ΒΙΒΛΙΟΘΗΚΗ
ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ
026000265541

ΣΥΓΧΡΟΝΙΣΜΟΣ ΔΙΕΡΓΑΣΙΩΝ ΜΕΣΩ ΔΟΣΟΛΗΨΙΩΝ

27
ΜΠΛΕ

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από τον

Ελευθέριο Κοσμά

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΗΝ ΘΕΩΡΙΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Οκτώβριος 2008

ΑΦΙΕΡΩΣΗ

Στους γονείς μου, στην αδελφή μου και στη Μαρία.

ΕΥΧΑΡΙΣΤΙΕΣ

Άρχικα θα ήθελα να ευχαριστήσω την επιβλέπουσα καθηγήτριά μου κα. Παναγιώτα Φατούρου για τη βοήθεια, την επιμονή και υπομονή της, τις πολύτιμες συμβουλές της και τη συμπαράστασή της κατά τη διάρκεια εκπόνησης της μεταπτυχιακής μου εργασίας. Ιδιαίτερα την ευχαριστώ, διότι η συνεργασία μας μου προσέφερε εμπειρία και γνώσεις που ξεπέρασαν κατά πολύ τις προσδοκίες μου.

Επίσης θα ήθελα να ευχαριστήσω τα υπόλοιπα μέλη της εξεταστικής μου επιτροπής, κ. Παναγιώτη Βασιλειάδη και κ. Λεωνίδα Παληό, για τις σημαντικές υποδείξεις τους.

Ευχαριστώ τους φίλους και γνωστούς μου για τις όμορφες αναμνήσεις που μου χάρισαν όλα αυτά τα χρόνια. Ακόμη τους ευχαριστώ για την κατανόηση και ηθική τους υποστήριξη κατά την περίοδο ολοκλήρωσης και συγγραφής της μεταπτυχιακής μου εργασίας.

Πάνω απ' όλα, είμαι ευγνώμων στους γονείς μου, Κωνσταντίνο και Θεώνη, για την ολόψυχη αγάπη και υποστήριξή τους όλα αυτά τα χρόνια.

Τέλος θα ήθελα να ευχαριστήσω τους ανθρώπους του Ιδρύματος Μποδοσάκη για την ηθική και υλική υποστήριξή τους.

ΠΕΡΙΕΧΟΜΕΝΑ

Σελ

ΑΦΙΕΡΩΣΗ ii
ΕΥΧΑΡΙΣΤΙΕΣ iii
ΠΕΡΙΕΧΟΜΕΝΑ iv
ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ vi
ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ vii
ΠΕΡΙΛΗΨΗ ix
EXTENDED ABSTRACT IN ENGLISH x
ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ 12
1.1. Γενικά 12
1.2. Προγραμματισμός Εφαρμογών για Πολυεπεξεργαστικά Συστήματα 14
1.2.1. Παραδοσιακές Τεχνικές Εξασφάλισης Συνέπειας 16
1.2.2. Software Transactional Μνήμη 18
1.3. Στόχοι Εργασίας 21
1.4. Διάρθρωση Εργασίας 23
ΚΕΦΑΛΑΙΟ 2. ΜΟΝΤΕΛΟ 25
2.1. Ασύγχρονο Σύστημα Διαμοιραζόμενης Μνήμης με Αποτυχίες Κατάρρευσης 25
2.2. Ατομικά Αντικείμενα 27
2.2.1. Γενικά 27
2.2.2. Ιδιότητες Ορθότητας 29
2.3. Software Transactional Μνήμη 32
2.3.1. Η Software Transactional Μνήμη ως διαμοιραζόμενο αντικείμενο 33
2.3.2. Ορισμοί 38
2.3.3. Ιδιότητες ατομικού αντικειμένου STM 40
2.3.4. Μετρικά Πολυπλοκότητας 42
2.3.5. Συγκρούσεις 43
2.3.6. Απαραίτητες συμβάσεις 45
2.3.7. Παράδειγμα χρήσης της Software Transactional Μνήμης 47
ΚΕΦΑΛΑΙΟ 3. ΚΑΤΗΓΟΡΙΕΣ ΣΧΕΔΙΑΣΤΙΚΩΝ ΕΠΙΛΟΓΩΝ 53
3.1. Τρόπος Ανάθεσης, Απόκτησης και Κατάργησης Ιδιοκτησιών 55
3.2. Χρόνος Απόκτησης Ιδιοκτησιών 58
3.3. Τρόποι Αποτροπής ή Ανίχνευσης και Επίλυσης Συγκρούσεων και Μηχανισμός Ελέγχου Συνέπειας 59
3.4. Πλήθος Ενδιάμεσων Επιπέδων 65
3.5. Μοντέλο Μνήμης 66
ΚΕΦΑΛΑΙΟ 4. ΑΛΓΟΡΙΘΜΟΣ DSTM 68
4.1. Περιγραφή αλγορίθμου DSTM 68
4.2. Απόδειξη ορθότητας αλγορίθμου DSTM 86

