημα ενέργειας. Δεν χρησιμοποιεί tableau για αυτού του είδους τις χρονικές εκφράσεις γιατί μπορούν να ελεγχθούν πιο αποδοτικά με απλούστερο τρόπο: με τον έλεγχο ύπαρξης κύκλων στον γράφο κατάστασης. Επειδή ο γράφος κατάστασης κατασκευάζεται για το liveness έλεγχο μπορεί εύκολα να γίνει πολύ μεγάλος για να το χειριστεί το TLC, το TLC περιοδικά ελέγχει τις liveness

ιδιότητες σε έναν μερικό γράφο καταστάσεων που έχει κατασκευαστεί μέχρι εκείνη την στιγμή. Όταν το TLC ανιχνεύσει ένα σφάλμα, ένα trace κατάστασης που δείχνει το σφάλμα τυπώνεται σαν μέρος της αναφοράς σφάλματος.

Η πραγματική σημασία του liveness ελέγχου είναι να αποδείξει την απουσία βρόγχων σε μονοπάτια διαδοχικών καταστάσεων άπειρου μήκους.

2.2 Ορισμοί ιδιότητας Liveness

Προτείνονται οι ακόλουθοι ορισμοί για την ιδιότητα liveness[7]:

Έστω S το σύνολο των καταστάσεων του συστήματος, S το σύνολο ακολουθιών άπειρου μήκους από καταστάσεις του συστήματος και S^* το σύνολο πεπερασμένων ακολουθιών από καταστάσεις του συστήματος. Μια εκτέλεση οποιουδήποτε προγράμματος μπορεί να μοντελοποιηθεί σαν ένα μέλος του S . Τα στοιχεία των S αποκαλούνται εκτελέσεις και τα στοιχεία του S^* μερικές εκτελέσεις και γράφουμε $\sigma \models P$ όταν η εκτέλεση σ ικανοποιεί την ιδιότητα P.

Ορισμός 1

Μια μερική εκτέλεση α είναι live για μια ιδιότητα P αν και μόνο αν υπάρχει μια ακολουθία από καταστάσεις β τέτοιες ώστε $\alpha\beta \models P$. Μια liveness ιδιότητα είναι μια ιδιότητα για την οποία κάθε μερική εκτέλεση είναι live. Επομένως P είναι μια liveness ιδιότητα αν και μόνο αν

Liveness: $(\forall a : \ a \in S^* : \ (\exists \beta : \ \beta \in S^\omega : \ \alpha\beta \models P))$

Ο ορισμός αυτός είναι ο λιγότερο περιοριστικός.

Μέσω της αντίφασης, θεωρείστε ότι P είναι μια liveness ιδιότητα που δεν ικανοποιεί τον παραπάνω ορισμό. Πρέπει να υπάρχει κάποια μερική εκτέλεση α, τέτοια ώστε

$$(\forall \beta : \ \beta \in S^\omega : \ \alpha\beta \not\models P))$$

Η εκτέλεση α είναι "κάτι κακό" το οποίο απαγορεύεται από την P . Έτσι, η P είναι εν μέρει safety ιδιότητα και όχι μια liveness ιδιότητα, όπως θεωρήθηκε.

Υπάρχουν ορισμοί για το liveness περισσότερο περιοριστικοί από τον παραπάνω.

Ορισμός 2

Ένας υποψήφιος που έχει διερευνηθεί είναι:

Uniform Liveness: $(\exists \beta : \ \beta \in S^\omega : \ (\forall \alpha : \ \alpha \in S^* : \ \alpha\beta \models P))$

η P είναι uniform liveness ιδιότητα αν και μόνο αν υπάρχει μια απλή εκτέλεση (β) που μπορεί να προσαρτηθεί σε κάθε μερική εκτέλεση (α) έτσι ώστε η ακολουθία που παράγεται να ικανοποιεί την P.

Ορισμός 3

Άλλος ορισμός που έχει προταθεί από τον Sistla

Absolute Liveness: $(\exists \gamma : \ \gamma \in S^{a} : \ \gamma \models P) \wedge (\forall \beta : \ \beta \in S^{a} : \ \beta \models P \Rightarrow (\forall a : \ a \in S^{*} : \ a\beta \models P))$

η P είναι absolute liveness ιδιότητα αν και μόνο αν αυτή είναι μη κενή (non-empty), δηλαδή υπάρχει τουλάχιστον μια εκτέλεση που την ικανοποιεί, και οποιαδήποτε εκτέλεση (β) στην P μπορεί να προσαρτηθεί σε οποιαδήποτε μερική εκτέλεση (α) για να λάβει μια ακολουθία στην P.

Είναι χρήσιμο να γίνει η σύγκριση αυτών των τυπικών ορισμών. Η L είναι μια liveness ιδιότητα αν οποιαδήποτε μερική εκτέλεση α μπορεί να επεκταθεί σε κάποια εκτέλεση β έτσι ώστε αβ να είναι L – η επιλογή της β πιθανόν να εξαρτάται από την α. Η U είναι μια uniform- liveness ιδιότητα αν υπάρχει μια απλή εκτέλεση β η οποία επεκτείνει όλες τις μερικές εκτελέσεις α τέτοιες ώστε αβ να είναι U. Και, Α είναι μια absolute-liveness ιδιότητα αν αυτή είναι μη κενή και οποιαδήποτε εκτέλεση β στην Α μπορεί να χρησιμοποιηθεί για να επεκτείνει όλες τις μερικές εκτελέσεις α. Οποιαδήποτε absolute-liveness ιδιότητα είναι μια uniform-liveness ιδιότητα και οποιαδήποτε uniform-liveness ιδιότητα είναι μια liveness ιδιότητα.

Ενώ η absolute liveness χαρακτηρίζει μια ενδιαφέρουσα τάξη ιδιοτήτων, δεν περιλαμβάνει όλες τις ιδιότητες που θα μπορούσαν να θεωρηθούν liveness. Οποιαδήποτε leads-to ιδιότητα (π.χ. μια εγγυημένη υπηρεσία) δεν είναι μια absolute-liveness ιδιότητα. Τέτοιες ιδιότητες χαρακτηρίζονται ως

> **Leads-to:** Οποιαδήποτε ύπαρξη ενός γεγονότος τύπου E_1 τελικά ακολουθείται από μια ύπαρξη ενός γεγονότος τύπου E_2.

Όταν το E_2 είναι ικανοποιήσιμο χαρακτηρίζεται ως μια liveness ιδιότητα, "κάτι καλό" που θέλουμε τελικά να συμβεί είναι το γεγονός E_2 . Για να δείτε ότι μια leads-to ιδιότητα δεν είναι μια absolute-liveness ιδιότητα θεωρείστε μια εκτέλεση β στην οποία κανένα γεγονός τύπου E_1 ή E_2 δεν συμβαίνει. Η leads-to θα πρέπει να ισχύει στη β. Ωστόσο προσαρτώντας την β σε μια μερική εκτέλεση αποτελούμενη από ένα απλό γεγονός τύπου E_1 παράγει μια εκτέλεση που δεν ικανοποιεί την ιδιότητα.

Επίσης θεωρείται ότι η uniform liveness δεν συλλαμβάνει πλήρως την έννοια του liveness. Ένα παράδειγμα μιας liveness ιδιότητας που δεν είναι uniform-liveness ιδιότητα χαρακτηρίζεται από

Predictive: Αν η ιδιότητα Α αρχικά ισχύει τότε μετά από κάποια μερική εκτέλεση, η ιδιότητα Β πάντοτε θα ισχύει. Διαφορετικά αν δεν ισχύει η Α, η Β ποτέ δεν θα ισχύει.

Αυτή είναι άλλη μια liveness ιδιότητα, επειδή αυτή απαιτεί κάποιο «καλό γεγονός» (είτε το «πάντοτε Β» ή «πάντοτε ») να συμβεί τελικά. Όμως, δεν είναι μια uniform-liveness ιδιότητα αφού δεν υπάρχει καμία απλή ακολουθία που να μπορεί επιτυχώς να επεκτείνει όλες τις μερικές εκτελέσεις.

2.3 Άλλες Ιδιότητες

Πολλές ιδιότητες δεν είναι ούτε safety ούτε liveness. Για παράδειγμα οποιαδήποτε ιδιότητα που χαρακτηρίζεται από[7]:

Ώσπου να (Until): Τελικά θα συμβεί ένα γεγονός τύπου E_2 και όλα τα προηγούμενα γεγονότα είναι του τύπου E_1.

είναι η τομή μιας safety ιδιότητας και μιας liveness ιδιότητας. Η safety ιδιότητα είναι: "$\neg E_1$ πριν E_2" δεν πρόκειται να συμβεί, και η liveness ιδιότητα είναι: το E_2 τελικά θα συμβεί. Η συνολική ορθότητα (total correctness) είναι επίσης η τομή μιας safety ιδιότητας (μερική ορθότητα) και μιας liveness ιδιότητας (τερματισμός). Στην πραγματικότητα κάθε ιδιότητα είναι η τομή (σύζευξη) μιας safety και μιας liveness ιδιότητας.

Τοπολογικοί χώροι

Τοπολογικοί χώροι (Topological spaces)[8] είναι μαθηματικές δομές που επιτρέπουν τον τυπικό ορισμό εννοιών όπως η σύγκλιση, η ορθότητα και η συνέχεια. Ο κλάδος των μαθηματικών που μελετά τοπολογικούς χώρους αποκαλείται τοπολογία (topology).

Ένας τοπολογικός χώρος είναι ένα σύνολο Χ μαζί με το τ, μια συλλογή από υποσύνολα του Χ, που ικανοποιούν τα ακόλουθα αξιώματα:

1. Το κενό σύνολο και το Χ ανήκουν στο τ.
2. Η ένωση οποιονδήποτε συλλογής συνόλων που ανήκουν στο τ, ανήκει επίσης στο τ.
3. Η τομή οποιουδήποτε ζεύγους συνόλων που ανήκουν στο τ, ανήκει επίσης στο τ.

Η συλλογή τ αποκαλείται τοπολογία στο Χ. Τα στοιχεία του Χ αποκαλούνται σημεία (points). Τα σύνολα στο τ αποκαλούνται ανοικτά σύνολα (open sets), και τα συμπληρώματα τους στο Χ αποκαλούνται κλειστά σύνολα (closed sets).

Μια γειτονιά (neighbourhood) ενός σημείου x είναι οποιοδήποτε σύνολο το οποίο έχει ένα ανοιχτό υποσύνολο που περιέχει το x. Το «σύστημα γειτονιά» (neighbourhood system) στο x αποτελείται από όλες τις γειτονιές του x.

Τυπικά, ένα υποσύνολο A ενός τοπολογικού χώρου X είναι πυκνό (dense) [9] στο X αν για οποιοδήποτε σημείο x στο X, κάθε γειτονιά του x περιλαμβάνει τουλάχιστον ένα σημείο από το A.

Η παραπάνω διαπίστωση προέκυψε από τον τοπολογικό χαρακτηρισμό του safety και του liveness που έδωσαν οι Alpern και Schneider. Σύμφωνα με την δημοσίευση[7] τους υπάρχει μια τοπολογία από το S^{ω} (το σύνολο ακολουθιών άπειρου μήκους από καταστάσεις του συστήματος) στην οποία οι safety ιδιότητες είναι ακριβώς closed σύνολα και οι liveness ιδιότητες είναι ακριβώς dense σύνολα. Τα βασικά open σύνολα αυτής της τοπολογίας είναι σύνολα όλων των εκτελέσεων που μοιράζονται ένα κοινό πρόθεμα. Ως συνήθως, ένα open σύνολο είναι η ένωση των βασικών open συνόλων, ένα closed σύνολο είναι το συμπλήρωμα ενός open συνόλου, και ένα dense σύνολο είναι αυτό που τέμνει κάθε μη κενό open σύνολο. Έτσι οι Alpern και Schneider απέδειξαν ότι

Θεώρημα: Κάθε ιδιότητα P είναι η τομή μιας safety ιδιότητας και μιας liveness ιδιότητας.

Επιπλέον απέδειξαν και ένα πιο γενικό αποτέλεσμα

Θεώρημα: Αν $|S|>1$, όπου S το σύνολο των καταστάσεων του συστήματος, τότε οποιαδήποτε ιδιότητα P είναι η τομή από δύο liveness ιδιότητες.

2.4 Κλείσιμο Μηχανής (Machine Closure)

Οι ενέργειες A, των οποίων οι fairness ιδιότητες εμφανίζονται στην έκφραση Liveness (σε μια προδιαγραφή γραμμένη στην μορφή $Init \wedge \Box Next \wedge Liveness$), έχουν ένα πράγμα από κοινού: πρέπει όλες να είναι υποενέργειες της ενέργειας επόμενης κατάστασης Next. Μια ενέργεια A είναι μια υποενέργεια της Next αν και μόνο αν κάθε A βήμα είναι ένα Next βήμα. Ισοδύναμα, η A είναι μια υποενέργεια της Next αν και μόνο αν η A συνεπάγεται την Next. Σε σχεδόν όλες τις προδιαγραφές η έκφραση Liveness θα πρέπει να είναι η σύζευξη των weak και/ή των strong fairness εκφράσεων για τις υποενέργειες της Next. Θα επεξηγήσουμε τώρα το γιατί.

Όταν βλέπουμε την προδιαγραφή, αναμένουμε το Init να περιορίζει την initial κατάσταση, το Next να περιορίζει το τι βήματα πιθανόν να συμβούν, και το Liveness να περιγράφει μόνο το τι τελικά πρέπει να συμβεί. Θεωρώντας την ακόλουθη έκφραση:

1. $(x = 0) \wedge \Box[x' = x + 1]_x \wedge \mathrm{WF}_x((x > 99) \wedge (x' = x - 1))$

Οι πρώτοι δύο συζευκτέοι της παραπάνω έκφρασης ισχυρίζονται ότι το x είναι αρχικά 0 και ότι οποιοδήποτε βήμα είτε αυξάνει το x κατά 1 ή το αφήνει αμετάβλητο. Ως εκ τούτου, αυτοί υπονοούν ότι αν το x ποτέ υπερβεί το 99, τότε αυτό για πάντα θα παραμείνει μεγαλύτερο του 99. Η weak fairness ιδιότητα (δεν είναι μια υποενέργεια της $x' = x + 1$) ισχυρίζεται ότι, αν αυτό συμβεί, τότε το x πρέπει τελικά να μειωθεί

κατά 1 –αντιφατικό για τον δεύτερο συζευκτέο. Ως εκ τούτου, η έκφραση αυτή υπονοεί ότι η x δεν μπορεί ποτέ να υπερβεί το 99, έτσι αυτό είναι ισοδύναμο με

2. $(x = 0) \wedge \Box[(x < 99) \wedge (x' = x + 1)]_x$

Συνδέοντας την weak fairness ιδιότητα με τους πρώτους δύο συζευκτέους της προηγούμενης έκφρασης(1) απαγορεύουμε ένα $x' = x + 1$ βήμα όταν το x = 99.

Μια προδιαγραφή της μορφής της έκφρασης (1) αποκαλείται machine closed αν και μόνο αν ο συζευκτέος Liveness **δεν** περιορίζει ούτε την initial κατάσταση ούτε το τι βήματα πιθανόν να συμβούν. Ένας πιο γενικός τρόπος για να εκφράσουμε αυτό είναι ο ακόλουθος. Μια πεπερασμένη συμπεριφορά σ ικανοποιεί μια safety ιδιότητα S αν και μόνο αν η συμπεριφορά λαμβάνεται προσθέτοντας άπειρα πολλά stuttering βήματα στο τέλος της σ που ικανοποιεί την S. Αν η S είναι μια safety ιδιότητα, τότε ορίζουμε το ζευγάρι από εκφράσεις S, L να είναι machine closed αν και μόνο αν κάθε πεπερασμένη συμπεριφορά που ικανοποιεί την S μπορεί να επεκταθεί σε μια άπειρη συμπεριφορά που ικανοποιεί την $S \wedge L$. Αποκαλούμε μια προδιαγραφή machine closed αν το ζεύγος των εκφράσεων $Init \wedge \Box[Next]_{vars}$ και Liveness είναι machine closed.

Σπάνια θέλουμε να γράψουμε μια προδιαγραφή που δεν είναι machine closed. Αν το κάνουμε, αυτό είναι συνήθως από λάθος. Μια προδιαγραφή εγγυάται να είναι machine closed αν η Liveness είναι η σύζευξη των weak και/ή των strong fairness ιδιοτήτων για υποενέργειες της Next. Αυτή η συνθήκη δεν διατηρείται για την προδιαγραφή του παραπάνω παραδείγματος, η οποία δεν είναι machine closed, επειδή η $(x > 99) \wedge (x' = x - 1)$ δεν είναι μια υποενέργεια της $x' = x + 1$.

Εκτός από τις ασυνήθιστες περιστάσεις, πρέπει να εκφράζεται το liveness με fairness ιδιότητες για τις υποενέργειες της ενέργειας επόμενης κατάστασης. Αυτές είναι οι πιο απλές προδιαγραφές, και ως εκ τούτου ευκολότερες να γραφούν και να κατανοηθούν. Οι περισσότερες προδιαγραφές συστήματος, ακόμα και αν είναι πολύ λεπτομερείς και περίπλοκες, μπορούν να γραφούν με αυτό τον απλό τρόπο.

Κεφάλαιο 3: TLC Model Checker

Περίληψη κεφαλαίου

Το κεφάλαιο αυτό ξεκινά με την παρουσίαση των κατηγοριών αλγορίθμων για model checking. Γίνεται μια εισαγωγή στο TLC και στην συνέχεια περιγράφεται ο τρόπος με τον οποίο το TLC εκτιμά γενικά τις εκφράσεις (χρονικές και μη), ο τρόπος με τον οποίο υπολογίζει τις καταστάσεις καθώς και το πως ελέγχει τις ιδιότητες. Στην συνέχεια παρουσιάζονται οι βασικές δομές δεδομένων που χρησιμοποιεί το TLC για την υλοποίηση των αλγορίθμων της και τέλος επεξηγείται ο αλγόριθμος που ακολουθείται στο TLC για το safety model checking αλλά και για το liveness model checking. Το κεφάλαιο αυτό αποσκοπεί στην αναλυτική παρουσίαση του TLC model checker.

3.1 Αλγόριθμοι Model Checking

Υπάρχουν δύο κατηγορίες αλγορίθμων model checking[10], το symbolic model checking και το explicit-state model checking. Το symbolic model checking είναι καλύτερο για την επαλήθευση hardware συστημάτων ενώ το explicit state model checking για την επαλήθευση συστημάτων λογισμικού. Οι hardware model checkers βασίζονται συνήθως σε μια στρατηγική συμβολικής αναζήτησης που εξετάζει τα σύνολα των καταστάσεων σε κάθε βήμα, ενώ οι model checkers λογισμικού βασίζονται συχνά στην explicit state αναζήτηση, όπου η πρόοδος γίνεται σε μια κατάσταση κάθε φορά. Το πιο σημαντικό είναι ότι οι hardware model checkers συνήθως χρησιμοποιούν ένα σύγχρονο μοντέλο εκτέλεσης, στο οποίο όλες οι διεργασίες παρουσιάζουν πρόοδο ταυτόχρονα, ενώ η εκτέλεση ενός μοντέλου σε ένα software model checker είναι συνήθως ασύγχρονη, όπου η εκτέλεση ανεξάρτητων μεταβάσεων μπορεί να πραγματοποιηθεί με όλες τις δυνατές σειρές εκτέλεσης.

Στο explicit state model checking, εκτελείται μια αναζήτηση (συνήθως dfs (depth first search, αναζήτηση σε βάθος)) στις καταστάσεις του υπό ελέγχου συστήματος. Η αναζήτηση προχωρά από κατάσταση σε κατάσταση. Η αναζήτηση εκτελείται με κατεύθυνση προς τα εμπρός, από μια κατάσταση στις επόμενες καταστάσεις (successors), εκτός και αν η στρατηγική αναζήτησης απαιτεί οπισθοδρόμηση. Επομένως μπορεί εύκολα να εκτιμηθεί ο αριθμός των καταστάσεων που συναντάει κατά την διάρκεια της επαλήθευσης. Επιπλέον, όλες οι καταστάσεις που συμμετέχουν στην αναζήτηση είναι προσβάσιμες από την αρχική κατάσταση.

Στο symbolic model checking, διατηρείται μια συμπιεσμένη αναπαράσταση του χώρου καταστάσεων. Σε κάθε στάδιο, η αναπαράσταση διατηρεί μια συλλογή από καταστάσεις. Μπορούμε να εφαρμόσουμε έναν μετασχηματισμό σε αυτό το σύνολο των καταστάσεων, για να βρούμε το σύνολο των καταστάσεων που είναι προσβάσιμες από αυτές με μια μετάβαση (forward αναζήτηση) ή το σύνολο των καταστάσεων το οποίο μπορεί να μετασχηματισθεί στο δοθέν σύνολο (backward αναζήτηση). Στην συνέχεια μπορεί να προσαρτηθεί το δοθέν σύνολο καταστάσεων ή να περιοριστεί εξαρτώμενο από την στρατηγική αναζήτησης και την ιδιότητα που είναι να ελεγχθεί.

3.2 Εισαγωγή στο TLC

Το TLC[1] είναι ένας explicit-state model checker για την TLA+ και είναι υλοποιημένος σε Java. Έχει σχεδιαστεί και υλοποιηθεί από τον Yuan Yu, με την βοήθεια των Leslie Lamport, Mark Hayden, και Mark Tuttle.
Το explicit-state model checking είναι δυνατό μόνο για πεπερασμένων καταστάσεων (bounded-state) προδιαγραφές. Επομένως το TLC δέχεται ως είσοδο μια TLA+ προδιαγραφή και ένα configuration αρχείο που ορίζουν ένα πεπερασμένο μοντέλο.
Το TLC είναι διαθέσιμο μέσω της ηλεκτρονικής τοποθεσίας (http://lamport.org). Το κεφάλαιο αυτό περιγράφει τον model checker TLC Version 2.01.

Το TLC χειρίζεται προδιαγραφές που έχουν την τυποποιημένη μορφή [Βλέπε 2.1]

$$Init \wedge \Box[Next]_{vars} \wedge Temporal$$

όπου Init είναι το αρχικό κατηγόρημα, το Next είναι η σχέση επόμενης κατάστασης, vars είναι οι μεταβλητές κατάστασης, και Temporal είναι η χρονική έκφραση που συνήθως προσδιορίζει μια liveness συνθήκη. Αν η προδιαγραφή δεν περιλαμβάνει Temporal έκφραση, τότε αυτή έχει την ακόλουθη μορφή $Init \wedge \Box[Next]_{vars}$, και επομένως μπορεί να αγνοηθεί η συζήτηση σχετικά με τον χρονικό έλεγχο κάποιας ιδιότητας. Το TLC μπορεί να εκτελεστεί χωρίς να έχει να ελέγξει κάποια ιδιότητα, στην περίπτωση ελέγχει για δύο είδη σφαλμάτων:

- "Silliness" σφάλματα. Μια χωρίς νόημα έκφραση (silly expression) είναι για παράδειγμα η $3 + \langle 1, 2 \rangle$, της οποίας η έννοια δεν καθορίζεται από την σημασιολογία της TLA+. Μια προδιαγραφή είναι λανθασμένη αν εξαρτάται από μια χωρίς νόημα έκφραση.
- Deadlock. Η απουσία αδιεξόδου είναι μια ιδιαίτερη ιδιότητα που συχνά επιθυμείται να ικανοποιεί μια προδιαγραφή, αυτή εκφράζεται από την ιδιότητα invariance $\Box(\text{ENABLED } Next)$. Ένα αντιπαράδειγμα για αυτήν την ιδιότητα είναι μια συμπεριφορά που επιδεικνύει αδιέξοδο δηλαδή το να φθάσει σε μια κατάσταση στην οποία το Next δεν είναι επιτρεπτό, επομένως κανένα περαιτέρω βήμα είναι δυνατό. Το TLC κανονικά ελέγχει για αδιέξοδο, αλλά αυτός ο έλεγχος μπορεί να είναι απενεργοποιημένος αφού για κάποια συστήματα το αδιέξοδο υποδεικνύει επιτυχή τερματισμό.

3.3 Εκτίμηση εκφράσεων

Ο έλεγχος μια προδιαγραφής απαιτεί εκτίμηση των εκφράσεων. Για παράδειγμα, το TLC κάνει invariance έλεγχο εκτιμώντας την invariant έκφραση σε κάθε προσβάσιμη κατάσταση δηλαδή, υπολογίζει την τιμή του κατηγορήματος, η οποία πρέπει να είναι TRUE. Για να γίνει αντιληπτό τι κάνει και τι δεν κάνει το TLC, πρέπει να κατανοηθεί το πώς αυτό εκτιμά τις εκφράσεις.
Το TLC εκτιμά τις εκφράσεις με ένα συγκεκριμένο τρόπο , γενικά εκτιμά υποεκφράσεις "από τα αριστερά προς τα δεξιά". Συγκεκριμένα:

- Εκτιμά την $p \wedge q$ εκτιμώντας πρώτα το p και, αν αυτό είναι TRUE, τότε εκτιμά το q.

- Εκτιμά την $p \vee q$ εκτιμώντας πρώτα το p και, αν αυτό είναι ίσο με FALSE, τότε εκτιμά το q. Εκτιμά την $p \Rightarrow q$ σαν $\neg p \vee q$.
- Εκτιμά την $\text{IF } p \text{ THEN } e_1 \text{ ELSE } e_2$ εκτιμώντας πρώτα το p και μετά εκτιμά είτε το e1 ή το e2.

Για να γίνει αντιληπτή η σημαντικότητα αυτών των κανόνων, ας μελετήσουμε ένα απλό παράδειγμα. Το TLC δεν μπορεί να εκτιμήσει την έκφραση x [1] αν το x είναι ίσο με $\langle\ \rangle$, αφού η $\langle\ \rangle$[1] δεν έχει νόημα. (Η άδεια ακολουθία $\langle\ \rangle$ είναι μια συνάρτηση της οποίας το πεδίο ορισμού είναι ένα κενό σύνολο και επομένως δεν περιέχει το 1.) Ο πρώτος κανόνας υπονοεί ότι, αν το x είναι ίσο με το $\langle\ \rangle$, τότε το TLC μπορεί να εκτιμήσει την έκφραση

$$(x \neq \langle\ \rangle) \wedge (x[1] = 0)$$

Αλλά όχι την (λογικά ισοδύναμη) έκφραση

$$(x[1] = 0) \wedge (x \neq \langle\ \rangle)$$

(Όταν εκτιμάται η τελευταία έκφραση, το TLC πρώτα προσπαθεί να υπολογίσει την $\langle\ \rangle$[1] = 0, αναφέροντας ένα σφάλμα επειδή δεν μπορεί να το κάνει.) Ευτυχώς, γενικά επιλέγεται να γραφεί η πρώτη έκφραση από την δεύτερη γιατί είναι ευκολότερη στο να γίνει αντιληπτή. Οι άνθρωποι καταλαβαίνουν μια έκφραση εκτιμώντας αυτήν διανοητικά από τα αριστερά προς τα δεξιά, με το ίδιο τρόπο που το κάνει και το TLC.

Το TLC εκτιμά την $\exists x \in S : p$ απαριθμώντας τα στοιχεία s1,…, sn του S με κάποια σειρά και μετά εκτιμά το p με το si να αντικαθιστά το x , διαδοχικώς για i = 1,…, n. Αυτό απαριθμεί τα στοιχεία του συνόλου S με ένα πολύ ευθύ τρόπο, και σταματά και δηλώνει ένα σφάλμα αν το σύνολο δεν είναι προφανώς πεπερασμένο.

Για παράδειγμα, το TLC μπορεί να απαριθμήσει τα στοιχεία από το $\{0, 1, 2, 3\}$ και το $0 \,..\, 3$. Αυτό απαριθμεί ένα σύνολο της μορφής $\{x \in S : p\}$ απαριθμώντας πρώτα το S, και έτσι αυτό μπορεί να απαριθμήσει το $\{i \in 0 \,..\, 5 : i < 4\}$ αλλά όχι το $\{i \in Nat : i < 4\}$.

Το TLC εκτιμά τις εκφράσεις $\forall x \in S : p$ και $\text{CHOOSE } x \in S : p$ (η οποία επιστρέφει μια αυθαίρετα επιλεγμένη τιμή x του S που ικανοποιεί την έκφραση p) απαριθμώντας πρώτα τα στοιχεία του S, με τον ίδιο τρόπο όπως αυτό εκτιμά την $\exists x \in S : p$. Η σημασιολογία της TLA+ αναφέρει ότι η $\text{CHOOSE } x \in S : p$ είναι μια αυθαίρετη τιμή αν δεν υπάρχει κανένα x στο S για το οποίο το p είναι true. Ωστόσο, αυτή η περίπτωση σχεδόν πάντα προκαλεί ένα λάθος, έτσι το TLC χειρίζεται αυτό ως ένα σφάλμα. Να σημειωθεί ότι η εκτίμηση της έκφρασης

$$\text{IF } n > 5 \text{ THEN CHOOSE } i \in 1 \,..\, n : i > 5 \text{ ELSE } 42$$

δεν θα αναπαράγει σφάλμα γιατί το TLC δεν θα εκτιμήσει την έκφραση CHOOSE αν $n \leq 5$. (Το TLC θα ανέφερε ένα σφάλμα αν προσπαθούσε να εκτιμήσει την CHOOSE έκφραση όταν το $n \leq 5$.)

Το TLC δεν μπορεί να εκτιμήσει μη οριοθετημένους "unbounded" ποσοδείκτες ή εκφράσεις CHOOSE δηλαδή, τις εκφράσεις που έχουν μια από τις ακόλουθες μορφές

$$\exists x : p \qquad \forall x : p \qquad \text{CHOOSE } x : p$$

Συγκεκριμένα, το TLC μπορεί να εκτιμήσει μια set-valued έκφραση μόνο αν αυτή η έκφραση ισούται με ένα πεπερασμένο σύνολο, και μπορεί να εκτιμήσει μια function-valued έκφραση μόνο αν η έκφραση είναι μια συνάρτηση της οποίας το πεδίο ορισμού

είναι ένα πεπερασμένο σύνολο. Το TLC μπορεί να εκτιμήσει εκφράσεις των ακόλουθων μορφών μόνο αν μπορεί να απαριθμήσει το σύνολο S:

$\exists x \in S : p$ $\quad \forall x \in S : p$ $\quad \text{CHOOSE } x \in S : p$
$\{x \in S : p\}$ $\quad \{e : x \in S\}$ $\quad [x \in S \mapsto e]$
$\text{SUBSET } S$ $\quad \text{UNION } S$

3.4 Εκτίμηση Χρονικών Εκφράσεων

Η προδιαγραφή και οι ιδιότητες που το TLC ελέγχει είναι χρονικές εκφράσεις. Στο σημείο αυτό περιγράφεται το σύνολο των χρονικών εκφράσεων που αυτό μπορεί να χειριστεί.

Μια χρονική έκφραση χαρακτηρίζεται ως nice αν και μόνο αν αυτή είναι μια σύζευξη από εκφράσεις που ανήκουν σε μια από τις ακόλουθες τέσσερις κλάσεις:

Κατηγόρημα Κατάστασης (State Predicate). Μια συνάρτηση κατάστασης που λαμβάνει booolean τιμές.
Invariance Έκφραση. Μια έκφραση της μορφής $\Box P$, όπου P είναι ένα κατηγόρημα κατάστασης.
Box-Action Έκφραση. Μια έκφραση της μορφής $\Box[A]_v$, όπου A είναι μια ενέργεια και υ είναι μια συνάρτηση κατάστασης.
Απλή Χρονική Έκφραση. Για να οριστεί αυτή η κλάση, πρέπει πρώτα να θέσουμε τους ακόλουθους ορισμούς:

- Οι *απλοί Boolean* τελεστές αποτελούνται από τους τελεστές
  $\wedge$ $\quad \vee$ $\quad \neg$ $\quad \Rightarrow$ $\quad \equiv$ $\quad \text{TRUE}$ $\quad \text{FALSE}$
  της προτασιακής λογικής μαζί με τον καθορισμό της ποσότητας (quantification) πάνω σε πεπερασμένα, σταθερά σύνολα.

- Μια *χρονική έκφραση κατάστασης (temporal state formula)* είναι αυτή που λαμβάνεται από τα κατηγορήματα κατάστασης εφαρμόζοντας απλούς Boolean τελεστές και τους χρονικούς τελεστές $\Box$, $\Diamond$ και $\rightsquigarrow$. Για παράδειγμα, αν το N είναι μια σταθερά, τότε
  $$\forall i \in 1 \,..\, N : \Box((x = i) \Rightarrow \exists j \in 1 \,..\, i : \Diamond(y = j))$$
  είναι μια χρονική έκφραση κατάστασης.
- Μια *απλή έκφραση ενέργειας (simple action formula)* είναι μια έκφραση που μπορεί να πάρει τις ακόλουθες μορφές:
  $\text{WF}_v(A)$ $\quad \text{SF}_v(A)$ $\quad \Box\Diamond\langle A\rangle_v$ $\quad \Diamond\Box[A]_v$
  όπου το A είναι μια ενέργεια και το υ είναι μια συνάρτηση κατάστασης. Οι συνθετικές εκφράσεις της $\text{WF}_v(A)$ και της $\text{SF}_v(A)$ είναι η $\langle A\rangle_v$ και η $\text{ENABLED } \langle A\rangle_v$ [βλέπε 1.6 και 1.7].

Επομένως μια απλή χρονική έκφραση ορίζεται να είναι μια έκφραση που δομείται από χρονικές εκφράσεις κατάστασης και απλές εκφράσεις ενέργειας εφαρμόζοντας απλούς Boolean τελεστές.

Το TLC **μπορεί να εκτιμήσει μια TLA χρονική έκφραση αν και μόνο αν** (i) η έκφραση είναι nice και (ii) το TLC μπορεί να εκτιμήσει όλες τις εκφράσεις από τις

οποίες συντίθεται η έκφραση. Για παράδειγμα, μια έκφραση της μορφής $P \rightsquigarrow Q$ είναι nice, έτσι το TLC μπορεί να εκτιμήσει αυτή αν και μόνο αν αυτό μπορεί να εκτιμήσει την P και την Q.

Το TLC μπορεί επομένως να εκτιμήσει την χρονική έκφραση
$\forall i \in 1 \,..\, N \,:\, \Diamond(y = i) \Rightarrow \mathrm{WF}_y((y' = y + 1) \wedge (y \geq i))$
αν το N είναι μια σταθερά, επειδή αυτή είναι μια απλή χρονική έκφραση (και επομένως nice) και το TLC μπορεί να εκτιμήσει όλες τις συνθετικές εκφράσεις. Το TLC δεν μπορεί να εκτιμήσει την $\Diamond\langle x' = 1\rangle_x$, αφού αυτή δεν είναι μια nice έκφραση. Αυτό δεν μπορεί να εκτιμήσει την έκφραση $\mathrm{WF}_x(x'[1] = 0)$ αν πρέπει να εκτιμήσει την ενέργεια $\langle x'[1] = 0\rangle_x$ με ένα βήμα $s \rightarrow t$ στο οποίο το $x = \langle\,\rangle$ στην κατάσταση t .

3.5 Υπολογισμός καταστάσεων

Όταν το TLC εκτιμά μια invariant, υπολογίζει την τιμή της invariant έκφρασης, η οποία είναι είτε TRUE ή FALSE. Όταν το TLC εκτιμά το initial κατηγόρημα ή την ενέργεια επόμενης κατάστασης, υπολογίζει ένα σύνολο από καταστάσεις για το αρχικό κατηγόρημα, το σύνολο όλων των αρχικών καταστάσεων, και για την ενέργεια επόμενης κατάστασης, το σύνολο των δυνατών επόμενων καταστάσεων -*successor states* (primed states) που είναι προσβάσιμες από μια δοθείσα (unprimed) κατάσταση εκκίνησης. Στην συνέχεια θα περιγραφεί πως το TLC κάνει αυτό για την ενέργεια επόμενης κατάστασης, η εκτίμηση του αρχικού κατηγορήματος είναι ανάλογη.

Όπως έχουμε αναφέρει μια κατάσταση είναι μια ανάθεση τιμών σε μεταβλητές. **Το TLC υπολογίζει τις successors από μια δοθείσα κατάσταση s αναθέτοντας σε όλες τις unprimed μεταβλητές τις τιμές τους στην κατάσταση s, και στην συνέχεια εκτιμά την ενέργεια επόμενης κατάστασης.** Το TLC εκτιμά την ενέργεια επόμενης κατάστασης όπως την εκτίμηση των εκφράσεων που περιγράφηκε πιο πάνω, με δύο εξαιρέσεις που περιγράφουμε στην συνέχεια. Αυτή η περιγραφή θεωρεί ότι το TLC έχει ήδη εκτελέσει όλες τις αναθέσεις και τις αντικαταστάσεις που προσδιορίστηκαν από την CONSTANT εντολή (με την οποία ορίζεται μια constant παράμετρο) στο configuration αρχείο. Έτσι, η ενέργεια επόμενης κατάστασης είναι μια έκφραση που περιλαμβάνει μόνο μεταβλητές, primed μεταβλητές, model τιμές, και ενσωματωμένους TLA+ τελεστές και σταθερές.

Η πρώτη διαφορά στην εκτίμηση της ενέργειας επόμενης κατάστασης είναι ότι το TLC δεν εκτιμά τις διαζεύξεις από τα αριστερά στα δεξιά. Αντί αυτού, όταν αυτό εκτιμά μια υποέκφραση $A_1 \vee \ldots \vee A_n$, αυτό χωρίζει τον υπολογισμό σε n ξεχωριστές εκτιμήσεις, κάθε μια από τις οποίες λαμβάνει μια υποέκφραση που αντιστοιχεί σε μία από τις A_i . Παρομοίως, όταν αυτό εκτιμά την $\exists x \in S : p$, αυτό χωρίζει τον υπολογισμό σε ξεχωριστές εκτιμήσεις για κάθε ένα στοιχείο του S. Μια συνεπαγωγή

$P \Rightarrow Q$ υπολογίζεται σαν την διάζευξη $(\neg P) \lor Q$. Για παράδειγμα, το TLC χωρίζει την εκτίμηση της

$$(A \Rightarrow B) \lor (C \land (\exists\, i \in S : D(i)) \land E)$$

σε ξεχωριστές εκτιμήσεις από τρείς διαζευκτέους, από τους οποίους αποτελείται η παραπάνω έκφραση, δηλαδή, $\neg A, B$ και $C \land (\exists\, i \in S : D(i)) \land E$.

Για να εκτιμηθεί ο τελευταίος διαζευκτεός από τους τρείς διαζευκτέους, πρώτα εκτιμάται το C. Αν ληφθεί η τιμή TRUE, τότε χωρίζει αυτή την εκτίμηση σε ξεχωριστές εκτιμήσεις της $D(i) \land E$, για κάθε i μέσα στο S. Εκτιμά την $D(i) \land E$ εκτιμώντας πρώτα το D(i) και, αν αυτή λάβει την τιμή TRUE, τότε εκτιμά το E.

Η δεύτερη διαφορά στο τρόπο με τον οποίο το TLC εκτιμά την ενέργεια επόμενης κατάστασης είναι ότι, για οποιοδήποτε μεταβλητή x, αν αυτό εκτιμά μια έκφραση της μορφής $x' = e$ όταν δεν έχει ανατεθεί μια τιμή στο x', τότε η εκτίμηση παράγει την τιμή TRUE και το TLC αναθέτει στο x' την τιμή που αποκτήθηκε από την εκτίμηση της έκφρασης e. Το TLC εκτιμά μια έκφραση της μορφής $x' \in S$ σαν να ήταν $\exists\, v \in S : x' = v$. Εκτιμά την UNCHANGED x σαν $x' = x$ για οποιαδήποτε μεταβλητή x, και την $\text{UNCHANGED } \langle e_1, \ldots, e_n \rangle$ σαν

$$(\text{UNCHANGED } e_1) \land \ldots \land (\text{UNCHANGED } e_n)$$

για οποιαδήποτε έκφραση e_i . Επομένως, το TLC εκτιμά την $\text{UNCHANGED } \langle x, \langle y, z \rangle \rangle$ σαν να ήταν

$$(x' = x) \land (y' = y) \land (z' = z)$$

Σε περίπτωση που εκτιμάται μια έκφραση της μορφής $x' = e$, το TLC αναφέρει ένα σφάλμα αν αυτό απαριθμήσει μια primed μεταβλητή που δεν τις έχει ανατεθεί ακόμα μια τιμή. Μια εκτίμηση σταματά, όταν δεν βρεθούν καταστάσεις, αν ένας συζευκτέος εκτιμηθεί σε false. Μια εκτίμηση που ολοκληρώνεται και λάβει την τιμή true βρίσκει την κατάσταση που προσδιορίζεται από τις τιμές που ανατέθηκαν στις primed μεταβλητές. Στην τελευταία περίπτωση, το TLC αναφέρει ένα σφάλμα αν σε κάποια primed μεταβλητή δεν έχει ανατεθεί μια τιμή.

Για να διευκρινίσουμε πώς αυτό λειτουργεί, ας μελετήσουμε πως το TLC εκτιμά την ενέργεια επόμενης κατάστασης

$$\begin{array}{l} \lor \land\ x' \in 1\ ..\ Len(y) \\ \quad \land\ y' = Append(Tail(y), x') \\ \lor \land\ x' = x + 1 \\ \quad \land\ y' = Append(y, x') \end{array}$$

Έστω θεωρούμε κατάσταση εκκίνησης με τιμές $x = 1$ και $y = \langle 2, 3 \rangle$. Το TLC χωρίζει τον υπολογισμό εκτιμώντας τους δύο διαζευκτέους χωριστά (κάθε διαζευκτέος αποτελείται από δύο συζευκτούς). Ξεκινά εκτιμώντας τον πρώτο διαζευκτέο της εκτιμώντας τον πρώτο συζευκτέο, ο οποίος χειρίζεται ως $\exists\, i \in 1\, ..\, Len(y) : x' = i$. Αφού Len(y) = 2, η εκτίμηση του πρώτου διαζευκτέου χωρίζεται σε δυο ξεχωριστές εκτιμήσεις :

$$\begin{array}{ll} \wedge\ x' = 1 & \wedge\ x' = 2 \\ \wedge\ y' = Append(Tail(y), x') & \wedge\ y' = Append(Tail(y), x') \end{array}$$

Το TLC εκτιμά την πρώτη από αυτές τις ενέργειες ως εξής. Εκτιμά τον πρώτο συζευκτέο, αναθέτοντας στην x' την τιμή 1. Στην συνέχεια εκτιμά τον δεύτερο συζευκτέο, αναθέτοντας στην y' την τιμή $Append(Tail(\langle 2, 3 \rangle), 1)$. Επομένως, εκτιμώντας την πρώτη ενέργεια βρίσκει την successor κατάσταση με $x = 1$ και $y = \langle 3, 1 \rangle$. Παρόμοια, εκτιμώντας την δεύτερη ενέργεια βρίσκει την successor κατάσταση με $x = 2$ και $y = \langle 3, 2 \rangle$. Με παρόμοιο τρόπο, το TLC εκτιμά τον δεύτερο διαζευκτέο της ενέργειας επόμενης κατάστασης για να βρει την successor κατάσταση με $x = 2$ και $y = \langle 2, 3, 2 \rangle$. Επομένως, η εκτίμηση της αρχικής ενέργειας επόμενης κατάστασης βρίσκει τρείς successor καταστάσεις.

Το TLC υπολογίζει τις initial καταστάσεις χρησιμοποιώντας μια παρόμοια διαδικασία για να εκτιμήσει το initial κατηγόρημα. Αντί να ξεκινήσει από τις δοθείσες τιμές των unprimed μεταβλητών (π.χ. x,y) και να αναθέτει τιμές σε primed μεταβλητές (π.χ. x′,y′) , αυτό αναθέτει τιμές στις unprimed μεταβλητές.

Το TLC είναι επίσης πιθανόν να πρέπει να εκτιμήσει μια ενέργεια όταν ελέγχει για μια ιδιότητα. Σε αυτήν τη περίπτωση, εκτιμά την ενέργεια όπως θα έκανε με οποιαδήποτε έκφραση.

3.6 Έλεγχος ιδιοτήτων

Μέχρι αυτό το σημείο έχουμε επεξηγήσει πως το TLC εκτιμά τις εκφράσεις και υπολογίζει τις initial καταστάσεις και τις successor καταστάσεις. Σε αυτήν την ενότητα περιγράφουμε πως το TLC χρησιμοποιεί την εκτίμηση για να ελέγξει τις ιδιότητες.

Όπως ήδη έχουμε αναφέρει η είσοδος στο TLC αποτελείται από μια TLA+ module και ένα configuration αρχείο [Βλέπε 1.3]. Το TLC θεωρεί ότι η προδιαγραφή έχει την μορφή

$$Init \wedge \Box[Next]_{vars} \wedge Temporal$$

Το configuration αρχείο ορίζει το πεπερασμένων καταστάσεων στιγμιότυπο του μοντέλου που είναι να αναλυθεί, και δηλώνει ποιες από τις εκφράσεις που ορίζονται μέσα στο μοντέλο αναπαριστούν την προδιαγραφή του συστήματος και ποιες είναι οι

ιδιότητες που είναι να επαληθευτούν πάνω σε αυτό το πεπερασμένων καταστάσεων στιγμιότυπο.

Για παράδειγμα, το configuration αρχείο για το HourClock όπως έχουμε δείξει περιλαμβάνει την δήλωση

SPECIFICATION HC

Μέσω της οποίας το TLC λαμβάνει την HC ως την προδιαγραφή που είναι να εξεταστεί.

Η ιδιότητα ή οι ιδιότητες που πρέπει να ελεγχθούν προσδιορίζονται στο ίδιο αρχείο με μια εντολή PROPERTY.

Προσδιορίζοντας ένα κατηγόρημα κατάστασης P σε μια PROPERTY εντολή λέμε στο TLC να ελέγξει ότι η προδιαγραφή συνεπάγεται το P, που σημαίνει ότι το P είναι αληθές σε κάθε initial κατάσταση κάθε συμπεριφοράς που ικανοποιεί την προδιαγραφή.

Σε αυτό το σημείο ορίζουμε κάποιες εκφράσεις που λαμβάνονται από το configuration αρχείο για την πραγματοποίηση του model checking. Σε αυτούς τους ορισμούς, ένας συζευκτέος προδιαγραφής (specification conjunct) είναι ένας συζευκτέος της έκφρασης που προσδιορίζεται από την εντολή SPECIFICATION, ένας συζευκτέος ιδιότητας (property conjunct) είναι ένας συζευκτέος από μια έκφραση που προσδιορίζεται από μια εντολή PROPERTY. Οι ορισμοί αυτοί χρησιμοποιούν τις τέσσερις κλάσεις των nice χρονικών εκφράσεων που ορίστηκαν πιο πάνω στην ενότητα 3.4.

Init
Το κατηγόρημα που προσδιορίζει την αρχική κατάσταση της προδιαγραφής. Αυτό το κατηγόρημα προσδιορίζεται από μια INIT ή μια SPECIFICATION εντολή. Στην τελευταία περίπτωση, είναι η σύζευξη από όλους τους συζευκτέους προδιαγραφής που είναι κατηγορήματα κατάστασης.
Έτσι για παράδειγμα αν η προδιαγραφή που εξετάζουμε έχει την μορφή

$$Init \wedge \Box[Next]_{vars}$$

χωρίς καμία liveness συνθήκη, τότε αντί να χρησιμοποιήσουμε μια SPECIFICATION εντολή, μπορούμε να ορίσουμε το initial κατηγόρημα και την ενέργεια επόμενης κατάστασης τοποθετώντας τις ακόλουθες δύο εντολές στο configuration αρχείο:

INIT Init

NEXT Next

Next
Η ενέργεια επόμενης κατάστασης της προδιαγραφής. Προσδιορίζεται από μια εντολή NEXT ή από μια εντολή SPECIFICATION. Στην τελευταία περίπτωση, αυτή είναι μια ενέργεια N τέτοια που υπάρχει ένας συζευκτέος προδιαγραφής της μορφής $\Box[N]_v$. Δεν πρέπει να υπάρχουν περισσότεροι του ενός τέτοιοι συζευκτέοι.

Temporal

Η σύζευξη κάθε συζευκτέου προδιαγραφή που δεν είναι ούτε κατηγόρημα κατάστασης ούτε box-action έκφραση. Αυτή είναι συνήθως η liveness συνθήκη της προδιαγραφής. Για παράδειγμα έστω ότι θέλουμε να προσδιορίσουμε ένα ρολόι στο οποίο θέλουμε να δηλώσουμε ότι ένα HCnxt βήμα πρέπει τελικά να συμβεί αν η HCnxt είναι συνεχώς ενεργοποιημένη χωρίς διακοπή. Ένα HCnxt βήμα είναι ένα βήμα που αλλάζει την hr όπως έχουμε αναφέρει [Βλέπε 1.3]:

Στο module γράφουμε
```
HCini  ==  hr \in (1 .. 12)
HCnxt  ==  hr' = IF hr # 12 THEN hr + 1 ELSE 1
HC == HCini/\ [][HCnxt]_hr /\ WF_hr(HCnxt)
```

Στο configuration αρχείο γράφουμε
```
SPECIFICATION HC
```

Επομένως η Temporal έκφραση είναι η WF_hr(HCnxt) που αποτελεί την liveness συνθήκη για την προδιαγραφή HC(HourClock). Ο ισχυρισμός συνθηκών fairness μπορεί να αποκλείσει το άπειρο stuttering.

Invariant

Η σύζευξη από κάθε κατηγόρημα κατάστασης I που είτε προσδιορίζεται από μια εντολή INVARIANT ή χάριν κάποιας ιδιότητα συζευκτέου ίση με $\Box I$. Για παράδειγμα έστω ότι θέλουμε η μεταβλητή hr να παίρνει τιμές από το 1 έως το 12 και θέλουμε αυτό να είναι ισχύει πάντα (να είναι πάντα αληθές). Υπάρχουν δύο τρόποι όπως αναφέραμε πιο πάνω:

Α′ τρόπος	**Β′ τρόπος**

Στο module γράφουμε
`HCini  ==  hr \in (1 .. 12)`
 	
Στο configuration αρχείο γράφουμε
`INVARIANT` `HCini`
	Στο module γράφουμε
	`HCini  ==  hr \in (1 .. 12)`
	`TypeInvariance == []HCini`
	
	Στο configuration αρχείο γράφουμε
	`PROPERTY` `TypeInvariance`

ImpliedInit

Η σύζευξη από κάθε συζευκτέο ιδιότητας που είναι ένα κατηγόρημα κατάστασης.

Στην πράξη είναι πιθανό να κληθούμε να ελέγξουμε την ορθότητα ενός συστήματος χωρίς να έχουμε καμία επίσημη προδιαγραφή του τι ακριβώς το σύστημα υποτίθεται ότι πρέπει να κάνει. Σε αυτήν την περίπτωση, μπορούμε να γράψουμε μια εκ των υστέρων (ex post facto) προδιαγραφή. Στην πραγματικότητα πρόκειται για μια απλοποιημένη έκδοση της εξεταζόμενης προδιαγραφής. Ο έλεγχος μας εστιάζεται στο αν η αρχική εξεταζόμενη προδιαγραφή συνεπάγεται/ ικανοποιεί την ex post facto προδιαγραφή.

Έστω για παράδειγμα για την προδιαγραφή HourClock ισχύει ότι δεν έχουμε μια επίσημη προδιαγραφή τότε δημιουργούμε την ex post facto προδιαγραφή HCCorrectness, που είναι μια πιο απλοποιημένη προδιαγραφή από την HourClock, και θέλουμε να ελέγξουμε αν η HourClock συνεπάγεται την HCCorrectness. Αυτός ο έλεγχος μπορεί να γίνει με τον ακόλουθο τρόπο στο configuration αρχείο της HourClock:

Στο module γράφουμε
```
INSTANCE HCCorrectness
```

Στο configuration αρχείο γράφουμε
```
PROPERTY HCCSpec
```

Η `HCCSpec` αποτελεί την χρονική έκφραση που περιγράφει την `HCCorrectness` και είναι ίση με

```
HCCSpec == HCiniC /\ [][HCnxtC]_hr
```

όπου `HCiniC` το αρχικό κατηγόρημα το οποίο ορίζει την αρχική κατάσταση του συστήματος της HCCorrectness και `HCnxtC` η σχέση επόμενης κατάστασης η οποία ορίζει μόνο τις δυνατές λειτουργίες του συστήματος της HCCorrectness.

Έτσι ελέγχοντας αν η HourClock ικανοποιεί την `HCCSpec` ελέγχουμε αν η HourClock συνεπάγεται την HCCorrectness.

Συγκεκριμένά αν μας ενδιάφερε μόνο να ελέγξουμε αν ισχύει το αρχικό κατηγόρημα `HCiniC` της HCCorrectness θα μπορούσαμε να γράψουμε το εξής

Στο module γράφουμε
```
INSTANCE HCCorrectness
```

Στο configuration αρχείο γράφουμε
```
PROPERTY HCiniC
```

ImpliedAction

Η σύζευξη από κάθε ενέργεια $[A]_u$ τέτοια ώστε κάποιος συζευκτέος ιδιότητας να ισοδυναμεί με $\Box[A]_u$.

Το παράδειγμα που παρουσιάσαμε πιο πάνω στην έκφραση ImpliedInit μπορούμε να το εκμεταλλευτούμε και για την καλύτερη κατανόηση της ImpliedAction.

Συγκεκριμένα όμως αν μας ενδιάφερε μόνο να ελέγξουμε αν ισχύει η σχέση επόμενης κατάστασης `HCnxtC` της HCCorrectness θα μπορούσαμε να γράψουμε το εξής

Στο module γράφουμε
```
INSTANCE HCCorrectness
ImpliedAction == [][HCnxtC]_hr
```

Στο configuration αρχείο γράφουμε
```
PROPERTY ImpliedAction
```

ImpliedTemporal

Η σύζευξη κάθε συζευκτέου ιδιότητας που είναι μια απλή χρονική έκφραση, αλλά δεν είναι της μορφής □I , όπου I είναι ένα κατηγόρημα κατάστασης. Αποτελεί και αυτήν μια liveness ιδιότητα. Έστω ότι θέλουμε να ορίσουμε ότι στο HourClock συμβαίνουν άπειρα <<HCnxt>>_hr βήματα, που αλλάζουν την hr:

Στο module γράφουμε

```
HCnxt  ==  hr' = IF hr # 12 THEN hr + 1 ELSE 1
AlwaysTick == []<><<HCnxt>>_hr

THEOREM  HC => AlwaysTick
```

Στο configuration αρχείο γράφουμε

```
SPECIFICATION HC
PROPERTY AlwaysTick
```

Το `[]<><<HCnxt>>_hr` αποτελεί την ImpliedTemporal έκφραση της προδιαγραφής.

Constraint

Η σύζευξη των κατηγορημάτων κατάστασης που προσδιορίζονται από εντολές CONSTRAINT.

Όταν θέλουμε να περιορίσουμε ένα μοντέλο για να το κάνουμε πεπερασμένο, δηλαδή, να επιτρέπει μόνο έναν πεπερασμένο αριθμό από τις δυνατές καταστάσεις, μπορούμε να προσδιορίσουμε ένα κατηγόρημα κατάστασης που αποκαλείται constraint το οποίο θα περιορίζει το μήκος των ακολουθιών.

Για παράδειγμα, έστω ότι θέλουμε να περιορίσουμε το μήκος κάποιων ακολουθιών έτσι ώστε να έχουν το πολύ μήκος 2 (`\leq` είναι η ASCII αναπαράσταση του συμβόλου **<=**) .

Στο module γράφουμε

```
SeqConstraint == /\ Len(msgQ) \leq 2
                 /\ Len(ackQ) \leq 2
```

Στο configuration αρχείο γράφουμε

```
CONSTRAINT SeqConstraint
```

ActionConstraint

Η σύζευξη από όλες τις ενέργειες που προσδιορίζονται από εντολές ACTION-CONSTRAINT. Ένα constraint ενέργειας είναι παρόμοιο με ένα constraint, με την διαφορά ότι αυτό εξαλείφει τις δυνατές μεταβάσεις παρά τις καταστάσεις.

Όταν το TLC ελέγχει μια ιδιότητα, στην πραγματικότητα δεν επαληθεύει ότι η προδιαγραφή υπονοεί την ιδιότητα. Αντί αυτού, ελέγχει ότι (i) το safety μέρος της προδιαγραφής συνεπάγεται το safety μέρος της ιδιότητας και (ii) η προδιαγραφή συνεπάγεται το liveness μέρος της ιδιότητας. Για παράδειγμα, υποθέτοντας ότι η προδιαγραφή Spec και η ιδιότητα Prop είναι

$$Spec \triangleq Init \wedge \Box[Next]_{vars} \wedge Temporal$$
$$Prop \triangleq ImpliedInit \wedge \Box[ImpliedAction]_{pvars} \wedge ImpliedTemporal$$

όπου Temporal και ImpliedTemporal είναι liveness ιδιότητες. Σε αυτήν την περίπτωση, το TLC ελέγχει τις δύο εκφράσεις

$$Init \wedge \Box[Next]_{vars} \Rightarrow ImpliedInit \wedge \Box[ImpliedAction]_{pvars}$$
$$Spec \Rightarrow ImpliedTemporal$$

Αυτό σημαίνει ότι δεν μπορεί να χρησιμοποιηθεί το TLC για να ελέγξει ότι μια non-machine-closed προδιαγραφή ικανοποιεί την safety ιδιότητα. Γιατί τότε ο Liveness συζευκτέος θα περιόριζε την initial κατάσταση ή/και τα βήματα με αποτέλεσμα ο έλεγχος του TLC να είναι λάθος [Βλέπε 2.4(Κλείσιμο Μηχανής)].

Στην πράξη, σχεδόν όλα τα αντιδραστικά (reactive) συστήματα μπορούν να προσδιοριστούν χρησιμοποιώντας συστήματα ενεργειών (action systems) μαζί με απλές weak και strong fairness ιδιότητες. Οι περισσότερες προδιαγραφές είναι machine closed. **Μια machine-closed προδιαγραφή μπορεί πάντα να γραφτεί σαν ένα σύστημα ενεργειών μαζί με fairness ιδιότητες μόνο σε διαζεύξεις των ενεργειών συστήματος[5].**

3.7 Βασικές δομές δεδομένων του TLC

Το TLC στην μέθοδο model checking προσπαθεί να βρει όλες τις προσβάσιμες (reachable) καταστάσεις, δηλαδή, όλες τις καταστάσεις που μπορούν να εμφανιστούν στις συμπεριφορές που ικανοποιούν την έκφραση

$$Init \wedge \Box[Next]_{vars}$$

Για την υλοποίηση της μεθόδου model checking το TLC διατηρεί τις ακόλουθες τρεις δομές δεδομένων (η τρίτη δημιουργείται μόνο για την υλοποίηση του liveness model checking):

- Το **σύνολο G** που είναι μέρος του γράφου reachability κατάστασης που το TLC έχει υπολογίσει μέχρι την τρέχουσα χρονική στιγμή. Στην ουσία περιλαμβάνει τα fingerprints των καταστάσεων. Τα fingerprints του είναι στοχαστικά μοναδικά αθροίσματα ελέγχου (checksum) που ουσιαστικά είναι κωδικοί (hash-code) των 64bits.

  Ο αλγόριθμος που ακολουθεί το TLC αναπαριστά το G σαν μια ένωση από δύο ασύνδετα (disjoint) σύνολα καταστάσεων, το ένα κρατείτε σαν ένα in-memory hash table και το άλλο σαν ένα ταξινομημένο αρχείο δίσκου που έχει ένα in-memory index. Για να ελεγχθεί αν μια κατάσταση είναι μέσα στο G, το TLC πρώτα ελέγχει τον in-memory table. Αν η κατάσταση δεν είναι εκεί, το TLC χρησιμοποιεί μια δυαδική αναζήτηση στο in-memory index για να βρει το μπλοκ του δίσκου που

πιθανόν να περιέχει αυτήν την κατάσταση, διαβάζει αυτό το μπλοκ, και χρησιμοποιεί δυαδική αναζήτηση για να δει αν η κατάσταση είναι στο μπλοκ. Για να προσθέσει μια κατάσταση στο σύνολο, το TLC την προσθέτει στον in-memory table. Όταν ο πίνακας είναι γεμάτος, τα περιεχόμενα του ταξινομούνται και συγχωνεύονται με το αρχείο δίσκου, και το in-memory index του αρχείου ενημερώνεται. Πρόσβαση στο αρχείο του δίσκου από πολλαπλά worker νήματα προστατεύεται από ένα readers-writers πρωτόκολλο κλειδώματος. [Βλέπε παράρτημα Α(Α.3) για περισσότερα].

- Η **ουρά U** (ουρά FIFO) είναι μια ακολουθία από καταστάσεις των οποίων οι successors καταστάσεις δεν έχουν υπολογιστεί ακόμη. Η ουρά U υλοποιείται σαν ένα αρχείο δίσκου του οποίου οι πρώτες και οι τελευταίες αρκετές χιλιάδες καταχωρήσεις κρατούνται στην μνήμη. Αυτή είναι μια FIFO ουρά που χρησιμοποιεί ένα background νήμα για να κάνει prefetch καταχωρήσεις και άλλο για να γράφει καταχωρήσεις στον δίσκο. Αφιερώνοντας ένα κλάσμα του επεξεργαστή για αυτά τα background νήματα γενικά εξασφαλίζεται ότι ένα νήμα worker ποτέ δεν περιμένει για disk I/O όταν γίνεται πρόσβαση στην ουρά. [Βλέπε παράρτημα Α(Α.3) για περισσότερα]

- Για το πραγματοποίηση του liveness model checking συγκεκριμένα λαμβάνεται υπόψη και μια τρίτη δομή, ένας κατευθυνόμενος γράφος καταστάσεων που ονομάζεται **behavior γράφος**. Περιοδικά κατά την διάρκεια της δημιουργίας του behavior γράφου, και όταν έχει ολοκληρωθεί η δημιουργία του behavior γράφου, το TLC ελέγχει την ιδιότητα ImpliedTemporal ως ακολούθως. Έστω T ένα σύνολο αποτελούμενο από κάθε συμπεριφορά τ που είναι ακολουθία καταστάσεων σε ένα μονοπάτι άπειρου μήκους στον behavior γράφο ξεκινώντας με μια initial κατάσταση. (Για παράδειγμα, το T περιλαμβάνει το μονοπάτι $s \rightarrow s \rightarrow s \rightarrow \ldots$ για κάθε initial κατάσταση s στον behavior γράφο.) Να σημειωθεί ότι κάθε συμπεριφορά στο T ικανοποιεί την $Init \wedge \Box[Next]_{vars}$. Το TLC για το liveness model checking ελέγχει επίσης ότι κάθε συμπεριφορά στο T ικανοποιεί την έκφραση $\text{Temporal} \Rightarrow \text{ıpliedTemporal}$.

  Η δημιουργία του behavior γράφου τερματίζεται μόνο αν το σύνολο των προσβάσιμων καταστάσεων είναι πεπερασμένο. Αλλιώς, το TLC θα τρέχει για πάντα δηλαδή, ώσπου να εξαντλήσει τους πόρους ή το σταματήσουν.

3.8 Αλγόριθμος Safety Model Checking

Ο αλγόριθμος που ακολουθείται για την υλοποίηση του safety model checking είναι ο ακόλουθος:

Αφού γίνει η ανάλυση και η σημασιολογική επεξεργασία των module που είναι να εξεταστούν από το TLC ακολουθεί η δημιουργία και αποθήκευση τμήματος state space.

1.Δημιουργία και αποθήκευση τμήματος state space

Το TLC εκτελεί τον αλγόριθμο του πίνακα 1, ξεκινώντας με κενά το G και την U:

1. Ελέγχει ότι κάθε assume στην προδιαγραφή ικανοποιείται από τις τιμές που ανατέθηκαν στις σταθερές παραμέτρους.

   Μια ASSUME εντολή θα πρέπει να χρησιμοποιείται μόνο για υποθέσεις σχετικά με τις σταθερές (constants). Δεδομένο είναι ότι η έκφραση δεν πρέπει να περιέχει μεταβλητές.

2. Υπολογίζει το σύνολο των αρχικών καταστάσεων εκτιμώντας το αρχικό κατηγόρημα Init (Βλέπε 3.5). Για κάθε αρχική κατάσταση s που βρήκε:

   a) Αν το κατηγόρημα Constraint είναι αληθές στην κατάσταση s, ελέγχεται αν το σύνολο G περιέχει την κατάσταση. Αν δεν την περιέχει την τοποθετεί στο σύνολο G, στην ουρά U. (γραμμές 10-16)

   b) Εκτιμά τα κατηγορούμενα Invariant και ImpliedInit στην κατάσταση s, αναφέρει πιθανά σφάλματα και σταματά αν είτε το ένα είτε το άλλο είναι ψευδές.

3. Όσο η U δεν είναι κενή, κάνει τα ακόλουθα:

   a) Αφαιρεί την πρώτη κατάσταση από την U. (έστω αυτή s)

   b) Βρίσκει το σύνολο T όλων των successor καταστάσεων της s εκτιμώντας την ενέργεια επόμενης κατάστασης ξεκινώντας από την s (Βλέπε 3.5).

   c) Για κάθε κατάσταση t στο T, κάνει τα ακόλουθα:

      i. Αν το κατηγόρημα Constraint είναι αληθές στην κατάσταση t και το βήμα $s \rightarrow t$ ικανοποιεί το ActionConstraint, τότε αν η t δεν είναι μέσα στο G, τότε την προσθέτει στο G και στην ουρά U (γραμμές 34-40).

      ii. Αν η t δεν είναι μέσα στο G, τότε ελέγχεται αν το Invariant είναι ψευδές στην κατάσταση t ή το ImpliedAction είναι ψευδές για το βήμα $s \rightarrow t$, τότε αναφέρει ένα σφάλμα και σταματά (γραμμές 41-45).

   d) Αν το T είναι κενό και η επιλογή για τον έλεγχο ύπαρξης deadlock είναι ενεργοποιημένη, τότε αναφέρει ένα σφάλμα αδιεξόδου και η όλη διαδικασία σταματά.

Το TLC μπορεί να χρησιμοποιήσει πολλαπλά νήματα, και τα βήματα 3(b)-(d) μπορούν να εκτελεστούν ταυτόχρονα από διαφορετικά νήματα για διαφορετικές καταστάσεις s.

Το TLC δεν εκτελεί συνήθως και τα τρία βήματα που περιγράφηκαν παραπάνω. Κάνει το βήμα 2 μόνο για μια μη σταθερή module, στην οποία περίπτωση το configuration αρχείο πρέπει να προσδιορίζει μια Init έκφραση. Το TLC κάνει το βήμα 3 μόνο αν το configuration αρχείο προσδιορίζει μια Next έκφραση, την οποία πρέπει να εκτελέσει αν

το configuration αρχείο προσδιορίζει μια Invariant, μια ImpliedAction, ή μια ImpliedTemporal έκφραση.

Πίνακας 1: Αλγόριθμος δημιουργίας και αποθήκευσης τμήματος state space

```
1.  Set G= new Set(); Queue U=new Queue();
2.  Vector initials=new Vector.addAll(Spec.getInitial());
3.  For all initials
4.  {
5.          State s = initials.getElement();
6.          if !s.satisfies(Constraints)
7.          {
8.              throw new ConstraintsViolated(s);
9.          }
10.         else
11.         {
12.             if !G.contains(s)
13.             {
14.                 G.put(s); U.enqueue(s);
15.             }
16.         }
17.         if !G.contains(s)
18.         {
19.             if !s.satisfies(invariant){throw new InvariantViolated(s);}
20.             if !s.satisfies(ImpliedInit){throw new ImpliedInit Violated(s);}
21.         }
22. }
23. While !U.isEmpty()
24. {
25.         State s = U.dequeueElement();
26.         Set T=Spec. getSuccessors (s);
27.         While T not empty
28.         {
29.             State t=T.getElement();
30.             If (!t.satisfies(Constraints)|| !t.satisfies(ActionConstraints))
31.             {
32.                 throw new ConstraintsViolated(s);
```

```
33.             }
34.             else
35.             {
36.                 if !G.contains(t)
37.                 {
38.                     G.put(t); U.enqueue(t);
39.                 }
40.             }
41.             if !G.contains(t)
42.             {
43.                 if !t.satisfies(invariant){throw new InvariantViolated(t);}
44.                 if !s.satisfies(impliedActions(t)){throw new ImpliedActionsViolated(t);}
45.             }
46.         }
47.         if T.isEmpty() && checkDeadlock){ throw new DeadlockViolated(s);}
48. }
```

3.9 Αλγόριθμος Liveness Model Checking

Στην περίπτωση που έχουμε liveness model checking το διάνυσμα που αντιστοιχεί στην ιδιότητα ImpliedTemporal και αφορά τις liveness ιδιότητες δεν είναι κενό. Για την καλύτερη κατανόηση της διαδικασίας του liveness model checking μπορείτε να δείτε στο Παράρτημα Α την ενότητα Α.3 που αντιστοιχεί στην «Αναλυτική περιγραφή εκτέλεσης».
Αφού γίνει η ανάλυση και η σημασιολογική επεξεργασία των modules που είναι να εξεταστούν από το TLC ακολουθεί η αρχικοποίηση για το Liveness Model Checking.

1.Αρχικοποίηση για Liveness Model Checking

Συγκεκριμένα στο στάδιο αυτό κανονικοποιούνται οι εκφράσεις των temporals και των impliedTemporals, δηλαδή μετατρέπονται σε **διαζευκτική κανονική μορφή (disjunctive normal form, DNF)** για να ελεγχθεί η εγκυρότητα τους, και να υπολογιστεί η σειρά και ο τρόπος με τον οποίο πρέπει να ελεγχθούν. Αυτή η μέθοδο επιστρέφει ένα handle (μια συλλογή από OrderOfSolution), ο οποίος μπορεί να χρησιμοποιηθεί και σε άλλες liveness εργασίες. Συγκεκριμένα:

1. Ο αλγόριθμος αποσυνθέτει την λίστα των temporals και των impliedTemporals σε μια διάζευξη από συζευκτέους της ακόλουθης μορφής (**Μετατροπή σε DNF**):

   $$(\Diamond\Box a \therefore \Box\Diamond b \therefore tf1) \because (\Diamond\Box c \therefore \Box\Diamond d \therefore tf2)...$$

2. Στην συνέχεια, συγκεντρώνονται όλες μαζί οι διαζεύξεις που έχουν το ίδιο tf (temporal formula). Αυτό θα συμβεί σε περιπτώσεις όπως στην (WF ∴ SF) =>

(WF ∴ SF ∴ TF), αφού το WF και το SF θα διασπαστούν σε πολλές περιπτώσεις και το TF θα παραμείνει ίδιο σε όλη την διάρκεια. (Στην πραγματικότητα, ελέγχεται η ισότητα συντακτικά μόνο στα TFs.)

Επομένως για αποδοτικότητα, εντοπίζονται οι διαζευκτέοι που έχουν την ίδια χρονική έκφραση (tf).

Το OrderOfSolution ομαδοποιεί αυτούς τους διαζευκτέους που έχουν την ίδια χρονική έκφραση tf.

Οι Boolean τελεστές έχουν την συνήθη σημασία τους στην TLA. Στην συνέχεια εφαρμόζεται η ισοδυναμία του προτασιακού λογισμού (επιμερισμός του ∧ πάνω στο ∨).

$$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$$

Για κάθε χρονική έκφραση (tf) δημιουργείται ένα OrderOfSolution:

$$(\Diamond\Box a \therefore \Box\Diamond b \because \Diamond\Box c \therefore \Box\Diamond d) \therefore tf$$

Κάθε συζευκτέος ($\Diamond\Box a \therefore \Box\Diamond b$) αναπαρίσταται με ένα PossibleErrorModel(Pem).

Είναι πιθανό ένα απλό order of solution να έχει πολλά αντίγραφα των ◇□(EA, Eventually Always) και των □◇(AE, Always Eventually) που κατανέμονται στις διαζεύξεις και τις συζεύξεις των pems του. Για να αποφευχθεί η σπατάλη, χρησιμοποιούνται δύο πίνακες αναζήτησης(lookup tables): ο checkState και ο checkAction. Τα στοιχεία αυτών των πινάκων είναι στην ουσία state predicates, δηλαδή Boolean εκφράσεις που περιλαμβάνουν σταθερές και μεταβλητές. Ο έλεγχος αυτών των κατηγορημάτων μπορεί να επιστρέψει true ή false γεγονός που διευκολύνει την πραγματοποίηση του liveness model checking.

Συγκεκριμένα ελέγχονται τα στοιχεία αυτών των πινάκων όταν εξετάζεται κάθε κατάσταση και οι μεταβάσεις της. Οι τιμές των pems που δείχνουν τι θα πρέπει να εξεταστεί είναι απλοί ακέραιοι δείκτες (indexs) σε αυτούς τους δύο πίνακες.

3. Ανάλογα με το πλήθος των OrderOfSolution δημιουργείται αντίστοιχο πλήθος από **DiskGraph** όπου αποθηκεύει έναν behavior γράφο στον δίσκο. Χρησιμοποιούνται δύο αρχεία δίσκου για να αποθηκευτεί ο κάθε γράφος. Για κάθε κόμβο στον γράφο, το πρώτο αρχείο (**nodes_0**) αποθηκεύει τις successors καταστάσεις και πληροφορίες που έχουν υπολογιστεί για τον κόμβο, και το δεύτερο αρχείο (**ptrs_0**) αποθηκεύει το fingerprint του κόμβου και τον δείκτη στην θέση του κόμβου στο πρώτο αρχείο.

Ένας **GraphNode** είναι ένας κόμβος στον behavior γράφο.

2.Δημιουργία και αποθήκευση τμήματος state space

Ο υπολογισμός του TLC διατηρεί τα ακόλουθα invariants:

- Τις καταστάσεις του G που ικανοποιούν το Constraint κατηγόρημα.
- Για κάθε κατάσταση s στο G, την ακμή από το s στο s μέσα στον behavior γράφο.
- Αν υπάρχει μια ακμή στον behavior γράφο από την κατάσταση s σε μια διαφορετική κατάσταση t, τότε η t είναι μια successor κατάσταση του s που ικανοποιεί το constraint ενέργειας. Με άλλα λόγια, το βήμα $s \rightarrow t$ ικανοποιεί τον $Next \wedge ActionConstraint$.
- Κάθε κατάσταση s του behavior γράφου είναι προσβάσιμη από μια initial κατάσταση (μια που ικανοποιεί το Init κατηγόρημα) από μια διαδρομή στον behavior γράφο.
- Η U είναι μια ακολουθία από διακριτές καταστάσεις που είναι στοιχεία/καταστασεις στο G.
- Για κάθε κατάσταση s στο G που δεν είναι στην U, και για κάθε κατάσταση t που ικανοποιεί το Constraint τέτοια που το βήμα $s \rightarrow t$ ικανοποιεί την $Next \wedge ActionConstraint$, η κατάσταση t (είναι μέσα στον G) και η ακμή από το s στο t είναι μέσα στον behavior γράφο.

Το TLC εκτελεί τον αλγόριθμο του πίνακα 2, ξεκινώντας με κενούς το G και την U:

1. Ελέγχει ότι κάθε assume στην προδιαγραφή ικανοποιείται από τις τιμές που ανατέθηκαν στις σταθερές παραμέτρους.

   Μια ASSUME εντολή θα πρέπει να χρησιμοποιείται μόνο για υποθέσεις σχετικά με τις σταθερές (constants). Δεδομένο είναι ότι η έκφραση δεν πρέπει να περιέχει μεταβλητές.

2. Υπολογίζει το σύνολο των αρχικών καταστάσεων εκτιμώντας το αρχικό κατηγόρημα Init (Βλέπε 3.5). Για κάθε αρχική κατάσταση s που βρήκε:

   a) Αν το κατηγόρημα Constraint είναι αληθές στην κατάσταση s, ελέγχεται αν το σύνολο G περιέχει την κατάσταση. Αν δεν την περιέχει την τοποθετεί στο σύνολο G, στην ουρά U καθώς και στον behavior γράφο αν η έκφραση ImpliedTemporal δεν είναι κενή, δηλαδή έχουμε και liveness checking. (γραμμές 10-19)

   b) Εκτιμά τα κατηγορούμενα Invariant και ImpliedInit στην κατάσταση s, αναφέρει πιθανά σφάλματα και σταματά αν είτε το ένα είτε το άλλο είναι ψευδές.

3. Όσο η U δεν είναι κενή, κάνει τα ακόλουθα:

   a) Αφαιρεί την πρώτη κατάσταση από την U. (έστω αυτή s)

   b) Βρίσκει το σύνολο T όλων των successor καταστάσεων της s εκτιμώντας την ενέργεια επόμενης κατάστασης ξεκινώντας από την s (Βλέπε 3.5).

   c) Για κάθε κατάσταση t στο T, κάνει τα ακόλουθα:

      iii. Αν το κατηγόρημα Constraint είναι αληθές στην κατάσταση t και το βήμα $s \rightarrow t$ ικανοποιεί το ActionConstraint, τότε αν η t δεν είναι μέσα στο G, τότε την προσθέτει στο G και στην ουρά U καθώς και σε ένα σύνολο LiveNextStates που περιέχει τις επόμενες καταστάσεις της s αν η έκφραση

ImpliedTemporal δεν είναι κενή, δηλαδή έχουμε και liveness checking (γραμμές 38-48).

iv. Αν η t δεν είναι μέσα στο G, τότε ελέγχεται αν το Invariant είναι ψευδές στην κατάσταση t ή το ImpliedAction είναι ψευδές για το βήμα $s \rightarrow t$, τότε αναφέρει ένα σφάλμα και σταματά (γραμμές 49-53).

d) Αν το T είναι κενό και η επιλογή για τον έλεγχο ύπαρξης deadlock είναι ενεργοποιημένη, τότε αναφέρει ένα σφάλμα αδιεξόδου και η όλη διαδικασία σταματά.

e) Τέλος αν η έκφραση ImpliedTemporal δεν είναι κενή, δηλαδή έχουμε και liveness checking, στο γράφο προστίθενται όλες τις successor καταστάσεις της s που περιέχονται στο σύνολο LiveNextStates καθώς και τις μεταβάσεις σε αυτές από την s, δηλαδή τις $s \rightarrow t$ μεταβάσεις εκτιμώντας όλα τα κατηγορήματα και τις ενέργειες που εμφανίζονται στις εκφράσεις Temporal και ImpliedTemporal (ελέγχει αν ικανοποιούνται τα στοιχεία των checkState και checkAction πινάκων). Στο γράφο προσθέτει με τις άλλες μεταβάσεις και την μετάβαση $s \rightarrow s$ (stuttering step). (γραμμές 56-61)

Το TLC μπορεί να χρησιμοποιήσει πολλαπλά νήματα, και τα βήματα 3(b)-(e) μπορούν να εκτελεστούν ταυτόχρονα από διαφορετικά νήματα για διαφορετικές καταστάσεις s.

Πίνακας 2: Αλγόριθμος δημιουργίας και αποθήκευσης τμήματος state space

```
1.  Set G= new Set(); Queue U=new Queue(); BehaviourGraph Graph=new BehaviourGraph();
2.  Vector initials=new Vector.addAll(Spec.getInitial());
3.  For all initials
4.  {
5.          State s = initials.getElement();
6.          if !s.satisfies(Constraints)
7.          {
8.              throw new ConstraintsViolated(s);
9.          }
10.         else
11.         {
12.             if !G.contains(s)
13.             {
14.                 G.put(s); U.enqueue(s);
15.                 if !isEmpty(ImpliedTemporal){          //for liveness
16.                         Graph.add(s);
17.                 }
18.             }
```

```
19.          }
20.          if !G.contains(s)
21.          {
22.                  if !s.satisfies(invariant){throw new InvariantViolated(s);}
23.                  if !s.satisfies(ImpliedInit){throw new ImpliedInit Violated(s);}
24.          }
25.  }
26.  While !U.isEmpty()
27.  {
28.          State s = U.dequeueElement();
29.          Set T=Spec. getSuccessors (s);
30.          Set LiveNextStates;
31.          While T not empty
32.          {
33.                  State t=T.getElement();
34.                  If (!t.satisfies(Constraints)|| !t.satisfies(ActionConstraints))
35.                  {
36.                          throw new ConstraintsViolated(s);
37.                  }
38.                  else
39.                  {
40.                          if !G.contains(t)
41.                          {
42.                              G.put(t); U.enqueue(t);
43.                              if !ImpliedTemporal.isEmpty()          //for liveness
44.                              {
45.                                      LiveNextStates.add(t);
46.                              }
47.                          }
48.                  }
49.                  if !G.contains(t)
50.                  {
51.                      if !t.satisfies(invariant){throw new InvariantViolated(t);}
52.                      if !s.satisfies(impliedActions(t)){throw new ImpliedActionsViolated(t);}
53.                  }
54.          }
```

```
55.         if T.isEmpty() && checkDeadlock){ throw new DeadlockViolated(s);}
56.         if !ImpliedTemporal.isEmpty()          //for liveness
57.         {
58.             Graph.add(LiveNextStates);
59.             LiveNextStates.add(s);
60.             Graph.addTransition(s,LiveNextStates);
61.         }
62. }
```

3. Υλοποίηση Liveness Model Checking

Όπως έχουμε ήδη αναφέρει μια liveness ιδιότητα εκφράζει ότι το σύστημα θα εμφανίσει μια συγκεκριμένη συμπεριφορά μέσα σε μια πεπερασμένη περίοδο χρόνου. Για να ανιχνευθούν παραβιάσεις liveness ιδιοτήτων χρειάζεται να εξεταστούν οι ιδιότητες αυτές σε εκτελέσεις συστήματος άπειρου μήκους.

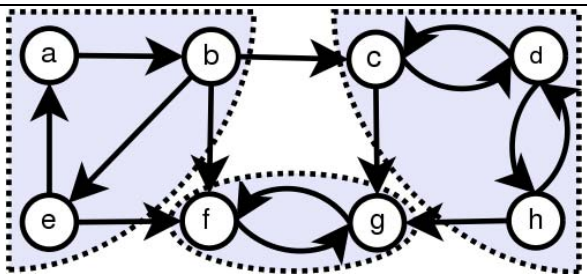
Ο behavior γράφος που κατασκευάζεται από το TLC για το liveness έλεγχο είναι ένας κατευθυνόμενος γράφος ο οποίος αποτελείται από ένα πεπερασμένο σύνολο κορυφών/κόμβων και ένα σύνολο κατευθυνόμενων ακμών. Μεταξύ δύο κόμβων του γράφου είναι δυνατόν να υπάρχουν περισσότερα του ενός μονοπάτια.

Εφόσον μιλάμε για μονοπάτια άπειρου μήκους σε έναν κατευθυνόμενο γράφο για τον έλεγχο του liveness είναι πολύ πιθανό να προκύψουν κύκλοι (κλειστά μονοπάτια) στον γράφο, δηλαδή μονοπάτια όπου όλοι οι κόμβοι είναι διακριτοί εκτός από $v_0=v_k$ (v_0 ο αρχικός κόμβος και v_k ο τελικός κόμβος του μονοπατιού). Στο TLC κατά την διαδικασία του Liveness Model checking η εύρεση ενός κύκλου αντιστοιχεί στην εύρεση ενός counterexample στην επαλήθευση μιας προδιαγραφής.

Η πραγματική σημασία του liveness ελέγχου είναι να αποδείξει την απουσία κύκλων σε μονοπάτια διαδοχικών καταστάσεων άπειρου μήκους.

Αν σε έναν κατευθυνόμενο γράφο για κάθε δύο κόμβους i και j υπάρχει τουλάχιστον ένα κατευθυνόμενο μονοπάτι από τον κόμβο i στον κόμβο j και ένα κατευθυνόμενο μονοπάτι από τον κόμβο j στον κόμβο i τότε ο γράφος χαρακτηρίζεται ως ισχυρά συνεκτικός (strongly connected)[11]. Τα δύο αυτά μονοπάτια δεν είναι απαραίτητα διαζευγμένα, δηλαδή μπορεί να μην διαθέτουν κοινούς κόμβους(ακμές). Δηλαδή, οι κόμβοι βρίσκονται πάνω σε ένα κλειστό μονοπάτι (κύκλο).

Ως ισχυρά συνεκτική συνιστώσα (Strongly Connected Component,SCC)[11] σε έναν γράφο ορίζεται ο μέγιστος υπογράφος κόμβων και ακμών τέτοιος ώστε κάθε κόμβος να είναι προσβάσιμος από κάθε άλλο κόμβο. Η εικόνα που ακολουθεί αναπαριστά έναν γράφο με διακριτές τις ισχυρά συνεκτικές συνιστώσες του.



Στην θεωρία των γράφων ένας κατευθυνόμενος γράφος είναι άκυκλος(acyclic), δηλαδή δεν έχει κατευθυνόμενους κύκλους, αν και μόνο αν αυτός δεν έχει καμία ισχυρά συνεκτική συνιστώσα. (επειδή ένας κύκλος είναι ισχυρά συνεκτικός και κάθε ισχυρά συνεκτικός υπογράφος περιέχει τουλάχιστον ένα κύκλο.) Σε αυτήν την περίπτωση μιλάμε για έναν κατευθυνόμενο άκυκλο γράφο (DAG, Directed Acyclic Graph).

Έτσι το TLC, κατά την υλοποίηση του liveness model checking, εστιάζει στην ανίχνευση των ισχυρά συνεκτικών συνιστωσών μιας και συνεπάγεται την ανίχνευση των όποιων πιθανών κύκλων, δηλαδή την εύρεση των όποιων πιθανών counterexamples.

Παρατήρηση: Ο αλγόριθμος που χρησιμοποιείται από το TLC για την εύρεση των ισχυρά συνεκτικών συνιστωσών μοιάζει αρκετά με τον αλγόριθμο **Gabow**(1999)[12]. Ο συγκεκριμένος αλγόριθμος χρησιμοποιεί δύο στοίβες. Η πρώτη στοίβα αποθηκεύει το τρέχον μονοπάτι ενώ η δεύτερη στοίβα σκοπό έχει να εντοπίσει ποιες κορυφές είναι σημεία εισόδου σε ισχυρά συνεκτικές συνιστώσες. Ο αλγόριθμος ακολουθεί την εξερεύνηση ΑσΒ (Αναζήτηση σε Βάθος). Όταν εντοπιστεί μια ανεξερεύνητη κορυφή υ, εισάγεται και στις δύο στοίβες. Εάν βρεθεί μια οπισθοακμή –και συνεπώς, βρισκόμαστε εντός συνιστώσας– με διαδοχικά ποπ στην δεύτερη στοίβα, παραμένει μόνο η κορυφή-απόληξη της οπισθοακμής ήτοι, η εγγύτερη προς την ρίζα του δένδρου ΑσΒ κορυφή του αντίστοιχου κύκλου. Κατά την περάτωση της επεξεργασίας μια κορυφής υ, ελέγχεται μήπως η υ είναι στην κορυφή της δεύτερης στοίβας. Εάν ναι, τότε αυτό σημαίνει πως η υ είναι σημείο εισόδου , και όλες οι κορυφές του μονοπατιού ΑσΒ στην πρώτη στοίβα, από την κεφαλή της μέχρι την υ, αποτελούν ισχυρά συνεκτική συνιστώσα. Εάν όχι, τότε δεν γίνεται τίποτα. Ο αλγόριθμος του Gabow ανακαλύπτει τις ισχυρά συνεκτικές συνιστώσες ενός γραφήματος G σε γραμμικό ως προς την πολυπλοκότητα του χρόνο και χώρο.

1. Λαμβάνεται το επόμενο orderOfSolution. Για κάθε pem(possibleErrorModel) του orderOfSolution:

2. Αρχικοποιείται μια ουρά, η nodeQueue με τους αρχικούς κόμβους του γράφου. Όσο αυτή η ουρά έχει στοιχεία, για κάθε στοιχείο/κατάσταση curState της που λαμβάνεται ως root κόμβος για τον υπολογισμό του scc:

3. Τοποθετείται η curState καθώς και η τιμή του μέγιστου μήκους (**MAX_PTR**=4611686018427387904)του αρχείου fileForNodes σε μια στοίβα dfsStack

που χρησιμοποιείται για την υλοποίηση του dfs (depth-first search) αλγορίθμου για την διάσχιση του γράφου.

4. Αρχικοποιείται μια μεταβλητή **newLink** να είναι ίση με MAX_PTR.

5. Όσο το μέγεθος της dfsStack είναι μεγαλύτερο από 2, εξάγεται πρώτα ένα στοιχείο το οποίο ανατίθεται σε μια μεταβλητή **lowLink** και ένα που ανατίθεται στην κατάσταση curState :

   Αν η κατάσταση αυτή **δεν έχει εξερευνηθεί** και

   Αν δεν της έχει ανατεθεί κάποιο link.

   i. Της ανατίθεται ένα link.

   ii. Τοποθετείται η curState ξανά στην dfsStack αλλά με αλλαγμένο κατάλληλο πεδίο της, έτσι ώστε να δηλώσει ότι έχει εξερευνηθεί η συγκεκριμένη κατάσταση.

   iii. Επιπλέον τοποθετείται και σε μια άλλη στοίβα την comStack που περιλαμβάνει όλες τις καταστάσεις που έχουν εξερευνηθεί.

   iv. Αρχικοποιείται μια μεταβλητή **nextLowLink** να είναι ίση με newLink. Παράλληλα αυξάνεται κατά ένα η newLink.

   v. Βρίσκει το σύνολο T όλων των successor καταστάσεων της curState μέσω του behavior γράφου. Για κάθε succState κατάσταση του συνόλου T:

      Αν αυτή η succState κατάσταση δεν έχει εξερευνηθεί, ελέγχονται τα EAAction για αυτήν την κατάσταση.

      Εάν ο έλεγχος είναι **αληθής** τότε αν το link που έχει ανατεθεί στην succState είναι FilePointer, δηλαδή το link είναι μικρότερο του MAX_PTR, τότε η succState τοποθετείται στην dfsStack (γραμμές 35-38). Διαφορετικά αν το link που έχει ανατεθεί στην succState είναι μικρότερο του nextLowLink τότε ανατίθεται η τιμή του link στην nextLowLink.

      Εάν ο έλεγχος είναι **ψευδής** τότε αν το link που έχει ανατεθεί στην succState είναι FilePointer τότε η succState τοποθετείται στην nodeQueue(γραμμές 44-47).

   vi. Τοποθετείται η nextLowLink στην dfsStack.

   Αν έχει ανατεθεί κάποιο link στην curState

   i. αν το link που έχει ανατεθεί στην curState είναι μικρότερο του lowLink τότε ανατίθεται η τιμή του link στην lowLink.

   ii. Τοποθετείται η lowLink στην dfsStack.

   Αν **έχει εξερευνηθεί** τότε οι καταστάσεις στην comStack από την κορυφή μέχρι την curState σχηματίζουν ένα SCC.

A. Αν το link της curState είναι ίσο με την τιμή του lowLink εξάγεται μια κατάσταση curStateC από την comStack.Στην συνέχεια γίνεται έλεγχος για το αν το component είναι trivial.Συγκρίνεται η curStateC με την τρέχουσα κατάσταση curState αν είναι ίδιες αλλά και αν η curStateC δεν παρουσιάζει stuttering, έτσι ώστε να αποφασισθεί αν το component είναι trivial.

Αν το component είναι **trivial**

Το link της curState γίνεται ίσο με MaxLink (=9223372036854775807).

Αν το component **δεν είναι trivial**

i. Λαμβάνονται όλοι οι κόμβοι/καταστάσεις που περιέχονται στο συστατικό από την comStack και τοποθετούνται σε έναν NodePtrTable(hashtable) με όνομα com. Επιπλέον το link της curState γίνεται ίσο με MaxLink (γραμμές 70-76).

ii. Κάθε ένα στοιχείο curNode του com ελέγχεται αν δεν είναι null. Στην περίπτωση αυτή ελέγχονται τα AEState, τα AEAction για κάθε successor κόμβο που δείχνει ο συγκεκριμένος κόμβος καθώς και αν εκπληρώνονται τα promises σε περίπτωση που υπάρχει tableau. Αν όλα ικανοποιούνται έχουμε ένα counterexample και τότε αναφέρεται ένα σφάλμα και σταματά ο έλεγχος. (Συγκεκριμένα ελέγχεται η μετάβαση στην οποία η curState δείχνει στον εαυτό της. (δηλαδή να υπάρχει κύκλος) Για καλύτερη κατανόηση κοιτάξτε την αναλυτική περιγραφή εκτέλεσης στο Παράρτημα Α).

B. Εξάγεται από την dfsStack το plowLink.

i. Αν το plowLink είναι μεγαλύτερο του lowLink , τότε γίνεται ίσο με το lowLink.

ii. Τοποθετείται ξανά στην dfsStack

Πίνακας 2: Αλγόριθμος Υλοποίηση Liveness Model Checking

```
1.  While(true)
2.  {
3.     OrderOfSolution oos = new OrderOfSolution (spec.getNextOOS());
4.     for  all  oos.pems
5.     {
6.       Queue nodeQueue=new Queue(Graph.getInitNodes());
7.       while (nodeQueue.hasElements())
8.       {
9.          State curState= nodeQueue.dequeue();
10.         Stack dfsStack=new Stack();   Stack comStack =new Stack();
```

```
11.          dfsStack.push(curState);
12.          dfsStack.push(MAX_PTR);
13.          long newLink= MAX_PTR;
14.          while (dfsStack.size()> 2)
15.          {
16.              long lowLink = dfsStack.pop();
17.              State curState = dfsStack.pop();
18.              if(!curState.hasExplored())
19.             {
20.                  if (!curState.HasLink())            //Αν δεν της έχει ανατεθεί link
21.                  {
22.                     curState.putLink();
23.                     dfsStack.push(lowLink);
24.                    dfsStack.push(curState);
25.                    comStack.push(curState);
26.                    long nextLowLink = newLink++;
27.                    Set T=Graph.getSuccessors (curState);
28.                    For all T
29.                    {
30.                       State succState =T.getElement();
31.                       if (!succState.hasExplored())
32.                       {
33.                           if (curState.CheckEAAction())
34.                           {
35.                                 if (succState.link<MAX_PTR)
36.                                 {
37.                                       dfsStack.enqueue(succState);
38.                                  }
39.                                  else if(succState. link < nextLowLink)
40.                                 {
41.                                      nextLowLink= succState. link;
42.                                 }
43.                           }
44.                           else  if (succState. link <MAX_PTR)
45.                           {
46.                                 nodeQueue.enqueue(succState);
```

```
47.                         }
48.                       }
49.                     }
50.                     dfsStack.push(nextLowLink);
51.                   }
52.                   else       //Αν της έχει ανατεθεί link
53.                   {
54.                     if (curState.link < lowLink) lowLink = curState.link;
55.                     dfsStack.pushLong(lowLink);
56.                   }
57.                 }
58.                 else
59.                 {
                      //Αν έχει εξερευνηθεί
60.                   if (curState.link == lowLink)
61.                   {
62.                       State curStateC=comStack.pop();
63.                       if (curStateC == curState &&!isStuttering(curStateC)) //trivial
64.                       {
65.                           curState.link=MaxLink;
66.                       }
67.                       else                  //non trivial
68.                       {
69.                           NodePtrTable com =new   NodePtrTable ();
70.                           while (true)
71.                           {
72.                               com.putNode(curStateC);
73.                               curState.link=MaxLink;
74.                               if (curState == curStateC ) break;
75.                               curStateC=comStack.pop();
76.                           }
77.                           for all  com
78.                           {
79.                              Node curNode = com.getNode();
80.                              if (curNode == null) continue;
```

```
81.                                 if curNode.satisfies(AEState) && curNode.satisfies(AEAction)&&
                                       && curNode.satisfies(Promises)
82.                                 {
83.                                     throw new CounterExampleExist();
84.                                 }
85.                             }
86.                         }
87.                     }
88.                     long plowLink = dfsStack.pop();
89.                     if (lowLink < plowLink) plowLink = lowLink;
90.                     dfsStack.push(plowLink);
91.                 }
92.             }       //end of  while (dfsStack.size()> 2)
93.         }       // end of while (nodeQueue.hasElements())
94.     }       //end of for
95. }       //end of while
```

Περατότητα Αλγορίθμου (Finiteness)

Ο συγκεκριμένος αλγόριθμος καλύπτει όλους τους κύκλους(sccs). Η nodeQueue αρχικοποιείται με όλες τις αρχικές καταστάσεις και κάθε μια από αυτές εξερευνείται (αποδίδοντας της ένα link) και λαμβάνεται ως root κόμβος στον υπολογισμό των sccs και επομένως εξετάζονται όλοι οι πιθανοί κύκλοι.

Κατά την εκτέλεση του αλγορίθμου στον έλεγχο των successor καταστάσεων μπορεί να τοποθετηθούν αντίγραφα των αρχικών καταστάσεων στο τέλος της nodeQueue προτού να εξερευνηθούν. Όταν έρχεται η σειρά ενός αντιγράφου, η αρχική κατάσταση σίγουρα θα έχει εξερευνηθεί και οπότε ελέγχεται το link της συγκεκριμένης κατάστασης βάσει του οποίου συμπεραίνεται ότι ο συγκεκριμένος κόμβος έχει ελεγχθεί με αποτέλεσμα να μην επαναλαμβάνεται η διαδικασία υπολογισμού για scc για τον συγκεκριμένο κόμβο. Ο έλεγχος τέτοιων αντιγράφων συναντάται στα τελευταία βήματα εκτέλεσης του αλγορίθμου στην εξέταση κάποιου pem.

Κεφάλαιο 4: Liveness Model Checking σε κατανεμημένο περιβάλλον

Περίληψη κεφαλαίου

Στο κεφάλαιο αυτό παρουσιάζεται η παράλληλη υλοποίηση που προϋπήρχε στο TLC για κατανεμημένα περιβάλλοντα. Αναφέρεται σύντομα η αρχιτεκτονική RMI που χρησιμοποιήθηκε για την παράλληλη υλοποίηση του liveness model checking σε κατανεμημένα περιβάλλοντα και παρουσιάζεται η παραλληλοποίηση του Liveness Model Checking (βήματα αλγορίθμου) που υλοποιήθηκε στα πλαίσια αυτής της εργασίας. Τέλος περιγράφεται ο τρόπος εκτέλεσης του TLCServer και του TLCWorker.

4.1 Παράλληλη υλοποίηση στην εκτέλεση του TLC για κατανεμημένα περιβάλλοντα

Το TLC παρέχει την παράλληλη υλοποίηση σε κατανεμημένο περιβάλλον μόνο της διαδικασίας δημιουργίας του State Space.

Για την υλοποίηση αυτού του αλγορίθμου χρησιμοποιήθηκε το μοντέλο RMI της Java, το οποίο βασίζεται στην τεχνολογία της διασύνδεσης απομακρυσμένων αντικειμένων και είναι σχεδιασμένο για να προάγει την διαλειτουργικότητα μεταξύ αντικειμένων, κατασκευασμένων σε Java, μέσα σε ένα κατανεμημένο και ετερογενές περιβάλλον. Συγκεκριμένα η επίκληση απομακρυσμένων μεθόδων (Java Remote Method Invocation - RMI) επιτρέπει σε ένα αντικείμενο της Java που τρέχει σε έναν υπολογιστή, να καλέσει μια μέθοδο ενός άλλου αντικειμένου της Java που εκτελείται σε άλλο μηχάνημα. Η προσπέλαση της μεθόδου αυτής γίνεται με την βοήθεια του δικτύου.

Στο μοντέλο RMI η μεταβίβαση ορισμάτων και επιστρεφόμενων τιμών σε ένα κατανεμημένο περιβάλλον γίνεται αυτόματα από μηχανισμούς του RMI καθώς και με την βοήθεια άλλων χαρακτηριστικών της Java όπως την σειριοποίηση αντικειμένου (Object serialization) [13] (τεχνολογία που επιτρέπει την μετατροπή ενός αντικείμενου σε μια σειρά από bytes με σκοπό την μεταφορά του μέσω του δικτύου ή την αποθήκευση του σε κάποιο αρχείο ή memory buffer αλλά και την ανάκτηση του στην αρχική του μορφή). Η διαδικασία με την οποία σειριοποιείται ένα αντικείμενο αποκαλείται σειριοποίηση (**serialization ή marshalling**) . Η αντίθετη λειτουργία, ανάκτησης μιας δομής δεδομένων από μια σειρά bytes, αποκαλείται αποσειριοποίηση (**deserialization ή unmarshalling).**

Συγκεκριμένα τοπικά αντικείμενα για να περαστούν παράμετροι σε απομακρυσμένα αντικείμενα, στο μοντέλο RMI, θα πρέπει να είναι serializable. Δηλαδή θα πρέπει να

υλοποιούν την διεπαφή java.io.Serializable. Επιπλέον τα μέλη ενός "σειριοποιήσιμου" αντικειμένου θα πρέπει να είναι "σειριοποιήσιμα".

Στο κατανεμημένο περιβάλλον, το αντικείμενο που καλεί ένα άλλο χαρακτηρίζεται ως πελάτης (client), ενώ αυτό που δέχεται την κλήση λέγεται εξυπηρετητής (server). Στο TLC έχουμε τον TLCServer που αντιστοιχεί στον server και τους TLCWorkers που αντιστοιχούν στους clients σύμφωνα με το μοντέλο RMI.

TLCServer

Ο TLCServer διατηρεί τις ακόλουθες δομές δεδομένων: το σύνολο G που είναι μέρος του γράφου reachability κατάστασης που το TLC έχει υπολογίσει μέχρι την τρέχουσα χρονική στιγμή και μια ουρά (μια ακολουθία) U από ανεξερεύνητες καταστάσεις, δηλαδή όλες τις καταστάσεις στο G των οποίων οι successors δεν έχουν υπολογιστεί ακόμη από το TLC. (Το σύνολο G και η U είναι ίδια με τις δομές της εκτέλεσης του single workstation που παρουσιάστηκαν στο κεφάλαιο 3, ενότητα 3.7.)

Με την πληκτρολόγηση της κατάλληλης εντολής [βλέπε 4.4] ξεκινά η εκτέλεση του TLCServer:

1. Αρχικά ο TLCServer πραγματοποιεί την ανάλυση και την σημασιολογική επεξεργασία των modules που είναι να εξεταστούν από το TLC. Στην συνέχεια, εκτελεί τα επόμενα βήματα, ξεκινώντας με κενό το σύνολο G και κενή την U:

2. Υπολογίζει το σύνολο των αρχικών καταστάσεων εκτιμώντας το αρχικό κατηγόρημα Init. Για κάθε αρχική κατάσταση s που βρήκε:

   a) Αν το κατηγόρημα Constraint είναι αληθές στην κατάσταση s, ελέγχει αν το σύνολο G περιέχει την κατάσταση. Αν δεν την περιέχει την τοποθετεί στο σύνολο G και στην ουρά U .

   b) Εκτιμά τα κατηγορούμενα Invariant και ImpliedInit στην κατάσταση s. Αν τουλάχιστον ένα από τα δύο είναι ψευδές η εκτέλεση σταματά και αναφέρεται ένα σφάλμα.

3. Δημιουργεί ένα μητρώο (Registry) στον τοπικό υπολογιστή που δέχεται αιτήσεις σε συγκεκριμένο port (εξ ορισμού 10997). Το μητρώο είναι ένας ειδικός εξυπηρετητής που αναζητά αντικείμενα βάσει ονόματος. Οι RMI servers πρέπει να καταχωρηθούν σε μια υπηρεσία αναζήτησης (lookup service), ώστε να μπορούν οι clients να τους εντοπίζουν. Συγκεκριμένα στην Java υπάρχει μια τέτοια υπηρεσία, η rmiregistry, η οποία εκτελείται ως ξεχωριστή διαδικασία και επιτρέπει στις εφαρμογές να καταχωρούν RMI υπηρεσίες ή να αποκτούν αναφορά σε μια δηλωμένη υπηρεσία.

4. Καταχωρεί τον server που είναι τύπου TLCServer με το όνομα "TLCServer" στο μητρώο.

5. Και περιμένει για εισερχόμενες αιτήσεις.

TLCWorker

Ο TLCWorker διατηρεί την ακόλουθη δομή δεδομένων: το σύνολο G που είναι μέρος του γράφου reachability κατάστασης που το TLC έχει υπολογίσει μέχρι την τρέχουσα χρονική στιγμή. (Το σύνολο G είναι ίδιο με την δομή της εκτέλεσης του single workstation που παρουσιάστηκε στο κεφάλαιο 3, ενότητα 3.7.)

Μετά την εκτέλεση του TLCServer, με την πληκτρολόγηση της κατάλληλης εντολής [βλέπε 4.5] ξεκινά η εκτέλεση του TLCWorker.:

1. Ο TLCWorker πρέπει να αποκτήσει μια αναφορά στο απομακρυσμένο αντικείμενο. Αυτό γίνεται μέσω του μητρώου RMI. Ο TLCWorker ζητάει μια συγκεκριμένη υπηρεσία με όνομα "TLCServer" και δέχεται ένα URL για τον απομακρυσμένο πόρο, δηλαδή το TLCServer αντικείμενο. Μόλις μια αναφορά σε ένα απομακρυσμένο αντικείμενο γίνει γνωστή, αυτό μπορεί να χρησιμοποιηθεί σαν να είναι τοπικό αντικείμενο.

2. Ο TLCWorker πραγματοποιεί την ανάλυση και την σημασιολογική επεξεργασία των modules που είναι να εξεταστούν από το TLC.

3. Μέσω της αναφοράς στο απομακρυσμένο αντικείμενο "TLCServer", ο TLCWorker αρχικοποιεί το τοπικό σύνολο του G ίσο με το σύνολο G του TLCServer.

4. Στην συνέχεια καλεί μια συνάρτηση register του server με όρισμα τον εαυτό του (worker). Με αυτόν τον τρόπο καταχωρείται ο TLCWorker στον εξυπηρετητή.

5. Από την στιγμή που ο TLCWorker καταχωρηθεί στον server, δημιουργείται ένα TLCServerThread για αυτόν που είναι ένα νήμα στον TLCServer για την εξυπηρέτηση του συγκεκριμένου TLCWorker. Με την εκτέλεση αυτού του νήματος εκτελούνται τα ακόλουθα βήματα:

6. Όσο η U του TLCServer δεν είναι κενή, το TLCServerThread κάνει τα ακόλουθα:

   a) Αφαιρεί ένα σύνολο καταστάσεων *states* από την U μεγέθους BlockSize (1024) προκειμένου να εξεταστούν αυτές οι καταστάσεις στον συγκεκριμένο worker.

   b) Ο TLCServerThread καλεί μια συνάρτηση getNextStates του TLCWorker περνώντας ως όρισμα το σύνολο καταστάσεων states για να υπολογίσει τις successor καταστάσεις του συνόλου states που έχει αναλάβει.

   c) Συγκεκριμένα με την getNextStates ο TLCWorker για κάθε μια s κατάσταση από τις καταστάσεις του συνόλου states βρίσκει το σύνολο T με τις successor καταστάσεις της, εκτιμώντας την ενέργεια επόμενης κατάστασης.

d) Αν το T είναι κενό και η επιλογή για τον έλεγχο ύπαρξης deadlock είναι ενεργοποιημένη, τότε αναφέρει ένα σφάλμα αδιεξόδου και η όλη διαδικασία σταματά.

e) Για κάθε κατάσταση t στο T, ο TLCWorker κάνει τα ακόλουθα:

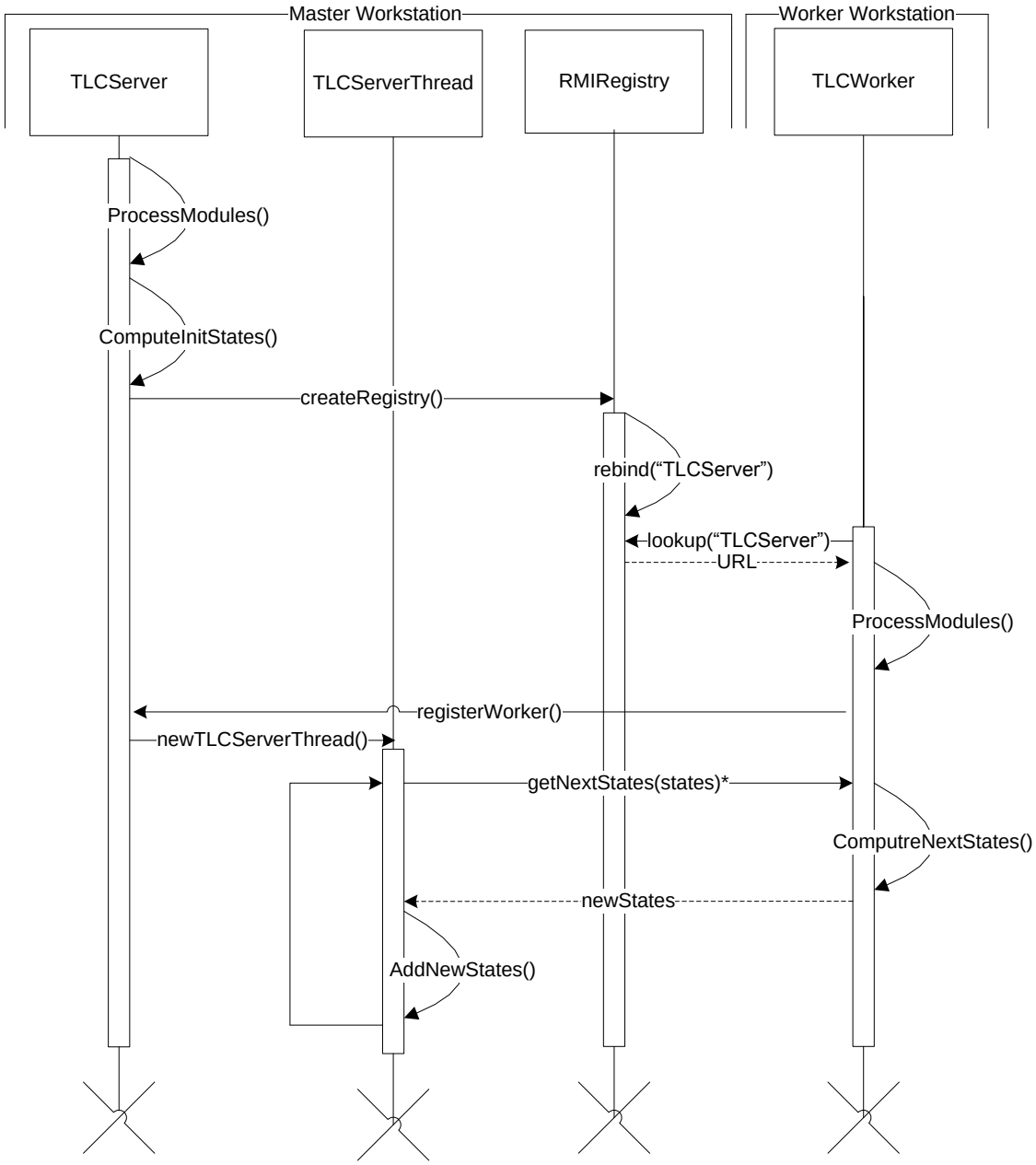
   i. Διαγράφει τις καταστάσεις του συνόλου T που υπάρχουν ήδη στο δικό του σύνολο G.

   ii. Σε αυτές που έχουν μείνει ελέγχει αν το κατηγόρημα Constraint είναι αληθές στην κατάσταση t και αν το βήμα $s \rightarrow t$ ικανοποιεί το ActionConstraint.

   iii. Το σύνολο των καταστάσεων που παράγονται επιστρέφεται στον TLCServerThread.

f) Από το επιστρεφόμενο σύνολο καταστάσεων, τοποθετούνται στο σύνολο G καθώς και στην ουρά U του TLCServer μόνο οι καταστάσεις που δεν εμπεριέχονται στο σύνολο G του TLCServer.

Στην επόμενη σελίδα ακολουθεί το διάγραμμα ακολουθίας της όλης διαδικασίας.



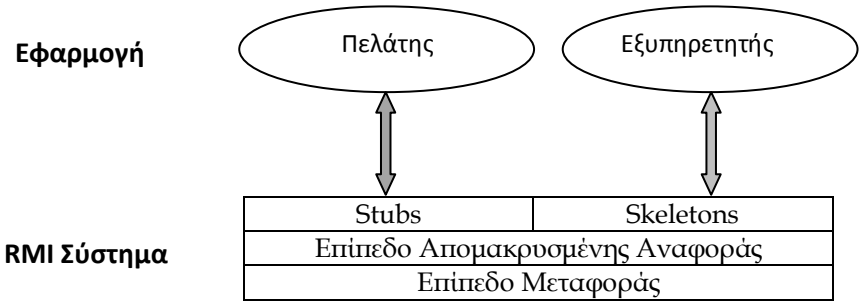
Σ' αυτήν την παράλληλη υλοποίηση του TLC δεν εμπεριέχεται καθόλου το Liveness model checking. Όμως, όπως φαίνεται και από τις μετρήσεις στο κεφάλαιο 5, κατά την εκτέλεση του TLC για μια προδιαγραφή που περιλαμβάνει Liveness ιδιότητα/ες σχεδόν όλος ο χρόνος εκτέλεσης καταναλώνεται στην εκτέλεση του Liveness model checking. Για τον λόγο αυτό, στα πλαίσια της εργασίας αυτής, υλοποιούμαι την παραλληλοποίηση του Liveness model checking σε κατανεμημένα περιβάλλοντα σύμφωνα και πάλι με το μοντέλο RMI. Στην συνέχεια ακολουθεί μια σύντομη παρουσίαση της αρχιτεκτονικής του RMI για να γίνει καλύτερα κατανοητό το πως ακριβώς θα υλοποιηθεί το παράλληλο Liveness Model Checking.

4.2 Αρχιτεκτονική του RMI

Το RMI[14] είναι ένα API της Java με το οποίο παρέχεται ένα μοντέλο προγραμματισμού για κατανεμημένα συστήματα. Στην ουσία πρόκειται για μια

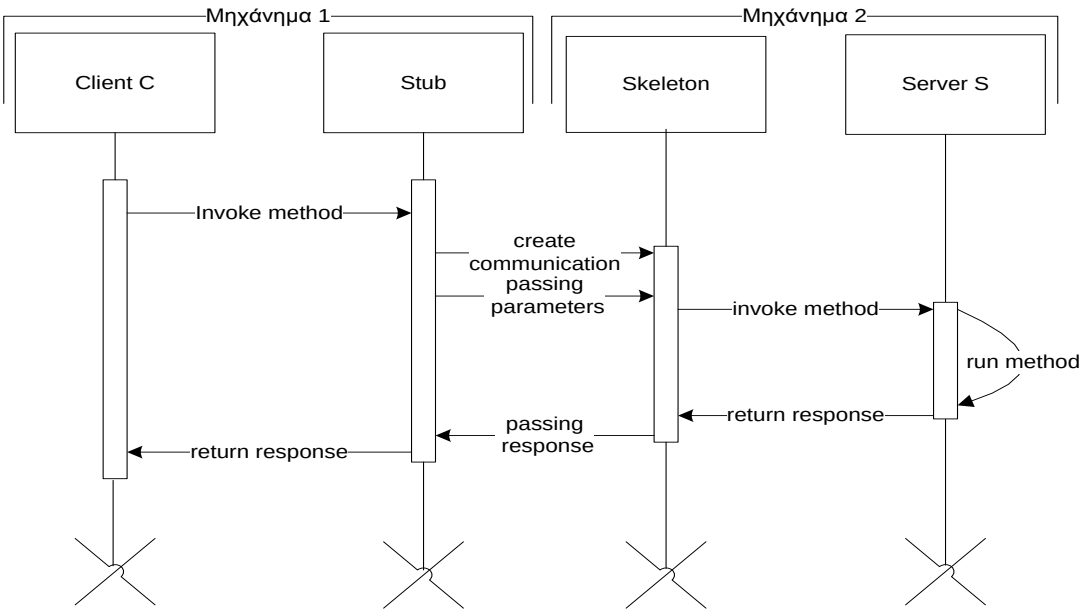
τεχνολογία η οποία επιτρέπει στην Java Virtual Machine ενός υπολογιστή να καλεί μεθόδους αντικειμένων οι οποίες εκτελούνται στις JVM άλλων απομακρυσμένων υπολογιστών. Το RMI χρησιμοποιεί σειριοποίηση αντικειμένου (object serialization) για την σειριοποίηση και την αποσειριοποίηση των παραμέτρων [Βλέπε 4.1] και δεν μεταβάλλει τους τύπους, αποκρύπτοντας έτσι διαφορές στα χαμηλότερα επίπεδα αρχιτεκτονικής.

Το σύστημα RMI αποτελείται από τρία διακριτά επίπεδα. Αυτά είναι:
- Το επίπεδο **Σκελετού/Στελέχους** (Skeleton/Stub)
- Το επίπεδο **Απομακρυσμένης Αναφοράς** (Remote Reference Layer)
- Το επίπεδο **Μεταφοράς** (Transport Layer)



Αν υποθέσουμε για παράδειγμα ότι ένα αντικείμενο C στο μηχάνημα1 θέλει να καλέσει μια μέθοδο του αντικειμένου S, που βρίσκεται στο μηχάνημα2. Το αντικείμενο S είναι ένα απομακρυσμένο αντικείμενο και υλοποιεί μια ή περισσότερες απομακρυσμένες διασυνδέσεις (remote interfaces). Οι απομακρυσμένες διασυνδέσεις περιέχουν μεθόδους οι οποίες μπορεί να κληθούν από στιγμιότυπα άλλων αντικειμένων. Επειδή όμως το αντικείμενο C βρίσκεται σε άλλο μηχάνημα, δεν είναι σε θέση να καλέσει απ' ευθείας μια τέτοια μέθοδο. Για το λόγο αυτόν δημιουργείται μια κλάση- στέλεχος (stub) του αντικειμένου S στο μηχάνημα όπου βρίσκεται το C. Το στέλεχος αυτό δρα ως υποκατάστατο (proxy) του απομακρυσμένου αντικειμένου S.

Ακολουθεί μια διαγραμματική απεικόνιση του συγκεκριμένου παραδείγματος.



Το στέλεχος (stub) στο μηχάνημα1 έχει τρεις βασικές εργασίες:

- Εμφανίζει τις ίδιες απομακρυσμένες διασυνδέσεις όπως και το S, άρα από την σκοπιά του C είναι το ίδιο με το S.
- Σε συνεργασία με την εικονική μηχανή της Java (JVM) και το σύστημα RMI, μεταφέρει τα ορίσματα για τις κλήσεις των απομακρυσμένων μεθόδων από το μηχάνημα1 στο μηχάνημα 2.
- Λαμβάνει τα αποτελέσματα των κλήσεων των απομακρυσμένων μεθόδων και τα επιστρέφει στο αντικείμενο C.

Στο μηχάνημα2 υπάρχει μια κλάση-σκελετός (skeleton) που είναι υπεύθυνη για τρία πράγματα:

- Δέχεται τις κλήσεις των απομακρυσμένων μεθόδων καθώς και τα ορίσματα τα οποία έρχονται από τον πελάτη.
- Εκτελεί τις κλήσεις των απομακρυσμένων μεθόδων προς το τοπικό αντικείμενο, περνώντας τα ορίσματα που έχει λάβει.
- Δέχεται τα αποτελέσματα των μεθόδων και τα επιστρέφει με τη βοήθεια της JVM και του RMI στο μηχάνημα1.

Επομένως κάθε ένα από τα τρία επίπεδα [15] κάνει τα ακόλουθα:

Το **επίπεδο Σκελετού/Στελέχους** είναι υπεύθυνο για την μετάδοση των δεδομένων από τον υπολογιστή-πελάτη προς το απομακρυσμένο αντικείμενο του εξυπηρετητή, καθώς και για την επιστροφή των αποτελεσμάτων από το απομακρυσμένο αντικείμενο στον πελάτη.

Το **επίπεδο απομακρυσμένης αναφοράς** ασχολείται με τη μεταφορά των δεδομένων σε

πιο χαμηλό επίπεδο καθώς και με τα πρωτόκολλα που απαιτούνται για να γίνει η επικοινωνία Σκελετού/Στελέχους.

Το **επίπεδο μεταφοράς** είναι γενικά υπεύθυνο για την δημιουργία, τη διαχείριση, και την παρακολούθηση των συνδέσεων. Παρακολουθεί επίσης και χειρίζεται όλες τις κλήσεις μεταξύ του εξυπηρετητή και των πελατών. Βασίζεται στο TCP/IP.

Τα στελέχη και οι σκελετοί δημιουργούνται με ένα βοηθητικό πρόγραμμα που ονομάζεται μεταγλωττιστής RMI (rmic tool)[16]. Για να μπορεί να καλείται ένα απομακρυσμένο αντικείμενο πρέπει να καταχωρηθεί με ένα συγκεκριμένο όνομα στο μητρώο RMI που βρίσκεται στον εξυπηρετητή. Το απομακρυσμένο αντικείμενο μπορεί τότε να προσπελαστεί με τη χρήση ενός URL που ακολουθεί το πρωτόκολλο RMI. Για παράδειγμα, αν το απομακρυσμένο αντικείμενο έχει καταχωρηθεί στο μητρώο με το όνομα "AddServer" και η διεύθυνση IP του εξυπηρετητή είναι a.b.c.d, το αντικείμενο-πελάτης μπορεί να δημιουργήσει ένα στέλεχος γι' αυτό το απομακρυσμένο αντικείμενο με την χρήση του URL:

Rmi://a.b.c.d./AddServer.

4.3 Παραλληλοποίηση Liveness Model Checking

TLCServer

Ο TLCServer στη liveness checking παράλληλη εκδοχή του, διατηρεί τις ακόλουθες δομές δεδομένων: ένα σύνολο **G** που είναι μέρος του γράφου reachability κατάστασης που το TLC έχει υπολογίσει μέχρι την τρέχουσα χρονική στιγμή, μια ουρά (μια ακολουθία) **U** από ανεξερεύνητες καταστάσεις, δηλαδή όλες τις καταστάσεις στο G των οποίων οι successors δεν έχουν υπολογιστεί ακόμη από το TLC αλλά και ένα ArrayList **sucstates,** που περιλαμβάνει όλες τις μεταβάσεις των καταστάσεων που έχουν εξερευνηθεί, το οποίο αποστέλλεται σε όλους τους workers (για την δημιουργία του behavior γράφου τους) πριν την υλοποίηση του liveness model checking. (Το σύνολο G και η U είναι ίδιες με τις δομές της εκτέλεσης του single workstation που παρουσιάστηκαν στο κεφάλαιο 3, ενότητα 3.7.)

Με την πληκτρολόγηση της κατάλληλης εντολής [βλέπε 4.4] ξεκινά η εκτέλεση του TLCServer.:

1. Ο TLCServer πραγματοποιεί την ανάλυση και την σημασιολογική επεξεργασία των modules που είναι να εξεταστούν από το TLC.

2. Στην συνέχεια κάνει την **Αρχικοποίηση για Liveness Model Checking** όπου υπολογίζονται τα OrderOfSolutions όπως παρουσιάστηκε στο κεφάλαιο 3, ενότητα 3.9.). Ο TLCServer εκτελεί τα επόμενα βήματα, ξεκινώντας με κενά το σύνολο G και την ουρά U:

3. Ελέγχει ότι κάθε assume στην προδιαγραφή ικανοποιείται από τις τιμές που ανατέθηκαν στις σταθερές παραμέτρους.

   Μια ASSUME εντολή θα πρέπει να χρησιμοποιείται μόνο για υποθέσεις σχετικά με τις σταθερές (constants). Θυμίζουμε ότι αυτή η έκφραση δεν πρέπει να περιέχει μεταβλητές.

4. Υπολογίζει το σύνολο των αρχικών καταστάσεων εκτιμώντας το αρχικό κατηγόρημα Init. Για κάθε αρχική κατάσταση s που βρήκε:

   a) Ελέγχει αν το κατηγόρημα Constraint είναι αληθές στην κατάσταση s και ελέγχει αν το σύνολο G περιέχει την κατάσταση. Αν δεν την περιέχει την τοποθετεί στο σύνολο G και στην ουρά U .

   b) Εκτιμά τα κατηγορούμενα Invariant και ImpliedInit στην κατάσταση s, αναφέρει ένα σφάλμα και σταματά αν είτε το ένα είτε το άλλο είναι ψευδές.

5. Δημιουργεί ένα μητρώο (Registry) στον τοπικό υπολογιστή που δέχεται αιτήσεις σε συγκεκριμένο port (εξ ορισμού 10997), όπως αναφέρθηκε πιο πάνω.

6. Καταχωρεί τον server που είναι τύπου TLCServer, που έχουν γίνει όλες οι προηγούμενες λειτουργίες, με το όνομα "TLCServer" στο μητρώο.

7. Περιμένει για εισερχόμενες αιτήσεις.

TLCWorker

Ο TLCWorker πέρα από το σύνολο **G**, που είναι μέρος του γράφου reachability κατάστασης που το TLC έχει υπολογίσει μέχρι την τρέχουσα χρονική στιγμή, ορίζει και μια δεύτερη δομή για το liveness model checking: έναν **behavior** γράφο. (Το G και ο behavior γράφος είναι ίδιες με τις δομές της εκτέλεσης του single workstation που παρουσιάστηκαν στο κεφάλαιο 3, ενότητα 3.7.)

Μετά την εκτέλεση του TLCServer, με την πληκτρολόγηση της κατάλληλης εντολής (σε κάθε υπολογιστή) [βλέπε 4.5] ξεκινά η εκτέλεση του TLCWorker.:

1. Ο TLCWorker πρέπει να αποκτήσει μια αναφορά στο απομακρυσμένο αντικείμενο. Αυτό μπορεί να γίνει με την υπηρεσία αναζήτησης που προσφέρει το μητρώο του RMI. Ο TLCWorker ζητάει μια συγκεκριμένης υπηρεσίας με όνομα "TLCServer" και δέχεται ένα URL για τον απομακρυσμένο πόρο. Μόλις μια αναφορά σε ένα απομακρυσμένο αντικείμενο γίνει γνωστή, αυτό μπορεί να χρησιμοποιηθεί σαν τοπικό αντικείμενο, γεγονός που απλοποιεί σημαντικά την όλη διαδικασία

2. Πραγματοποιεί την ανάλυση και την σημασιολογική επεξεργασία των module που είναι να εξεταστούν από το TLC.

3. Στην συνέχεια ο TLCWorker κάνει την "**Αρχικοποίηση για Liveness Model Checking"** όπου υπολογίζονται τα OrderOfSolutions όπως παρουσιάστηκε στο κεφάλαιο 3, ενότητα 3.9.). Ο TLCWorker εκτελεί τα επόμενα βήματα, ξεκινώντας με κενά τα σύνολα G που διαχειρίζεται ο GManager:

4. Υπολογίζει το σύνολο των αρχικών καταστάσεων εκτιμώντας το αρχικό κατηγόρημα Init. Για κάθε αρχική κατάσταση s που βρήκε:

   a) Ελέγχει αν το κατηγόρημα Constraint είναι αληθές στην κατάσταση s και ελέγχεται αν το σύνολο G περιέχει την κατάσταση. Αν δεν την περιέχει την τοποθετεί στο σύνολο G καθώς και στον behavior γράφο ως αρχική κατάσταση του behavior γράφου.

5. Εκτιμά τα κατηγορούμενα Invariant και ImpliedInit στην κατάσταση s, αναφέρει ένα σφάλμα και σταματά αν είτε το ένα είτε το άλλο είναι ψευδές

6. Μέσω της αναφοράς στο απομακρυσμένο αντικείμενο "TLCServer", ο TLCWorker κάνει το σύνολο του G να είναι ίσο με το σύνολο G του TLCServer.

7. Ο TLCWorker καλεί μια συνάρτηση register του server με όρισμα τον ευατό του (worker). Με αυτόν τον τρόπο καταχωρείται ο TLCWorker στον εξυπηρετητή.

8. Από την στιγμή που αυτός ο TLCWorker καταχωρηθεί, δημιουργείται ένα νήμα TLCServerThread από τον TLCServer για την εξυπηρέτηση του συγκεκριμένου TLCWorker. Κατά την εκτέλεση του νήματος TLCServerThread εκτελούνται τα εξής:

9. Όσο η U του TLCServer δεν είναι κενή το TLCServerThread (Δημιουργία State Space):

   a) Αφαιρεί ένα σύνολο καταστάσεων *states* από την U μεγέθους Blocksize(1024) εν προκειμένου να εξεταστούν αυτές οι καταστάσεις από τον συγκεκριμένο worker.

   b) Καλεί μια συνάρτηση getNextStates του TLCWorker μέσα από τον TLCServerThread για να υπολογίσει τις successor καταστάσεις του συνόλου states που έχει αναλάβει.

   c) Συγκεκριμένα ο TLCWorker με την getNextStates για κάθε μια s κατάσταση από τις καταστάσεις του συνόλου states βρίσκει το σύνολο T με τις successor καταστάσεις της, εκτιμώντας την ενέργεια επόμενης κατάστασης.

   d) Αν το T είναι κενό και η επιλογή για τον έλεγχο ύπαρξης deadlock είναι ενεργοποιημένη, τότε ο TLCWorker αναφέρει ένα σφάλμα αδιεξόδου και η όλη διαδικασία σταματά.

   e) Για κάθε κατάσταση t στο T, ο TLCWorker κάνει τα ακόλουθα:

      i. Διαγράφει τις καταστάσεις του συνόλου T που υπάρχουν ήδη στο δικό του σύνολο G.

ii. Σε αυτές που έχουν μείνει ελέγχει αν το κατηγόρημα Constraint είναι αληθές στην κατάσταση t και αν το βήμα $s \rightarrow t$ ικανοποιεί το ActionConstraint.

iii. Το σύνολο newStates των καταστάσεων που παρήχθησαν καθώς και ένα διάνυσμα liveNextStates με τις μεταβάσεις ($s \rightarrow t$) που πρέπει να εισαχθούν στον behavior γράφο επιστρέφονται στον TLCServerThread.

f) Από το επιστρεφόμενο σύνολο καταστάσεων, οι καταστάσεις οι οποίες δεν εμπεριέχονται στο σύνολο G του TLCServer τοποθετούνται στο σύνολο G καθώς και στην ουρά U του TLCServer. Όσο για τις μεταβάσεις για τον behavior γράφο τοποθετούνται στο ArrayList sucstates του TLCServer που περιλαμβάνει όλες τις μεταβάσεις των καταστάσεων που έχουν εξερευνηθεί.

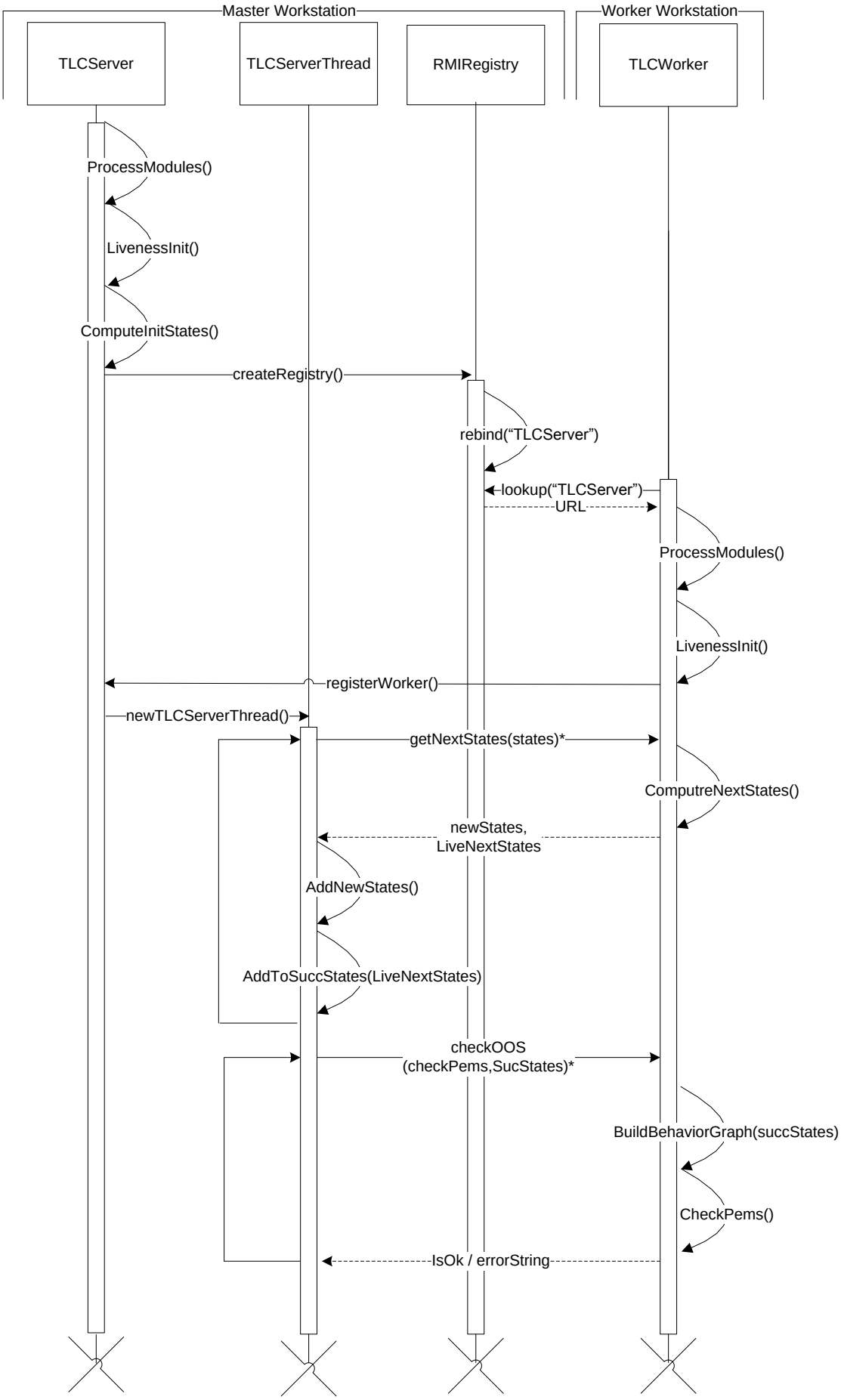
10. Εφόσον ολοκληρωθεί η διαδικασία δημιουργίας του State Space (η U του TLCServer δεν περιλαμβάνει πλέον στοιχεία) ξεκινά το Liveness model checking.

a) Το TLCServerThread αφαιρεί ένα σύνολο από τα pems των OrderOfSolution μεγέθους BlocksizePem (1024) και τοποθετούνται στο διάνυσμα checkPems εν προκειμένου να εξεταστούν στον συγκεκριμένο worker.

Την πρώτη φορά ως αρχικοποίηση ο worker λαμβάνει επιπρόσθετα με το διανύσμα checkPems και το ArrayList sucstates του TLCServer με τις μεταβάσεις όλων των καταστάσεων που έχουν βρεθεί. Ο worker προσθέτει αυτές τις μεταβάσεις στον behavior γράφο αφού εκτιμήσει όλα τα κατηγορήματα και τις ενέργειες που εμφανίζονται στις εκφράσεις Temporal και ImpliedTemporal.

b) Για κάθε ένα pem από το σύνολο checkPems πραγματοποιεί τα βήματα του σταδίου **"Υλοποίηση Liveness Model Checking"** όπως παρουσιάστηκε στο κεφάλαιο 3, ενότητα 3.9.

Στην επόμενη σελίδα ακολουθεί το διάγραμμα ακολουθίας της όλης διαδικασίας.



4.4 Εκτέλεση TLCServer

TLCServer.java : Αυτή η κλάση επεκτείνει την κλάση java.rmi.server. UnicastRemoteObject[17] (η οποία παρέχει ένα σύνολο από μεθόδους που επιτρέπουν στο RMI να δουλεύει σωστά. Συγκεκριμένα σειριοποιεί και αποσειριοποιεί τις απομακρυσμένες αναφορές ενός αντικειμένου.) και αυτόματα εξάγεται για RMI (Remote Method Invocation) πρόσβαση. Αυτό ο τρόπος μας καθιστά ικανούς για client-server πρόσβαση πάνω σε δίκτυο μεταξύ του TLC Server και των TLC Workers. Αυτή υλοποιεί το TLCServerRMI έτσι ώστε να επιτρέπει απομακρυσμένη πρόσβαση στο server object από τους TLC workers.

Για την εκτέλεση του TLCServer πληκτρολογείται μια εντολή της μορφής :

java tlc.tool.TLCServer options spec_file

όπου

spec_file είναι το όνομα του αρχείου που περιέχει την TLA+ προδιαγραφή.

options είναι μια ακολουθία αποτελούμενη από μηδέν ή περισσότερες από τις ακόλουθες επιλογές:

-config *config_file*

Προσδιορίζει ότι το configuration αρχείο ονομάζεται *config_file*, το οποίο πρέπει να είναι ένα αρχείο με επέκταση .cfg. Η επέκταση .cfg μπορεί να παραληφθεί από το *config_file*. Αν αυτή η επιλογή παραληφθεί υποθέτετε ότι έχει το ίδιο όνομα με το *spec_file*, εκτός της επέκτασης .cfg.

-deadlock

Λέει στο TLC να μην ελέγξει για αδιέξοδο. Εάν αυτή η επιλογή δεν προσδιορίζεται, το TLC θα σταματήσει αν βρει ένα αδιέξοδο δηλαδή, μια προσβάσιμη κατάσταση χωρίς successor κατάσταση.

-recover *run_id*

Αυτή η επιλογή κάνει το TLC να ξεκινήσει την εκτέλεση της προδιαγραφής όχι από την αρχή, αλλά από όπου σταμάτησε το τελευταίο σημείο ελέγχου (checkpoint). Όταν το TLC λαμβάνει ένα σημείο ελέγχου, τυπώνει το αναγνωριστικό εκτέλεσης. Η τιμή του run_id πρέπει να είναι αυτό το αναγνωριστικό εκτέλεσης.

-coverage *num*

Αυτή η επιλογή κάνει TLC να τυπώνει "coverage" πληροφορίες κάθε num λεπτά και στο τέλος της εκτέλεσης του. Για κάθε ενέργεια συζευκτέου που "αναθέτει μια τιμή" σε μια μεταβλητή, το TLC τυπώνει τον αριθμό των φορών που ο συζευκτέος έχει πραγματικά χρησιμοποιηθεί στην κατασκευή μιας νέας κατάστασης.

-terse

Κανονικά, το TLC επεκτείνει πλήρως τις τιμές που εμφανίζονται στα μηνύματα σφάλματος ή στην έξοδο από την εκτίμηση των Print εκφράσεων. Η terse επιλογή κάνει το TLC αντί αυτού να τυπώνει μερικώς εκτιμώμενες, συντομότερες εκδόσεις αυτών των τιμών.

-nowarning

Υπάρχουν TLA+ εκφράσεις που είναι νόμιμες αλλά είναι αρκετά αμφίβολες, που η παρουσία τους πιθανόν να υποδεικνύει ένα σφάλμα. Το TLC κανονικά εξάγει μια προειδοποίηση όταν απαριθμεί μια τέτοια αμφίβολη περίπτωση, αυτή η επιλογή αποκρύπτει αυτές τις προειδοποιήσεις.

-fp num
Αρχικοποιεί τον κώδικα για τα 64-bit fingerprints με την τιμή num.

4.5 Εκτέλεση TLCWorker

TLCWorker.java : Αυτή η κλάση επεκτείνει την κλάση java.rmi.server. UnicastRemoteObject[17] και υλοποιεί την TLCWorkerRMI διεπαφή.

Για την εκτέλεση του TLCWorker πληκτρολογείται μια εντολή της μορφής :

java tlc.tool.TLCWorker options spec_file server_name

όπου
spec_file είναι το όνομα του αρχείου που περιέχει την TLA+ προδιαγραφή.
server_name είναι το hostname του TLC server με τον οποίο θα έρθει σε επικοινωνία.
options είναι μια ακολουθία αποτελούμενη από μηδέν ή περισσότερες από τις ακόλουθες επιλογές:

-config ***config_file***
Προσδιορίζει ότι το configuration αρχείο ονομάζεται *config_file*, το οποίο πρέπει να είναι ένα αρχείο με επέκταση .cfg. Η επέκταση .cfg μπορεί να παραληφθεί από το *config_file*. Αν αυτή η επιλογή παραληφθεί υποθέτετε ότι έχει το ίδιο όνομα με το *spec_file*, εκτός της επέκτασης .cfg.

Κεφάλαιο 5: Μετρήσεις

Περίληψη κεφαλαίου

Στο κεφάλαιο αυτό παρουσιάζονται τα αποτελέσματα των μετρήσεων που προέκυψαν κατά την εκτέλεση μιας προδιαγραφής μικρής κλίμακας (5000 καταστάσεις) και στην συνέχεια μιας προδιαγραφής μεγάλης κλίμακας (30000 καταστάσεις). Στην τελική ενότητα αυτού του κεφαλαίου γίνεται παρουσίαση των συμπερασμάτων που έχουν εξαχθεί βάσει των μετρήσεων.

5.1 Μεθοδολογία μετρήσεων

Οι μετρήσεις πραγματοποιήθηκαν σε δύο μηχανήματα του εργαστηρίου του μεταπτυχιακού με τα ακόλουθα χαρακτηριστικά:

Χαρακτηριστικά	
Επεξεργαστής	Intel Xeon CPU , E5520 @ 2.27GHz
CPU cores	4
Μνήμη (GB)	4
Τύπος Δίσκου	SATA2

Η προδιαγραφή που χρησιμοποιήθηκε για τις μετρήσεις ήταν η LiveHourClock.tla για την οποία υπάρχει λεπτομερής ανάλυση τόσο της ίδιας της προδιαγραφής όσο και της εκτέλεσης της στο TLC στο Παράρτημα Α.

Παρατήρηση: Το RMI χρησιμοποιεί για να καταχωρήσει στο registry ένα remote object εξ' ορισμού τη διεύθυνση 127.0.0.1(localhost), IP που δεν είναι προσβάσιμη από άλλο υπολογιστή, οπότε για αυτό τον λόγο στην εντολή για την εκτέλεση του RMIServer και στον RMIWorker εισάγουμε και την δήλωση της ιδιότητας hostname[18]. Η τιμή της συγκεκριμένης ιδιότητας αναπαριστά το host name αλφαριθμητικό το οποίο θα πρέπει να συνδέεται με τα απομακρυσμένα stubs ώστε να επιτρέπεται στους clients να καλούν μεθόδους σε ένα απομακρυσμένο αντικείμενο.

Από την μεριά του **RMIServer**, που εκτελείται στο 1ο μηχάνημα, πληκτρολογούμε την ακόλουθη εντολή:

```
java -Djava.rmi.server.hostname="inf-uth-corei7-1-pg.dyndns.org"  tlc.tool.TLCServer LiveHourClock.tla
```

Όσον αφορά τον **RMIClient,** για την εκτέλεση του, ανάλογα πόσους workers θέλουμε, εκτελούμε ένα script το οποίο αναθέτει σε κάθε core του 2ου μηχανήματος έναν worker μέσω της εντολής taskset[19]. Επιπλέον ορίζουμε με το –Dpid (**p**rocess **id**) μια ιδιότητα αναθέτοντας σε κάθε worker μια μοναδική τιμή η οποία προστίθεται στο τέλος του

ονόματος του metadir (το metadir προσδιορίζει τον κατάλογο *dir* στον οποίο το TLC αποθηκεύει τα αρχεία του που προκύπτουν κατά την εκτέλεση - ορίζεται να είναι ίσο με την τρέχουσα ημερομηνία και ώρα την στιγμή της δημιουργίας του) έτσι ώστε να υπάρχει διαφοροποίηση μεταξύ των client που τρέχουν στο ίδιο μηχάνημα καθώς χρειάζεται κατά την εκτέλεση του αλγορίθμου. Τέλος χρησιμοποιήσαμε την ανακατεύθυνση εξόδου έτσι ώστε τα αποτελέσματα να ανακατευθύνονται σε διαφορετικά αρχεία. Για παράδειγμα για την εκτέλεση τεσσάρων workers το script έχει την ακόλουθη μορφή:

script.sh

```
#!/bin/bash

taskset -c 0 java -Dpid=0 -Djava.rmi.server.hostname="inf-uth-corei7-2-pg.dyndns.org" tlc.tool.TLCWorker LiveHourClock.tla inf-uth-corei7-1-pg.dyndns.org &> worker0.txt &

taskset -c 1 java -Dpid=1 -Djava.rmi.server.hostname="inf-uth-corei7-2-pg.dyndns.org" tlc.tool.TLCWorker LiveHourClock.tla inf-uth-corei7-1-pg.dyndns.org &> worker1.txt &

taskset -c 2 java -Dpid=2 -Djava.rmi.server.hostname="inf-uth-corei7-2-pg.dyndns.org" tlc.tool.TLCWorker LiveHourClock.tla inf-uth-corei7-1-pg.dyndns.org &> worker2.txt &

taskset -c 3 java -Dpid=3 -Djava.rmi.server.hostname="inf-uth-corei7-2-pg.dyndns.org" tlc.tool.TLCWorker LiveHourClock.tla inf-uth-corei7-1-pg.dyndns.org &> worker3.txt &
```

Οι μετρήσεις που ακολουθούν έγιναν αρχικά για πεδίο τιμών 1 έως 5000 που λαμβάνει η μεταβλητή hr στις προδιαγραφές HourClock.tla και LiveHourCLock.tla και στην συνέχεια η ίδια διαδικασία επαναλήφθηκε για πεδίο τιμών από 1 έως 30000. Σε κάθε μια από αυτές τις εκδοχές ο πρώτος πίνακας αντιστοιχεί στην σειριακή εκτέλεση της προδιαγραφής για 1 worker και στην συνέχεια για multiprocessing τοπικά για 2, 3 και 4 workers. Στην σειριακή εκτέλεση κάθε worker λαμβάνει μια-μια τις καταστάσεις όπως και τα Pems.

Οι επόμενοι πίνακες παρουσιάζουν τα αποτελέσματα της παράλληλης εκτέλεσης σε κατανεμημένα περιβάλλοντα. Το block size των states που ανατίθενται σε κάθε worker παραμένει σταθερό στα 1024 όπως ήταν ορισμένο στην ήδη υπάρχουσα παράλληλη υλοποίηση του TLC ενώ πειραματιστήκαμε ως προς το μέγεθος του block size των Pems που ανατίθενται σε κάθε worker προς επεξεργασία.

Σε κάθε μια από αυτές τις περιπτώσεις ο πρώτος πίνακας παρουσιάζει τα αποτελέσματα στον master για πλήθος workers (WorkerCnt) από 1 έως 4. Για κάθε μέτρηση πραγματοποιούμε τρεις εκτελέσεις, ενώ τα στοιχεία της μέτρησης που καταγράφουμε είναι: **(i)** ο χρόνος ολοκλήρωσης της εκτέλεσης (**TStates**) του υπολογισμού των καταστάσεων (δημιουργία State Space), **(ii)** το μέγεθος της μνήμης που έχει δεσμευθεί (**MStates-Allocated**) και **(iii)** το ωφέλιμο μέγεθος μνήμης (**MStates-Used**) για την ίδια διαδικασία (Η Java δεσμεύει περισσότερη μνήμη απ' όση χρειάζεται μια δεδομένη

στιγμή για να ελαχιστοποιήσει τον αριθμό των δεσμεύσεων που θα χρειαστούν), **(iv)** ο χρόνος ολοκλήρωσης της εκτέλεσης (**TLiveness**) του liveness model checking (σημείωση: ο χρόνος TLiveness περιλαμβάνει το χρόνο εκτέλεσης υπολογισμού των καταστάσεων TStates), **(v)** το μέγεθος της μνήμης που έχει δεσμευθεί (**MLiveness-Allocated**) και **(vi)** το ωφέλιμο μέγεθος μνήμης (**MLiveness-Used**) μέχρι και αυτήν την διαδικασία, **(vii)** το συνολικό χρόνο εκτέλεσης (**TTotal**), (**viii**) το συνολικό μέγεθος της μνήμης που έχει δεσμευθεί (**MTotal-Allocated**) και (**ix**) το συνολικό ωφέλιμο μέγεθος μνήμης (**MTotal-Used**). Επίσης για κάθε παράμετρο υπολογίσαμε τον μέσο όρο (**Avg**) και την τυπική απόκλιση (**StDev**) των τιμών.

Οι υπόλοιποι πίνακες παρουσιάζουν τις μετρήσεις για κάθε worker ξεχωριστά. Για κάθε worker ο σχετικός πίνακας δεικτοδοτείται ως Worker1 για τον 1ο worker, Worker2 για τον 2ο worker κ.ο.κ. Στους συγκεκριμένους πίνακες καταγράφουμε τρεις μετρήσεις για τις παραμέτρους **TTotal**, **MTotal-Allocated** και **MTotal-Used**. Και σε αυτήν την περίπτωση υπολογίσαμε για κάθε παράμετρο τον μέσο όρο (**Avg**) και την τυπική απόκλιση (**StDev**) των τιμών.

5.2 Εκτέλεση Προδιαγραφής με 5000 states

Single Workstation (1pem)

WorkerCnt=1	1	2	3	Avg	StDev
Tstates(sec)	3.727	3.693	3.893	3.77	0.11
Mstates(KB)-Allocated	241536	241600	241600	241578.67	36.95
Mstates(KB)-Used	70665	71107	71107	70959.67	255.19
Tliveness(sec)	48.016	50.011	48.559	48.86	1.03
Mliveness-Allocated(KB)	81024	84480	89920	85141.33	4484.72
Mliveness-Used(KB)	74810	67719	57609	66712.67	8644.54
Ttotal(sec)	48.019	50.014	48.562	**48.87**	1.03
Mtotal-Allocated(KB)	81024	84480	89920	**85141.33**	4484.72
Mtotal-Used(KB)	74810	67719	57609	**66712.67**	8644.54

WorkerCnt=2	1	2	3	Avg	StDev
Tstates(sec)	3.691	3.75	3.738	3.73	0.03
Mstates(KB)-Allocated	241600	241664	241600	241621.33	36.95
Mstates(KB)-Used	73483	73119	74416	73672.67	668.98
Tliveness(sec)	48.441	48.675	48.924	48.68	0.24
Mliveness-Allocated(KB)	96384	79488	87168	87680.00	8459.63
Mliveness-Used(KB)	45152	39021	42079	42084.00	3065.50
Ttotal(sec)	48.444	48.678	48.926	**48.68**	0.24
Mtotal-Allocated(KB)	96384	79488	87168	**87680.00**	8459.63
Mtotal-Used(KB)	45152	39021	42079	**42084.00**	3065.50

WorkerCnt=3	1	2	3	Avg	StDev
Tstates(sec)	3.77	3.837	3.901	3.84	0.07
Mstates(KB)-Allocated	241600	241664	241600	241621.33	36.95
Mstates(KB)-Used	72772	75532	73813	74039.00	1393.81
Tliveness(sec)	48.133	48.293	48.248	48.22	0.08
Mliveness-Allocated(KB)	94272	88704	91200	91392.00	2788.96
Mliveness-Used(KB)	67750	75830	69787	71122.33	4202.25
Ttotal(sec)	48.736	48.296	48.251	**48.43**	0.27
Mtotal-Allocated(KB)	94272	88704	91200	**91392.00**	2788.96
Mtotal-Used(KB)	67750	75830	69787	**71122.33**	4202.25

WorkerCnt=4	1	2	3	Avg	StDev
Tstates(sec)	3.836	3.759	3.788	3.79	0.04
Mstates(KB)-Allocated	241600	241600	241408	241536.00	110.85
Mstates(KB)-Used	75141	75390	76964	75831.67	988.50
Tliveness(sec)	48.586	48.597	48.112	48.43	0.28
Mliveness-Allocated(KB)	93312	90944	96128	93461.33	2595.22
Mliveness-Used(KB)	58621	50670	63978	57756.33	6696.00
Ttotal(sec)	48.589	48.6	48.115	**48.43**	0.28
Mtotal-Allocated(KB)	93312	90944	96128	**93461.33**	2595.22
Mtotal-Used(KB)	58621	50670	63978	**57756.33**	6696.00

Παράλληλη εκτέλεση σε κατανεμημένο περιβάλλον (64 pems)

WorkerCnt=1	1	2	3	Avg	StDev
Tstates(sec)	1.113	1.114	1.092	1.11	0.01
Mstates(KB)-Allocated	60672	60672	60672	60672.00	0.00
Mstates(KB)-Used	8585	8604	8624	8604.33	19.50
Tliveness(sec)	66.283	66.834	66.126	66.41	0.37
Mliveness-Allocated(KB)	60672	60672	60672	60672.00	0.00
Mliveness-Used(KB)	13922	13938	13969	13943.00	23.90
Ttotal(sec)	66.645	67.179	66.465	**66.76**	0.37
Mtotal-Allocated(KB)	60672	60672	60672	**60672.00**	0.00
Mtotal-Used(KB)	13922	13938	13969	**13943.00**	23.90

WorkerCnt=2	1	2	3	Avg	StDev
Tstates(sec)	0.874	0.915	0.88	0.89	0.02
Mstates(KB)-Allocated	58752	58752	58944	58816.00	110.85
Mstates(KB)-Used	7488	7499	7101	7362.67	226.68
Tliveness(sec)	39.068	38.84	38.969	38.96	0.11
Mliveness-Allocated(KB)	58752	58752	58944	58816.00	110.85
Mliveness-Used(KB)	17420	17402	16948	17256.67	267.46
Ttotal(sec)	39.789	40.063	39.798	**39.88**	0.16
Mtotal-Allocated(KB)	58752	58752	58944	**58816.00**	110.85
Mtotal-Used(KB)	17420	17402	16948	**17256.67**	267.46

WorkerCnt=3	1	2	3	Avg	StDev
Tstates(sec)	0.856	0.738	0.761	0.79	0.06
Mstates(KB)-Allocated	75328	59968	59904	65066.67	8886.63
Mstates(KB)-Used	7567	7675	7771	7671.00	102.06
Tliveness(sec)	29.021	28.843	29.116	28.99	0.14
Mliveness-Allocated(KB)	75328	59520	59520	64789.33	9126.75
Mliveness-Used(KB)	22661	10070	10105	14278.67	7259.33
Ttotal(sec)	30.199	30.139	30.199	**30.18**	0.03
Mtotal-Allocated(KB)	75328	59520	59520	**64789.33**	9126.75
Mtotal-Used(KB)	22661	10070	10105	**14278.67**	7259.33

WorkerCnt=4	1	2	3	Avg	StDev
Tstates(sec)	0.665	0.778	0.832	0.76	0.09
Mstates(KB)-Allocated	73600	72320	59264	68394.67	7933.25
Mstates(KB)-Used	7327	9208	9744	8759.67	1269.34
Tliveness(sec)	23.707	23.959	23.99	23.89	0.16
Mliveness-Allocated(KB)	73600	72320	59456	68458.67	7822.76
Mliveness-Used(KB)	25573	27331	16355	23086.33	5895.40
Ttotal(sec)	25.716	25.934	25.813	**25.82**	0.11
Mtotal-Allocated(KB)	73600	72320	59456	**68458.67**	7822.76
Mtotal-Used(KB)	27035	28856	17062	**24317.67**	6349.22

WorkerCnt=1

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	66.392	66.897	66.188	66.49	0.36
Mtotal-Allocated(KB)	157056	158144	157120	157440.00	610.52
Mtotal-Used(KB)	52834	52552	59268	54884.67	3798.70

WorkerCnt=2

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	39.506	39.809	39.125	39.48	0.34
Mtotal-Allocated(KB)	157440	147392	158720	154517.33	6203.82
Mtotal-Used(KB)	59282	93847	65557	72895.33	18413.94

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	39.145	39.482	39.376	39.33	0.17
Mtotal-Allocated(KB)	158400	157056	157120	157525.33	758.16
Mtotal-Used(KB)	39574	40959	108068	62867.00	39151.34

WorkerCnt=3

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	29.511	29.807	30.25	29.86	0.37
Mtotal-Allocated(KB)	147520	157056	147776	150784.00	5433.22
Mtotal-Used(KB)	89210	114507	56816	86844.33	28918.16

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	29.195	29.461	30.548	29.73	0.72
Mtotal-Allocated(KB)	157312	147712	147776	150933.33	5524.18
Mtotal-Used(KB)	52900	90661	85956	76505.67	20578.02

Worker3	1	2	3	Avg	StDev
Ttotal(sec)	29.855	29.116	30.896	29.96	0.89
Mtotal-Allocated(KB)	147456	157184	158848	154496.00	6153.33
Mtotal-Used(KB)	81197	64990	61243	69143.33	10605.57

WorkerCnt=4

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	24.898	25.171	24.981	25.02	0.14
Mtotal-Allocated(KB)	157440	157056	157056	157184.00	221.70
Mtotal-Used(KB)	99620	111341	42925	84628.67	36588.83

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	25.281	24.559	25.27	25.04	0.41
Mtotal-Allocated(KB)	157376	147584	157376	154112.00	5653.41
Mtotal-Used(KB)	100233	62655	66952	76613.33	20567.75

Worker3	1	2	3	Avg	StDev
Ttotal(sec)	24.291	24.877	24.635	24.60	0.29
Mtotal-Allocated(KB)	157440	157056	157120	157205.33	205.73
Mtotal-Used(KB)	59289	71482	72982	67917.67	7510.19

Worker4	1	2	3	Avg	StDev
Ttotal(sec)	24.588	25.485	25.641	25.24	0.57
Mtotal-Allocated(KB)	147904	147968	157568	151146.67	5561.13
Mtotal-Used(KB)	68539	72086	103395	81340.00	19182.35

Παράλληλη εκτέλεση σε κατανεμημένο περιβάλλον (128 pems)

WorkerCnt=1	1	2	3	Avg	StDev
Tstates(sec)	1.1	1.095	1.098	1.10	0.00
Mstates(KB)-Allocated	60672	60672	60672	60672.00	0.00
Mstates(KB)-Used	8602	8600	8604	8602.00	2.00
Tliveness(sec)	63.249	60.567	62.224	62.01	1.35
Mliveness-Allocated(KB)	60672	60672	60672	60672.00	0.00
Mliveness-Used(KB)	13670	13688	13648	13668.67	20.03
Ttotal(sec)	63.612	60.908	62.568	**62.36**	1.36
Mtotal-Allocated(KB)	60672	60672	60672	**60672.00**	0.00
Mtotal-Used(KB)	13760	13688	13648	**13698.67**	56.76

WorkerCnt=2	1	2	3	Avg	StDev
Tstates(sec)	0.904	0.9	0.921	0.91	0.01
Mstates(KB)-Allocated	58944	58944	58944	58944.00	0.00
Mstates(KB)-Used	7113	7318	7364	7265.00	133.63
Tliveness(sec)	36.62	36.702	36.812	36.71	0.10
Mliveness-Allocated(KB)	58944	58944	58944	58944.00	0.00
Mliveness-Used(KB)	16667	16927	16991	16861.67	171.60
Ttotal(sec)	37.332	38.54	38.293	**38.06**	0.64
Mtotal-Allocated(KB)	58944	58944	58944	**58944.00**	0.00
Mtotal-Used(KB)	16667	16927	16991	**16861.67**	171.60

WorkerCnt=3	1	2	3	Avg	StDev
Tstates(sec)	0.736	0.802	0.75	0.76	0.03
Mstates(KB)-Allocated	59840	76480	75264	70528.00	9276.03
Mstates(KB)-Used	7600	7279	8051	7643.33	387.82
Tliveness(sec)	27.596	27.561	27.884	27.68	0.18
Mliveness-Allocated(KB)	59712	76480	75264	70485.33	9349.77
Mliveness-Used(KB)	9847	22385	22487	18239.67	7268.44
Ttotal(sec)	28.941	29.025	29.096	**29.02**	0.08
Mtotal-Allocated(KB)	59712	76480	75264	**70485.33**	9349.77
Mtotal-Used(KB)	9847	22385	22623	**18285.00**	7308.49

WorkerCnt=4	1	2	3	Avg	StDev
Tstates(sec)	0.683	0.728	0.68	0.70	0.03
Mstates(KB)-Allocated	75456	59392	73472	69440.00	8758.18
Mstates(KB)-Used	34640	10175	7906	17573.67	14823.36
Tliveness(sec)	22.565	22.352	22.435	22.45	0.11
Mliveness-Allocated(KB)	75520	58496	73472	69162.67	9294.19
Mliveness-Used(KB)	23576	16620	26206	22134.00	4953.02
Ttotal(sec)	25.154	25.02	25.131	**25.10**	0.07
Mtotal-Allocated(KB)	75520	58496	73472	**69162.67**	9294.19
Mtotal-Used(KB)	25213	17113	27194	**23173.33**	5341.05

WorkerCnt=1

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	63.494	60.635	62.292	62.14	1.44
Mtotal-Allocated(KB)	160384	162432	162368	161728.00	1164.38
Mtotal-Used(KB)	39740	80751	67408	62633.00	20918.32

WorkerCnt=2

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	36.688	38.36	37.976	37.67	0.88
Mtotal-Allocated(KB)	161920	162112	160512	161514.67	873.63
Mtotal-Used(KB)	81846	67758	35672	61758.67	23664.39

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	36.972	37.948	37.63	37.52	0.50
Mtotal-Allocated(KB)	160512	162560	162112	161728.00	1076.65
Mtotal-Used(KB)	40134	32045	62970	45049.67	16037.82

WorkerCnt=3

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	27.947	28.766	28.744	28.49	0.47
Mtotal-Allocated(KB)	162560	161920	162496	162325.33	352.48
Mtotal-Used(KB)	33606	66849	64133	54862.67	18458.83

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	28.61	28.023	28.407	28.35	0.30
Mtotal-Allocated(KB)	162496	162496	162176	162389.33	184.75
Mtotal-Used(KB)	62175	68549	73609	68111.00	5729.57

Worker3	1	2	3	Avg	StDev
Ttotal(sec)	28.263	28.329	28.024	28.21	0.16
Mtotal-Allocated(KB)	162048	162176	162176	162133.33	73.90
Mtotal-Used(KB)	96108	78221	72986	82438.33	12124.20

WorkerCnt=4

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	24.114	24.272	23.754	24.05	0.27
Mtotal-Allocated(KB)	162368	162688	162496	162517.33	161.06
Mtotal-Used(KB)	76057	73133	59846	69678.67	8639.93

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	23.84	23.66	24.405	23.97	0.39
Mtotal-Allocated(KB)	162368	162176	162432	162325.33	133.23
Mtotal-Used(KB)	75087	119377	75893	90119.00	25341.38

Worker3	1	2	3	Avg	StDev
Ttotal(sec)	24.738	24.638	24.755	24.71	0.06
Mtotal-Allocated(KB)	160512	161984	162176	161557.33	910.36
Mtotal-Used(KB)	49436	93374	82287	75032.33	22849.72

Worker4	1	2	3	Avg	StDev
Ttotal(sec)	24.463	23.975	24.031	24.16	0.27
Mtotal-Allocated(KB)	162624	162176	162048	162282.67	302.45
Mtotal-Used(KB)	42578	81758	82204	68846.67	22750.43

Παράλληλη εκτέλεση σε κατανεμημένο περιβάλλον (256 pems)

WorkerCnt=1	1	2	3	Avg	StDev
Tstates(sec)	1.152	1.102	1.107	1.12	0.03
Mstates(KB)-Allocated	60672	60672	60672	60672.00	0.00
Mstates(KB)-Used	8585	8585	8585	8585.00	0.00
Tliveness(sec)	58.677	60.509	61.356	60.18	1.37
Mliveness-Allocated(KB)	60672	60672	60672	60672.00	0.00
Mliveness-Used(KB)	13543	13543	13543	13543.00	0.00
Ttotal(sec)	59.044	60.878	61.74	**60.55**	1.38
Mtotal-Allocated(KB)	60672	60672	60672	**60672.00**	0.00
Mtotal-Used(KB)	13543	13543	13543	**13543.00**	0.00

WorkerCnt=2	1	2	3	Avg	StDev
Tstates(sec)	0.875	0.868	0.913	0.89	0.02
Mstates(KB)-Allocated	58944	58944	58752	58880.00	110.85
Mstates(KB)-Used	7247	7258	7650	7385.00	229.56
Tliveness(sec)	35.091	35.451	35.558	35.37	0.24
Mliveness-Allocated(KB)	58944	58944	58752	58880.00	110.85
Mliveness-Used(KB)	16679	16818	17134	16877.00	233.17
Ttotal(sec)	36.938	37.421	36.833	**37.06**	0.31
Mtotal-Allocated(KB)	58944	58944	58752	**58880.00**	110.85
Mtotal-Used(KB)	16679	16818	17134	**16877.00**	233.17

WorkerCnt=3	1	2	3	Avg	StDev
Tstates(sec)	0.729	0.716	0.708	0.72	0.01
Mstates(KB)-Allocated	59776	59968	75328	65024.00	8924.04
Mstates(KB)-Used	7623	7541	7671	7611.67	65.74
Tliveness(sec)	25.875	25.747	26.064	25.90	0.16
Mliveness-Allocated(KB)	59712	59520	75328	64853.33	9071.84
Mliveness-Used(KB)	9181	9276	22417	13624.67	7614.53
Ttotal(sec)	29.285	29.166	29.179	**29.21**	0.07
Mtotal-Allocated(KB)	59712	59520	75328	**64853.33**	9071.84
Mtotal-Used(KB)	9599	9696	22417	**13904.00**	7372.63

WorkerCnt=4	1	2	3	Avg	StDev
Tstates(sec)	0.748	0.679	0.693	0.71	0.04
Mstates(KB)-Allocated	107008	59392	107008	91136.00	27491.11
Mstates(KB)-Used	6412	9948	6548	7636.00	2003.41
Tliveness(sec)	21.718	21.738	21.704	21.72	0.02
Mliveness-Allocated(KB)	107008	58560	107008	90858.67	27971.47
Mliveness-Used(KB)	24112	16282	24420	21604.67	4612.14
Ttotal(sec)	24.223	24.35	24.328	**24.30**	0.07
Mtotal-Allocated(KB)	107008	58560	107008	**90858.67**	27971.47
Mtotal-Used(KB)	27496	16993	27810	**24099.67**	6156.56

WorkerCnt=1

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	58.738	60.571	61.419	60.24	1.37
Mtotal-Allocated(KB)	164288	164608	164224	164373.33	205.73
Mtotal-Used(KB)	74815	117114	111270	101066.33	22921.33

WorkerCnt=2

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	36.627	36.939	36.516	36.69	0.22
Mtotal-Allocated(KB)	166208	166144	164800	165717.33	795.08
Mtotal-Used(KB)	58336	71109	118594	82679.67	31751.64

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	36.248	37.309	36.148	36.57	0.64
Mtotal-Allocated(KB)	164864	164608	166464	165312.00	1005.84
Mtotal-Used(KB)	64425	130777	45857	80353.00	44644.46

WorkerCnt=3

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	28.538	28.833	28.272	28.55	0.28
Mtotal-Allocated(KB)	164928	166528	164736	165397.33	983.88
Mtotal-Used(KB)	94807	25473	88960	69746.67	38453.41

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	28.881	28.499	28.604	28.66	0.20
Mtotal-Allocated(KB)	166144	164800	165760	165568.00	692.27
Mtotal-Used(KB)	76347	22809	77390	58848.67	31215.62

Worker3	1	2	3	Avg	StDev
Ttotal(sec)	28.238	28.173	29.015	28.48	0.47
Mtotal-Allocated(KB)	166464	166592	166208	166421.33	195.52
Mtotal-Used(KB)	54245	60905	44990	53380.00	7992.68

WorkerCnt=4

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	23.791	23.634	22.952	23.46	0.45
Mtotal-Allocated(KB)	165056	166208	164864	165376.00	726.90
Mtotal-Used(KB)	126463	129620	83850	113311.00	25562.76

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	23.18	23.023	23.287	23.16	0.13
Mtotal-Allocated(KB)	166144	166080	165888	166037.33	133.23
Mtotal-Used(KB)	48241	109280	145599	101040.00	49199.27

Worker3	1	2	3	Avg	StDev
Ttotal(sec)	22.854	23.972	23.913	23.58	0.63
Mtotal-Allocated(KB)	166208	166080	165888	166058.67	161.06
Mtotal-Used(KB)	52465	50250	81944	61553.00	17693.82

Worker4	1	2	3	Avg	StDev
Ttotal(sec)	23.522	23.352	23.535	23.47	0.10
Mtotal-Allocated(KB)	166080	165184	165760	165674.67	454.05
Mtotal-Used(KB)	85650	66640	135504	95931.33	35564.62

Παράλληλη εκτέλεση σε κατανεμημένο περιβάλλον (512 pems)

WorkerCnt=1	1	2	3	Avg	StDev
Tstates(sec)	1.11	1.125	1.107	1.11	0.01
Mstates(KB)-Allocated	60672	60672	60672	60672.00	0.00
Mstates(KB)-Used	8601	8596	8588	8595.00	6.56
Tliveness(sec)	58.339	59.983	59.959	59.43	0.94
Mliveness-Allocated(KB)	60672	60672	60672	60672.00	0.00
Mliveness-Used(KB)	13465	13459	13443	13455.67	11.37
Ttotal(sec)	58.704	60.363	60.343	**59.80**	0.95
Mtotal-Allocated(KB)	60672	60672	60672	**60672.00**	0.00
Mtotal-Used(KB)	13465	13459	13443	**13455.67**	11.37

WorkerCnt=2	1	2	3	Avg	StDev
Tstates(sec)	0.916	0.893	0.905	0.90	0.01
Mstates(KB)-Allocated	58816	58944	58944	58901.33	73.90
Mstates(KB)-Used	7637	7355	7133	7375.00	252.59
Tliveness(sec)	35.277	35.722	35.272	35.42	0.26
Mliveness-Allocated(KB)	58816	58944	58944	58901.33	73.90
Mliveness-Used(KB)	17014	16751	16498	16754.33	258.02
Ttotal(sec)	36.144	37.413	36.041	**36.53**	0.76
Mtotal-Allocated(KB)	58816	58944	58944	**58901.33**	73.90
Mtotal-Used(KB)	17074	16751	16498	**16774.33**	288.71

WorkerCnt=3	1	2	3	Avg	StDev
Tstates(sec)	0.706	0.77	0.754	0.74	0.03
Mstates(KB)-Allocated	75328	59904	59904	65045.33	8905.05
Mstates(KB)-Used	7369	7756	7761	7628.67	224.89
Tliveness(sec)	25.213	25.467	25.33	25.34	0.13
Mliveness-Allocated(KB)	75328	59456	59776	64853.33	9072.74
Mliveness-Used(KB)	21186	9361	9286	13277.67	6848.92
Ttotal(sec)	30.304	30.449	30.522	**30.43**	0.11
Mtotal-Allocated(KB)	75328	59456	59776	**64853.33**	9072.74
Mtotal-Used(KB)	22135	9755	9680	**13856.67**	7169.35

WorkerCnt=4	1	2	3	Avg	StDev
Tstates(sec)	0.666	0.833	0.799	0.77	0.09
Mstates(KB)-Allocated	107008	59648	106880	91178.67	27306.43
Mstates(KB)-Used	6597	8711	6629	7312.33	1211.39
Tliveness(sec)	19.775	19.874	19.708	19.79	0.08
Mliveness-Allocated(KB)	107008	56704	106880	90197.33	29006.15
Mliveness-Used(KB)	24421	16737	24654	21937.33	4505.13
Ttotal(sec)	26.353	26.537	26.702	**26.53**	0.17
Mtotal-Allocated(KB)	107008	56704	106880	**90197.33**	29006.15
Mtotal-Used(KB)	27808	17367	28040	**24405.00**	6096.19

WorkerCnt=1

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	58.462	60.017	60.014	59.50	0.90
Mtotal-Allocated(KB)	167808	168128	168128	168021.33	184.75
Mtotal-Used(KB)	137721	133359	140172	137084.00	3450.88

WorkerCnt=2

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	35.666	37.117	35.701	36.16	0.83
Mtotal-Allocated(KB)	161088	168000	168128	165738.67	4028.10
Mtotal-Used(KB)	83625	132702	81459	99262.00	28980.13

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	36.044	36.758	35.393	36.07	0.68
Mtotal-Allocated(KB)	167296	168640	168640	168192.00	775.96
Mtotal-Used(KB)	64108	100524	86611	83747.67	18376.08

WorkerCnt=3

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	29.644	29.809	29.474	29.64	0.17
Mtotal-Allocated(KB)	168768	161728	168064	166186.67	3877.33
Mtotal-Used(KB)	116800	89670	61260	89243.33	27772.46

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	29.987	30.161	30.236	30.13	0.13
Mtotal-Allocated(KB)	168064	157056	164672	163264.00	5637.45
Mtotal-Used(KB)	111716	75857	93344	93639.00	17931.32

Worker3	1	2	3	Avg	StDev
Ttotal(sec)	29.278	29.478	29.836	29.53	0.28
Mtotal-Allocated(KB)	156480	168000	149632	158037.33	9282.50
Mtotal-Used(KB)	118321	141037	63893	107750.33	39643.45

WorkerCnt=4

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	25.95	25.588	25.626	25.72	0.20
Mtotal-Allocated(KB)	161088	148224	157440	155584.00	6629.79
Mtotal-Used(KB)	79717	62007	53359	65027.67	13436.12

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	24.977	23.494	25.262	24.58	0.95
Mtotal-Allocated(KB)	152640	155392	163392	157141.33	5585.38
Mtotal-Used(KB)	20868	109254	70121	66747.67	44289.45

Worker3	1	2	3	Avg	StDev
Ttotal(sec)	25.611	25.873	25.966	25.82	0.18
Mtotal-Allocated(KB)	167168	159360	167936	164821.33	4745.22
Mtotal-Used(KB)	27890	54716	56957	46521.00	16173.78

Worker4	1	2	3	Avg	StDev
Ttotal(sec)	25.299	25.248	26.288	25.61	0.59
Mtotal-Allocated(KB)	155456	168704	168448	164202.67	7575.92
Mtotal-Used(KB)	73256	79506	124899	92553.67	28185.65

Παράλληλη εκτέλεση σε κατανεμημένο περιβάλλον (1024 pems)

WorkerCnt=1	1	2	3	Avg	StDev
Tstates(sec)	1.076	1.092	1.125	1.10	0.02
Mstates(KB)-Allocated	60672	60672	60672	60672.00	0.00
Mstates(KB)-Used	8585	8585	8612	8594.00	15.59
Tliveness(sec)	59.843	59.939	59.487	59.76	0.24
Mliveness-Allocated(KB)	60672	60672	60672	60672.00	0.00
Mliveness-Used(KB)	13449	13449	13476	13458.00	15.59
Ttotal(sec)	60.221	60.312	59.822	**60.12**	0.26
Mtotal-Allocated(KB)	60672	60672	60672	**60672.00**	0.00
Mtotal-Used(KB)	13449	13499	13476	**13474.67**	25.03

WorkerCnt=2	1	2	3	Avg	StDev
Tstates(sec)	0.919	0.877	0.894	0.90	0.02
Mstates(KB)-Allocated	58752	58752	58752	58752.00	0.00
Mstates(KB)-Used	7648	7651	7651	7650.00	1.73
Tliveness(sec)	30.889	30.956	30.495	30.78	0.25
Mliveness-Allocated(KB)	58752	58752	58752	58752.00	0.00
Mliveness-Used(KB)	17037	17001	16995	17011.00	22.72
Ttotal(sec)	40.959	40.999	40.479	**40.81**	0.29
Mtotal-Allocated(KB)	58752	58752	58752	**58752.00**	0.00
Mtotal-Used(KB)	17108	17072	17066	**17082.00**	22.72

WorkerCnt=3	1	2	3	Avg	StDev
Tstates(sec)	0.743	0.735	0.715	0.73	0.01
Mstates(KB)-Allocated	75136	59840	59904	64960.00	8812.73
Mstates(KB)-Used	7968	7735	8159	7954.00	212.35
Tliveness(sec)	19.782	19.645	19.364	19.60	0.21
Mliveness-Allocated(KB)	75136	59712	59520	64789.33	8960.99
Mliveness-Used(KB)	21233	9172	9363	13256.00	6908.94
Ttotal(sec)	31.412	31.13	31.609	**31.38**	0.24
Mtotal-Allocated(KB)	75136	59712	59520	**64789.33**	8960.99
Mtotal-Used(KB)	22337	9794	9989	**14040.00**	7186.07

WorkerCnt=4	1	2	3	Avg	StDev
Tstates(sec)	0.67	0.737	0.742	0.72	0.04
Mstates(KB)-Allocated	107008	73472	75456	85312.00	18815.46
Mstates(KB)-Used	6582	8390	8932	7968.00	1230.52
Tliveness(sec)	19.449	19.616	19.474	19.51	0.09
Mliveness-Allocated(KB)	107008	73472	75456	85312.00	18815.46
Mliveness-Used(KB)	24.389	26.498	26.492	25.79	1.22
Ttotal(sec)	29.726	30.295	30.452	**30.16**	0.38
Mtotal-Allocated(KB)	107008	73472	75456	**85312.00**	18815.46
Mtotal-Used(KB)	27777	27564	27986	**27775.67**	211.00

WorkerCnt=1

Worker1	1	2	3	Avg	StDev
Ttotal(ms)	59969	59999	59547	59838.33	252.75
Mtotal-Allocated(KB)	169664	169536	169408	169536.00	128.00
Mtotal-Used(KB)	98731	53531	62883	71715.00	23859.24

WorkerCnt=2

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	40.299	40.757	40.215	40.42	0.29
Mtotal-Allocated(KB)	158848	152384	163456	158229.33	5561.87
Mtotal-Used(KB)	35886	98369	25345	53200.00	39470.97

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	40.664	40.41	39.867	40.31	0.41
Mtotal-Allocated(KB)	169536	169664	169472	169557.33	97.76
Mtotal-Used(KB)	98029	39506	101232	79589.00	34749.82

WorkerCnt=3

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	30.391	30.173	31.322	30.63	0.61
Mtotal-Allocated(KB)	168896	164032	169536	167488.00	3010.04
Mtotal-Used(KB)	84289	97200	99739	93742.67	8284.96

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	30.734	30.909	30.985	30.88	0.13
Mtotal-Allocated(KB)	169664	156800	151040	159168.00	9535.14
Mtotal-Used(KB)	98262	96021	37499	77260.67	34452.84

Worker3	1	2	3	Avg	StDev
Ttotal(sec)	31.115	30.564	30.603	30.76	0.31
Mtotal-Allocated(KB)	156416	169600	169664	165226.67	7630.33
Mtotal-Used(KB)	89188	65914	80352	78484.67	11748.83

WorkerCnt=4

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	29.101	29.251	29.071	29.14	0.10
Mtotal-Allocated(KB)	157056	149312	153088	153152.00	3872.40
Mtotal-Used(KB)	76293	43020	16874	45395.67	29780.65

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	28.71	29.601	29.714	29.34	0.55
Mtotal-Allocated(KB)	146240	169664	154816	156906.67	11851.12
Mtotal-Used(KB)	64354	33200	76912	58155.33	22505.61

Worker3	1	2	3	Avg	StDev
Ttotal(sec)	28.411	29.942	30.035	29.46	0.91
Mtotal-Allocated(KB)	147520	147072	148416	147669.33	684.33
Mtotal-Used(KB)	81858	99892	100185	93978.33	10497.54

Worker4	1	2	3	Avg	StDev
Ttotal(sec)	29.36	28.953	29.364	29.23	0.24
Mtotal-Allocated(KB)	164032	141632	169472	158378.67	14755.90
Mtotal-Used(KB)	89025	24319	101257	71533.67	41343.97

5.3 Εκτέλεση Προδιαγραφής με 30000 states

Single Workstation(1pem)

WorkerCnt=1	1	2	3	Avg	StDev
Tstates(sec)	220.979	228.959	207.178	219.04	11.02
Mstates(KB)-Allocated	114176	131520	125248	123648.00	8782.00
Mstates(KB)-Used	66574	101220	57858	75217.33	22936.79
Tliveness(sec)	2093.484	2101.591	2132.974	2109.35	20.86
Mliveness-Allocated(KB)	648512	653248	631232	644330.67	11588.30
Mliveness-Used(KB)	504698	528007	431121	487942.00	50569.72
Ttotal(sec)	2093.518	2101.627	2133.008	**2109.38**	20.86
Mtotal-Allocated(KB)	648512	653248	631232	**644330.67**	11588.30
Mtotal-Used(KB)	504698	528007	431121	**487942.00**	50569.72

WorkerCnt=2	1	2	3	Avg	StDev
Tstates(sec)	210.224	213.711	230.994	218.31	11.12
Mstates(KB)-Allocated	151424	109504	141184	134037.33	21854.70
Mstates(KB)-Used	98130	59201	80067	79132.67	19481.31
Tliveness(sec)	2079.136	2115.732	2105.809	2100.23	18.93
Mliveness-Allocated(KB)	696832	700416	716608	704618.67	10536.57
Mliveness-Used(KB)	327484	328531	256348	304121.00	41375.94
Ttotal(sec)	2079.171	2115.768	2105.844	**2100.26**	18.93
Mtotal-Allocated(KB)	696832	700416	716608	**704618.67**	10536.57
Mtotal-Used(KB)	327484	328531	256348	**304121.00**	41375.94

WorkerCnt=3	1	2	3	Avg	StDev
Tstates(sec)	254.097	227.416	279.41	253.64	26.00
Mstates(KB)-Allocated	108928	129728	108224	115626.67	12217.18
Mstates(KB)-Used	66465	54334	60893	60564.00	6072.19
Tliveness(sec)	2177.31	2115.134	2225.624	2172.69	55.39
Mliveness-Allocated(KB)	626048	657920	698880	660949.33	36510.38
Mliveness-Used(KB)	291408	346212	523780	387133.33	121470.59
Ttotal(sec)	2177.347	2115.169	2225.661	**2172.73**	55.39
Mtotal-Allocated(KB)	626048	657920	698880	**660949.33**	36510.38
Mtotal-Used(KB)	291408	346212	523780	**387133.33**	121470.59

WorkerCnt=4	1	2	3	Avg	StDev
Tstates(sec)	325.374	201.638	312.421	279.81	68.01
Mstates(KB)-Allocated	91712	123584	82432	99242.67	21584.83
Mstates(KB)-Used	34510	64695	32041	43748.67	18182.01
Tliveness(sec)	2421.111	2103.259	2425.793	2316.72	184.88
Mliveness-Allocated(KB)	392640	617728	404992	471786.67	126539.71
Mliveness-Used(KB)	308096	420588	333418	354034.00	59011.67
Ttotal(sec)	2421.149	2103.294	2425.832	**2316.76**	184.88
Mtotal-Allocated(KB)	392640	617728	404992	**471786.67**	126539.71
Mtotal-Used(KB)	308096	420588	333418	**354034.00**	59011.67

Παράλληλη εκτέλεση σε κατανεμημένο περιβάλλον (256 pems)

WorkerCnt=1	1	2	3	Avg	StDev
Tstates(sec)	1.978	1.921	2.008	1.97	0.04
Mstates(KB)-Allocated	133952	131648	136704	134101.33	2531.31
Mstates(KB)-Used	87824	59383	59912	69039.67	16269.86
Tliveness(sec)	2666.648	2643.202	2619.484	2643.11	23.58
Mliveness-Allocated(KB)	139776	131776	131264	134272.00	4773.47
Mliveness-Used(KB)	84622	65505	55278	68468.33	14894.75
Ttotal(sec)	2667.061	2643.648	2619.908	**2643.54**	23.58
Mtotal-Allocated(KB)	139776	131776	131264	**134272.00**	4773.47
Mtotal-Used(KB)	84622	65505	55278	**68468.33**	14894.75

WorkerCnt=2	1	2	3	Avg	StDev
Tstates(sec)	1.862	1.766	2.01	1.88	0.12
Mstates(KB)-Allocated	147968	150528	146048	148181.33	2247.61
Mstates(KB)-Used	97992	93008	82018	91006.00	8173.01
Tliveness(sec)	1528.982	1513.89	1509.536	1517.47	10.21
Mliveness-Allocated(KB)	143232	153344	147136	147904.00	5099.56
Mliveness-Used(KB)	57637	108641	99090	88456.00	27113.90
Ttotal(sec)	1533.704	1528.516	1521.615	**1527.95**	6.06
Mtotal-Allocated(KB)	143232	153344	147136	**147904.00**	5099.56
Mtotal-Used(KB)	57637	108641	99090	**88456.00**	27113.90

WorkerCnt=3	1	2	3	Avg	StDev
Tstates(sec)	1.476	1.632	1.557	1.56	0.08
Mstates(KB)-Allocated	135552	136384	135616	135850.67	462.99
Mstates(KB)-Used	76661	53729	84818	71736.00	16119.03
Tliveness(sec)	1137.003	1126.977	1129.761	1131.25	5.18
Mliveness-Allocated(KB)	136896	139200	137728	137941.33	1166.72
Mliveness-Used(KB)	94795	60102	98888	84595.00	21310.06
Ttotal(sec)	1143.131	1138.48	1146.492	**1142.70**	4.02
Mtotal-Allocated(KB)	136896	139200	137728	**137941.33**	1166.72
Mtotal-Used(KB)	94795	60102	98888	**84595.00**	21310.06

WorkerCnt=4	1	2	3	Avg	StDev
Tstates(sec)	1.67	1.52	1.695	1.63	0.09
Mstates(KB)-Allocated	151296	144192	140416	145301.33	5524.18
Mstates(KB)-Used	88379	76241	74230	79616.67	7654.73
Tliveness(sec)	954.632	955.858	975.517	962.00	11.72
Mliveness-Allocated(KB)	152384	147328	143936	147882.67	4251.23
Mliveness-Used(KB)	122086	108386	110733	113735.00	7326.77
Ttotal(sec)	969.554	968.655	984.433	**974.21**	8.86
Mtotal-Allocated(KB)	152384	147328	143936	**147882.67**	4251.23
Mtotal-Used(KB)	122086	108386	110733	**113735.00**	7326.77

WorkerCnt=1

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	2667.37	2643.707	2619.834	2643.64	23.77
Mtotal-Allocated(KB)	871744	770944	871808	838165.33	58215.39
Mtotal-Used(KB)	295119	254448	287287	278951.33	21578.81

WorkerCnt=2

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	1533.068	1527.921	1521.127	1527.37	5.99
Mtotal-Allocated(KB)	768000	755392	581376	701589.33	104298.49
Mtotal-Used(KB)	543928	253941	300085	365984.67	155821.02

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	1532.988	1528.371	1521.169	1527.51	5.96
Mtotal-Allocated(KB)	672128	681920	643776	665941.33	19810.28
Mtotal-Used(KB)	522340	519099	258277	433238.67	151529.91

WorkerCnt=3

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	1142.283	1137.824	1146.001	1142.04	4.09
Mtotal-Allocated(KB)	602560	707904	710080	673514.67	61458.18
Mtotal-Used(KB)	278449	466931	527943	424441.00	130061.00

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	1142.693	1137.431	1145.501	1141.88	4.10
Mtotal-Allocated(KB)	687936	620672	697088	668565.33	41728.51
Mtotal-Used(KB)	265228	256596	505547	342457.00	141306.01

Worker3	1	2	3	Avg	StDev
Ttotal(sec)	1143.1	1138.106	1145.643	1142.28	3.83
Mtotal-Allocated(KB)	746496	618432	676096	680341.33	64137.46
Mtotal-Used(KB)	556958	247534	297305	367265.67	166152.56

WorkerCnt=4

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	968.067	968.723	983.118	973.30	8.51
Mtotal-Allocated(KB)	648960	544704	673216	622293.33	68280.06
Mtotal-Used(KB)	247008	247569	326228	273601.67	45576.60

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	968.007	967.393	983.96	973.12	9.39
Mtotal-Allocated(KB)	670272	658560	575872	634901.33	51455.22
Mtotal-Used(KB)	564024	496544	243994	434854.00	168698.08

Worker3	1	2	3	Avg	StDev
Ttotal(sec)	966.263	968.281	982.738	972.43	8.99
Mtotal-Allocated(KB)	564864	652544	677312	631573.33	59084.39
Mtotal-Used(KB)	258889	502759	288299	349982.33	133123.14

Worker4	1	2	3	Avg	StDev
Ttotal(sec)	968.505	967.805	983.552	973.29	8.90
Mtotal-Allocated(KB)	662336	652352	662016	658901.33	5674.15
Mtotal-Used(KB)	271006	303651	282853	285836.67	16525.76

Παράλληλη εκτέλεση σε κατανεμημένο περιβάλλον (512 pems)

WorkerCnt=1	1	2	3	Avg	StDev
Tstates(sec)	1.881	1.986	1.982	1.95	0.06
Mstates(KB)-Allocated	132224	135360	132608	133397.33	1710.53
Mstates(KB)-Used	86111	77729	55062	72967.33	16062.85
Tliveness(sec)	2541.203	2534.344	2540.164	2538.57	3.70
Mliveness-Allocated(KB)	137920	142784	124416	135040.00	9516.65
Mliveness-Used(KB)	84334	72027	54593	70318.00	14943.97
Ttotal(sec)	2541.607	2534.758	2540.585	**2538.98**	3.69
Mtotal-Allocated(KB)	137920	142784	124416	**135040.00**	9516.65
Mtotal-Used(KB)	84334	72027	54593	**70318.00**	14943.97

WorkerCnt=2	1	2	3	Avg	StDev
Tstates(sec)	1.602	1.685	1.812	1.70	0.11
Mstates(KB)-Allocated	151552	149696	132416	144554.67	10553.27
Mstates(KB)-Used	89371	59543	61000	69971.33	16816.39
Tliveness(sec)	1447.865	1460.628	1447.89	1452.13	7.36
Mliveness-Allocated(KB)	154624	150272	132288	145728.00	11841.04
Mliveness-Used(KB)	105042	75133	90710	90295.00	14958.82
Ttotal(sec)	1460.247	1468.495	1460.036	**1462.93**	4.82
Mtotal-Allocated(KB)	154624	150272	132288	**145728.00**	11841.04
Mtotal-Used(KB)	105042	75133	90710	**90295.00**	14958.82

WorkerCnt=3	1	2	3	Avg	StDev
Tstates(sec)	1.614	1.402	1.575	1.53	0.11
Mstates(KB)-Allocated	143936	152704	154688	150442.67	5721.59
Mstates(KB)-Used	86655	79718	99983	88785.33	10299.09
Tliveness(sec)	1085.856	1084.252	1086.73	1085.61	1.26
Mliveness-Allocated(KB)	146112	154048	160832	153664.00	7367.51
Mliveness-Used(KB)	89933	117731	84855	97506.33	17698.15
Ttotal(sec)	1097.013	1101.226	1102.06	**1100.10**	2.71
Mtotal-Allocated(KB)	146112	154048	160832	**153664.00**	7367.51
Mtotal-Used(KB)	89933	117731	84855	**97506.33**	17698.15

WorkerCnt=4	1	2	3	Avg	StDev
Tstates(sec)	1.304	1.508	1.551	1.45	0.13
Mstates(KB)-Allocated	145792	136896	156992	146560.00	10069.99
Mstates(KB)-Used	77583	61996	71778	70452.33	7877.61
Tliveness(sec)	917.444	925.423	918.933	920.60	4.24
Mliveness-Allocated(KB)	148544	139456	160640	149546.67	10627.53
Mliveness-Used(KB)	105070	104308	94928	101435.33	5648.38
Ttotal(sec)	945.59	944.937	947.529	**946.02**	1.35
Mtotal-Allocated(KB)	148544	139456	160640	**149546.67**	10627.53
Mtotal-Used(KB)	105070	104308	94928	**101435.33**	5648.38

WorkerCnt=1

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	2541.623	2534.761	2540.657	2539.01	3.71
Mtotal-Allocated(KB)	583424	699968	641728	641706.67	58272.00
Mtotal-Used(KB)	265924	469224	246935	327361.00	123223.29

WorkerCnt=2

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	1460.359	1467.944	1459.483	1462.60	4.65
Mtotal-Allocated(KB)	662272	646720	680640	663210.67	16979.47
Mtotal-Used(KB)	266420	296795	518074	360429.67	137366.16

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	1459.916	1468.081	1459.657	1462.55	4.79
Mtotal-Allocated(KB)	676544	682432	670592	676522.67	5920.03
Mtotal-Used(KB)	311682	337810	306110	318534.00	16924.39

WorkerCnt=3

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	1096.14	1101.049	1101.515	1099.57	2.98
Mtotal-Allocated(KB)	613632	588032	648384	616682.67	30291.43
Mtotal-Used(KB)	332293	305813	295265	311123.67	19076.70

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	1096.207	1100.22	1101.133	1099.19	2.62
Mtotal-Allocated(KB)	643072	596352	448960	562794.67	101313.56
Mtotal-Used(KB)	288865	258812	269319	272332.00	15251.37

Worker3	1	2	3	Avg	StDev
Ttotal(sec)	1096.605	1100.623	1101.926	1099.72	2.77
Mtotal-Allocated(KB)	644672	547200	645440	612437.33	56498.49
Mtotal-Used(KB)	271312	247443	301444	273399.67	27060.96

WorkerCnt=4

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	944.398	944.238	946.673	945.10	1.36
Mtotal-Allocated(KB)	481472	556928	596480	544960.00	58430.60
Mtotal-Used(KB)	272063	307034	277695	285597.33	18777.06

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	944.858	943.189	946.126	944.72	1.47
Mtotal-Allocated(KB)	477376	548480	482432	502762.67	39673.00
Mtotal-Used(KB)	282814	326392	267194	292133.33	30679.61

Worker3	1	2	3	Avg	StDev
Ttotal(sec)	943.755	943.452	944.449	943.89	0.51
Mtotal-Allocated(KB)	575744	529152	460032	521642.67	58220.35
Mtotal-Used(KB)	302286	294689	291923	296299.33	5365.89

Worker4	1	2	3	Avg	StDev
Ttotal(sec)	945.178	943.94	946.404	945.17	1.23
Mtotal-Allocated(KB)	471296	436544	592832	500224.00	82061.60
Mtotal-Used(KB)	268391	291738	314430	291519.67	23020.28

Παράλληλη εκτέλεση σε κατανεμημένο περιβάλλον (1024 pems)

WorkerCnt=1	1	2	3	Avg	StDev
Tstates(sec)	1.992	1.967	2.021	1.99	0.03
Mstates(KB)-Allocated	136384	136256	136704	136448.00	230.76
Mstates(KB)-Used	77512	64271	77545	73109.33	7654.24
Tliveness(sec)	2409.152	2464.608	2434.716	2436.16	27.76
Mliveness-Allocated(KB)	143808	137920	144384	142037.33	3577.33
Mliveness-Used(KB)	71274	65718	70609	69200.33	3034.06
Ttotal(sec)	2409.584	2465.019	2435.138	**2436.58**	27.75
Mtotal-Allocated(KB)	143808	137920	144384	**142037.33**	3577.33
Mtotal-Used(KB)	71274	65718	70609	**69200.33**	3034.06

WorkerCnt=2	1	2	3	Avg	StDev
Tstates(sec)	1.739	1.745	1.861	1.78	0.07
Mstates(KB)-Allocated	151296	132544	142336	142058.67	9379.08
Mstates(KB)-Used	90901	84521	87752	87724.67	3190.09
Tliveness(sec)	1401.021	1406.728	1407.34	1405.03	3.49
Mliveness-Allocated(KB)	154240	135232	143616	144362.67	9525.97
Mliveness-Used(KB)	105231	81982	66474	84562.33	19506.92
Ttotal(sec)	1450.542	1413.297	1423.639	**1429.16**	19.23
Mtotal-Allocated(KB)	154240	135232	143616	**144362.67**	9525.97
Mtotal-Used(KB)	105231	81982	66474	**84562.33**	19506.92

WorkerCnt=3	1	2	3	Avg	StDev
Tstates(sec)	1.712	1.649	1.718	1.69	0.04
Mstates(KB)-Allocated	153664	135808	137984	142485.33	9741.95
Mstates(KB)-Used	102447	58981	54116	71848.00	26610.92
Tliveness(sec)	1039.088	1048.741	1038.73	1042.19	5.68
Mliveness-Allocated(KB)	156416	138112	141568	145365.33	9724.91
Mliveness-Used(KB)	85095	78954	74399	79482.67	5367.56
Ttotal(sec)	1073.655	1098.199	1095.547	**1089.13**	13.47
Mtotal-Allocated(KB)	156416	138112	141568	**145365.33**	9724.91
Mtotal-Used(KB)	85095	78954	74399	**79482.67**	5367.56

WorkerCnt=4	1	2	3	Avg	StDev
Tstates(sec)	1.93	2.089	1.62	1.88	0.24
Mstates(KB)-Allocated	162048	138752	156800	152533.33	12220.03
Mstates(KB)-Used	108439	82318	88811	93189.33	13599.78
Tliveness(sec)	901.615	901.522	889.212	897.45	7.13
Mliveness-Allocated(KB)	165760	144512	159424	156565.33	10908.64
Mliveness-Used(KB)	100474	98148	94377	97666.33	3076.91
Ttotal(sec)	972.032	967.591	964.192	**967.94**	3.93
Mtotal-Allocated(KB)	165760	144512	159424	**156565.33**	10908.64
Mtotal-Used(KB)	100474	98242	95295	**98003.67**	2597.71

WorkerCnt=1

Worker1	1	2	3	Avg	StDev
Ttotal(ms)	2409.494	2464.945	2435.056	2436.50	27.75
Mtotal-Allocated(KB)	628864	642496	640768	637376.00	7422.07
Mtotal-Used(KB)	302353	305508	330161	312674.00	15226.13

WorkerCnt=2

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	1450.363	1412.938	1422.987	1428.76	19.37
Mtotal-Allocated(KB)	513408	575488	539904	542933.33	31150.67
Mtotal-Used(KB)	282278	238314	329014	283202.00	45357.06

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	1450.021	1412.762	1423.072	1428.62	19.24
Mtotal-Allocated(KB)	571136	576192	508096	551808.00	37940.02
Mtotal-Used(KB)	253877	270891	261563	262110.33	8520.20

WorkerCnt=3

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	1072.614	1097.636	1094.927	1088.39	13.73
Mtotal-Allocated(KB)	533440	417728	478656	476608.00	57883.18
Mtotal-Used(KB)	263494	339143	308198	303611.67	38032.47

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	1072.607	1097.207	1094.507	1088.11	13.49
Mtotal-Allocated(KB)	476864	547968	497984	507605.33	36515.37
Mtotal-Used(KB)	285814	240683	347663	291386.67	53707.27

Worker3	1	2	3	Avg	StDev
Ttotal(sec)	1072.873	1097.432	1094.6	1088.30	13.44
Mtotal-Allocated(KB)	412928	492032	453696	452885.33	39558.23
Mtotal-Used(KB)	243956	298493	256032	266160.33	28644.52

WorkerCnt=4

Worker1	1	2	3	Avg	StDev
Ttotal(sec)	970.349	966.189	963.152	966.56	3.61
Mtotal-Allocated(KB)	511232	414720	531520	485824.00	62407.81
Mtotal-Used(KB)	300657	335609	283712	306659.33	26464.04

Worker2	1	2	3	Avg	StDev
Ttotal(sec)	970.697	964.085	962.793	965.86	4.24
Mtotal-Allocated(KB)	491136	555392	487104	511210.67	38315.23
Mtotal-Used(KB)	297141	319647	261929	292905.67	29091.16

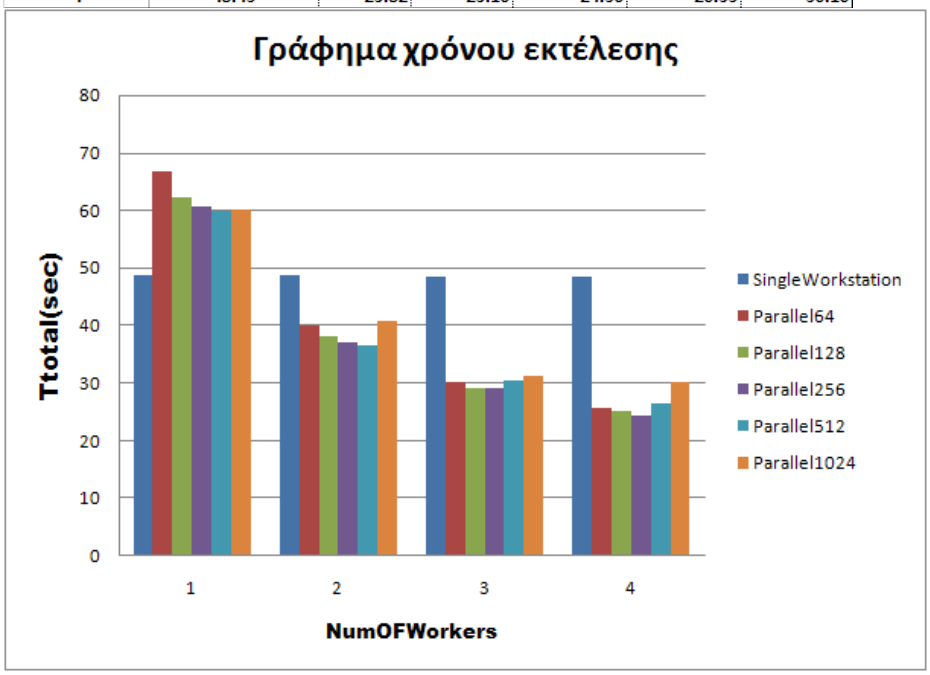
Worker3	1	2	3	Avg	StDev
Ttotal(sec)	970.687	966.279	962.715	966.56	3.99
Mtotal-Allocated(KB)	476160	509248	511488	498965.33	19781.73
Mtotal-Used(KB)	300006	316859	321746	312870.33	11405.66

Worker4	1	2	3	Avg	StDev
Ttotal(sec)	970.913	966.16	963.026	966.70	3.97
Mtotal-Allocated(KB)	432832	481280	458496	457536.00	24238.26
Mtotal-Used(KB)	303960	296637	261123	287240.00	22912.44

5.4 Συμπεράσματα

Για την προδιαγραφή με τις 5000 καταστάσεις έχουμε τα γραφήματα των εικόνων 1 και 2. Στην εικόνα 1 απεικονίζεται το γράφημα του συνολικού χρόνου εκτέλεσης για την περίπτωση εκτέλεσης του προγράμματος σε single workstation και την κατανεμημένη εκτέλεση του προγράμματος με διαφορετικές τιμές για το μέγεθος του BlockSizePem. Στο επάνω μέρος κάθε γραφήματος δίνεται και ένας πίνακας με τις σχετικές τιμές.

	Ttotal(sec)					
NumOFWorkers	SingleWorkstation	Parallel64	Parallel128	Parallel256	Parallel512	Parallel1024
1	48.87	66.76	62.36	60.55	59.80	60.12
2	48.68	39.88	38.06	37.06	36.53	40.81
3	48.43	30.18	29.02	29.21	30.43	31.38
4	48.43	25.82	25.10	24.30	26.53	30.16



Εικόνα 1

Παρατηρούμε ότι η απόδοση του single workstation είναι ίδια παρόλο που αυξάνεται ο αριθμός των workers και αυτό γιατί η εκτέλεση σε single workstation σε επίπεδο multiprocessing δεν εγγυάται ότι τα νήματα θα εκτελεστούν σε διαφορετικά physical cores του επεξεργαστή (κατά την έρευνα μας δεν βρήκαμε τρόπο στην Java να μπορείς να επέμβεις στο scheduling των νημάτων).

Επιπλέον βλέπουμε ότι με έναν worker η κατανεμημένη εκτέλεση έχει χειρότερο χρόνο εκτέλεσης από αυτόν της εκτέλεσης του single workstation γεγονός που θεωρείται αναμενόμενο λόγω του κόστους επικοινωνίας (overhead) που προκύπτει λόγω των λειτουργιών επικοινωνίας μεταξύ του server και του worker.

Ο καλύτερος χρόνος εκτέλεσης για αυτήν την προδιαγραφή παρατηρείται στην περίπτωση της κατανεμημένης εκτέλεσης με τέσσερις workers όπου το BlockSizePem τίθεται στα 256 Pems (Parallel256).

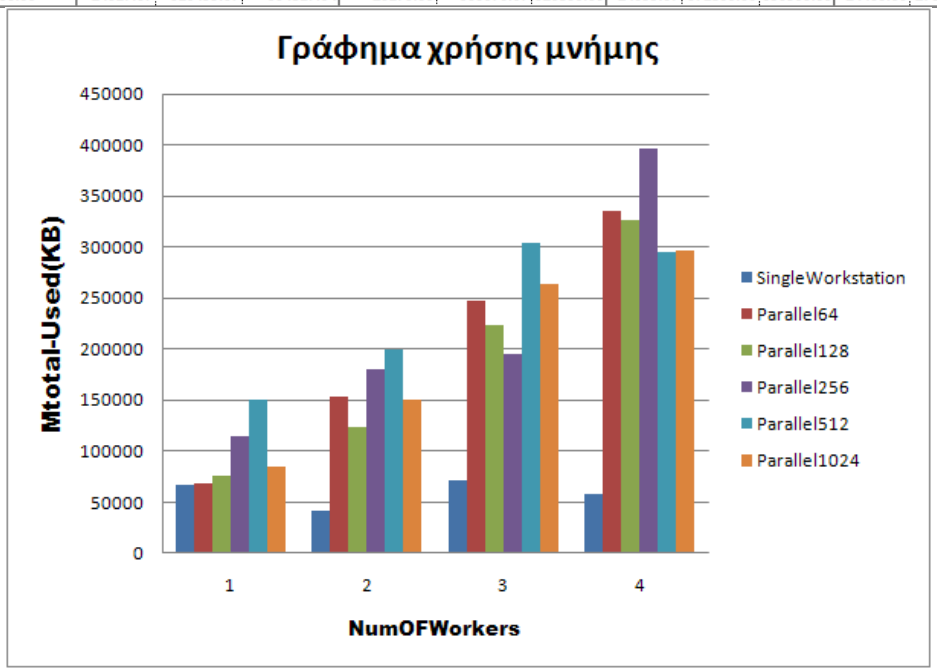
Η επιτάχυνση (speedup), δηλαδή ο λόγος της **καλύτερης** σε χρόνο εκτέλεσης του single workstation προς αυτήν της **καλύτερης** κατανεμημένης εκτέλεσης είναι:

$$Speedup_{3000} = \frac{T_{Single}}{T_{Parallel}} = \frac{48,43}{24,30} \cong .99$$

Βλέπουμε ότι ο παράλληλος αλγόριθμος liveness σε κατανεμημένα περιβάλλοντα είναι περίπου 2 φορές πιο γρήγορος από τον αλγόριθμο liveness του single workstation. Η επιτάχυνση του υπολογισμού δεν αυξάνεται γραμμικά ως προς των αριθμό των workers επειδή έχουμε overhead λόγω της επικοινωνίας.

Στην εικόνα 2 στην επόμενη σελίδα απεικονίζεται το γράφημα του συνολικού ωφέλιμου μεγέθους μνήμης για την περίπτωση εκτέλεσης του προγράμματος σε single workstation και την κατανεμημένη εκτέλεση του προγράμματος με διαφορετικές τιμές για το μέγεθος του BlockSizePem. Το συνολικό ωφέλιμο μέγεθος μνήμης (Total) στην περίπτωση της κατανεμημένης εκτέλεσης ισούται με το άθροισμα του συνολικού ωφέλιμου μεγέθους μνήμης στον server και στους workers.

	Mtotal-Used(KB)															
		Parallel64			Parallel128			Parallel256			Parallel512			Parallel1024		
NumOFWorkers	SingleWorkstation	Server	Workers	Total	Server	Workers	Total	Server	Workers	Total	Server	Workers	Total	Server	Workers	Total
1	66712.67	13943.00	54884.67	68827.67	13698.67	62633.00	76331.67	13543.00	101066.33	114609.33	13455.67	137084.00	150539.67	13474.67	71715.00	85189.67
2	42084.00	17256.67	135762.33	153019.00	16861.67	106808.33	123670.00	16877.00	163032.67	179909.67	16774.33	183009.00	199783.33	17082.00	132789.00	149871.00
3	71122.33	14278.67	232493.33	246772.00	18285.00	205412.00	223697.00	13904.00	181975.33	195879.33	13856.67	290632.00	304488.67	14040.00	249488.00	263528.00
4	57756.33	24317.67	310499.67	334817.34	23173.33	303676.67	326850.00	24099.67	371835.33	395935.00	24405.00	270850.00	295255.00	27775.67	269063.00	296838.67

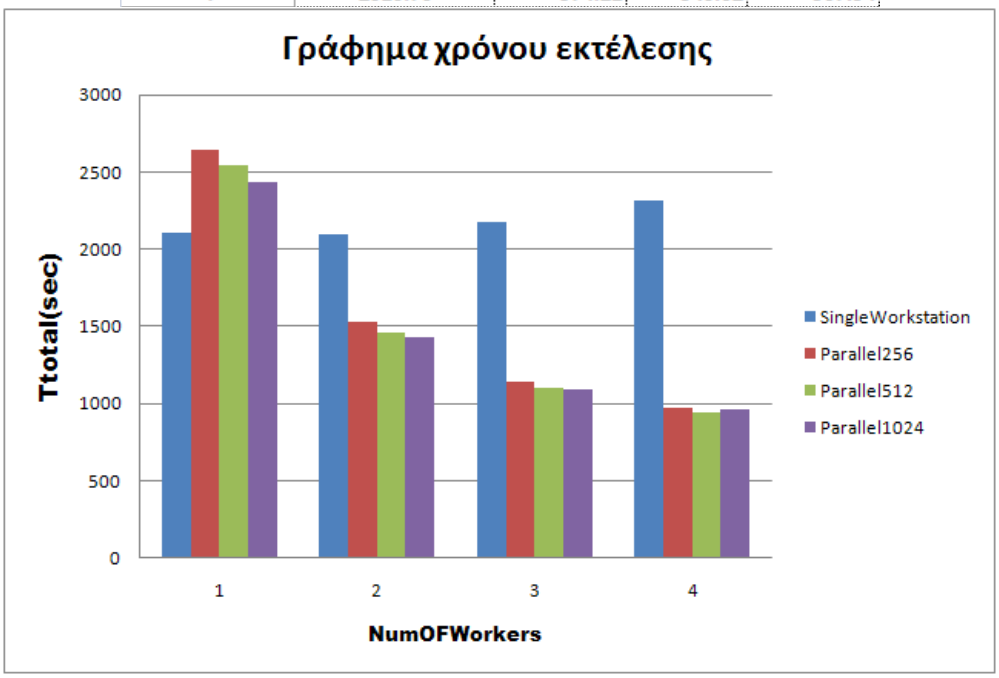


Εικόνα 2

Σε ότι αφορά την χρήση μνήμης, τα ποσά που καταναλώνονται στις εξεταζόμενες περιπτώσεις είναι αρκετά μεγάλα. Αυτό οφείλεται στο ότι επιλέξαμε να μην καλέσουμε ρητά τον garbage collector της Java, μιας και θα επιδρούσε άμεσα στον συνολικό χρόνο εκτέλεσης αφού αποτελεί μια διαδικασία που χρησιμοποιεί αρκετούς υπολογιστικούς πόρους στο να αποφασίσει ποια κομμάτια μνήμης πρέπει να ελευθερώσει.

Για την προδιαγραφή με τις 30000 καταστάσεις, ξεκινήσαμε τις μετρήσεις με BlockSizePem =256 Pems, γιατί είδαμε πειραματικά ότι είχε καλή απόδοση. Για την περίπτωση αυτήν έχουμε τα γραφήματα των εικόνων 3 και 4.

	Ttotal(sec)			
NumOFWorkers	SingleWorkstation	Parallel256	Parallel512	Parallel1024
1	2109.38	2643.54	2538.98	2436.58
2	2100.26	1527.95	1462.93	1429.16
3	2172.73	1142.70	1100.10	1089.13
4	2316.76	974.21	946.02	967.94

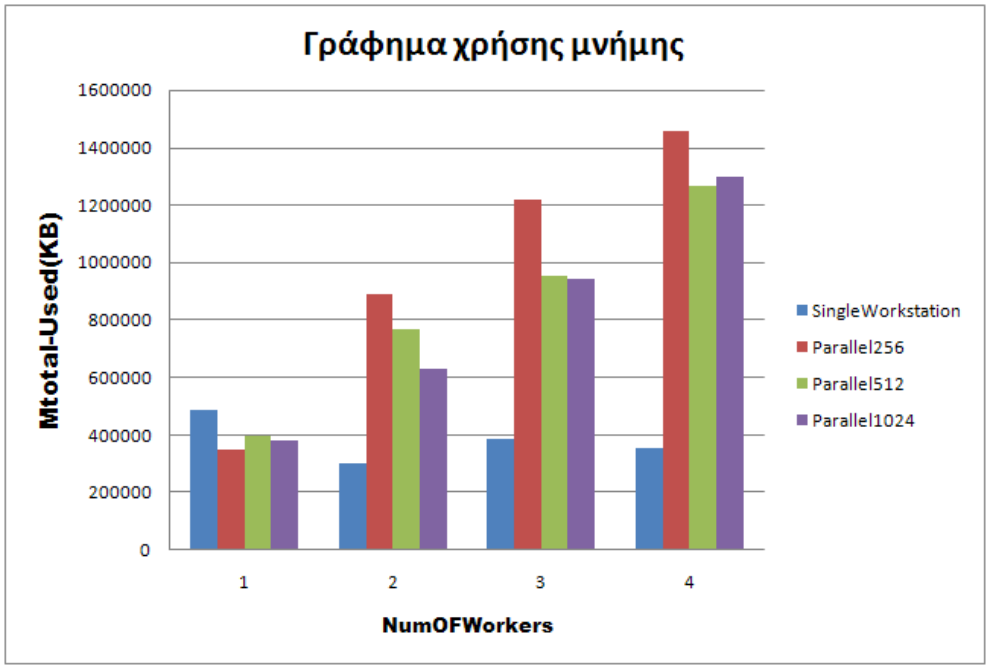


Εικόνα 3

Ο καλύτερος χρόνος εκτέλεσης για αυτήν την προδιαγραφή παρατηρείται στην περίπτωση της κατανεμημένης εκτέλεσης με τέσσερις workers όπου το BlockSizePem τίθεται ίσο με 512 Pems (Parallel512).

$$Speedup_{30000} = \frac{T_{Single}}{T_{Parallel}} = \frac{2100,26}{946,02} \cong 2,22$$

	Mtotal-Used(KB)									
		Parallel256			Parallel512			Parallel1024		
NumOFWorkers	SingleWorkstation	Server	Workers	Total	Server	Workers	Total	Server	Workers	Total
1	487942.00	68468.33	278951.33	347419.66	70318.00	327361.00	397679.00	69200.33	312674.00	381874.33
2	304121.00	88456.00	799223.33	887679.33	90295.00	678963.67	769258.67	84562.33	545312.33	629874.66
3	387133.33	84595.00	1134163.67	1218758.67	97506.33	856855.33	954361.66	79482.67	861158.67	940641.34
4	354034.00	113735.00	1344274.67	1458009.67	101435.33	1165549.67	1266985.00	98003.67	1199675.33	1297679.00



Εικόνα 4

Παράρτημα Α: Παραδείγματα Εκτέλεσης TLC

Α.1 Εκτέλεση TLC

Ακριβώς το πως εκτελείται το TLC εξαρτάται από το λειτουργικό σύστημα που χρησιμοποιείται και από το πώς το διαμορφώσατε. Πιθανόν να πληκτρολογήσετε μια εντολή της μορφής

program_name options spec_file

όπου

program_name είναι συγκεκριμένο για το σύστημα σας. Αυτό πιθανόν να είναι java tlc.TLC.

spec_file είναι το όνομα του αρχείου που περιέχει την TLA+ προδιαγραφή. Κάθε TLA+ module με όνομα M που εμφανίζεται στην προδιαγραφή πρέπει να είναι σε ένα ξεχωριστό αρχείο με όνομα M.tla. Η επέκταση .tla πιθανόν να παραλείπετε από το spec_file.

options είναι μια ακολουθία αποτελούμενη από μηδέν ή περισσότερες από τις ακόλουθες επιλογές:

-deadlock

Λέει στο TLC να μην ελέγξει για αδιέξοδο. Εάν αυτή η επιλογή δεν προσδιορίζεται, το TLC θα σταματήσει αν βρει ένα αδιέξοδο δηλαδή, μια προσβάσιμη κατάσταση χωρίς successor κατάσταση.

-simulate

Λέει στο TLC να τρέξει σε simulation τρόπο λειτουργίας, παράγοντας τυχαία επιλεκτικές συμπεριφορές, αντί να παράγει όλες τις προσβάσιμες καταστάσεις.

-depth *num*

Αυτή η επιλογή γίνεται η αιτία το TLC να παράγει συμπεριφορές μήκους με ανώτατο όριο το num σε simulation τρόπο λειτουργίας. Χωρίς αυτήν την επιλογή, το TLC θα παράγει τρεξίματα μήκους με ανώτατο όριο το 100. Αυτή η επιλογή είναι σημαντική μόνο όταν η επιλογή simulate χρησιμοποιείται.

-seed *num*

Σε simulation τρόπο λειτουργίας, οι συμπεριφορές που παράγονται από το TLC προσδιορίζονται από το αρχικό seed που δίνεται σε μια γεννήτρια ψευδοτυχαίων αριθμών. Κανονικά, το seed παράγεται τυχαία. Αυτή η επιλογή γίνεται η αιτία το TLC να κάνει το seed να είναι ίσο με num, ο οποίος πρέπει να είναι ένας ακέραιος από -2^{63} έως 2^{63} -1. Εκτελώντας το TLC δυο φορές σε simulation τρόπο λειτουργίας με το ίδιο seed και aril (δείτε την επιλογή aril παρακάτω) θα παράγει εντελώς ίδια αποτελέσματα. Αυτή η επιλογή είναι σημαντική μονό όταν χρησιμοποιείται η επιλογή simulate.

-aril *num*

Αυτή η επιλογή κάνει το TLC να χρησιμοποιήσει το num σαν το aril σε simulation τρόπο λειτουργίας. Το aril είναι ένας modifier του αρχικού seed. Όταν το TLC βρίσκει ένα σφάλμα στον simulation τρόπο λειτουργίας, τυπώνει και το αρχικό seed και ένα aril αριθμό. Η χρήση αυτού του αρχικού seed και του aril θα γίνει η αιτία το πρώτο ίχνος που παράχθηκε να είναι το ίχνος σφάλματος. Η προσθήκη Print εκφράσεων δεν θα αλλάζει συνήθως την σειρά με την οποία το TLC παράγει τα ίχνη. Έτσι, αν το ίχνος δεν σας λέει τι πήγε λάθος, μπορείτε να

τρέξετε το TLC ξανά σε μόνο ακριβώς αυτό το ίχνος για να τυπώσετε επιπρόσθετες πληροφορίες.

-coverage *num*

Αυτή η επιλογή κάνει το TLC να τυπώνει "coverage" πληροφορίες κάθε num λεπτά καθώς και στο τέλος της εκτέλεσης του. Για κάθε λειτουργία σύζευξης που "αναθέτει μια τιμή" σε μια μεταβλητή, το TLC τυπώνει τον αριθμό των φορών που η σύζευξη έχει πραγματικά χρησιμοποιηθεί στην κατασκευή μιας νέας κατάστασης. Οι τιμές που αυτό τυπώνει πιθανόν να μην είναι ακριβής, αλλά τα μεγέθη τους μπορούν να παρέχουν χρήσιμες πληροφορίες. Συγκεκριμένα, μια τιμή 0 δείχνει ότι μέρος της επόμενης κατάστασης ποτέ δεν "εκτελέστηκε". Αυτό πιθανόν να δείχνει ένα σφάλμα στην προδιαγραφή, ή πιθανόν να σημαίνει ότι το model που το TLC ελέγχει είναι τόσο μικρό στο να ασκήσει αυτό το μέρος της λειτουργίας.

-recover *run_id*

Αυτή η επιλογή κάνει το TLC να ξεκινήσει την εκτέλεση της προδιαγραφής όχι από την αρχή, αλλά από όπου το άφησε το τελευταίο σημείο ελέγχου (checkpoint). Όταν το TLC λαμβάνει ένα σημείο ελέγχου, τυπώνει το αναγνωριστικό εκτέλεσης. (Αυτό το αναγνωριστικό είναι το ίδιο καθ' όλη την διάρκεια μιας εκτέλεσης του TLC.) Η τιμή του run_id πρέπει να είναι αυτό το αναγνωριστικό εκτέλεσης.

-cleanup

Το TLC δημιουργεί έναν αριθμό από αρχεία όταν αυτό εκτελείται. Όταν η εκτέλεση του ολοκληρώνεται, αυτό τα σβήνει όλα αυτά. Αν το TLC βρει ένα σφάλμα, ή αν το σταματήσετε πριν να τερματίσει, το TLC μπορεί να αφήσει κάποια μεγάλα αρχεία. Η cleanup ιδιότητα κάνει το TLC να διαγράφει όλα τα αρχεία που δημιουργήθηκαν από προηγούμενες εκτελέσεις. Μην χρησιμοποιείται αυτήν την επιλογή αν κάποια στιγμή τρέχετε άλλο ένα αντίγραφο του TLC στον ίδιο τον φάκελο, αν το κάνετε, αυτό θα κάνει το άλλο το αντίγραφο να αποτύχει.

-difftrace *num*

Όταν το TLC βρει ένα σφάλμα, τυπώνει ένα ίχνος σφάλματος. Κανονικά, αυτό το ίχνος τυπώνεται ως μια ακολουθία από πλήρεις καταστάσεις, όπου μια κατάσταση καταγράφει τις τιμές όλων των δηλωμένων μεταβλητών. Η difftrace επιλογή κάνει το TLC να τυπώνει μια σύντομη έκδοση κάθε κατάστασης, καταγράφοντας μόνο τις μεταβλητές των οποίων οι τιμές είναι διαφορετικές από την προηγούμενη κατάσταση. Αυτό το κάνει ευκολότερο για να δείτε τι συνέβη σε κάθε βήμα, αλλά δυσκολότερο να βρείτε την πλήρη κατάσταση.

-terse

Κανονικά, το TLC επεκτείνει πλήρως τις τιμές που εμφανίζονται στα μηνύματα σφάλματος ή στην έξοδο από την εκτίμηση των Print εκφράσεων. Η terse επιλογή κάνει το TLC σε αντικατάσταση να τυπώνει μερικώς εκτιμώμενες, συντομότερες εκδόσεις αυτών των τιμών.

-workers *num*

Τα βήματα του αλγορίθμου εκτέλεσης του TLC μπορούν να επιταχυνθούν σε ένα υπολογιστή με πολυεπεξεργαστή με την χρήση πολλαπλών νημάτων. Η επιλογή κάνει το TLC να χρησιμοποιεί num σε πλήθος νήματα όταν βρίσκει τις προσβάσιμες καταστάσεις. Δεν υπάρχει λόγος να χρησιμοποιήσετε περισσότερα νήματα από τους πραγματικούς επεξεργαστές που υπάρχουν στο σύστημα σας. Αν η επιλογή παραληφθεί, το TLC χρησιμοποιεί ένα μόνο νήμα.

-config *config_file*

Προσδιορίζει ότι το configuration αρχείο ονομάζεται *config_file*, το οποίο πρέπει να είναι ένα αρχείο με επέκταση .cfg. Η επέκταση .cfg μπορεί να παραληφθεί από το *config_file*. Αν αυτή η επιλογή παραληφθεί υποθέτετε ότι έχει το ίδιο όνομα με το *spec_file*, εκτός της επέκτασης .cfg.

-nowarning

Υπάρχουν TLA+ εκφράσεις που είναι νόμιμες αλλά είναι αρκετά αμφίβολες και που η παρουσία τους πιθανόν να υποδεικνύει ένα σφάλμα. Για παράδειγμα, η έκφραση [f EXCEPT ![υ] = e] είναι πιθανά εσφαλμένη αν το υ δεν είναι ένα στοιχείο του πεδίου ορισμού της f . (Σε αυτήν την περίπτωση, η έκφραση απλώς ισούται με την f .) Το TLC κανονικά εξάγει μια προειδοποίηση όταν απαριθμεί μια τέτοια αμφίβολη έκφραση, αυτή η επιλογή αποκρύπτει αυτές τις προειδοποιήσεις.

-metadir *dir*

Προσδιορίζει τον κατάλογο *dir* στον οποίο το TLC αποθηκεύει τα αρχεία του, όπως αυτά που κρατούν την ουρά με τις προσβάσιμες καταστάσεις που δεν έχουν εξεταστεί. Ο εξ ορισμού κατάλογος είναι ./states, όπου "." είναι ο τρέχον κατάλογος στον οποίο το TLC τρέχει.

-dump *state_ file*

Κάνει το TLC να γράψει όλες τις προσβάσιμες καταστάσεις που αυτό βρήκε στο αρχείο *state_ file*.

-continue

Κάνει το TLC να συνεχίσει την εκτέλεση αφού διαπιστώσει ένα σφάλμα, τυπώνοντας ένα νέο μήνυμα για κάθε σφάλμα που αυτό βρίσκει.

-fp *num*

Κάνει το TLC να επιλέξει seed αριθμό τον *num* για τον fingerprint αλγόριθμο του, όπου num είναι ένας ακέραιος από 0 (ο εξ ορισμού) έως 130. Το TLC μπορεί να αποτύχει να ελέγξει όλες τις προσβάσιμες καταστάσεις, και έτσι να αποτύχει να βρει ένα σφάλμα, λόγω μιας σύγκρουσης fingerprint—δηλαδή, επειδή δυο διαφορετικές καταστάσεις έχουν το ίδιο fingerprint. Αν τρέχετε το TLC στην ίδια προδιαγραφή χρησιμοποιώντας δύο διαφορετικά seeds, και οι δυο εκτελέσεις δεν αναφέρουν κανένα σφάλμα και βρίσκουν τον ίδιο αριθμό καταστάσεων, στην περίπτωση αυτήν είναι πάρα πολύ απίθανο αν συμβεί μια σύγκρουση.

Α.2 Αποτελέσματα τροποποιημένων παραδειγμάτων

Παρατήρηση αποτελεσμάτων τροποποιημένων εκδόσεων των αρχείων HourClock.tla και LiveHourClock.tla.

```
---------------------- MODULE HourClock ----------------------
EXTENDS Naturals
VARIABLE hr
HCini  ==  hr \in (1 .. 3)
HCnxt  ==  hr' = IF hr # 3 THEN hr + 1 ELSE 1
HC  ==  HCini /\ [][HCnxt]_hr
--------------------------------------------------------------
THEOREM  HC => []HCini
```

```
 (*************************************************************)
(* This is a TLC configuration file for testing that HCini is an   *)
(*invariant of the specification HC                               *)
(*************************************************************)

SPECIFICATION HC
   \* This statement tells TLC that HC is the specification that it
   \* should check.

INVARIANT HCini
   \* This statement tells TLC to check that formula HCini is an
   \* invariant of the specification--in other words, that the
   \* specification implies []HCini.
```

HourClock.cfg

```
--------------------------- MODULE LiveHourClock --------------------
(*************************************************************)
(* This module adds the liveness condition to the hour clock       *)
(* specification of module HourClock.                              *)
(*                                                                 *)
(*************************************************************)

EXTENDS HourClock

LSpec == HC /\ WF_hr(HCnxt)

(*************************************************************)
(* The specification with the liveness condition conjoined.      *)
(*************************************************************)

(*************************************************************)
(* We now define some properties that LSpec satisfies.             *)
(*************************************************************)
AlwaysTick == []<><<HCnxt>>_hr

(*************************************************************)
(* Asserts that infinitely many <<HCnxt>>_hr steps occur.        *)
(*************************************************************)

AllTimes == \A n \in 1..3 : []<>(hr = 2)

(*************************************************************)
(* Asserts that, for each time n in 1..3, hr infinitely often   *)
(*equals 2.                                                      *)
(*************************************************************)

TypeInvariance == []HCini

(*************************************************************)
(* The temporal formula asserting that HCini is always true.  It *)
```

```
(* is stated in this way to show you another way of telling TLC  *)
(* to check an  invariant.                                       *)
(***************************************************************)

-----------------------------------------------------------------
THEOREM  LSpec => AlwaysTick /\ AllTimes /\ TypeInvariance
=================================================================
```

```
(***************************************************************)
(* This is a TLC configuration file for testing that the       *)
(* specification LSpec implies the properties AlwaysTick,      *)
(* AllTimes,and TypeInvariance.                                *)
(***************************************************************)

SPECIFICATION LSpec
   \* This statement tells TLC that LSpec is the specification that
   \* it should check.

PROPERTIES AlwaysTick AllTimes TypeInvariance
   \* This statement tells TLC to check that the specification
   \* implies each of the three properties AlwaysTick, AllTimes,
   \* and TypeInvariance (and hence that it implies their
   \* conjunction).
\* The keywords PROPERTY and PROPERTIES can be used
\* interchangeably.
```

LiveHourClock.cfg

Κανονική έξοδος

```
TLC Version 2.01 of April 9, 2008
Model-checking
Parsing file LiveHourclock.tla
Parsing file HourClock.tla
Parsing file C:\Users\ggeorgak\Desktop\ntina\workspace\tla\tlasany\StandardModules\Naturals.tla
Semantic processing of module Naturals
Semantic processing of module HourClock
Semantic processing of module LiveHourclock
Implied-temporal checking--satisfiability problem has 1 branches.
Finished computing initial states: 3 distinct states generated.
--Checking temporal properties for the complete state space...
Model checking completed. No error has been found.
  Estimates of the probability that TLC did not check all reachable states
  because two distinct states had the same fingerprint:
    calculated (optimistic):  4.87890977618477E-19
    based on the actual fingerprints:  4.514021842285041E-19
6 states generated, 3 distinct states found, 0 states left on queue.
The depth of the complete state graph search is 1.
```

Η module **HourClock** περιλαμβάνει δηλώσεις και ορισμούς:

- **hr,** μια μεταβλητή κατάστασης
- **HCini,** ένα κατηγόρημα κατάστασης
- **HCnxt,** μια ενέργεια (αποτελείται από την hr και την hr')
- **HC,** μια χρονική έκφραση που προσδιορίζει ότι
  - η αρχική κατάσταση ικανοποιεί το HCini

- κάθε μετάβαση ικανοποιεί την HCnxt ή αφήνει την hr αμετάβλητη

- Η module HourClock επίσης ισχυρίζεται ένα θεώρημα**: THEOREM HC => []Hcini**

  Η έκφραση HCini ισχυρίζεται ότι η hr είναι ένας ακέραιος με τιμή από το 1 έως το 3, και η $\Box HCini$ ισχυρίζεται ότι η HCini είναι πάντοτε αληθής. Έτσι, η $\Box HCini$ πρέπει να είναι αληθής για οποιαδήποτε συμπεριφορά ικανοποιεί την HC. Άλλος τρόπος να το πούμε είναι ότι η HC υπονοεί την $\Box HCini$, για οποιαδήποτε συμπεριφορά. Έτσι, η έκφραση $HC \Rightarrow \Box HCini$ πρέπει να ικανοποιείτε από κάθε συμπεριφορά. Μια χρονική έκφραση που ικανοποιείται από κάθε συμπεριφορά αποκαλείται θεώρημα, επομένως η $HC \Rightarrow \Box HCini$ πρέπει να είναι ένα θεώρημα. Οι επιστήμονες της επιστήμης της λογικής αποκαλούν μια έκφραση έγκυρη αν αυτή ικανοποιείται από κάθε συμπεριφορά, αυτοί διατηρούν τον όρο θεώρημα για έγκυρες εκφράσεις που μπορούν να αποδειχτούν.

  Αυτή η εντολή ισχυρίζεται ότι η έκφραση $HC \Rightarrow \Box HCini$ είναι αληθής στο πλαίσιο της εντολής. Πιο συγκεκριμένα, ισχυρίζεται ότι η έκφραση έπεται λογικά από τους ορισμούς στην module, τους ορισμούς στην Naturals module, και τους κανόνες της TLA+. Αν η έκφραση δεν ήταν αληθής, τότε η module θα ήταν εσφαλμένη.

Το HourClock configuration αρχείο περιλαμβάνει τις ακόλουθες δηλώσεις:

- SPECIFICATION HC
  Λέει στο TLC να λάβει την HC σαν την προδιαγραφή.

  Η έκφραση SPECIFICATION εντολής πρέπει να περιέχει ακριβώς έναν συζευκτέο που είναι μια box-action έκφραση. Αυτός ο συζευκτέος προσδιορίζει την ενέργεια επόμενης κατάστασης.

- INVARIANT Hcini
  Η INVARIANT εντολή πρέπει να προσδιορίζει ένα κατηγόρημα κατάστασης.

  Για να ελέγξει το invariance με μια PROPERTY εντολή, η προσδιορισμένη ιδιότητα πρέπει να είναι της μορφής $\Box P$. Ο προσδιορισμός ενός κατηγορήματος κατάστασης P σε μια PROPERTY εντολή λέει στο TLC να ελέγξει ότι η προδιαγραφή συνεπάγεται το P, που σημαίνει ότι το P είναι αληθές στην initial κατάσταση κάθε συμπεριφοράς που ικανοποιεί την προδιαγραφή. Μια PROPERTY εντολή μπορεί να προσδιορίσει οποιεσδήποτε εκφράσεις τις οποίες το TLC μπορεί να εκτιμήσει.

Σημείωση:

- Το ρολόι πιθανόν τελικά να σταματήσει να χτυπά (ticking)
- Αυτό δεν πρέπει να αποτύχει σε άλλη περίπτωση

Μια έκφραση **Init /\ [][Next]_v**

Προσδιορίζει τις αρχικές καταστάσεις και τις επιτρεπτές μεταβάσεις ενός συστήματος.

Αυτή επιτρέπει τις μεταβάσεις που δεν αλλάζουν το v: stuttering μεταβάσεις.

- *Άπειρο stuttering μπορεί να αποκλειστεί με τον ισχυρισμό συνθηκών fairness.*

Για παράδειγμα η HC == HCini/\ [][HCnxt]_hr /\ WF_hr(HCnxt) προσδιορίζει ένα hour clock που ποτέ δεν σταματά να χτυπά. (Module LiveHourClock)

Η module **LiveHourClock**, που προσθέτει την Liveness συνθήκη στην προδιαγραφή του hour clock στην module HourCLock, περιλαμβάνει τα ακόλουθα:

- **LSpec,** η προδιαγραφή με την Liveness συνθήκη
- Κάποιες ιδιότητες που η Lspec πρέπει να ικανοποιεί
  - **AlwaysTick**, που ισχυρίζεται ότι άπειρα <<HCnxt>>_hr βήματα λαμβάνουν χώρα.
  - **AllTimes**, ισχυρίζεται ότι, για κάθε n από 1..3, το hr απείρως συχνά ισοδυναμεί με το 2.
  - **TypeInvariance,** μια χρονική έκφραση που προσδιορίζει ότι
    - το Hcini είναι πάντα αληθές
- Η module LiveHourClock επίσης ισχυρίζεται ένα θεώρημα**:**

  LSpec => AlwaysTick /\ AllTimes /\ TypeInvariance

1. Χωρίς το Liveness property (WF_hr(HCnxt)) στο αρχείο LiveHourClock.tla:

Έχουμε άπειρο stuttering.

Έξοδος

```
TLC Version 2.01 of April 9, 2008
Model-checking
Parsing file LiveHourclock.tla
Parsing file HourClock.tla
Parsing file C:\Users\ggeorgak\Desktop\ntina\workspace\tla\tlasany\StandardModules\Natura
Semantic processing of module Naturals
Semantic processing of module HourClock
Semantic processing of module LiveHourclock
Implied-temporal checking--satisfiability problem has 1 branches.
Finished computing initial states: 3 distinct states generated.
--Checking temporal properties for the complete state space...
Error: Temporal properties were violated.
The following behaviour constitutes a counter-example:

STATE 1: <Initial predicate>
hr = 1

STATE 2: Stuttering
6 states generated, 3 distinct states found, 0 states left on queue.
```

2. Χωρίς το stuttering στο HourClock.tla (το _hr στο HC)

Παράγεται σημασιολογικό σφάλμα και αποτυγχάνει η σημασιολογική ανάλυση.

Έξοδος

```
TLC Version 2.01 of April 9, 2008
Model-checking
Parsing file LiveHourclock.tla
Parsing file HourClock.tla
Parsing file C:\Users\ggeorgak\Desktop\ntina\workspace\tla\tlasany\StandardModules\Naturals.tla
Semantic processing of module Naturals
Semantic processing of module HourClock
Semantic errors:

*** Errors: 1

line 6, col 18 to line 6, col 19 of module HourClock

Unsupported expression type N_GenPrefixOp



Semantic processing of module LiveHourclock
Semantic errors:

*** Errors: 1

line 6, col 18 to line 6, col 19 of module HourClock

Unsupported expression type N_GenPrefixOp



Error: Parsing or semantic analysis failed.
```

3. Ταυτόχρονη αφαίρεση των δύο παραπάνω

Έξοδος

Η έξοδος είναι ίδια με την προηγούμενη περίπτωση.

Α.3 Αναλυτική περιγραφή εκτέλεσης

Στην συγκεκριμένη ενότητα του παραρτήματος γίνεται μια αναλυτική περιγραφή εκτέλεσης των παραδειγμάτων (HourClock.tla και LiveHourCLock.tla) που παρουσιάστηκαν στο Ζήτημα 1ο.

Κατά την αρχικοποίηση του model checker αρχικοποιούνται δύο βασικές δομές. Το TLC χρησιμοποιεί ένα σύνολο G από fingerprints καταστάσεων και μια ουρά U από καταστάσεις.

Συγκεκριμένα:

- Το σύνολο G τύπου **MultiFPSet.Java** (μεταβλητή ***theFPSet*)** είναι μέρος του γράφου reachability κατάστασης που το TLC έχει υπολογίσει μέχρι την τρέχουσα χρονική στιγμή. Στην ουσία περιλαμβάνει τα fingerprints των καταστάσεων. Τα fingerprints του TLC είναι στοχαστικά μοναδικά αθροίσματα ελέγχου(checksum). Ένα MultiFPSet είναι ένα σύνολο από 64-bit fingerprints.

Το DiskFPSet είναι ένας υποτύπος του FPSet που χρησιμοποιεί ένα περιορισμένο κομμάτι μνήμης. Οποιαδήποτε fingerprints που δεν ταιριάζουν στην μνήμη γράφονται στα backing αρχεία δίσκου. Όπως απαιτείται από την FPSet κλάση, οι μέθοδοι αυτής της κλάσης είναι thread-safe.

Αυτή η υλοποίηση χρησιμοποιεί ένα απλό ταξινομημένο αρχείο δίσκου στο οποίο δυαδική αναζήτηση παρεμβολής (interpolated binary search) εκτελείται. Κρατά ένα ξεχωριστό BufferedRandomAccessFile (Αυτή η κλάση επεκτείνει ένα RandomAccessFile για να χρησιμοποιήσει έναν private buffer έτσι ώστε οι περισσότερες λειτουργίες να μην απαιτούν πρόσβαση στον δίσκο.) αντικείμενο ανοιχτό στο αρχείο δίσκου για κάθε νήμα worker.

Η υλοποίηση χρησιμοποιεί έναν έξυπνο συγχρονισμό (χρησιμοποιώντας την ReadersWriterLock κλάση) έτσι αναζητήσεις στον δίσκο μπορούν να εκτελεστούν παράλληλα.

Χρησιμοποιείται το MSB ενός fingerprint για να δείξει αν αυτό έχει μεταφερθεί (flushed) στον δίσκο. Κάνοντας αυτό, χάνεται ένα bit από το fingerprint. Ωστόσο, λαμβάνεται αυτό το bit πίσω αν χρησιμοποιείται MultiFPSet, όπως και συμβαίνει για το σύνολο G.

- Η ουρά U (ουρά FIFO) τύπου **DiskStateQueue.java** (μεταβλητή ***theStateQueue*)** είναι μια ακολουθία από καταστάσεις των οποίων οι successors δεν έχουν υπολογιστεί ακόμη. Η **DiskStateQueue** είναι μια κλάση που κάνει extends την StateQueue για να παρέχει μια υλοποίηση μιας State Queue όπου όταν το in memory μέρος της State Queue είναι γεμάτο , κάνει flush αυτό στον δίσκο.

  Η ουρά U υλοποιείται σαν ένα αρχείο δίσκου του οποίου οι πρώτες και οι τελευταίες αρκετές χιλιάδες καταχωρήσεις κρατούνται στην μνήμη. Αυτή είναι μια FIFO ουρά που χρησιμοποιεί ένα background νήμα για να κάνει prefetch καταχωρήσεις και άλλο για να γράφει καταχωρήσεις στον δίσκο. Αφιερώνοντας ένα κλάσμα του επεξεργαστή για αυτά τα background νήματα γενικά εξασφαλίζεται ότι ένα νήμα worker ποτέ δεν περιμένει για disk I/O όταν γίνεται πρόσβαση στην ουρά.

Εκτέλεση του HourClock.tla (Safety Model Checking)

Αφού γίνει ο σημασιολογική ανάλυση των τριών module (Naturals, HourClock) που περιέχει το παράδειγμα μας τυπώνεται το ακόλουθο μήνυμα και ξεκινά το model checking.

```
Semantic processing of module Naturals
Semantic processing of module HourClock
```

Property	Value
Init (initPredVec)	hr \in (1 .. 3)
Next (nextPred)	hr' = IF hr # 3 THEN hr + 1 ELSE 1
Temporal	[]
Invariant	hr \in (1 .. 3)
ImpliedInit	[]
ImpliedAction	[]
ImpliedTemporal	[]

Constraint (ModelConstraints)	[]
ActionConstraint	null

Το πλήθος των ImpliedTemporal είναι μηδενικό και επομένως δεν θα έχουμε Liveness model checking αλλά απλό safety model checking.

Ο αλγόριθμος ξεκινά με κενά το σύνολο G και την ουρά U:

Αφού πρώτα ελέγξει ότι κάθε assume στην προδιαγραφή ικανοποιείται από τις τιμές που ανατέθηκαν στις σταθερές παραμέτρους (βήμα το οποίο δεν πραγματοποιείται μιας και δεν έχουμε κανένα assumption) υπολογίζει τις αρχικές καταστάσεις εκτιμώντας το αρχικό κατηγόρημα Init.

Στην συνέχεια το TLC παράγει και ελέγχει όλες τις πιθανές καταστάσεις που ικανοποιούν το init κατηγόρημα και θέτει το G και την U να περιλαμβάνουν όλες αυτές τις καταστάσεις.

Στην περίπτωση μας έχουμε τρεις αρχικές καταστάσεις (η s1:hr=1, η s2:hr=2 και η s3:hr= 3).

Για κάθε μια από τις αρχικές καταστάσεις:

1. Αν το κατηγόρημα Constraint (στην περίπτωση μας δεν έχουμε κάποιο constraint) είναι αληθές στην κατάσταση s1, προσθέτει τον κόμβο s1 (το fingerprint) στο σύνολο G (συγκεκριμένα στο in-memory hash table). Εφόσον η κατάσταση s1 δεν υπήρχε στο σύνολο

   - τοποθετείται και στην ουρά U (συγκεκριμένα στον enqBuf της).
   - Και εκτιμά τα κατηγορήματα Invariant και ImpliedInit (στην περίπτωση μας δεν έχουμε κάποιο ImpliedInit) στην κατάσταση s1.

Το ίδιο επαναλαμβάνεται και για τις άλλες δύο καταστάσεις s2 και s3. Αφού ολοκληρωθεί η διαδικασία και για τις τρεις αρχικές καταστάσεις τυπώνεται κατάλληλο μήνυμα που αναφέρει το πλήθος των αρχικών καταστάσεων που έχουν παραχθεί.

```
Finished computing initial states: 3 distinct states generated.
```

Στην συνέχεια λαμβάνεται κάθε κατάσταση από την ουρά, και παράγονται όλες οι δυνατές επόμενες καταστάσεις της κατάστασης, ελέγχονται τα invariants, και ενημερώνεται το σύνολο των καταστάσεων και η ουρά καταστάσεων.

2. Όσο η U δεν είναι κενή, κάνει τα ακόλουθα:

   a) Αφαιρεί την πρώτη κατάσταση από την U, δηλαδή την s1, εν προκειμένου η s1 να είναι η τρέχουσα κατάσταση.

   b) Βρίσκει το σύνολο T όλων των successor καταστάσεων της s1 εκτιμώντας την ενέργεια επόμενης κατάστασης ξεκινώντας από την s1.

   c) Στην περίπτωση της s1 έχουμε ένα στοιχείο στο T, το s2. Αυξάνεται ανάλογα με το πλήθος των επόμενων καταστάσεων το συνολικό πλήθος των παραγόμενων καταστάσεων.

d) Για κάθε κατάσταση t στο T, κάνει τα ακόλουθα:

   i. Ελέγχεται αν η κατάσταση t ικανοποιεί τα κατηγορήματα Constraint και ActionConstraint (που στην περίπτωση μας είναι κενά). Ο έλεγχος επιστρέφει θετικό αποτέλεσμα (true)

      A. Στην συνέχεια προσπαθεί να προσθέσει την t μέσα στο σύνολο G. Στην περίπτωση μας υπάρχει ήδη η κατάσταση αυτή στο σύνολο G (στο in-memory hash table) οπότε δεν την προσθέτει.

   ii. Τέλος ελέγχεται αν η κατάσταση t ικανοποιεί το ImpliedAction. (στην περίπτωση μας είναι κενό).

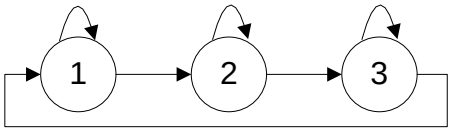
Η παραπάνω διαδικασία του βήματος 2 επαναλαμβάνεται και για τις άλλες δύο καταστάσεις s2 και s3. Μόνο που για την s2 έχουμε την επόμενη κατάσταση s3 και για την s3 την s1.

Επομένως συνολικά παράγονται 6 καταστάσεις.(τρεις αρχικές συν τις τρεις που προέκυψαν κατά τον υπολογισμό των succesor καταστάσεων.

```
6 states generated, 3 distinct states found, 0 states left on queue.
```

Εκτέλεση του LiveHourClock.tla (Liveness Model Checking)

Γράφημα μετάβασης που αντιστοιχεί στο συγκεκριμένο παράδειγμα.



Περιορισμοί του Liveness ελέγχου

Αν μια προδιαγραφή παραβιάζει μια safety ιδιότητα, τότε υπάρχει μια πεπερασμένη συμπεριφορά που παρουσιάζει την παραβίαση. Αυτή η συμπεριφορά μπορεί να παραχθεί με ένα πεπερασμένο model. Αυτή είναι ως εκ τούτου, γενικά, δυνατό να ανακαλύψει την παραβίαση με το TLC. Αυτή μπορεί να είναι μη δυνατό να ανακαλύψει μια παραβίαση μιας liveness ιδιότητας με οποιοδήποτε πεπερασμένο model.

Για να δούμε γιατί, μελετάμε την ακόλουθη απλή προδιαγραφή EvenSpec που ξεκινά με το x ίσο με μηδέν και επαναληπτικά το αυξάνει κατά 2:

$$EvenSpec \triangleq (x = 0) \wedge \Box[x' = x + 2]_x \wedge \mathrm{WF}_x(x' = x + 2)$$

Προφανώς, το x ποτέ δεν ισούται με 1 σε οποιαδήποτε συμπεριφορά ικανοποιεί την EvenSpec. Έτσι, η EvenSpec δεν ικανοποιεί την liveness ιδιότητα $\Diamond(x = 1)$. Υποθέτοντας ότι ζητάμε στο TLC να ελέγξει αν η EvenSpec συνεπάγεται την $\Diamond(x = 1)$. Για να επιτύχουμε το TLC να τερματιστεί, πρέπει να παρέχουμε έναν περιορισμό ο οποίος το περιορίζει για να αναπαράγει ένα πεπερασμένο σύνολο από προσβάσιμες καταστάσεις. Όλες οι μη πεπερασμένες συμπεριφορές που ικανοποιούν την $(x = 0) \wedge \Box[x' = x + 2]_x$ που το TLC παράγει τότε καταλήγουν σε ένα μη πεπερασμένο αριθμό stuttering βημάτων. Σε οποιαδήποτε τέτοια συμπεριφορά, η ενέργεια

$x' = x + 2$ καθίσταται συνήθως ικανή, αλλά μόνο ένας πεπερασμένος αριθμός $x' = x + 2$ βημάτων συμβαίνουν, και έτσι η $\mathrm{WF}_x(x' = x + 2)$ είναι ψευδής. Το TLC επομένως δεν θα αναφέρει ένα σφάλμα επειδή η έκφραση

$$\mathrm{WF}_x(x' = x + 2) \Rightarrow \Diamond(x = 1)$$

ικανοποιείται από όλες τις μη πεπερασμένες συμπεριφορές που αυτό παράγει.

Όταν πραγματοποιούμε χρονικό έλεγχο, πρέπει να γίνουμε σίγουροι ότι το model μας θα επιτρέπει μη πεπερασμένες συμπεριφορές που ικανοποιούν την liveness συνθήκη της προδιαγραφής.

Είναι καλή ιδέα να επιβεβαιώνουμε ότι το TLC εκτελεί τον liveness έλεγχο που αναμένουμε. Το πετυχαίνουμε αυτό ελέγχοντας μια liveness ιδιότητα που η προδιαγραφή δεν ικανοποιεί και να σιγουρευτούμε ότι αυτό αναφέρει ένα σφάλμα.

Το TLC για τον liveness model checking κατασκευάζει έναν behavior γράφο (DiskGraph).

Αφού γίνει ο σημασιολογική ανάλυση των τριών module (Naturals, HourClock, LiveHourClock) που περιέχει το παράδειγμα μας ξεκινά το model checking.

```
Semantic processing of module Naturals
Semantic processing of module HourClock
Semantic processing of module LiveHourClock
```

Property	Value
Init (initPredVec)	hr \in (1 .. 3)
Next (nextPred)	hr' = IF hr # 3 THEN hr + 1 ELSE 1
Temporal	WF_hr(HCnxt)
Invariant	hr \in (1 .. 3)
ImpliedInit	[]
ImpliedAction	[]
ImpliedTemporal	[]<><<HCnxt>>_hr \A n \in 1..3 : []<>(hr = 2)
Constraint (ModelConstraints)	[]
ActionConstraint	null

Το πλήθος των ImpliedTemporal δεν είναι μηδενικό και επομένως θα έχουμε Liveness model checking.

Μια property της μορφής $\Box\Diamond F$ για οποιαδήποτε έκφραση F θεωρείται ότι είναι μια liveness property που πρέπει να ελεγχθεί από το TLC μέσω της ακόλουθης διαδικασίας:

Κατά την διάρκεια του parsing του configuration αρχείου όταν συναντάται το token PROPERTY ή PROPERTIES λαμβάνει τα επόμενα tokens που αντιστοιχούν στα properties που πρέπει να ελεγχθούν και τοποθετούνται σε έναν

ειδικό διάνυσμα (props). Αφού προηγηθεί η σημασιολογική ανάλυση των modules, όπου παράγεται ο σημασιολογικός γράφος για κάθε εξωτερική module, γίνεται η επεξεργασία καθεμίας ιδιότητας που έχει βρεθεί πιο πάνω. Συγκεκριμένα αποδίδεται μια τιμή level ανάλογα με την μορφή της ιδιότητας σύμφωνα με την οποία τιμή κατατάσσεται η ιδιότητα σε συγκεκριμένη κατηγορία ιδιοτήτων π.χ level=3 (impliedTemporal), level=1 (invariant).

0 --> constant

1 --> state expression

2 --> action expression

3 --> temporal expression

Πραγματοποιείται η αρχικοποίηση για liveness έλεγχο. Συγκεκριμένα κανονικοποιούνται οι εκφράσεις των temporals και των impliedTemporals, δηλαδή μετατρέπονται σε **διαζευκτική κανονική μορφή (disjunctive normal form, DNF)** για να ελεχθεί η εγκυρότητα τους, και να υπολογιστεί η σειρά και ο τρόπος με τον οποίο πρέπει να ελεγχθούν. Αυτή η μέθοδο επιστρέφει έναν handle (μια συλλογή από OrderOfSolution), ο οποίος μπορεί να χρησιμοποιηθεί και σε άλλες liveness εργασίες.

Θεωρία: πραγματοποιείται έλεγχος για την ύπαρξη counterexamples για το:
spec /\ livespec => []inv /\ livecheck
π.χ. (spec /\ livespec /\ <>-inv) \/ (spec /\ livespec /\ -livecheck)
Το πρώτο μέρος της διάζευξης (inv) ελέγχεται ήδη από τον model checker on the fly. Μετατρέπεται το δεύτερο μισό σε κανονική μορφή. Στην πραγματικότητα παραλείπεται το spec σε αυτό που παράγεται. Αυτό υπονοείται. Έτσι, η μόνη εργασία είναι να αλλαχθεί το livespec /\ -livecheck:
live1 /\ live2 ... /\ (-check1 \/ -check2 ...)
σε κανονική μορφή.

Η κλάση LiveExprNode αναπαριστά κόμβους που αντιστοιχούν σε ένα είδος έκφρασης π.χ, μια σταθερά constant, ή μια έκφραση κατάστασης, μια έκφραση ενέργειας ή μια χρονική έκφραση.
Συγκεκριμένα μπορεί να είναι

LNConj	μια σύζευξη. (περιλαμβάνει μια λίστα από συζευκτέους)
LNDisj	μια διάζευξη. (περιλαμβάνει μια λίστα από διαζευκτέους)
LNAll	Πάντοτε: []e
LNEven	Τελικά: <>e
LNNeg	Άρνηση: -e
LNState	Κατηγόρημα κατάστασης. Συγκεκριμένοι τύποι: LNStateAST, LNStateEnabled. Υπάρχει κάποια ιεραρχία για τον τύπο LNState. Αυτό γιατί ο LNStateAST (ο οποίος έχει μόνο έναν ASTNode για το κατηγόρημα κατάσταση) πρέπει να εκτιμηθεί διαφορετικά από τον LNStateEnabled (ο οποίος έχει ENABLED ast μέσα του)
LNAction	Κατηγόρημα μετάβασης.
LNNext	επόμενος. ()e. Χρησιμοποιείται μόνο για την κατασκευή του tableau. Δεν αποτελεί μέρος της TLA.

Σημείωση: Στην συνέχεια για λόγους ευκολίας στην απεικόνιση αντί των LNConj και LNDisj χρησιμοποιούμε τα σύμβολα που αντιστοιχούν σε αυτούς τους κόμβους, δηλαδή τα /\ και \/ αντίστοιχα.

Temporals

```
(WF_hr(HCnxt))
```

LNAll LNEven(LNNeg LNStateEnabled(HCnxt)_hr \/ LNAction(HCnxt)_hr)

ImpliedTemporals

```
([]<><<HCnxt>>_hr, \A n \in 1..3 : []<>(hr = 2))
```

(LNNeg LNAll LNEven LNAction HCnxt $AngleAct hr) \/

LNNeg (LNAll LNEven LNStateAST (hr=2) /\ LNAll LNEven LNStateAST (hr=2) /\ LNAll LNEven LNStateAST(hr=2))

Οσον αφορά το OrderOfSolution

Ο αλγόριθμος αποσυνθέτει την fairness spec /\ ~check (δηλαδή την λίστα των temporals και των impliedTemporals) σε μια διάζευξη από συζευκτέους της ακόλουθης μορφής (Μετατροπή σε DNF):

(<>[ ]a /\ [ ]<>b /\ tf1) \/ (<>[ ]c /\ [ ]<>d /\ tf2) ..

Για το παράδειγμα μας η αρχική σύζευξη:

(LNAll LNEven(LNNeg LNStateEnabled ([HCnxt]_hr))\/ (LNAction([HCnxt]_hr))

/\

[LNNeg LNAll LNEven LNAction (HCnxt $AngleAct hr) \/

(LNNeg (LNAll LNEven LNStateAST(hr=2) /\ LNAll LNEven LNStateAST(hr=2) /\ LNAll LNEven LNStateAST(hr=2)]

Μετασχηματίζεται στην ακόλουθη διάζευξη:

(LNAll LNEven(LNNeg LNStateEnabled ([HCnxt]_hr))\/ (LNAction([HCnxt]_hr))

/\ LNEven LNAll LNNeg LNAction (HCnxt $AngleAct hr))

\/

(LNAll LNEven(LNNeg LNStateEnabled [HCnxt]_hr) \/ LNAction([HCnxt]_hr))

/\ LNEven LNAll LNNeg LNStateAST(hr=2))

\/

(LNAll LNEven (LNNeg LNStateEnabled [HCnxt]_hr) \/ LNAction([HCnxt]_hr))

/\ LNEven LNAll LNNeg LNStateAST(hr=2))

\/

(LNAll LNEven(LNNeg LNStateEnabled [HCnxt]_hr \/ LNAction([HCnxt]_hr))

/\ LNEven LNAll LNNeg LNStateAST(hr=2))

Στην συνέχεια, συγκεντρώνονται όλες μαζί οι διαζεύξεις που έχουν το ίδιο tf. Αυτό θα συμβεί σε περιπτώσεις όπως στην(WF /\ SF) => (WF /\ SF /\ TF), αφού το WF και το SF θα διασπαστούν σε πολλές περιπτώσεις και το TF θα παραμείνει ίδιο σε όλη την διάρκεια. (Στην πραγματικότητα, ελέγχεται η ισότητα συντακτικά μόνο στα TFs.)

Επομένως, εντοπίζονται οι διαζευκτέοι που έχουν την ίδια χρονική έκφραση (temporal formula,tf). Το OrderOfSolution ομαδοποιεί αυτούς τους διαζευκτέους που έχουν την ίδια χρονική έκφραση. Για κάθε χρονική έκφραση (tf) δημιουργείται ένα OrderOfSolution:

(<>[ ]a/\[ ]<>b \/ <>[ ]c/\[ ]<>d) /\ tf

Κάθε συζευκτέος (<>[]a/\[]<>b) αναπαρίσταται με ένα PossibleErrorModel.

Στην περίπτωση μας το tf είναι null (σε περίπτωση που θα είχαμε tf θα είχαμε tableau).

Έτσι έχουμε το **πρώτο PossibleErrorModel**

LNAll LNEven (LNNeg LNStateEnabled [HCnxt]_hr) \/ LNAction([HCnxt]_hr) /\

LNEven LNAll LNNeg LNAction (HCnxt $AngleAct hr)

Δεύτερο PossibleErrorModel

LNAll LNEven (LNNeg LNStateEnabled [HCnxt]_hr) \/ LNAction([HCnxt]_hr) /\

LNEven LNAll LNNeg LNStateAST (hr=2)

Τρίτο PossibleErrorModel

LNAll LNEven (LNNeg LNStateEnabled [HCnxt]_hr) \/ LNAction([HCnxt]_hr) /\

LNEven LNAll LNNeg LNStateAST (hr=2)

Τέταρτο PossibleErrorModel

LNAll LNEven (LNNeg LNStateEnabled [HCnxt]_hr) \/ LNAction([HCnxt]_hr) /\

LNEven LNAll LNNeg LNStateAST (hr=2)

Είναι πιθανό ένα απλό order of solution να έχει πολλά αντίγραφα των <>[](EA, Eventually Always) και των []<>(AE, Always Eventually) που κατανέμονται στις διαζεύξεις και τις συζεύξεις των pems του. Για να αποφευχθεί η σπατάλη, χρησιμοποιούνται δύο πίνακες αναζήτησης(lookup tables): ο checkState και ο checkAction. Συγκεκριμένα όταν εξετάζεται κάθε κατάσταση και οι μεταβάσεις της, τα στοιχεία αυτών των πινάκων ελέγχονται. Συνεπώς οι τιμές των pems που δείχνουν τι θα πρέπει να εξεταστεί είναι απλοί ακέραιοι δείκτες (indexs) σε αυτούς τους δύο πίνακες.

Ο πίνακας checkState στην περίπτωση μας για την συγκεκριμένη OrderOfSolution είναι κενός. Όσο αφορά τον πίνακα checkAction έχουμε τα ακόλουθα στοιχεία:

checkAction	
0) LNNeg LNStateEnabled [HCnxt]_hr\/ LNAction([HCnxt]_hr)	**(AEAction)**
1) LNNeg LNAction (HCnxt $AngleAct hr)	**(EAAction)**
2) LNNeg LNStateAST (hr=2)	**(EAAction)**
3) LNNeg LNStateAST (hr=2)	**(EAAction)**
4) LNNeg LNStateAST (hr=2)	**(EAAction)**

Μόλις ολοκληρωθεί η αρχικοποίηση του liveness ελέγχου τυπώνεται

```
Implied-temporal checking--satisfiability problem has 1 branches.
```

που αναφέρεται στο μήκος αυτού του handle δηλαδή σημαίνει ότι περιλαμβάνει ένα OrderOf Solution.

Ανάλογα με το μήκος δημιουργείται αντίστοιχο πλήθος από **DiskGraph** όπου αποθηκεύει έναν behavior γράφο στον δίσκο. Χρησιμοποιούνται δύο αρχεία δίσκου για να αποθηκευτεί ο κάθε γράφος. Για κάθε κόμβο στον γράφο, το πρώτο αρχείο (**nodes_0**) αποθηκεύει τις successors καταστάσεις και πληροφορίες που έχουν υπολογιστεί για τον κόμβο, και το δεύτερο αρχείο (**ptrs_0**) αποθηκεύει το fingerprint του κόμβου και τον δείκτη στην θέση του κόμβου στο πρώτο αρχείο.

Προσωρινά αποθηκεύονται μέρη του γράφου στην μνήμη.

Ένας **GraphNode** είναι ένας κόμβος στον behavior γράφο. Σε κάθε κόμβο αποθηκεύονται μόνο τα fingerprints των καταστάσεων, παρά οι πραγματικές καταστάσεις. Έτσι, για κάθε κατάσταση που συναντούμε, χρειάζεται να υπολογιστούν όλα τα <>[] (EA, Eventually Always) και τα []<>(AE, Always Eventually) που καταγράφονται στο order of solution. Για κάθε outgoing ακμή, καταγράφουμε το fingerprint του κόμβου στόχου και τα checkActions κατά μήκος της.

Αφού πρώτα ελέγξει ότι κάθε ASSUME στην προδιαγραφή ικανοποιείται από τις τιμές που ανατέθηκαν στις σταθερές παραμέτρους (βήμα το οποίο δεν πραγματοποιείται μιας και δεν έχουμε κανένα assumption) υπολογίζει τις αρχικές καταστάσεις εκτιμώντας το αρχικό κατηγόρημα Init.

Στην συνέχεια το TLC παράγει και ελέγχει όλες τις πιθανές καταστάσεις που ικανοποιούν το init κατηγόρημα και θέτει το G και την U να περιλαμβάνουν όλες αυτές τις καταστάσεις.

Στην περίπτωση μας έχουμε τρεις αρχικές καταστάσεις (η s1:hr=1, η s2:hr=2 και η s3:hr= 3).

Για κάθε μια από τις αρχικές καταστάσεις:

1. Αν το κατηγόρημα Constraint (στην περίπτωση μας δεν έχουμε κάποιο constraint) είναι αληθές στην κατάσταση s1, προσθέτει τον κόμβο s1 (το fingerprint) στο σύνολο G (συγκεκριμένα στο in-memory hash table). Εφόσον η κατάσταση s1 δεν υπήρχε στον γράφο

   - τοποθετείται και στην ουρά U (συγκεκριμένα στον enqBuf της ουράς)

- κατασκευάζεται ο behavior γράφος για liveness checking καταγράφοντας ότι η κατάσταση s1 είναι μια initial κατάσταση στον behavior γράφο .
- Και εκτιμά τα κατηγόρηματα Invariant και ImpliedInit(στην περίπτωση μας δεν έχουμε κάποιο ImpliedInit) στην κατάσταση s1.

Το ίδιο επαναλαμβάνεται και για τις άλλες δύο καταστάσεις s2 και s3. Αφού ολοκληρωθεί η διαδικασία και για τις τρεις αρχικές καταστάσεις τυπώνεται κατάλληλο μήνυμα που αναφέρει το πλήθος των καταστάσεων που έχουν παραχθεί.

```
Finished computing initial states: 3 distinct states generated.
```

Στην συνέχεια λαμβάνεται κάθε κατάσταση από την ουρά, και παράγονται όλες οι δυνατές επόμενες καταστάσεις της κατάστασης, ελέγχονται τα invariants, και ενημερώνεται το σύνολο των καταστάσεων και η ουρά καταστάσεων.

2. Όσο η U δεν είναι κενή, κάνει τα ακόλουθα:

a) Αφαιρεί την πρώτη κατάσταση από την U, δηλαδή την s1, εν προκειμένου η s1 να είναι η τρέχουσα κατάσταση.

b) Βρίσκει το σύνολο T όλων των successor καταστάσεων της s1 εκτιμώντας την ενέργεια επόμενης κατάστασης ξεκινώντας από την s1.

c) Στην περίπτωση του s1 έχουμε ένα στοιχείο στο T, το s2. Αυξάνεται ανάλογα με το πλήθος των επόμενων καταστάσεων το συνολικό πλήθος των παραγόμενων καταστάσεων.

d) Για κάθε κατάσταση t στο T, κάνει τα ακόλουθα:

iii. Ελέγχεται αν η κατάσταση t ικανοποιεί τα κατηγορήματα Constraint και ActionConstraint (που στην περίπτωση μας είναι κενά). Ο έλεγχος επιστρέφει θετικό αποτέλεσμα (true)

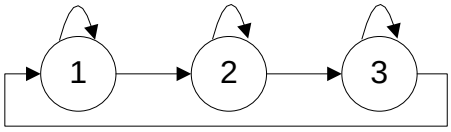
A. Στην συνέχεια προσπαθεί να προσθέσει την t μέσα στο G. Στην περίπτωση μας υπάρχει ήδη η κατάσταση αυτή στο σύνολο G (στο in-memory hash table) οπότε δεν την προσθέτει.

iv. Τέλος ελέγχεται αν η κατάσταση t ικανοποιεί το ImpliedAction. (στην περίπτωση μας είναι κενό).

e) Προσθέτει τις ακμές s1->t και s1->s1 (stuttering βήμα) στον behavior γράφο ελέγχοντας ταυτόχρονα αν ικανοποιούνται τα temporals και τα impliedTemporals .

Η παραπάνω διαδικασία του βήματος 2 επαναλαμβάνεται και για τις άλλες δύο καταστάσεις s2 και s3. Μόνο που για την s2 έχουμε την επόμενη κατάσταση s3 και για την s3 την s1.

Επομένως συνολικά παράγονται 6 καταστάσεις.(τρεις αρχικές συν τις τρεις που προέκυψαν κατά τον υπολογισμό των successor καταστάσεων.



Στην συνέχεια ξεκινά ο έλεγχος για τις liveness ιδιότητες όπου και τυπώνεται

--Checking temporal properties for the complete state space...

Λαμβάνεται το σύνολο των OrderOfSolution (εμείς έχουμε ένα για το παράδειγμα μας) και για κάθε ένα από αυτά λαμβάνει τα PossibleErrorModel της λύσης.

Για το δικό μας OrderOfSolution έχουμε τα ακόλουθα PossibleErrorModel. Η τιμή κάθε στοιχείου AEAction και EAAction αντιστοιχεί σε μια θέση στον checkAction που βρήκαμε πιο πάνω :

pems	PossibleErrorModel[4] (id=90)
[0]	PossibleErrorModel (id=95)
AEAction	(id=99)
[0]	0
AEState	(id=101)
EAAction	(id=102)
[0]	1
[1]	PossibleErrorModel (id=96)
AEAction	(id=103)
[0]	0
AEState	(id=104)
EAAction	(id=105)
[0]	2
[2]	PossibleErrorModel (id=97)
AEAction	(id=106)
[0]	0
AEState	(id=107)
EAAction	(id=108)
[0]	3
[3]	PossibleErrorModel (id=98)
AEAction	(id=109)
[0]	0
AEState	(id=110)
EAAction	(id=111)
[0]	4

Για κάθε ένα από τα στοιχεία αυτά(PossibleErrorModel) υπολογίζει τα strongely connected components, και ελέγχει κάθε ένα από αυτά αν περιέχει ένα counterexample.

Ένα ισχυρά συνδεδεμένο συστατικό (**strongly connected component**) σε ένα γράφο είναι ένα μέγιστο υποσύνολο από κόμβους και ακμές τέτοιο ώστε κάθε κόμβος να είναι προσβάσιμος από κάθε άλλο κόμβο. Τα συστατικά συλλαμβάνουν πλήρως κάθε πιθανό κύκλο.

Συγκεκριμένα ξεκινώντας για το

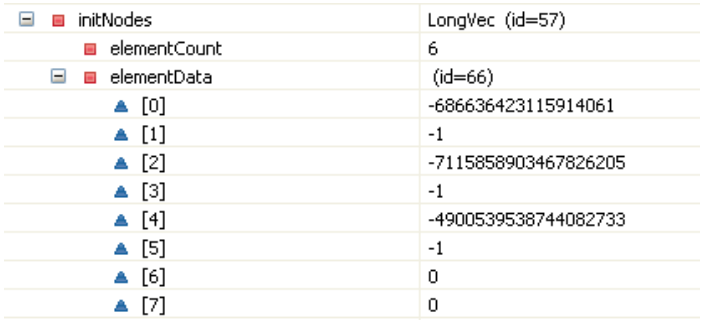
1ο PossibleErrorModel

0) LNNeg LNStateEnabled [HCnxt]_hr\/ LNAction([HCnxt]_hr) **(AEAction)**

1) LNNeg LNAction (HCnxt $AngleAct hr) **(EAAction)**

Δημιουργείται ένας in-memory node-pointer πίνακας από το node-pointer αρχείο που αντιστοιχεί στον behavior γράφο (όπου είναι ένα BufferedRandomAccessFile το οποίο μοιάζει με ένα RandomAccessFile, αλλά αυτό χρησιμοποιεί έναν private buffer έτσι ώστε οι περισσότερες λειτουργίες να μην απαιτούν πρόσβαση στον δίσκο).

Αρχικοποιείται μια ουρά nodeQueue από Integers και Longs στην μνήμη με τις αρχικούς κόμβους του γράφου που είναι 6. Οι αρχικοί κόμβοι είναι :

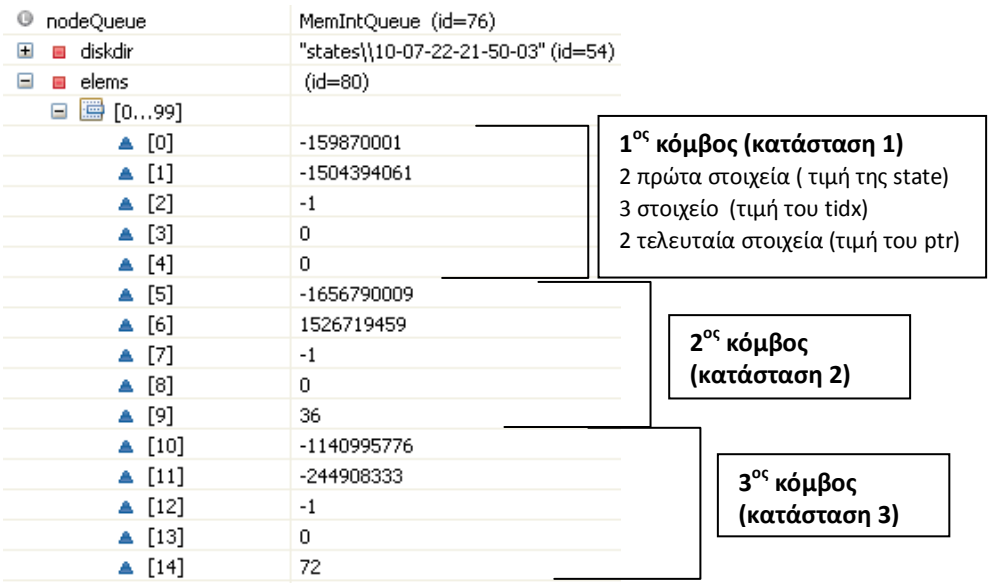


Στην πραγματικότητα λαμβάνονται στην πράξη τρεις κόμβοι μιας και οι υπόλοιπες τρεις καταστάσεις /κόμβοι είναι οι ίδιες καταστάσεις με τις προηγούμενες που προέκυψαν κατά την διαδικασία εύρεσης των successor καταστάσεων. Στον πίνακα στην συνέχεια φαίνεται η αντιστοίχιση καθεμιάς κατάστασης στο αντίστοιχο fingerprint της.

State	fp (fingerprint)
1	-686636423115914061
2	-7115858903467826205
3	-4900539538744082733

Συγκεκριμένα τοποθετούνται για κάθε κατάσταση: η τρέχουσα κατάσταση *state* (long)**, το αναγνωριστικό της επόμενης *tidx*** (int)(Ο οποίος έχει νόημα όταν χρησιμοποιείται tableau στην διαδικασία του liveness model checking. Η τιμή -1 σημαίνει ότι δεν υπάρχει tableau) **και το link που έχει αποδοθεί στην συγκεκριμένη κατάσταση *ptr***(long)**.**

Τρέχουσα κατάσταση nodeQueue



Για όλα τα στοιχεία της παραπάνω ουράς nodeQueue παράγει τα SCCs και ελέγχει αν περιέχουν κάποιο "bad" κύκλο. Διαδικασία που επιτυγχάνεται μέσα από ένα αριθμό επαναλήψεων συγκεκριμένα όσο αυτή η nodeQueue ουρά έχει στοιχεία.

Σημείωση: Η τρέχουσα κατάσταση της dfsStack και της nodeQueue είναι η περιοχή που οριοθετείται στο γραφικό στιγμιότυπο, έτσι ώστε να υπάρχει διάκριση από τις παλιές τιμές που συνεχίζουν να υφίστατε.

1η Επανάληψη

Μετά την αρχικοποίηση το αρχικό μήκος είναι 15 (πέντε στοιχεία για κάθε κόμβο).

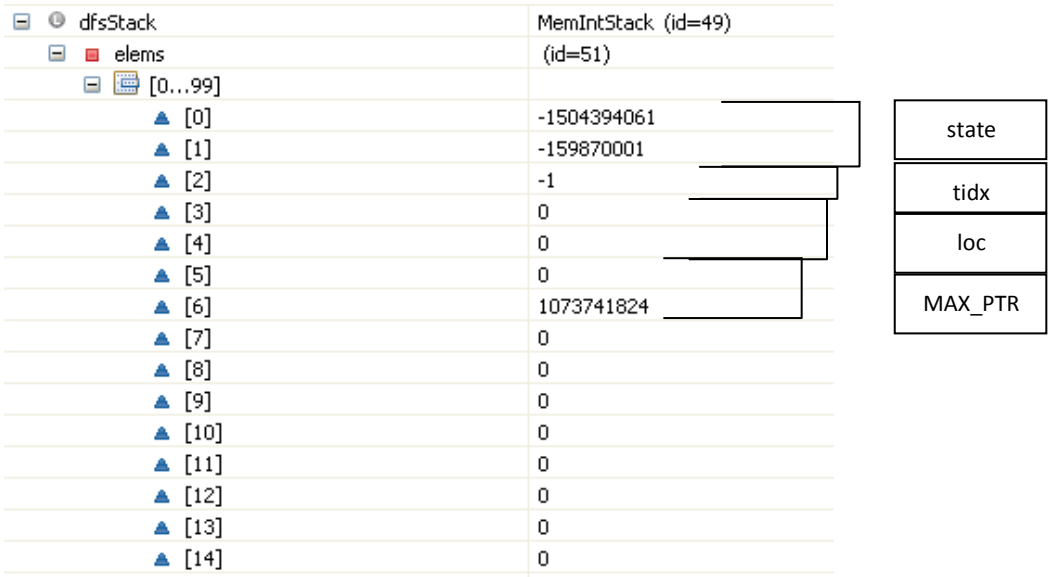
1. Αφού αφαιρέσει μια κατάσταση από την ουρά, ξεκινά τον υπολογισμό των SCCs με την κατάσταση αυτή με <state, tidx> ως τον root κόμβο. Τα στοιχεία για την πρώτη κατάσταση (**κατάσταση 1**) είναι:

   state: -686636423115914061, **tidx**: -1, **loc(το ptr)**:0

2. Τοποθετεί τα τρία στοιχεία της κατάστασης καθώς και την τιμή *MAX_PTR* (long)(Το μέγιστο μήκος του αρχείου fileForNodes του γράφου.) σε μια στοίβα **dfsStack** (την οποία κάθε φορά κάνει reset για κάθε κατάσταση της ουράς nodeQueue).

3. Αρχικοποιεί μια μεταβλητή **newLink** να είναι ίση με την τιμή του *MAX_PTR*.

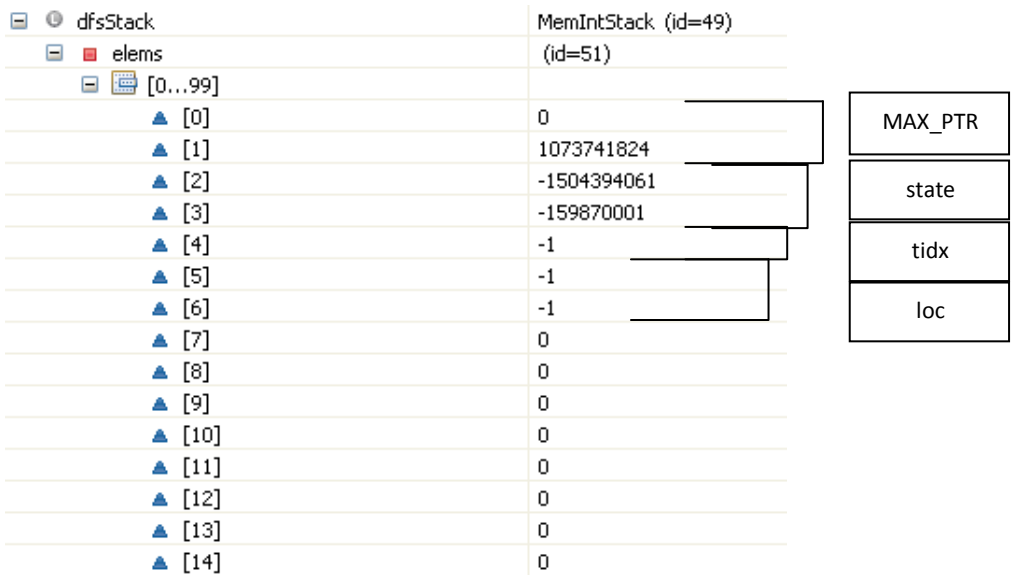
Τρέχουσα κατάσταση dfsStack



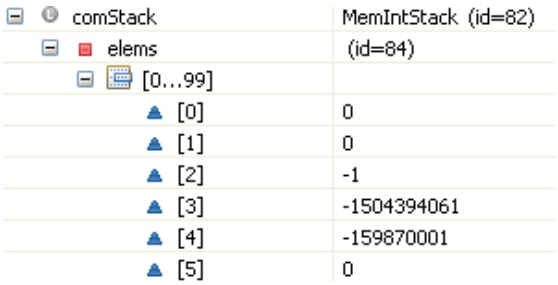
4. Το πλήθος των στοιχειών της στοίβας dfsStack είναι μεγαλύτερο από 2 (στην συγκεκριμένη περίπτωση είναι 7 όπως φαίνεται από την παραπάνω εικόνα):

- Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο (**κατάσταση 1**) της στοίβας.
- Ελέγχεται το loc αν είναι μικρότερο του μηδέν έτσι ώστε να εξαχθεί το συμπέρασμα αν ο κόμβος έχει εξερευνηθεί. Στην περίπτωση μας δεν είναι μικρότερο του 0 οπότε ο κόμβος δεν έχει εξερευνηθεί
  - Στο συγκεκριμένο κόμβο δεν έχει ανατεθεί ένα link και έτσι του ανατίθεται ένα link.
  - Επανατοποθετεί την τρέχουσα κατάσταση(κόμβο) στην **dfsStack**, αλλά κάνει τη κατάσταση explored κάνοντας το loc ίσο με -1.

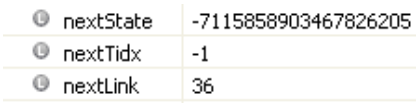
Τρέχουσα κατάσταση dfsStack



Επίσης τοποθετεί την τρέχουσα κατάσταση σε μια άλλη στοίβα την **comStack** που περιέχει τις καταστάσεις αυτές που έχουν εξερευνηθεί με την αρχική τιμή του loc.(χωρίς να κάνει το loc ίσο με -1).

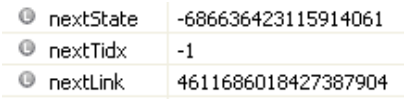


- Αρχικοποιεί μια μεταβλητή **nextLowLink** με την τιμή της **newLink** και αυξάνει την τελευταία κατά ένα.
- Κοιτάζει όλες τις successors καταστάσεις αυτής της κατάστασης (που είναι δύο). Για την πρώτη:
  - Λαμβάνει τα στοιχεία που χαρακτηρίζουν την συγκεκριμένη successor κατάσταση **(κατάσταση 2)** . (**StateFP**, **tidx** και το **link** που έχει ανατεθεί στην συγκεκριμένη κατάσταση/κόμβο)



  - Το nextlink (το loc δηλαδή) είναι μεγαλύτερο του μηδενός (δεν έχει εξερευνηθεί η συγκεκριμένη successor κατάσταση) οπότε ελέγχει τα EAAction
    - Επιστρέφεται **false** και επειδή το nextlink της successor κατάστασης είναι *FilePointer* (*είναι μικρότερο του MAX_PTR = 4611686018427387904*) τοποθετεί την successor κατάσταση στην **nodeQueue** (έτσι ώστε να εξερευνηθεί και να τις ανατεθεί κάποιο link)

Η ίδια διαδικασία επαναλαμβάνεται και για την δεύτερη κατάσταση (βλέπουμε από την τιμή της nextState ότι πρόκειται για την ίδια την υπό εξέταση κατάσταση, **την κατάσταση 1**).

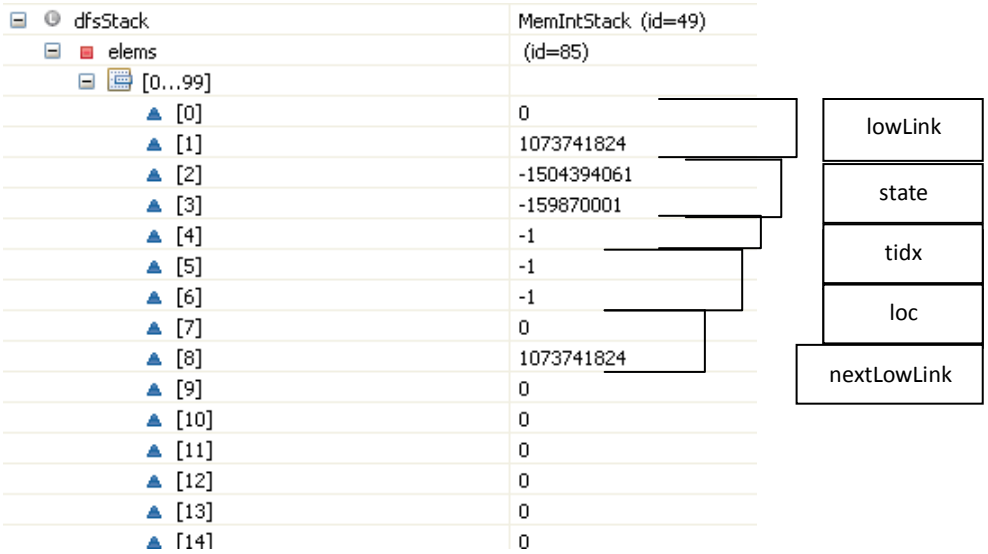


Με την μόνη διαφορά ότι επιστρέφεται true από τον έλεγχο των EAAction

- το nextlink δεν είναι *FilePointer (δεν είναι μικρότερο του MAX_PTR = 4611686018427387904)* και έτσι απλά ελέγχει αν το nextLink είναι μικρότερο του nextLowLink (που αρχικά είναι ίσο με *MAX_PTR*) όπου επίσης δεν ισχύει γιατί είναι ίσα και δεν κάνει το nextLowLink ίσο με το nextLink.

- Τοποθετεί την τιμή της nextLowLink ως είχε στην dfsStack.

Τρέχουσα κατάσταση dfsStack



5. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 9) είναι μεγαλύτερο από 2:

- Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο (**πρόκειται για την κατάσταση 1**) της στοίβας.

lowLink	4611686018427387904
curLoc	-1
curTidx	-1
curState	-686636423115914061

Το curloc είναι μικρότερο του 0 οπότε ο κόμβος έχει εξερευνηθεί

- Το curLink (4611686018427387904) που έχει αποδοθεί στην συγκεκριμένη κατάσταση είναι ίσο με το lowLink και οπότε οι καταστάσεις στην comStack από την κορυφή μέχρι την curState σχηματίζουν ένα SCC. Γίνεται έλεγχος για "bad" κύκλους. Για το τρέχον pem (PossibleErrorModel), αυτή η μέθοδο ελέγχει αν το τρέχον scc ικανοποιεί τα AEs του . (Γνωρίζουμε ότι το τρέχον scc ικανοποιεί το EA του pem.) Αν ικανοποιούνται, αυτό το pem περιέχει ένα counterexample, και αυτή η μέθοδο τότε καλεί την printErrorTrace για να τυπώσει ένα σφάλμα και επιστρέφει false. Στην περίπτωση μας δεν ικανοποιείται.

  Συγκεκριμένα πραγματοποιείται η ακόλουθη διαδικασία:

  Εξάγεται η πρώτη κατάσταση της **comStack** και συγκρίνεται με την τρέχουσα κατάσταση αν είναι ίδιες αλλά και αν ο κόμβος <state, tidx> δεν παρουσιάζει stutterring έτσι ώστε να αποφασισθεί αν το component είναι trivial.

state1	-686636423115914061
tidx1	-1
loc1	0

  Στην συγκεκριμένη περίπτωση ο συγκεκριμένος κόμβος παρουσιάζει stuttering οπότε δεν είναι trivial.

Τοποθετούνται όλοι οι κόμβοι του component σε έναν hashtable. Ξεκινά ο έλεγχος αυτού του component. Το μήκος αυτού του hashtable είναι 128. Ελέγχει ένα ένα τα στοιχεία και αν δεν είναι null τότε εξάγει τον κόμβο και ελέγχει τα AEState (στην περίπτωση μας είναι μηδενικό το πλήθος αυτών) και τα AEAction για κάθε επόμενο κόμβο που δείχνει ο κόμβος και ο οποίος υπάρχει μέσα στο hashtable. Ελέγχεται αν εκπληρώνεται μέσω των promises (ο έλεγχος δεν μπαίνει σε αυτό το βήμα μιας και δεν έχουμε promises είναι για την περίπτωση ύπαρξης tableau). Αν ικανοποιούνται όλες οι συνθήκες τότε έχουμε ένα counterexample. Σε εμάς δεν ικανοποιούνται οπότε δεν έχουμε ένα counterexample.

- Εξάγεται το plowLink (που είναι ίσο με DiskGraph.*MAX_PTR*) από την **dfsStack** και συγκρίνεται με το lowLink του συγκεκριμένου κόμβου για να ελεγχθεί αν το τελευταίο είναι μικρότερο του. Αυτό δεν ισχύει γιατί είναι ίσα και δεν κάνει το plowLink ίσο με το lowLink. Και έτσι, απλά επανατοποθετεί το plowLink ως είχε στην **dfsStack**.

Τρέχουσα κατάσταση dfsStack

dfsStack	MemIntStack (id=46)
elems	(id=55)
[0...99]	
[0]	0
[1]	1073741824

6. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 2, περιέχει μόνο το nextLowLink) δεν είναι μεγαλύτερο από 2 οπότε βγαίνει από αυτό το loop.

2η Επανάληψη

Τρέχουσα κατάσταση nodeQueue

nodeQueue	MemIntQueue (id=76)
diskdir	"states\\10-07-22-21-50-03" (id=54)
elems	(id=80)
[0...99]	
[0]	-159870001
[1]	-1504394061
[2]	-1
[3]	0
[4]	0
[5]	-1656790009
[6]	1526719459
[7]	-1
[8]	0
[9]	36
[10]	-1140995776
[11]	-244908333
[12]	-1
[13]	0
[14]	72
[15]	-1656790009
[16]	1526719459
[17]	-1
[18]	0
[19]	36

Η ουρά nodeQueue έχει ακόμη στοιχεία οπότε επαναλαμβάνεται η παραπάνω διαδικασία. (Τρέχον μήκος ουράς 15)

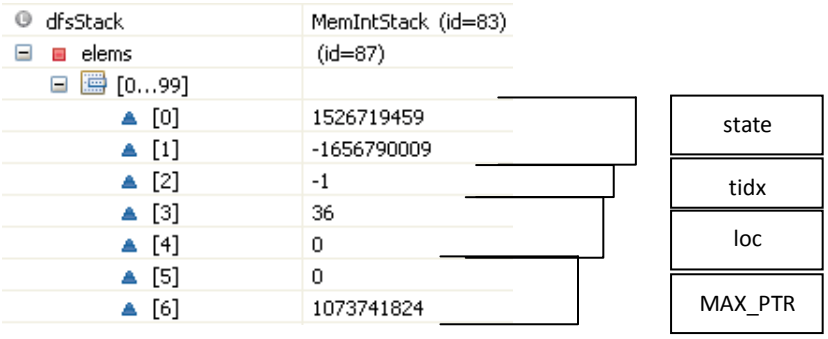
1. Αφού αφαιρέσει μια κατάσταση από την ουρά, ξεκινά τον υπολογισμό των SCCs με την κατάσταση αυτή (**κατάσταση 2**) ως τον root κόμβο. Τα στοιχεία για την πρώτη κατάσταση είναι:

   state: -7115858903467826205, **tidx**: -1, **loc(το ptr)**: 36

2. Τοποθετεί τα τρία στοιχεία της κατάστασης καθώς και την τιμή *MAX_PTR* (long)(Το μέγιστο μήκος του αρχείου fileForNodes του γράφου.) σε μια στοίβα **dfsStack** (την οποία κάθε φορά κάνει reset για κάθε κατάσταση της ουράς nodeQueue).

3. Αρχικοποιεί μια μεταβλητή **newLink** να είναι ίση με την τιμή του *MAX_PTR*.

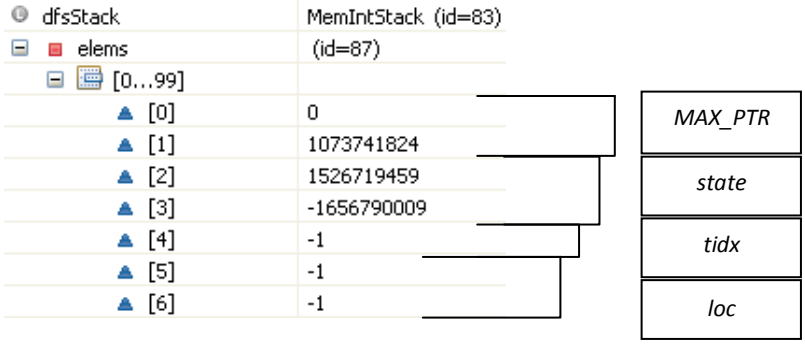
Τρέχουσα κατάσταση dfsStack



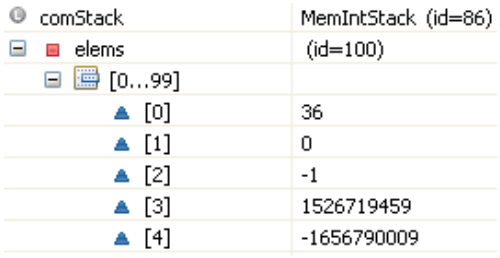
4. Το πλήθος των στοιχειών της στοίβας dfsStack είναι μεγαλύτερο από 2 (στην συγκεκριμένη περίπτωση είναι 7 όπως φαίνεται από την παραπάνω εικόνα):

   - Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο της στοίβας.
   - Ελέγχεται το loc αν είναι μικρότερο του μηδέν έτσι ώστε να εξαχθεί το συμπέρασμα αν ο κόμβος έχει εξερευνηθεί. Στην περίπτωση μας δεν είναι μικρότερο του 0 οπότε ο κόμβος δεν έχει εξερευνηθεί
     - Στο συγκεκριμένο κόμβο δεν έχει ανατεθεί ένα link και έτσι του ανατίθεται ένα link.
     - Επανατοποθετεί την τρέχουσα κατάσταση(κόμβο) στην **dfsStack**, αλλά κάνει τη κατάσταση explored κάνοντας το loc ίσο με -1.

Τρέχουσα κατάσταση dfsStack

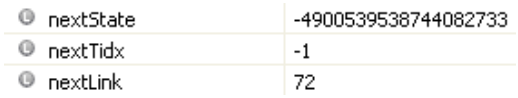


Επίσης τοποθετεί την τρέχουσα κατάσταση σε μια άλλη στοίβα την **comStack** που περιέχει τις καταστάσεις αυτές που έχουν εξερευνηθεί με την αρχική τιμή του loc.(χωρίς να κάνει το loc ίσο με -1.



comStack	MemIntStack (id=86)
elems	(id=100)
[0...99]	
[0]	36
[1]	0
[2]	-1
[3]	1526719459
[4]	-1656790009

- Αρχικοποιεί μια μεταβλητή **nextLowLink** με την τιμή της **newLink** και αυξάνει την τελευταία κατά ένα.
- Ελέγχει όλες τις successors καταστάσεις αυτής της κατάστασης (που είναι δύο). Για την πρώτη:
  - Λαμβάνει τα στοιχεία που χαρακτηρίζουν την συγκεκριμένη successor κατάσταση (**κατάσταση 3**) . (**StateFP**, **tidx** και το **link** που έχει ανατεθεί στην συγκεκριμένη κατάσταση/κόμβο)



nextState	-4900539538744082733
nextTidx	-1
nextLink	72

  - Το nextlink (το loc δηλαδή) είναι μεγαλύτερο του μηδενός (δεν έχει εξερευνηθεί η συγκεκριμένη successor κατάσταση) οπότε ελέγχει τα EAAction
    - Επιστρέφεται **false** και επειδή το nextlink της successor κατάστασης είναι *FilePointer* (*είναι μικρότερο του MAX_PTR = 4611686018427387904*) τοποθετεί την successor κατάσταση στην **nodeQueue**. (έτσι ώστε να εξερευνηθεί και να τις ανατεθεί κάποιο link)

Η ίδια διαδικασία επαναλαμβάνεται και για την δεύτερη κατάσταση (βλέπουμε από την τιμή της nextState ότι πρόκειται για την ίδια την υπό εξέταση κατάσταση, **την κατάσταση 2**).

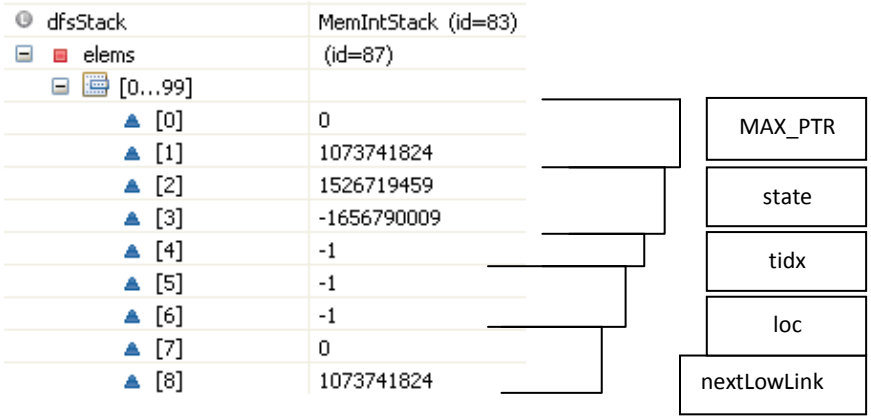
nextState	-7115858903467826205
nextTidx	-1
nextLink	4611686018427387904

Με την μόνη διαφορά ότι επιστρέφεται **true** από τον έλεγχο των EAAction

- το nextlink δεν είναι *FilePointer* (*δεν είναι μικρότερο του MAX_PTR = 4611686018427387904*) και έτσι απλά ελέγχει αν το nextLink είναι μικρότερο του nextLowLink (που αρχικά είναι ίσο με *MAX_PTR*) όπου επίσης δεν ισχύει γιατί είναι ίσα και δεν κάνει το nextLowLink ίσο με το link.

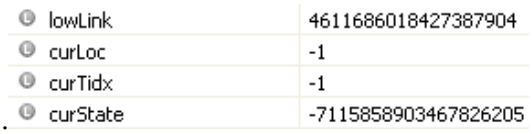
- Τοποθετεί την τιμή της nextLowLink ως είχε στην dfsStack.

Τρέχουσα κατάσταση dfsStack



5. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 9) είναι μεγαλύτερο από 2:

- Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο (**πρόκειται για την κατάσταση 2**) της στοίβας

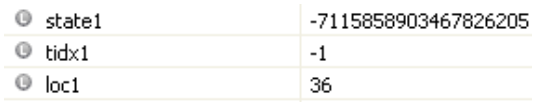


- Το curloc είναι μικρότερο του 0 οπότε ο κόμβος έχει εξερευνηθεί

  - Το curLink (4611686018427387904) που έχει αποδοθεί στην συγκεκριμένη κατάσταση είναι ίσο με το lowLink και οπότε οι καταστάσεις στην comStack από την κορυφή μέχρι την curState σχηματίζουν ένα SCC. Γίνεται έλεγχος για "bad" κύκλους. Για το τρέχον pem (PossibleErrorModel), αυτή η μέθοδο ελέγχει αν το τρέχον scc ικανοποιεί τα AEs του. (Γνωρίζουμε ότι το τρέχον scc ικανοποιεί το EA του pem.) Αν ικανοποιούνται, αυτό το pem περιέχει ένα counterexample, και αυτή η μέθοδο τότε καλεί την printErrorTrace για να τυπώσει ένα σφάλμα και επιστρέφει false. Στην περίπτωση μας δεν ικανοποιείται.

    Συγκεκριμένα πραγματοποιείται η ακόλουθη διαδικασία:

    Εξάγεται η πρώτη κατάσταση της **comStack** και συγκρίνεται με την τρέχουσα κατάσταση αν είναι ίδιες αλλά και αν ο κόμβος <state, tidx> δεν παρουσιάζει stutterring έτσι ώστε να αποφασισθεί αν το component είναι trivial.



    Στην συγκεκριμένη περίπτωση παρουσιάζει stuttering οπότε δεν είναι trivial.

    Τοποθετούνται όλοι οι κόμβοι του component σε έναν hashtable. Ξεκινά ο έλεγχος αυτού του component. Το μήκος αυτού του hashtable είναι 128. Ελέγχει ένα ένα τα στοιχεία και αν δεν είναι null τότε εξάγει τον κόμβο και ελέγχει τα AEState (στην περίπτωση μας είναι μηδενικό το πλήθος αυτών) και τα AEAction για κάθε επόμενο

κόμβο που δείχνει ο κόμβος και ο οποίος υπάρχει μέσα στο hashtable. Ελέγχεται αν εκπληρώνεται μέσω των promises (ο έλεγχος δεν μπαίνει σε αυτό το βήμα μιας και δεν έχουμε promises είναι για την περίπτωση ύπαρξης tableau). Αν ικανοποιούνται όλες οι συνθήκες τότε έχουμε ένα counterexample. Σε εμάς δεν ικανοποιούνται οπότε δεν έχουμε ένα counterexample.

- Εξάγεται το plowLink (που είναι ίσο με DiskGraph.*MAX_PTR)* από την **dfsStack** και συγκρίνεται με το lowLink του συγκεκριμένου κόμβου για να ελεγχθεί αν το τελευταίο είναι μικρότερο του. Αυτό δεν ισχύει γιατί είναι ίσα και δεν κάνει το plowLink ίσο με το lowLink. Και έτσι, απλά επανατοποθετεί το plowLink ως είχε στην **dfsStack.**

Τρέχουσα κατάσταση dfsStack

dfsStack	MemIntStack (id=46)
elems	(id=55)
[0...99]	
[0]	0
[1]	1073741824

6. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 2, περιέχει μόνο το nextLowLink) δεν είναι μεγαλύτερο από 2 οπότε βγαίνει από αυτό το loop.

3η Επανάληψη

Τρέχουσα κατάσταση nodeQueue

nodeQueue	MemIntQueue (id=76)
diskdir	"states\\10-07-22-21-50-03" (id=54)
elems	(id=80)
[0...99]	
[0]	-159870001
[1]	-1504394061
[2]	-1
[3]	0
[4]	0
[5]	-1656790009
[6]	1526719459
[7]	-1
[8]	0
[9]	36
[10]	-1140995776
[11]	-244908333
[12]	-1
[13]	0
[14]	72
[15]	-1656790009
[16]	1526719459
[17]	-1
[18]	0
[19]	36
[20]	-1140995776
[21]	-244908333
[22]	-1
[23]	0
[24]	72

Η ουρά nodeQueue έχει ακόμη στοιχεία οπότε επαναλαμβάνεται η παραπάνω διαδικασία.(Τρέχον μήκος ουράς 15)

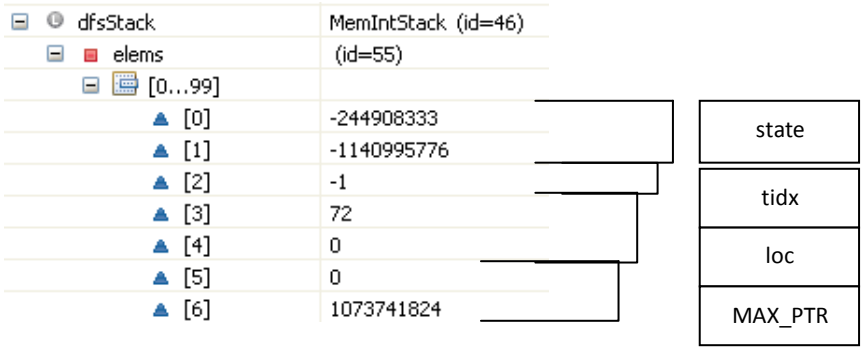
1. Αφού αφαιρέσει μια κατάσταση από την ουρά, ξεκινά τον υπολογισμό των SCCs με την κατάσταση αυτή (**κατάσταση 3**) ως τον root κόμβο. Τα στοιχεία για την πρώτη κατάσταση είναι:

   state: -4900539538744082733, **tidx**: -1, **loc(το ptr)**: 72

2. Τοποθετεί τα τρία στοιχεία της κατάστασης καθώς και την τιμή DiskGraph.*MAX_PTR* (long) σε μια στοίβα **dfsStack**.

3. Αρχικοποιεί μια μεταβλητή **newLink** να είναι ίση με την τιμή του *MAX_PTR*.

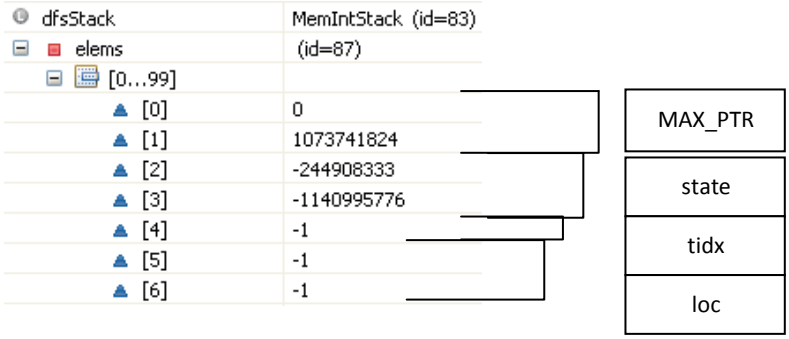
Τρέχουσα κατάσταση dfsStack



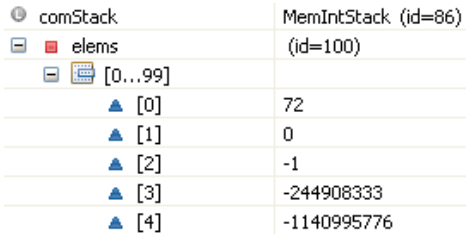
4. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 7) είναι μεγαλύτερο από 2:

   - Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο της στοίβας.
   - Το loc είναι μεγαλύτερο από 0 οπότε ο κόμβος δεν έχει εξερευνηθεί
     - Στο συγκεκριμένο κόμβο δεν έχει ανατεθεί ήδη ένα link και οπότε του ανατίθεται ένα link.
     - Επανατοποθετεί την τρέχουσα κατάσταση(κόμβο) στην **dfsStack**, αλλά κάνει τη κατάσταση explored κάνοντας το loc ίσο με -1.

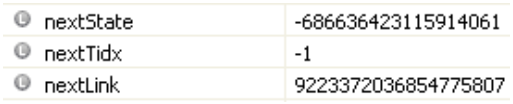
Τρέχουσα κατάσταση dfsStack



Επίσης τοποθετεί την τρέχουσα κατάσταση σε μια άλλη στοίβα την **comStack** που περιέχει τις καταστάσεις αυτές που έχουν εξερευνηθεί με την αρχική τιμή του loc.(χωρίς να κάνει το loc ίσο με -1.

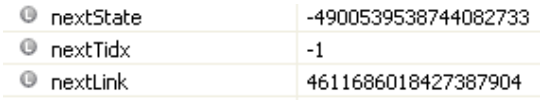


- Αρχικοποιεί μια μεταβλητή **nextLowLink** με την τιμή της **newLink** και αυξάνει την τελευταία κατά ένα.
- Ελέγχει όλες τις successors καταστάσεις αυτής της κατάστασης (που είναι δύο). Συγκεκριμένα για την πρώτη:
  - Λαμβάνει τα στοιχεία που χαρακτηρίζουν την συγκεκριμένη successor (**κατάσταση 1**). (**StateFP**, **tidx** και το **link**)



  - Το nextLink είναι μεγαλύτερο του μηδενός (η συγκεκριμένη successor κατάσταση δεν έχει εξερευνηθεί) και επομένως ελέγχει τα EAAction
    - Όπου επιστρέφεται false και για τον λόγο ότι το nextLink δεν είναι *FilePointer* δεν κάνει κάτι και δεν τοποθετεί την successor κατάσταση στην **nodeQueue**.

Η ίδια διαδικασία επαναλαμβάνεται και για την δεύτερη κατάσταση (βλέπουμε από την τιμή της nextState ότι **πρόκειται** για την ίδια την υπό εξέταση κατάσταση, **την κατάσταση 3**).

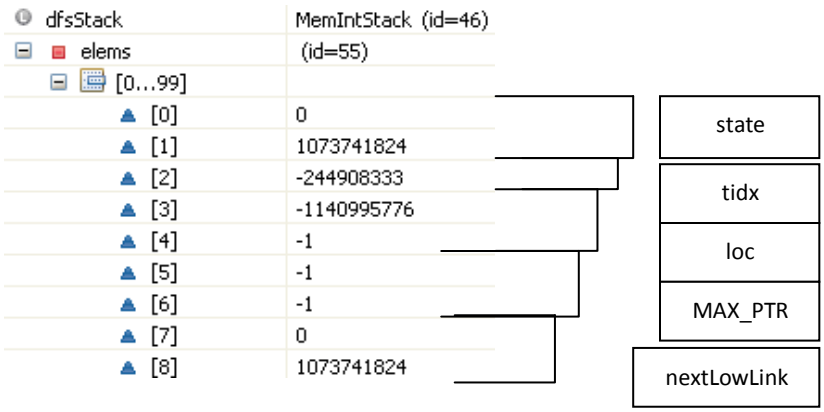


Με την μόνη διαφορά ότι επιστρέφεται true από τον έλεγχο των EAAction

- αν το nextlink ήταν *FilePointer* τότε θα τοποθετούσε την successor κατάσταση στην dfsStack. Αλλά δεν είναι *FilePointer (δεν είναι μικρότερο του MAX_PTR = 4611686018427387904)* και έτσι απλά ελέγχει αν το nextLink είναι μικρότερο του nextLowLink (που αρχικά είναι ίσο με DiskGraph.*MAX_PTR*) όπου επίσης δεν ισχύει και δεν κάνει το nextLowLink ίσο με το nextLink (είναι ίσα).

- Τοποθετεί την nextLowLink στην dfsStack.

Τρέχουσα κατάσταση dfsStack

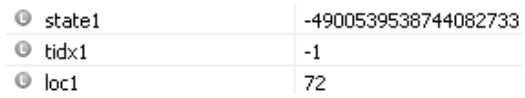


5. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 9) είναι μεγαλύτερο από 2:

- Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο (**κατάσταση 3**) της στοίβας.
- Το loc είναι μικρότερο του 0 οπότε ο κόμβος έχει εξερευνηθεί
  - Το curLink (4611686018427387904) που έχει αποδοθεί στην συγκεκριμένη κατάσταση είναι ίσο με το lowLink και οπότε οι καταστάσεις στην comStack από την κορυφή μέχρι την curState σχηματίζουν ένα SCC. Γίνεται έλεγχος για "bad" κύκλους. Για το τρέχον pem (PossibleErrorModel), αυτή η μέθοδο ελέγχει αν το τρέχον scc ικανοποιεί τα AEs του. (Γνωρίζουμε ότι το τρέχον scc ικανοποιεί το EA του pem.) Αν ικανοποιούνται, αυτό το pem περιέχει ένα counterexample, και αυτή η μέθοδο τότε καλεί την printErrorTrace για να τυπώσει ένα σφάλμα και επιστρέφει false. Στην περίπτωση μας δεν ικανοποιείται.

    Συγκεκριμένα πραγματοποιείται η ακόλουθη διαδικασία:

    Εξάγεται η πρώτη κατάσταση της **comStack** και συγκρίνεται με την τρέχουσα κατάσταση αν είναι ίδιες αλλά και αν ο κόμβος <state, tidx> δεν παρουσιάζει stutterring έτσι ώστε να αποφασισθεί αν το component είναι trivial.



    Στην συγκεκριμένη περίπτωση παρουσιάζει stuttering οπότε δεν είναι trivial.

    Τοποθετούνται όλοι οι κόμβοι του component σε έναν hashtable. Ξεκινά ο έλεγχος αυτού του component. Το μήκος αυτού του hashtable είναι 128. Ελέγχει ένα ένα τα στοιχεία και αν δεν είναι null τότε εξάγει τον κόμβο και ελέγχει τα AEState (στην περίπτωση μας είναι μηδενικό το πλήθος αυτών) και τα AEAction για κάθε επόμενο κόμβο που δείχνει ο κόμβος και ο οποίος υπάρχει μέσα στο hashtable. Ελέγχεται αν εκπληρώνεται μέσω των promises (ο έλεγχος δεν μπαίνει σε αυτό το βήμα μιας και δεν έχουμε promises είναι για την περίπτωση ύπαρξης tableau). Αν ικανοποιούνται όλες οι συνθήκες τότε έχουμε ένα counterexample. Σε εμάς δεν ικανοποιούνται οπότε δεν έχουμε ένα counterexample.

  - Εξάγεται το plowLink (που είναι ίσο με DiskGraph.*MAX_PTR)* από την **dfsStack** και συγκρίνεται με το lowLink του συγκεκριμένου κόμβου για να ελεγχθεί αν το τελευταίο

είναι μικρότερο του. Αυτό δεν ισχύει γιατί είναι ίσα και δεν κάνει το plowLink ίσο με το lowLink. Και έτσι, απλά επανατοποθετεί το plowLink ως είχε στην **dfsStack.**

Τρέχουσα κατάσταση dfsStack

dfsStack	MemIntStack (id=46)
elems	(id=55)
[0...99]	
[0]	0
[1]	1073741824

6. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 2) δεν είναι μεγαλύτερο από 2 οπότε βγαίνει από αυτό το loop.

4η Επανάληψη

Τρέχουσα κατάσταση nodeQueue

nodeQueue	MemIntQueue (id=76)
diskdir	"states\\10-07-22-21-50-03" (id=54)
elems	(id=80)
[0...99]	
[0]	-159870001
[1]	-1504394061
[2]	-1
[3]	0
[4]	0
[5]	-1656790009
[6]	1526719459
[7]	-1
[8]	0
[9]	36
[10]	-1140995776
[11]	-244908333
[12]	-1
[13]	0
[14]	72
[15]	-1656790009
[16]	1526719459
[17]	-1
[18]	0
[19]	36
[20]	-1140995776
[21]	-244908333
[22]	-1
[23]	0
[24]	72

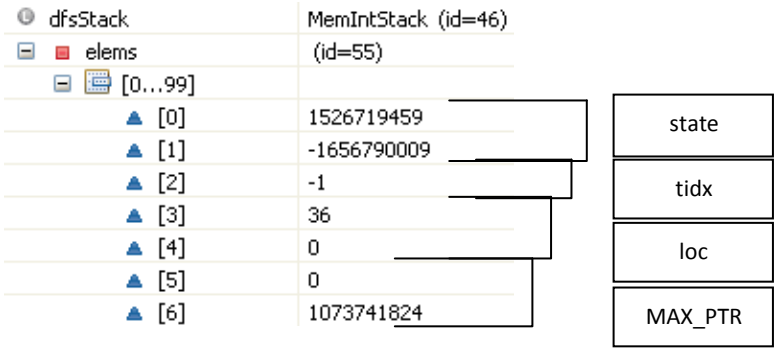
Η ουρά nodeQueue έχει ακόμη στοιχεία οπότε επαναλαμβάνεται η παραπάνω διαδικασία. (Τρέχον μήκος ουράς 10)

1. Αφού αφαιρέσει μια κατάσταση από την ουρά, ξεκινά τον υπολογισμό των SCCs με την κατάσταση αυτή (**κατάσταση 2**) ως τον root κόμβο. Τα στοιχεία για την πρώτη κατάσταση είναι:

state: -7115858903467826205, **tidx**: -1, **loc(το ptr)**: 36

2. Τοποθετεί τα τρία στοιχεία της κατάστασης καθώς και την τιμή DiskGraph.*MAX_PTR* (long) σε μια στοίβα **dfsStack**.

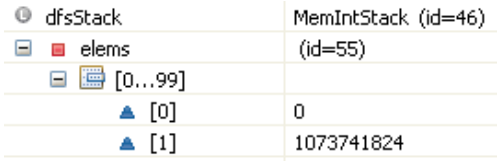
Τρέχουσα κατάσταση dfsStack



3. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 7) είναι μεγαλύτερο από 2:

- Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο (**κατάσταση 2**) της στοίβας.
- Το loc είναι μεγαλύτερο από ίσο από 0 οπότε ο κόμβος δεν έχει εξερευνηθεί
  - **Στο συγκεκριμένο κόμβο όμως έχει ανατεθεί ήδη ένα link** , δεν κάνει τίποτα από το να επιστρέψει απλά τον υπάρχον link.
  - ελέγχει αν αυτό το link που έχει επιστραφεί είναι μικρότερο του lowlink. Το τελευταίο δεν ισχύει οπότε δεν το αναθέτει στο lowlink. Τέλος τοποθετεί το lowlink στην dfsStack.

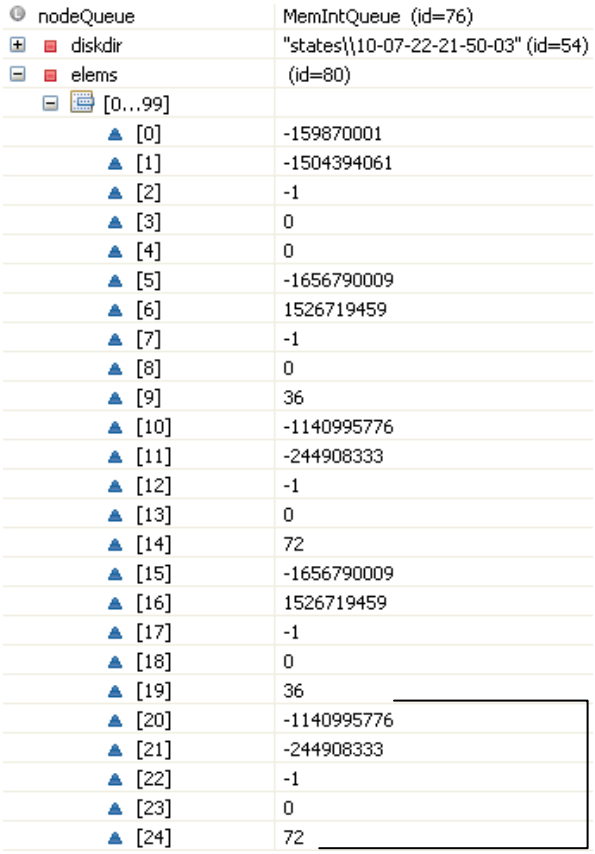
Τρέχουσα κατάσταση dfsStack



4. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 2) δεν είναι μεγαλύτερο από 2 οπότε βγαίνει από αυτό το loop.

5η Επανάληψη

Τρέχουσα κατάσταση nodeQueue



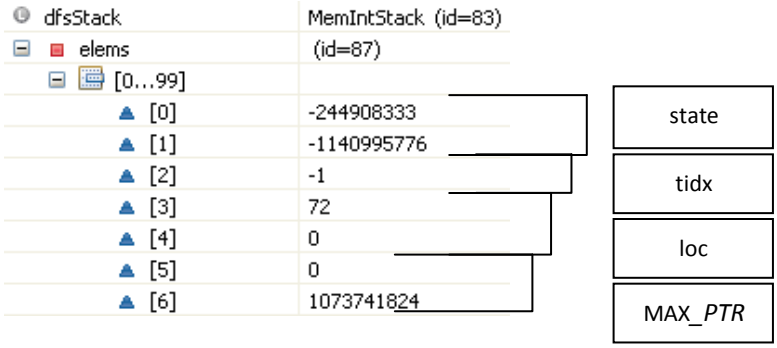
Η ουρά nodeQueue έχει ακόμη στοιχεία (μεγαλύτερη του 0) οπότε επαναλαμβάνεται ακριβώς η παραπάνω διαδικασία.(Τρέχον μήκος ουράς 5)

1. Αφού αφαιρέσει μια κατάσταση από την ουρά, ξεκινά τον υπολογισμό των SCCs με την κατάσταση αυτή (**κατάσταση 3**) ως τον root κόμβο. Τα στοιχεία για την πρώτη κατάσταση είναι:

   state: -4900539538744082733, **tidx**: -1, **loc(το ptr)**: 72

2. Τοποθετεί τα τρία στοιχεία της κατάστασης καθώς και την τιμή DiskGraph.*MAX_PTR* (long) σε μια στοίβα **dfsStack**.

Τρέχουσα κατάσταση dfsStack



3. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 7) είναι μεγαλύτερο από 2:

   - Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο (**κατάσταση 3**) της στοίβας.

- Το loc είναι μεγαλύτερο από ίσο από 0 οπότε ο κόμβος δεν έχει εξερευνηθεί
  - **Στο συγκεκριμένο κόμβο όμως έχει ανατεθεί ήδη ένα link** , δεν κάνει τίποτα από το να επιστρέψει απλά τον υπάρχον link.
  - ελέγχει αν αυτό το link που έχει επιστραφεί είναι μικρότερο του lowlink. Το τελευταίο δεν ισχύει οπότε δεν το αναθέτει στο lowlink. Τέλος τοποθετεί το lowlink στην dfsStack.

Τρέχουσα κατάσταση dfsStack

dfsStack	MemIntStack (id=46)
elems	(id=55)
[0...99]	
[0]	0
[1]	1073741824

4. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 2) δεν είναι μεγαλύτερο από 2 οπότε βγαίνει από αυτό το loop.

Έπειτα από όλες τις παραπάνω επαναλήψεις το μήκος της ουράς nodeQueue είναι 0 και οπότε βγαίνει από το loop.

2ο PossibleErrorModel

0) LNNeg LNStateEnabled [HCnxt]_hr\/ LNAction([HCnxt]_hr) **(AEAction)**

2) LNNeg LNStateAST (hr=2) **(EAAction)**

Δημιουργείται ένας in-memory node-pointer πίνακας από το node-pointer αρχείο που αντιστοιχεί στον behavior γράφο (όπου είναι ένα BufferedRandomAccessFile το οποίο μοιάζει με ένα RandomAccessFile, αλλά αυτό χρησιμοποιεί έναν private buffer έτσι ώστε οι περισσότερες λειτουργίες να μην απαιτούν πρόσβαση στον δίσκο).

Αρχικοποιείται η ουρά nodeQueue από Integers και Longs στην μνήμη με τις αρχικούς κόμβους του γράφου που είναι 6. Οι αρχικοί κόμβοι είναι :

initNodes	LongVec (id=57)
elementCount	6
elementData	(id=66)
[0]	-686636423115914061
[1]	-1
[2]	-7115858903467826205
[3]	-1
[4]	-4900539538744082733
[5]	-1
[6]	0
[7]	0

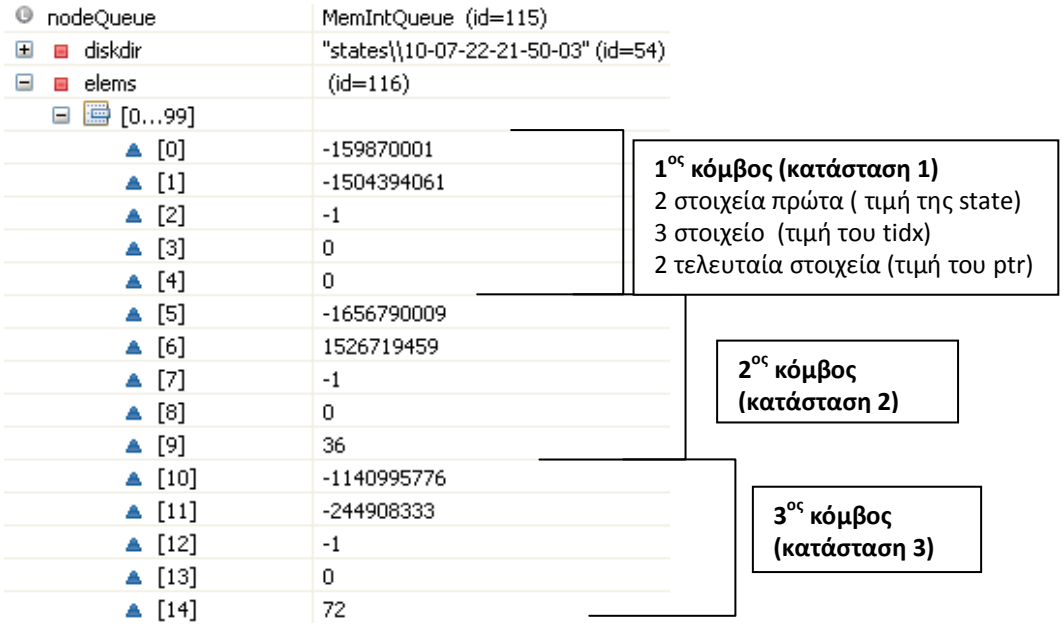
Στην πραγματικότητα λαμβάνονται στην πράξη τρεις κόμβοι μιας και οι υπόλοιπες τρεις καταστάσεις /κόμβοι είναι οι ίδιες καταστάσεις με τις προηγούμενες που προέκυψαν κατά την διαδικασία εύρεσης των

successor καταστάσεων. Στον πίνακα στην συνέχεια φαίνεται η αντιστοίχιση καθεμιάς κατάστασης στο αντίστοιχο fingerprint της.

State	fp (fingerprint)
1	-686636423115914061
2	-7115858903467826205
3	-4900539538744082733

Συγκεκριμένα τοποθετούνται για κάθε κατάσταση: η τρέχουσα κατάσταση *state* (long)**, το αναγνωριστικό της επόμενης *tidx*** (int)(Ο οποίος έχει νόημα όταν χρησιμοποιείται tableau στην διαδικασία του liveness model checking. Η τιμή -1 σημαίνει ότι δεν υπάρχει tableau) **και το link που έχει αποδοθεί στην συγκεκριμένη κατάσταση *ptr***(long)**.**

Τρέχουσα κατάσταση nodeQueue



1η Επανάληψη

Συγκεκριμένα όσο αυτή η ουρά nodeQueue έχει στοιχεία . Αρχικό μήκος 15 (πέντε στοιχεία για κάθε κόμβο) έχει μετά την αρχικοποίηση.

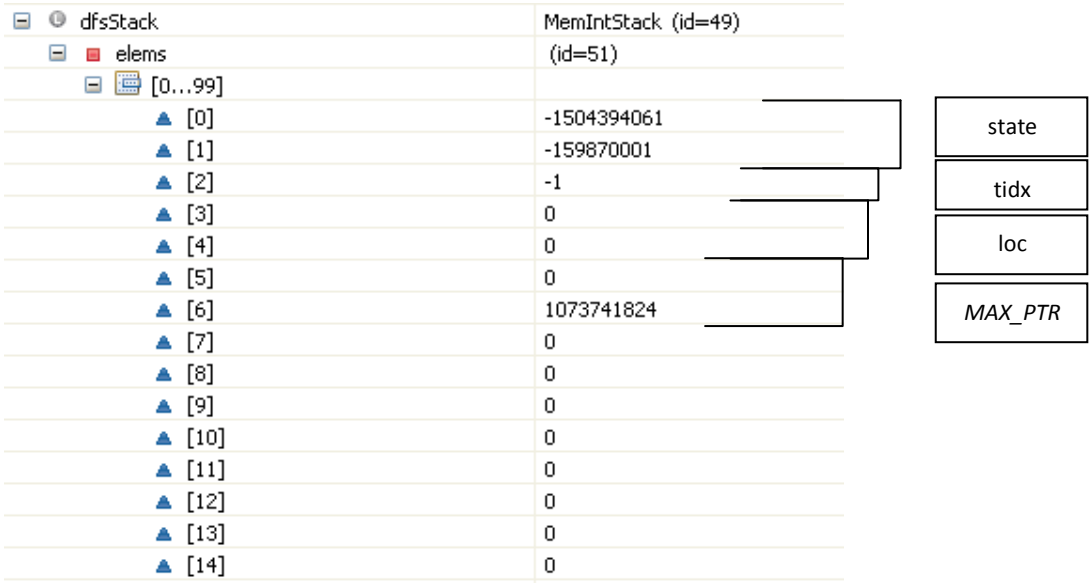
1. Αφού αφαιρέσει μια κατάσταση από την ουρά, ξεκινά τον υπολογισμό των SCCs με την κατάσταση αυτή με <state, tidx> ως τον root κόμβο. Τα στοιχεία για την πρώτη κατάσταση (**κατάσταση 1**) είναι:

   state: -686636423115914061, **tidx**: -1, **loc(το ptr)**:0

2. Τοποθετεί τα τρία στοιχεία της κατάστασης καθώς και την τιμή DiskGraph.*MAX_PTR* (long)(Το μέγιστο μήκος του αρχείου fileForNodes του γράφου.) σε μια στοίβα **dfsStack** (την οποία κάθε φορά κάνει reset για κάθε κατάσταση της ουράς nodeQueue).

3. Αρχικοποιεί μια μεταβλητή **newLink** να είναι ίση με την τιμή του *MAX_PTR*.

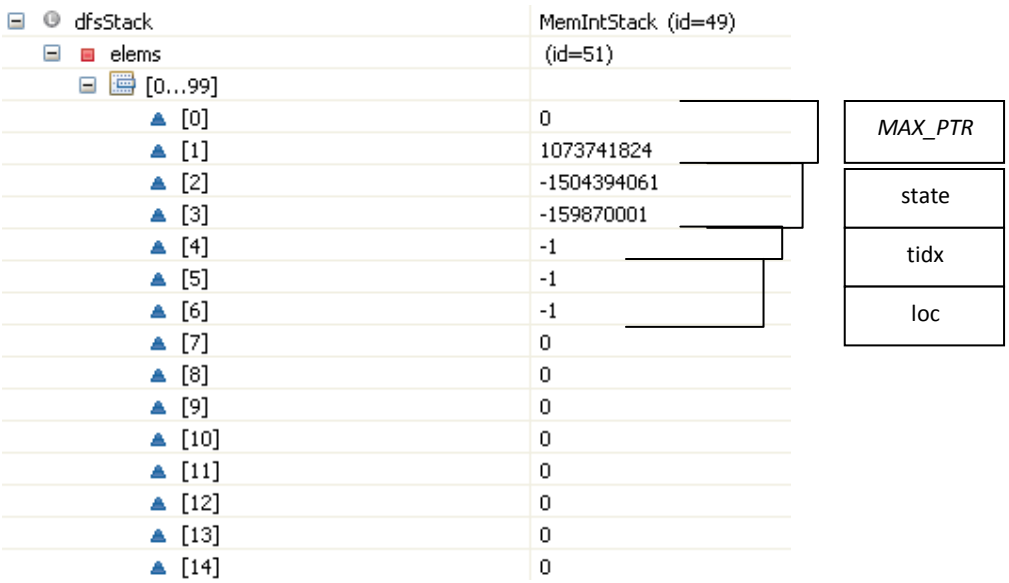
Τρέχουσα κατάσταση dfsStack



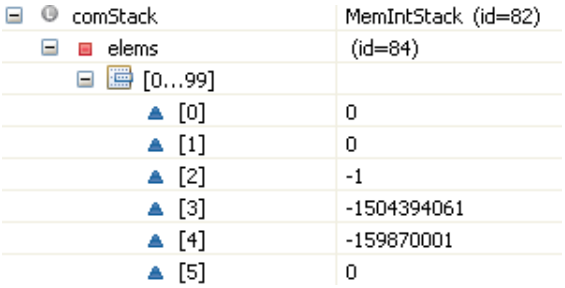
4. Το πλήθος των στοιχειών της στοίβας **dfsStack** είναι μεγαλύτερο από 2 (στην συγκεκριμένη περίπτωση είναι 7 όπως φαίνεται από την παραπάνω εικόνα):

- Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο της στοίβας.
- Το loc ελέγχεται αν είναι μικρότερο του μηδέν έτσι ώστε να συμπεράνει αν ο κόμβος έχει εξερευνηθεί. Στην περίπτωση μας δεν είναι μικρότερο του 0 οπότε ο κόμβος δεν έχει εξερευνηθεί
  - Στο συγκεκριμένο κόμβο δεν έχει ανατεθεί ένα link και έτσι του ανατίθεται ένα link.
  - Επανατοποθετεί την τρέχουσα κατάσταση(κόμβο) στην **dfsStack**, αλλά κάνει τη κατάσταση explored κάνοντας το loc ίσο με -1.

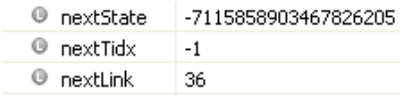
Τρέχουσα κατάσταση dfsStack



Επίσης τοποθετεί την τρέχουσα κατάσταση σε μια άλλη στοίβα την **comStack** που περιέχει τις καταστάσεις αυτές που έχουν εξερευνηθεί.(χωρίς να κάνει το loc ίσο με -1.

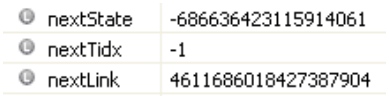


- Αρχικοποιεί μια μεταβλητή nextLowLink ίση με την newLink και αυξάνει την τελευταία κατά ένα.
- Κοιτάζει όλες τις successors καταστάσεις αυτής της κατάστασης (που είναι δύο). Για την πρώτη:
  - Λαμβάνει τα στοιχεία που χαρακτηρίζουν την συγκεκριμένη successor κατάσταση (**κατάσταση 2**) . (**StateFP**, **tidx** και το **link** που έχει ανατεθεί στην συγκεκριμένη κατάσταση/κόμβο)



  - Το nextlink είναι μεγαλύτερο του μηδενός(δηλαδή δεν έχει εξερευνηθεί) οπότε ελέγχει τα EAAction
    - Όπου επιστρέφεται **true** και επειδή το nextlink της successor κατάστασης είναι *FilePointer* (*είναι μικρότερο του MAX_PTR = 4611686018427387904*) τοποθετεί την successor κατάσταση στην **dfsStack**.

Η ίδια διαδικασία επαναλαμβάνεται και για την δεύτερη κατάσταση (βλέπουμε από την τιμή της nextState ότι πρόκειται για την ίδια την υπό εξέταση κατάσταση, **την κατάσταση 1**).

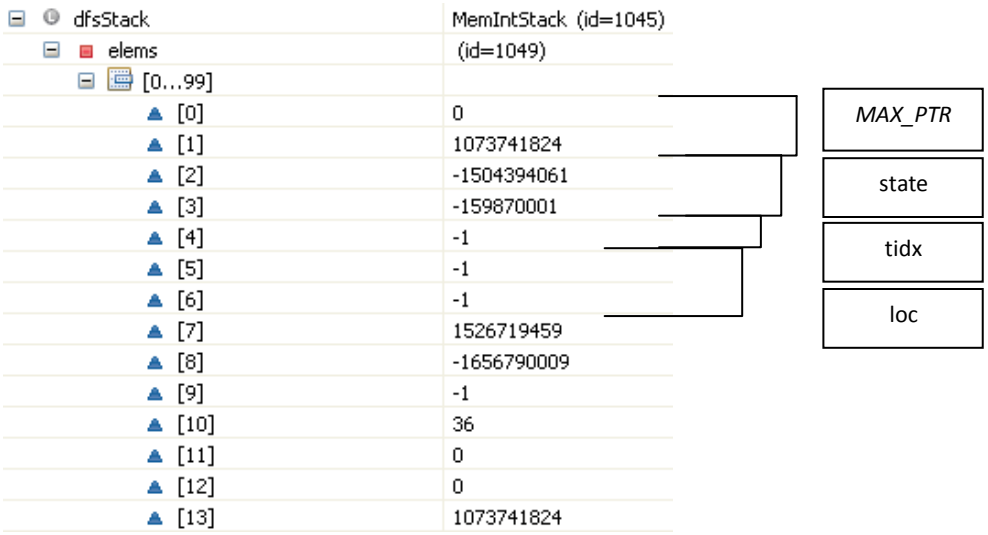


Οπού και πάλι επιστρέφεται **true** από τον έλεγχο των EAAction

- το nextLink δεν είναι *FilePointer* (*δεν είναι μικρότερο του MAX_PTR = 4611686018427387904*) και έτσι απλά ελέγχει αν το nextLink είναι μικρότερο του nextLowLink (που αρχικά είναι ίσο με DiskGraph.MAX_PTR) όπου επίσης δεν ισχύει γιατί είναι ίσα και δεν κάνει το nextLowLink ίσο με το nextLink.

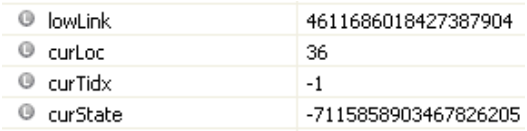
- Τοποθετεί την τιμή της nextLowLink στην dfsStack.

Τρέχουσα κατάσταση dfsStack



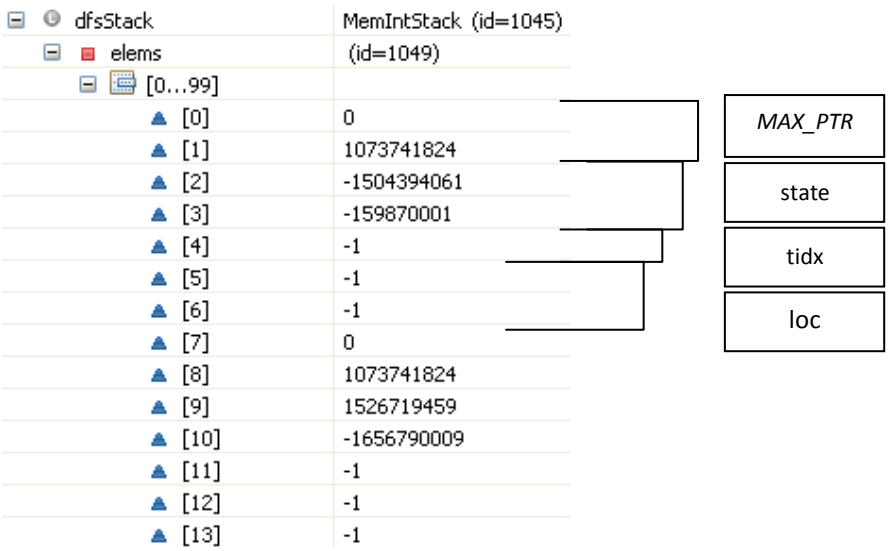
5. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 14) είναι μεγαλύτερο από 2:

- Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο (πρόκειται για την **κατάσταση 2**) της στοίβας.

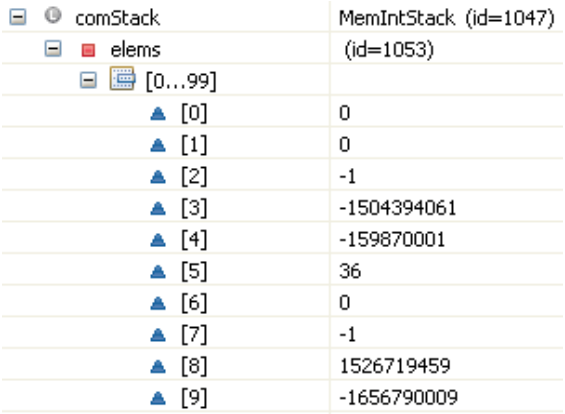


- Το curloc δεν είναι μικρότερο του 0 οπότε ο κόμβος δεν έχει εξερευνηθεί
  - Στο συγκεκριμένο κόμβο δεν έχει ανατεθεί ένα link και έτσι του ανατίθεται ένα link.
  - Επανατοποθετεί την τρέχουσα κατάσταση(κόμβο) στην **dfsStack**, αλλά κάνει τη κατάσταση explored κάνοντας το loc ίσο με -1.

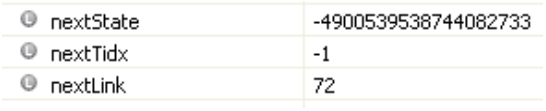
Τρέχουσα κατάσταση dfsStack



Επίσης τοποθετεί την τρέχουσα κατάσταση σε μια άλλη στοίβα την **comStack** που περιέχει τις καταστάσεις αυτές που έχουν εξερευνηθεί.(χωρίς να κάνει το loc ίσο με -1.

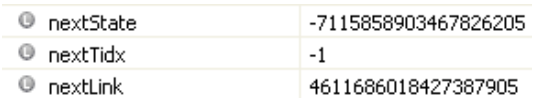


- Αρχικοποιεί μια μεταβλητή nextLowLink ίση με την newLink και αυξάνει την τελευταία κατά ένα.
- Κοιτάζει όλες τις successors καταστάσεις αυτής της κατάστασης (που είναι δύο). Για την πρώτη:
  - Λαμβάνει τα στοιχεία που χαρακτηρίζουν την συγκεκριμένη successor κατάσταση (**κατάσταση 3**) . (**StateFP**, **tidx** και το **link** που έχει ανατεθεί στην συγκεκριμένη κατάσταση/κόμβο)



  - Το nextlink είναι μεγαλύτερο του μηδενός οπότε ελέγχει τα EAAction
    - Όπου επιστρέφεται **false** και επειδή το nextlink της successor κατάστασης είναι *FilePointer* (*είναι μικρότερο του MAX_PTR = 4611686018427387904*) τοποθετεί την successor κατάσταση στην **nodeQueue**.

Η ίδια διαδικασία επαναλαμβάνεται και για την δεύτερη κατάσταση (βλέπουμε από την τιμή της nextState ότι πρόκειται για την ίδια την υπό εξέταση κατάσταση, **την κατάσταση 2**).

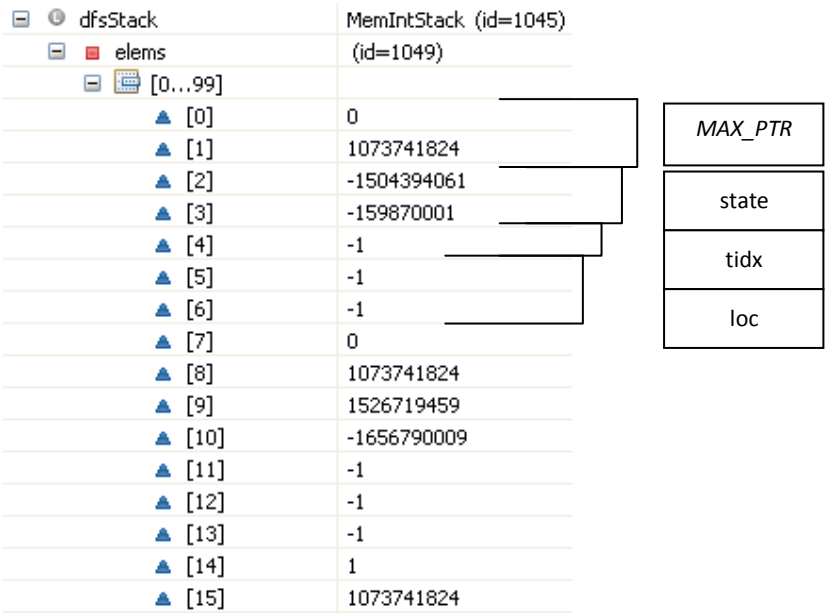


Οπού και πάλι επιστρέφεται **false** από τον έλεγχο των EAAction

- το nextlink δεν είναι *FilePointer* (*δεν είναι μικρότερο του MAX_PTR = 4611686018427387904*) και οπότε δεν τοποθετεί την successor κατάσταση στην **nodeQueue**.

- Τοποθετεί την τιμή της nextLowLink στην dfsStack.

Τρέχουσα κατάσταση dfsStack



6. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 16) είναι μεγαλύτερο από 2:

- Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο (**πρόκειται για την κατάσταση 2**) της στοίβας.(αυξήθηκε το lowlink κατά ένα)

lowLink	4611686018427387905
curLoc	-1
curTidx	-1
curState	-7115858903467826205

- Το curloc είναι μικρότερο του 0 οπότε ο κόμβος έχει εξερευνηθεί
- Λαμβάνει το link του τρέχοντα κόμβου (curLink= 4611686018427387905) και επειδή είναι ίσο με το lowLink
  - Οι καταστάσεις στην comStack από την κορυφή μέχρι την curState σχηματίζουν ένα SCC. Γίνεται έλεγχος για "bad" κύκλους. Για το τρέχον pem (PossibleErrorModel), αυτή η μέθοδο ελέγχει αν το τρέχον scc ικανοποιεί τα AEs του . (Γνωρίζουμε ότι το τρέχον scc ικανοποιεί το EA του pem.) Αν ικανοποιείται, αυτό το pem περιέχει ένα counterexample, και αυτή η μέθοδο τότε καλεί την printErrorTrace για να τυπώσει ένα σφάλμα και επιστρέφει false. Στην περίπτωση μας δεν ικανοποιείται.

    Συγκεκριμένα πραγματοποιείται η ακόλουθη διαδικασία:

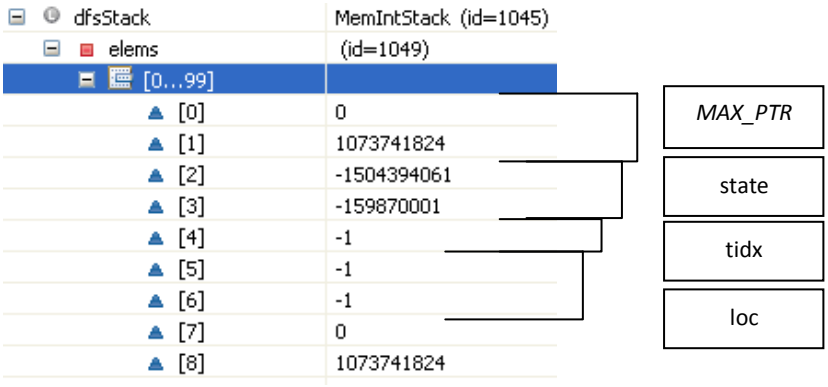
    Εξάγεται η πρώτη κατάσταση της comStack και συγκρίνεται με την τρέχουσα κατάσταση αν είναι ίδιες αλλά και αν ο κόμβος <state, tidx> δεν έχει stutterring έτσι ώστε να αποφασισθεί αν το component είναι trivial.

state1	-7115858903467826205
tidx1	-1
loc1	36

Στην συγκεκριμένη περίπτωση ισχύουν όλες οι συνθήκες οπότε είναι trivial. Και απλά κάνει το loc της συγκεκριμένης κατάστασης ίσο με MAX_LINK.

- Εξάγεται το plowLink (που είναι ίσο με DiskGraph.*MAX_PTR*) από την **dfsStack** και συγκρίνεται με το lowLink του συγκεκριμένου κόμβου για να ελεγχθεί αν το τελευταίο είναι μικρότερο του. Αυτό δεν ισχύει γιατί είναι ίσα και δεν κάνει το plowLink ίσο με το lowLink. Και έτσι, απλά επανατοποθετεί το plowLink ως είχε στην **dfsStack**.

Τρέχουσα κατάσταση dfsStack



7. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 9) είναι μεγαλύτερο από 2:

- Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο (**πρόκειται για την κατάσταση 1**) της στοίβας.(αυξήθηκε το lowlink κατά ένα)

- Το curloc είναι μικρότερο του 0 οπότε ο κόμβος έχει εξερευνηθεί

- Λαμβάνει το link του τρέχοντα κόμβου (curLink= 4611686018427387904) και επειδή είναι ίσο με το lowLink

  - Οι καταστάσεις στην comStack από την κορυφή μέχρι την curState σχηματίζουν ένα SCC. Γίνεται έλεγχος για "bad" κύκλους. Για το τρέχον pem (PossibleErrorModel), αυτή η μέθοδο ελέγχει αν το τρέχον scc ικανοποιεί τα AEs του . (Γνωρίζουμε ότι το τρέχον scc ικανοποιεί το EA του pem.) Αν ικανοποιείται, αυτό το pem περιέχει ένα counterexample, και αυτή η μέθοδο τότε καλεί την printErrorTrace για να τυπώσει ένα σφάλμα και επιστρέφει false. Στην περίπτωση μας δεν ικανοποιείται.

    Συγκεκριμένα πραγματοποιείται η ακόλουθη διαδικασία:

    Εξάγεται η πρώτη κατάσταση της comStack και συγκρίνεται με την τρέχουσα κατάσταση αν είναι ίδιες αλλά και αν ο κόμβος <state, tidx> δεν παρουσιάζει stutterring έτσι ώστε να αποφασισθεί αν το component είναι trivial.

state1	-686636423115914061
tidx1	-1
loc1	0

Στην συγκεκριμένη περίπτωση παρουσιάζει stuttering οπότε δεν είναι trivial.

- Τοποθετούνται όλοι οι κόμβοι του component σε έναν hashtable. Ξεκινά ο έλεγχος αυτού του component. Το μήκος αυτού του hashtable είναι 128. Ελέγχει ένα ένα τα στοιχεία και αν δεν είναι null τότε εξάγει τον κόμβο και ελέγχει τα AEState (στην περίπτωση μας είναι μηδενικό το πλήθος αυτών) και τα AEAction για κάθε επόμενο κόμβο που δείχνει ο κόμβος και ο οποίος υπάρχει μέσα στο hashtable. Ελέγχεται αν εκπληρώνεται μέσω των promises δεν μπαίνει ο έλεγχος σε αυτό το βήμα μιας και δεν έχουμε promises (είναι για την περίπτωση ύπαρξης tableau). Αν ικανοποιούνται όλες οι συνθήκες τότε έχουμε ένα counterexample. Σε εμάς δεν ικανοποιούνται οπότε δεν έχουμε ένα counterexample.

- Εξάγεται το plowLink (που είναι ίσο με DiskGraph.*MAX_PTR*) από την **dfsStack** και συγκρίνεται με το lowLink του συγκεκριμένου κόμβου για να ελεγχθεί αν το τελευταίο είναι μικρότερο του. Αυτό δεν ισχύει γιατί είναι ίσα και δεν κάνει το plowLink ίσο με το lowLink. Και έτσι, απλά επανατοποθετεί το plowLink ως είχε στην **dfsStack**.

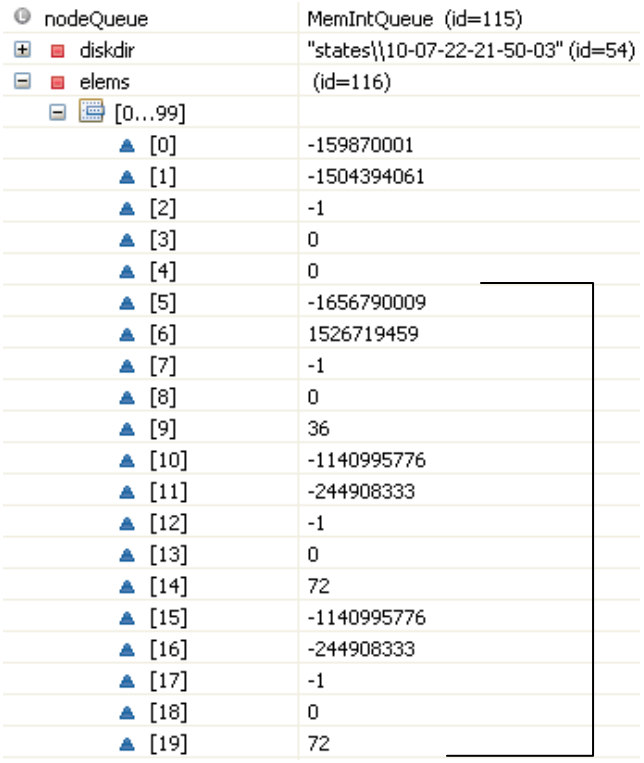
Τρέχουσα κατάσταση dfsStack

dfsStack	MemIntStack (id=46)
elems	(id=55)
[0...99]	
[0]	0
[1]	1073741824

8. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 2, περιέχει μόνο το nextLowLink) δεν είναι μεγαλύτερο από 2 οπότε βγαίνει από αυτό το loop.

2η Επανάληψη

Τρέχουσα κατάσταση nodeQueue



Η ουρά nodeQueue έχει ακόμη στοιχεία οπότε επαναλαμβάνεται η παραπάνω διαδικασία. (Τρέχον μήκος ουράς 15)

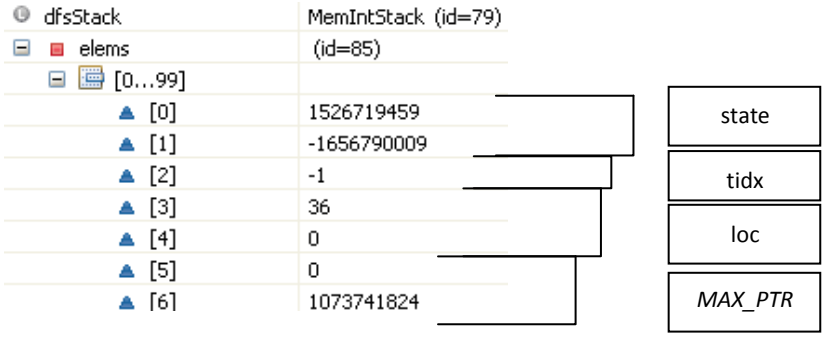
1. Αφού αφαιρέσει μια κατάσταση από την ουρά, ξεκινά τον υπολογισμό των SCCs με την κατάσταση αυτή (**κατάσταση 2**) ως τον root κόμβο. Τα στοιχεία για την πρώτη κατάσταση είναι:

   state: -7115858903467826205, **tidx**: -1, **loc(το ptr)**: 36

2. Τοποθετεί τα τρία στοιχεία της κατάστασης καθώς και την τιμή DiskGraph.*MAX_PTR* (long)(Το μέγιστο μήκος του αρχείου fileForNodes του γράφου.) σε μια στοίβα **dfsStack** (την οποία κάθε φορά κάνει reset για κάθε κατάσταση της ουράς nodeQueue).

3. Αρχικοποιεί μια μεταβλητή **newLink** να είναι ίση με την τιμή του *MAX_PTR*.

Τρέχουσα κατάσταση dfsStack



4. Το πλήθος των στοιχειών της στοίβας **dfsStack** είναι μεγαλύτερο από 2 (στην συγκεκριμένη περίπτωση είναι 7 όπως φαίνεται από την παραπάνω εικόνα):

   - Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο της στοίβας.
   - Το loc ελέγχεται αν είναι μικρότερο του μηδέν έτσι ώστε να συμπεράνει αν ο κόμβος έχει εξερευνηθεί. Στην περίπτωση μας δεν είναι μικρότερο του 0 οπότε ο κόμβος δεν έχει εξερευνηθεί
     - Στο συγκεκριμένο κόμβο έχει ανατεθεί ένα link.
     - Συγκρίνεται το link (=9223372036854775807) που επιστράφηκε με το lowlink (=4611686018427387904) αν είναι μικρότερο. Πράγμα το οποίο δεν ισχύει και απλά επανατοποθετείται η lowLink στην **dfsStack**.

Τρέχουσα κατάσταση dfsStack

dfsStack	MemIntStack (id=46)
elems	(id=55)
[0...99]	
[0]	0
[1]	1073741824

5. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 2, περιέχει μόνο το nextLowLink) δεν είναι μεγαλύτερο από 2 οπότε βγαίνει από αυτό το loop.

3η Επανάληψη

Τρέχουσα κατάσταση nodeQueue

nodeQueue	MemIntQueue (id=115)
diskdir	"states\\10-07-22-21-50-03" (id=54)
elems	(id=116)
[0...99]	
[0]	-159870001
[1]	-1504394061
[2]	-1
[3]	0
[4]	0
[5]	-1656790009
[6]	1526719459
[7]	-1
[8]	0
[9]	36
[10]	-1140995776
[11]	-244908333
[12]	-1
[13]	0
[14]	72
[15]	-1140995776
[16]	-244908333
[17]	-1
[18]	0
[19]	72

Η ουρά nodeQueue έχει ακόμη στοιχεία οπότε επαναλαμβάνεται η παραπάνω διαδικασία.(Τρέχον μήκος ουράς 10)

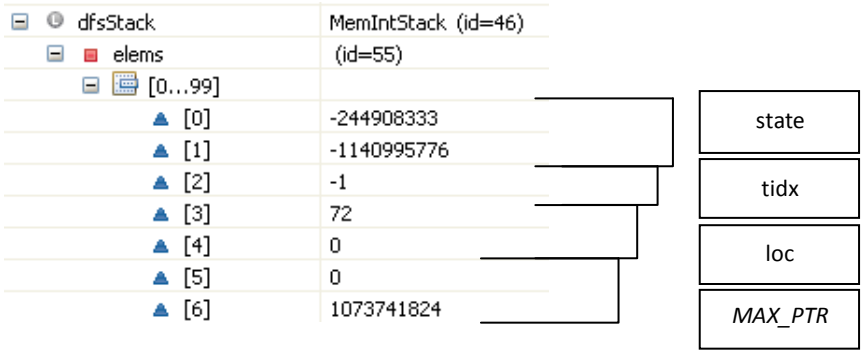
1. Αφού αφαιρέσει μια κατάσταση από την ουρά, ξεκινά τον υπολογισμό των SCCs με την κατάσταση αυτή (**κατάσταση 3**) ως τον root κόμβο. Τα στοιχεία για την πρώτη κατάσταση είναι:

   state: -4900539538744082733, **tidx**: -1, **loc(το ptr)**: 72

2. Τοποθετεί τα τρία στοιχεία της κατάστασης καθώς και την τιμή DiskGraph.*MAX_PTR* (long) σε μια στοίβα **dfsStack**.

3. Αρχικοποιεί μια μεταβλητή **newLink** να είναι ίση με την τιμή του *MAX_PTR*.

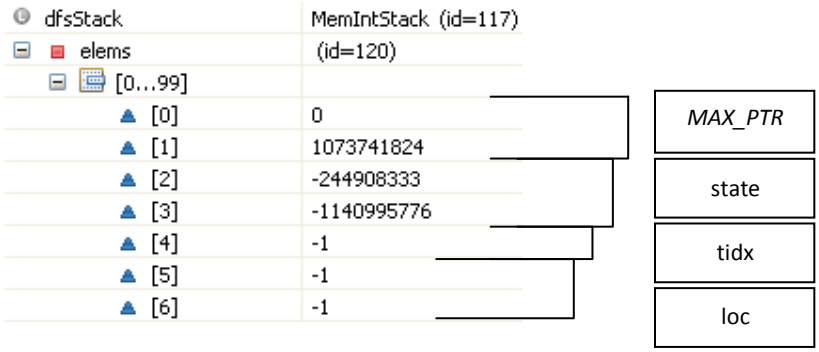
Τρέχουσα κατάσταση dfsStack



4. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 7) είναι μεγαλύτερο από 2:

   - Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο (**κατάσταση 3**) της στοίβας.
   - Το loc είναι μεγαλύτερο από 0 οπότε ο κόμβος δεν έχει εξερευνηθεί
     - Στο συγκεκριμένο κόμβο δεν έχει ανατεθεί ήδη ένα link και οπότε του ανατίθεται ένα link.
     - Επανατοποθετεί την τρέχουσα κατάσταση(κόμβο) στην **dfsStack**, αλλά κάνει τη κατάσταση explored κάνοντας το loc ίσο με -1.

Τρέχουσα κατάσταση dfsStack



Επίσης τοποθετεί την τρέχουσα κατάσταση σε μια άλλη στοίβα την **comStack** που περιέχει τις καταστάσεις αυτές που έχουν εξερευνηθεί.(χωρίς να κάνει το loc ίσο με -1.

comStack	MemIntStack (id=118)
elems	(id=121)
[0...99]	
[0]	72
[1]	0
[2]	-1
[3]	-244908333
[4]	-1140995776

- Αρχικοποιεί μια μεταβλητή nextLowLink ίση με την newLink και αυξάνει την τελευταία κατά ένα.
- Κοιτάζει όλες τις successors καταστάσεις αυτής της κατάστασης (που είναι δύο). Συγκεκριμένα για την πρώτη:
  - Λαμβάνει τα στοιχεία που χαρακτηρίζουν την συγκεκριμένη successor (**κατάσταση 1**). (**StateFP**, **tidx** και το **link**)

nextState	-686636423115914061
nextTidx	-1
nextLink	9223372036854775807

  - Το link είναι μεγαλύτερο του μηδενός και επομένως ελέγχει τα EAAction
    - Όπου επιστρέφεται **true** και για τον λόγο ότι το link δεν είναι *FilePointer* ελέγχεται αν το nextLink είναι μικρότερο από το nextLowLink γεγονός που δεν ισχύει και επομένως δεν γίνεται κάτι(δηλαδή η ανάθεση του nextLink στο nextLowLink).

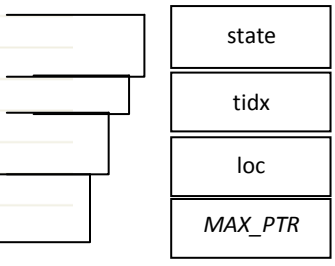
Η ίδια διαδικασία επαναλαμβάνεται και για την δεύτερη κατάσταση (βλέπουμε από την τιμή της nextState ότι **πρόκειται** για την ίδια την υπό εξέταση κατάσταση, **την κατάσταση 3**).

nextState	-4900539538744082733
nextTidx	-1
nextLink	4611686018427387904

- Τοποθετεί την nextLowLink στην dfsStack.

Τρέχουσα κατάσταση dfsStack

dfsStack	MemIntStack (id=46)
elems	(id=55)
[0...99]	
[0]	0
[1]	1073741824
[2]	-244908333
[3]	-1140995776
[4]	-1
[5]	-1
[6]	-1
[7]	0
[8]	1073741824



5. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 9) είναι μεγαλύτερο από 2:

- Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο (**κατάσταση 3**) της στοίβας.
- Το loc είναι μικρότερο του 0 οπότε ο κόμβος έχει εξερευνηθεί
- Λαμβάνει το link του τρέχοντα κόμβου (curLink= 4611686018427387904) και επειδή είναι ίσο με το lowLink
  - Οι καταστάσεις στην comStack από την κορυφή μέχρι την curState σχηματίζουν ένα SCC. Γίνεται έλεγχος για "bad" κύκλους. Για το τρέχον pem (PossibleErrorModel), αυτή η μέθοδο ελέγχει αν το τρέχον scc ικανοποιεί τα AEs του . (Γνωρίζουμε ότι το τρέχον scc ικανοποιεί το EA του pem.) Αν ικανοποιείται, αυτό το pem περιέχει ένα counterexample, και αυτή η μέθοδο τότε καλεί την printErrorTrace για να τυπώσει ένα σφάλμα και επιστρέφει false. Στην περίπτωση μας δεν ικανοποιείται.

    Συγκεκριμένα πραγματοποιείται η ακόλουθη διαδικασία:

    Εξάγεται η πρώτη κατάσταση της comStack και συγκρίνεται με την τρέχουσα κατάσταση αν είναι ίδιες αλλά και αν ο κόμβος <state, tidx> δεν παρουσιάζει stutterring έτσι ώστε να αποφασισθεί αν το component είναι trivial.

state1	-4900539538744082733
tidx1	-1
loc1	72

    Στην συγκεκριμένη περίπτωση παρουσιάζει stuttering οπότε δεν είναι trivial.

    Τοποθετούνται όλοι οι κόμβοι του component σε έναν hashtable. Ξεκινά ο έλεγχος αυτού του component. Το μήκος αυτού του hashtable είναι 128. Ελέγχει ένα ένα τα στοιχεία και αν δεν είναι null τότε εξάγει τον κόμβο και ελέγχει τα AEState (στην περίπτωση μας είναι μηδενικό το πλήθος αυτών) και τα AEAction για κάθε επόμενο κόμβο που δείχνει ο κόμβος και ο οποίος υπάρχει μέσα στο hashtable. Ελέγχεται αν εκπληρώνεται μέσω των promises δεν μπαίνει ο έλεγχος σε αυτό το βήμα μιας και δεν έχουμε promises (είναι για την περίπτωση ύπαρξης tableau). Αν ικανοποιούνται όλες οι συνθήκες τότε έχουμε ένα counterexample. Σε εμάς δεν ικανοποιούνται οπότε δεν έχουμε ένα counterexample.

- Εξάγεται το plowLink (που είναι ίσο με DiskGraph.*MAX_PTR*) από την **dfsStack** και συγκρίνεται με το lowLink του συγκεκριμένου κόμβου για να ελεγχθεί αν το τελευταίο είναι μικρότερο του. Αυτό δεν ισχύει γιατί είναι ίσα και δεν κάνει το plowLink ίσο με το lowLink. Και έτσι, απλά επανατοποθετεί το plowLink ως είχε στην **dfsStack**.

Τρέχουσα κατάσταση dfsStack

dfsStack	MemIntStack (id=46)
elems	(id=55)
[0...99]	
[0]	0
[1]	1073741824

6. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 2) δεν είναι μεγαλύτερο από 2 οπότε βγαίνει από αυτό το loop.

4η Επανάληψη

Τρέχουσα κατάσταση nodeQueue

nodeQueue	MemIntQueue (id=115)
diskdir	"states\\10-07-22-21-50-03" (id=54)
elems	(id=116)
[0...99]	
[0]	-159870001
[1]	-1504394061
[2]	-1
[3]	0
[4]	0
[5]	-1656790009
[6]	1526719459
[7]	-1
[8]	0
[9]	36
[10]	-1140995776
[11]	-244908333
[12]	-1
[13]	0
[14]	72
[15]	-1140995776
[16]	-244908333
[17]	-1
[18]	0
[19]	72

Η ουρά nodeQueue έχει ακόμη στοιχεία οπότε επαναλαμβάνεται η παραπάνω διαδικασία. (Τρέχον μήκος ουράς 5)

1. Αφού αφαιρέσει μια κατάσταση από την ουρά, ξεκινά τον υπολογισμό των SCCs με την κατάσταση αυτή (**κατάσταση 2**) ως τον root κόμβο. Τα στοιχεία για την πρώτη κατάσταση είναι:

   state: -4900539538744082733, **tidx**: -1, **loc(το ptr)**: 72

2. Τοποθετεί τα τρία στοιχεία της κατάστασης καθώς και την τιμή DiskGraph.*MAX_PTR* (long) σε μια στοίβα **dfsStack**.

3. Αρχικοποιεί μια μεταβλητή **newLink** να είναι ίση με την τιμή του *MAX_PTR*.

Τρέχουσα κατάσταση dfsStack

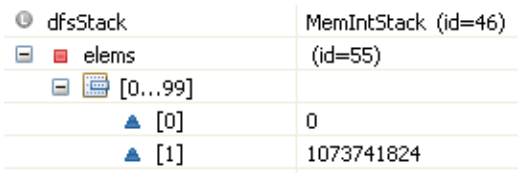
dfsStack	MemIntStack (id=79)
elems	(id=85)
[0...99]	
[0]	-244908333
[1]	-1140995776
[2]	-1
[3]	72
[4]	0
[5]	0
[6]	1073741824

state
tidx
loc
MAX_PTR

4. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 7) είναι μεγαλύτερο από 2:

   - Εξάγει τα στοιχεία που αντιστοιχούν στον πρώτο κόμβο της στοίβας.
   - Το loc είναι μεγαλύτερο από ίσο από 0 οπότε ο κόμβος δεν έχει εξερευνηθεί
     - **Στο συγκεκριμένο κόμβο όμως έχει ανατεθεί ήδη ένα link** , δεν κάνει τίποτα από το να επιστρέψει απλά τον υπάρχον link.
     - ελέγχει αν αυτό το link που έχει επιστραφεί είναι μικρότερο του lowlink. Το τελευταίο δεν ισχύει οπότε δεν το αναθέτει στο lowlink. Τέλος τοποθετεί το lowlink στην dfsStack.

Τρέχουσα κατάσταση dfsStack



5. Το πλήθος των στοιχειών της στοίβας **dfsStack** (στην συγκεκριμένη περίπτωση είναι 2) δεν είναι μεγαλύτερο από 2 οπότε βγαίνει από αυτό το loop.

Έπειτα από όλες τις παραπάνω επαναλήψεις το μήκος της ουράς nodeQueue είναι 0 και οπότε βγαίνει από το loop.

Το τρίτο και τέταρτο PossibleErrorModels είναι ακριβώς τα ίδια με το δεύτερο και επομένως ακολουθείται η ακριβώς διαδικασία με το δεύτερο PossibleErrorModel.

Μετά από την ολοκλήρωση της όλης διαδικασίας δεν επαληθεύεται κάποιο counterexample και τελικά εμφανίζεται :

```
6 states generated, 3 distinct states found, 0 states left on queue.
```

Παράρτημα Β: Επεξήγηση Βασικών Εννοιών

Β.1 Σημασιολογικό Tableau

Η διαδικασία απόδειξης του σημασιολογικού tableau[3], όπως περιγράφηκε από τον Smullyan , είναι ένα πλήρως αυτοματοποιημένο σύστημα απόδειξης διάψευσης. Εν ολίγοις, μια απόδειξη μιας έκφρασης X αποτελείται από την παρουσίαση ότι η άρνηση της X δεν μπορεί να ικανοποιηθεί στο model. Ένα tableau είναι μια μορφή δυαδικού δέντρου με μια έκφραση σε κάθε κόμβο.

Η Μέθοδος Tableau

Η μέθοδο tableau βασίζεται στην ιδέα ότι μια έκφραση είναι έγκυρη αν και μόνο αν η άρνηση της δεν ικανοποιείται.

Ορισμός : Ένα κλάδος(branch), ή μονοπάτι (path), είναι κλειστός αν και ένα λεκτικό(literal) και η άρνηση του συμβαίνουν στον κλάδο, και σε διαφορετική περίπτωση είναι ανοικτός. Ένας κλειστός κλάδος σηματοδοτεί ότι μια αντίκρουση έχει συμβεί.

Ορισμός : Ένα tableau είναι κλειστό αν και μόνο αν όλοι οι κλάδοι του είναι κλειστοί και σε διαφορετική περίπτωση είναι ανοιχτό.

Μια απόδειξη tableau μιας έκφρασης X είναι απλά ένα κλειστό tableau της άρνησης της X. Αν ένα tableau περιέχει ένα ανοιχτό κλάδο τότε ένα counter-example μπορεί να κατασκευαστεί, το οποίο είναι μια αποτίμηση που ικανοποιεί την ¬X.

Τα tableau συνδέονται με αυτό που είναι γνωστό σαν διαζευκτική κανονική μορφή (disjunctive normal format,DNF). Η DNF είναι μια διάζευξη από συζεύξεις.

Το ⊤ υποδηλώνει το true και το ⊥ υποδηλώνει το false. Μια ατομική έκφραση είναι ένα προτασιακό γράμμα, η άρνηση της, ⊤, ή ⊥.

X = B1 ∧ B2 ∧ B3 ∧ ... ∧ Bn.

Bi = A1 ∨ A2 ∨ A3 ∨ ... ∨ Am

Ai είναι μια ατομική έκφραση ή άρνηση της

Σχήμα: Disjunctive Normal Form

Συστατικά εκφράσεων

Εκφράσεις συζευκτικού τύπου αποκαλούνται α-εκφράσεις και διαζευκτικού τύπου εκφράσεις αποκαλούνται β-εκφράσεις. Μια α- έκφραση εκτιμάται σε true αν και μόνο αν τα συστατικά της, α1 και α2, και τα δύο εκτιμώνται σε true. Ομοίως, β-έκφραση εκτιμάται σε true αν και μόνο αν τουλάχιστον ένα από τα δύο συστατικά της , β1 ή β2, εκτιμώνται σε true.

α	α_1	α_2
$X \wedge Y$	X	Y
$\neg\ (X \vee Y)$	$\neg X$	$\neg Y$
$\neg (X \rightarrow Y)$	X	$\neg Y$
β	β_1	β_2
$X \vee Y$	X	Y
$\neg\ (X \wedge Y)$	$\neg X$	$\neg Y$
$X \rightarrow Y$	$\neg X$	Y

Σχήμα: Συστατικά των α και β εκφράσεων

Restricted TLA εκφράσεις που δεν έχουν την παραπάνω μορφή θα αναφέρονται ως R-εκφράσεις. Οι R-εκφράσεις πρέπει να εκτιμώνται όσον αφορά ένα μονοπάτι π έτσι προτάσσουμε την έκφραση με την κατάσταση, για παράδειγμα η $\Box\phi$ εκτιμάται στην αρχική κατάσταση π_0 σαν $\pi_0(\Box\phi)$ και ένα state κατηγόρημα P εκτιμάται σαν $\pi_0(P)$.
Οι R-εκφράσεις μπορούν επιπλέον να ταξινομηθούν ως R_α-εκφράσεις, οι οποίες δρουν συζευκτικά και R_β- εκφράσεις οι οποίες δρουν διαζευκτικά. Τα συστατικά από το καθένα παρουσιάζονται στο ακόλουθο σχήμα. Οι R_α- εκφράσεις εκτιμώνται σε true αν και μόνο αν και τα δυο από τα συστατικά της εκτιμώνται σε true. Οι R_β-εκφράσεις εκτιμώνται σε true αν και μόνο αν τουλάχιστον ένα από τα συστατικά της εκτιμάται σε true.

R_α	$R_{\alpha 1}$	$R_{\alpha 2}$
$\pi_i(\Box\phi)$	$\pi_i(\phi)$	$\pi_{i+1}(\Box\phi)$
$\pi_i(\Box[A]_f)$	$(\pi_i, \pi_{i+1})(A \vee (f' = f))$	$\pi_{i+1}(\Box[A]_f)$
$\pi_i(\neg\Diamond\phi)$	$\pi_i(\neg\phi)$	$\pi_{i+1}(\neg\Diamond\phi)$

R_β	$R_{\alpha 1}$	$R_{\alpha 2}$
$\pi_i(\Diamond\phi)$	$\pi_i(\phi)$	$\pi_{i+1}(\Diamond\phi)$
$\pi_i(\neg\Box[A]_f)$	$(\pi_i, \pi_{i+1})(\neg A \wedge \neg(f' = f))$	$\pi_{i+1}(\neg\Box[A]_f)$
$\pi_i(\neg\Box\phi)$	$\pi_i(\neg\phi)$	$\pi_{i+1}(\neg\Box\phi)$

Σχήμα: Συστατικά των R_α και R_β εκφράσεων

Κανόνες επέκτασης (Expansion Rules)
Οι tableau κανόνες επέκτασης ορίζονται ως ακολούθως:
Αν το $\neg\neg Z$ λαμβάνει χώρα σε ένα κλάδο αντικατέστησε το με Z.
Αν το $\neg\top$ λαμβάνει χώρα σε ένα κλάδο αντικατέστησε το με $\bot$.
Αν το $\neg\bot$ λαμβάνει χώρα σε ένα κλάδο αντικατέστησε το με $\top$.
Αν μια α-έκφραση λαμβάνει χώρα σε ένα κλάδο αντικατέστησε την με α_1 ακολουθούμενο από α_2 στον ίδιο κλάδο.
Αν μια β-έκφραση λαμβάνει χώρα σε ένα κλάδο χώρισε τον ένα κλάδο σε δύο, έναν με β_1 και ένα με β_2.
Αν μια R_α-έκφραση λαμβάνει χώρα σε ένα κλάδο αντικατέστησε την με $R_{\alpha 1}$ ακολουθημένη από $R_{\alpha 2}$ στον ίδιο κλάδο.
Αν μια R_β έκφραση λαμβάνει χώρα σε ένα κλάδο χώρισε τον κλάδο σε δυο, ένα με $R_{\beta 1}$ και ένα με $R_{\beta 2}$

$$\frac{\neg\neg Z}{Z} \quad \frac{\neg\top}{\bot} \quad \frac{\neg\bot}{\top} \quad \frac{\alpha}{\alpha_1 \\ \alpha_2} \quad \frac{\beta}{\beta_1 \mid \beta_2} \quad \frac{R_\alpha}{R_{\alpha 1} \\ R_{\alpha 2}} \quad \frac{R_\beta}{R_{\beta 1} \mid R_{\beta 2}}$$

Σχήμα: Tableau Κανόνες επέκτασης (Expansion Rules)

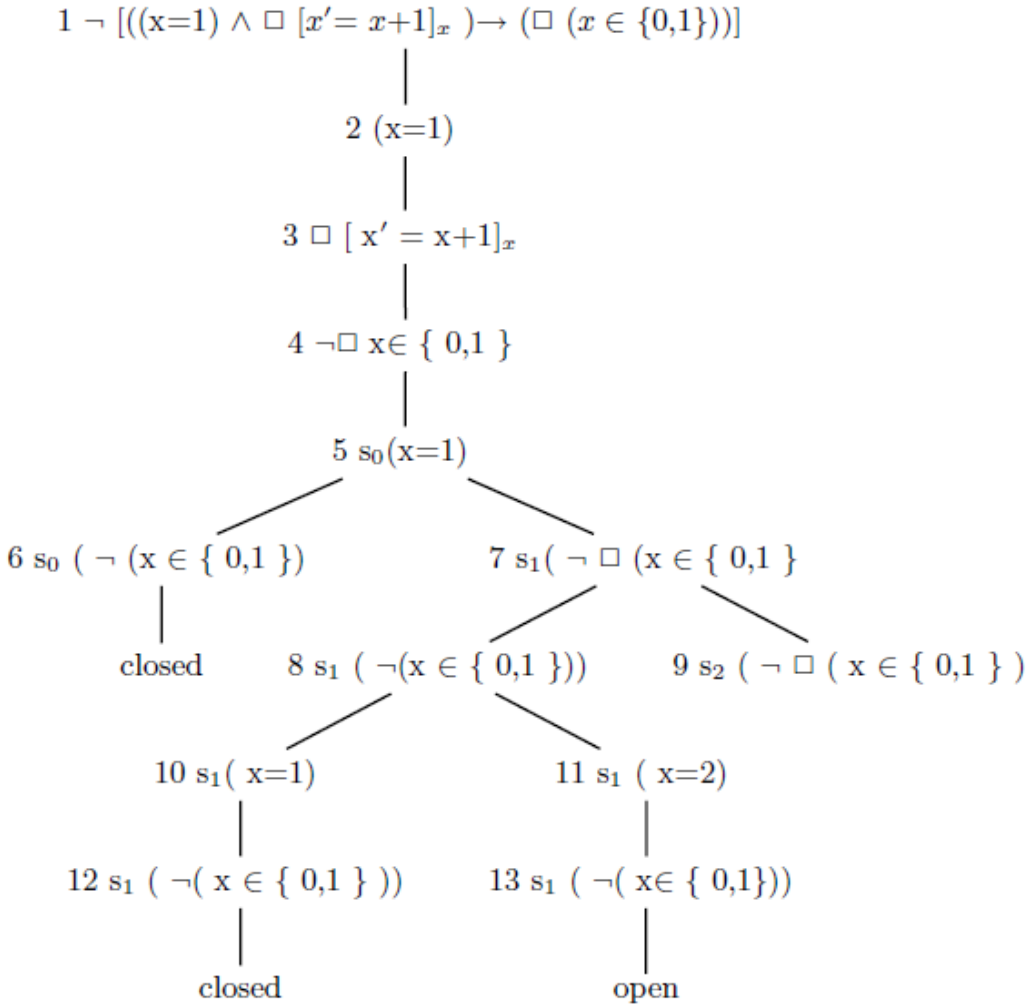
Ερμηνεία
Μια ερμηνεία μιας έκφρασης Χ είναι μια ανάθεση τιμών σε όλες τις μεταβλητές οι οποίες λαμβάνουν χώρα μέσα στην Χ. Μια έκφραση είναι έγκυρη αν κάθε ερμηνεία αυτής εκτιμάται σε true.

Παράδειγμα Tableau
Μελετάμε μια έκφραση της μορφής που τυπικά συναντάται χρησιμοποιώντας την TLA+, δηλαδή, της μορφής: Init $\wedge \Box$ [Next] $\rightarrow$ Properties.

Σε αυτό το παράδειγμα υποθέτουμε μια μεταβλητή x τη οποία ορίζουμε να έχει μια δυνατή τιμή 0 ή 1, το οποίο μπορούμε να γράψουμε ως x ∈ 0,1. Ορίζουμε Init να είναι x=1. Η μόνη δυνατή ενέργεια (action) για το model μας είναι να αυξάνεται το x κατά 1.
Η ιδιότητα που επιθυμούμε να επαληθευτεί είναι ότι η τιμή της x είναι πάντα είτε 0 είτε 1. Η έκφραση που πρέπει να επαληθευτεί είναι:

$$((x'=1) \land \Box\, [x'= x+1]_x \,) \rightarrow \Box\, (\, (x \in 0,1)$$

Το tableau παρουσιάζεται στο ακόλουθο σχήμα, η αρίθμηση δεν είναι μέρος της απόδειξης αλλά χρησιμοποιείται μόνο για τον σκοπό της απεικόνισης. Τα 2,3,4 είναι από το 1 από την εφαρμογή του α-κανόνα, το 5 απλά προσθέτει την κατάσταση πρόθεμα (state prefix) στο 2, το 6 και το 7 είναι από το 4 από τον R_β-κανόνα, 8 και 9 είναι από το 6 από τον R_β-κανόνα, 10 και 11 είναι από το 3 από τον β-κανόνα (θυμηθείτε ότι [x= x + 1]x ≡ ((x= x + 1) ∨ (x′ = x)), 12, 13 είναι από το 7 από τον R_β-κανόνα.
Ο πιο αριστερός κλάδο, καταλήγει στο 6, έχει μια ύπαρξη του s_0(x = 1) και s_0(¬(x ∈ 0, 1)). Αυτό σημαίνει ότι και οι δύο εκτιμώνται σε true στην κατάσταση s0. Αυτός ο κλάδος είναι κλειστός επειδή (x = 1) και (¬(x ∈ 0, 1)) παράγουν μια αντίφαση.
Ο κλάδος που καταλήγει στο 12 ισχυρίζεται ότι το ίδιο είναι true στην κατάσταση s_1, επομένως ο κλάδος είναι επίσης κλειστός. Ο κλάδος που καταλήγει στο 13 έχει s_1(x = 2) και s_1(¬(x ∈ 0, 1)) να συμβαίνουν σε αυτό. Αυτό σημαίνει στην κατάσταση s1 και οι δύο είναι true, αλλά αυτό είναι απόλυτα καλό επειδή αν το x είναι 2 τότε το x δεν είναι 0 ή 1. Αυτός ο κλάδος τότε δίνει ένα counter-example της έκφρασης και συμπεραίνουμε ότι η προδιαγραφή του συστήματος δεν ικανοποιεί την επιθυμητή ιδιότητα. Δεν γνωρίζουμε αν ο κλάδος στο 9 είναι κλειστός ή δεν είναι αλλά αφού ήδη έχουμε έναν ανοιχτό κλάδο δεν χρειάζεται να συνεχίσουμε.



Σχήμα : Ένα Tableau για την ((x=1) ∧ □ [x′ = x+1]$_x$)→ □ ((x ∈ 0,1)

Πηγές

[1] L. Lamport. *Specifying Systems*. Addison-Wesley, Boston, 2002.

[2] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999

[3] Keith Miller. *A Tableau Model Checker for the Temporal Logic of Actions* ,August 2006

[4] S. Owicki and L. Lamport. *Proving Liveness Properties of Concurrent Programs*. ACM TOPLAS, 1982

[5] Leslie Lamport. *Fairness and hyperfairness*. SRC Research Report 152, Digital, Systems Research Center, March 1998.

[6] L.Lamport, J.Matthews, M.R.Tuttle, and Y. Yu, "*Specifying and verifying systems with TLA+*" in ACM SIGOPS European Workshop, 2002, pp. 45-48.

[7] Bowen Alpern and Fred B. Schneider. *Defining liveness*. Information Processing Letters, 21(4):181-185, 1985.

[8] http://en.wikipedia.org/wiki/Topology_(structure)

[9] http://en.wikipedia.org/wiki/Dense_set#cite_note-CEIT-0

[10] Cindy Eisner, Doron Peled. *Comparing Symbolic and Explicit Model Checking of a Software System*. In SPIN, pages 230-239, 2002.

[11] http://en.wikipedia.org/wiki/Strongly_connected_component

[12] Παναγιώτης Δ. Μποζάνης: "*Αλγόριθμοι - Σχεδιασμός και Ανάλυση*", Εκδόσεις Τζιόλα, 2003.

[13] http://en.wikipedia.org/wiki/Serialization

[14] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

[15] http://java.sun.com/developer/onlineTraining/rmi/RMI.html#RMIArchitectureLayers

[16] http://download.oracle.com/javase/1.4.2/docs/tooldocs/solaris/rmic.html

[17] http://download.oracle.com/javase/1.4.2/docs/api/java/rmi/server/UnicastRemoteObject.html

[18] http://download.oracle.com/javase/1.4.2/docs/guide/rmi/javarmiproperties.html

[19] http://linuxcommand.org/man_pages/taskset1.html

Γλωσσάρι όρων

DAG (Directed Acyclic Graph): Κατευθυνόμενος γράφος χωρίς κύκλους.

DNF (Disjuctive Normal Form, Διαζευκτική Κανονική Μορφή): Μια έκφραση είναι σε Διαζευκτική Κανονική Μορφή αν είναι μια διάζευξη από ελάχιστους όρους ((minterm) είναι ένα γράμμα ή η σύζευξη γραμμάτων) κανένας από τους οποίους δεν απορροφά κανέναν άλλο.

Fairness: Είναι μια από τις σημαντικότερες έννοιες στην ταυτόχρονη επεξεργασία. Σημαίνει ότι κάθε διεργασία έχει την ευκαιρία να σημειώσει πρόοδο ανεξάρτητα από τις άλλες διεργασίες.

Liveness ιδιότητα: Δηλώνει ότι κάτι καλό θα συμβεί τελικά, δηλαδή ότι το σύστημα τελικά θα εισαχθεί σε μια επιθυμητή κατάσταση.

Model: Μια απεικόνιση της συμπεριφοράς ενός συστήματος.

Model checking: Είναι η αυτόματη, βασισμένη σε models προσέγγιση για την επαλήθευση ιδιοτήτων.

RMI(Remote Method Invocation): Η επίκληση απομακρυσμένων μεθόδων επιτρέπει σε ένα αντικείμενο της Java που τρέχει σε έναν υπολογιστή, να καλέσει μια μέθοδο ενός άλλου αντικειμένου της Java που εκτελείται σε άλλο μηχάνημα. Η προσπέλαση της μεθόδου αυτής γίνεται με την βοήθεια του δικτύου.

Safety ιδιότητα: Δηλώνει ότι δεν πρόκειται ποτέ να συμβεί κάτι κακό, δηλαδή ότι το σύστημα δεν θα εισαχθεί ποτέ σε μια μη επιθυμητή κατάσταση.

Stuttering βήμα: Είναι γνωστό ως idling βήμα, είναι ένα βήμα το οποίο αφήνει το σύστημα στην ίδια κατάσταση. Δηλαδή f′ = f για όλες τις μεταβλητές f.

Βήμα: Είναι ένα ζεύγος από καταστάσεις (s,t) όπου s είναι η παλιά κατάσταση και t η νέα κατάσταση. Η s περιγράφεται από unprimed μεταβλητές και η t από primed μεταβλητές.

Ενέργεια: Είναι μια συσχέτιση μεταξύ δύο καταστάσεων και αναπαριστά μια ατομική λειτουργία ενός ταυτόχρονου συστήματος.

Ιδιότητα: Είναι μια έκφραση χρονικής λογικής η οποία ορίζει ένα κατηγόρημα πάνω σε συμπεριφορές.

Ισχυρά συνεκτική συνιστώσα: Μια ισχυρά συνεκτική συνιστώσα (**strongly connected component**) σε ένα γράφο είναι ένας μέγιστο υπογράφος από κόμβους και ακμές τέτοιος ώστε κάθε κόμβος να είναι προσβάσιμος από κάθε άλλο κόμβο.

Κατάσταση: Είναι μια συγκεκριμένη ανάθεση τιμών σε όλες τις μεταβλητές σε ένα στιγμιότυπο του χρόνου.

Κατηγόρημα: Είναι μια συνάρτηση που επιστρέφει μια λογική τιμή.

Προδιαγραφή: Είναι μια γραπτή περιγραφή του τι ένα σύστημα πρέπει να κάνει.

Σημασιολογικό tableau: Ένα πλήρως αυτοματοποιημένο σύστημα απόδειξης διάψευσης.

Συμπεριφορά: Μια άπειρη ακολουθία καταστάσεων.