4.2.1. Σειριοποιησιμότητα 87
4.2.2. Ελευθερία Ανταγωνισμού 111
4.3. Πολυπλοκότητα αλγορίθμου DSTM 112
ΚΕΦΑΛΑΙΟ 5. ΑΛΓΟΡΙΘΜΟΣ SSTM 113
5.1. Περιγραφή Αλγορίθμου SSTM 113
5.2. Απόδειξη Αλγορίθμου SSTM 126
5.2.1. Σειριοποιησιμότητα 127
5.2.2. Ελευθερία Κλειδωμάτων 171
5.3. Πολυπλοκότητα αλγορίθμου SSTM 175
ΚΕΦΑΛΑΙΟ 6. ΑΛΓΟΡΙΘΜΟΣ NBSTM 176
6.1. Περιγραφή αλγορίθμου NBSTM 176
6.2. Απόδειξη ορθότητας αλγορίθμου NBSTM 199
6.2.1. Σειριοποιησιμότητα 201
6.2.2. Ελευθερία Κλειδωμάτων 222
6.3. Πολυπλοκότητα αλγορίθμου NBSTM 224
ΚΕΦΑΛΑΙΟ 7. ΕΠΙΣΚΟΠΗΣΗ ΥΠΑΡΧΟΝΤΩΝ ΑΛΓΟΡΙΘΜΩΝ STM 225
7.1. Μη-εμποδιστικοί Αλγόριθμοι 226
7.1.1. FSTM-OSTM 226
7.1.2. ASTM 235
7.2. Εμποδιστικοί Αλγόριθμοι 240
7.2.1. TL 240
7.2.2. TL II 246
7.3. Συνοπτικοί Παρουσίαση Αλγορίθμων 248
ΚΕΦΑΛΑΙΟ 8. ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΔΟΥΛΕΙΑ 251
ΑΝΑΦΟΡΕΣ 255
ΣΥΝΤΟΜΟ ΒΙΟΓΡΑΦΙΚΟ 257

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

Πίνακας

Σελ

Πίνακας 3.1 Σχεδιαστικές επιλογές και ιδιότητες αλγορίθμων STM 250

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

Σχήμα Σελ

Σχήμα 2.1: Πρώτο παράδειγμα σειριοποιήσιμης εκτέλεσης. α) Χωρίς τα σημεία σειριοποίησης, β) με τα σημεία σειριοποίησης 31

Σχήμα 2.2: Δεύτερο παράδειγμα σειριοποιήσιμης εκτέλεσης. 31

Σχήμα 2.3: Παράδειγμα μη-σειριοποιήσιμης εκτέλεσης. 32

Σχήμα 2.4: α) Η μορφή του κόμβου τύπου node της συνδεδεμένης λίστας. β) Η μορφή του κόμβου τύπου tmnode της συνδεδεμένης λίστας. 47

Σχήμα 2.5: Κώδικας της συνάρτησης εισαγωγής ενός στοιχείου x σε μια ταξινομημένη κατά αύξουσα διάταξη απλά συνδεδεμένη λίστα, η οποία μπορεί να προσπελάσετε από μία μόνο διεργασία. 48

Σχήμα 2.6: Κώδικας της συνάρτησης εισαγωγής ενός στοιχείου x σε μια ταξινομημένη κατά αύξουσα διάταξη απλά συνδεδεμένη λίστα, η οποία μπορεί να προσπελάσετε ταυτόχρονα από πολλές διεργασίες, με βάση το βασικό μοντέλο STM. 49

Σχήμα 2.7: Κώδικας της μακροεντολής Check. 50

Σχήμα 2.8: Ενημέρωση των δεδομένων που περιγράφει η t-μεταβλητή tvarprevP, στο επεκτεταμένο αντικείμενο STM. 52

Σχήμα 3.1: Η συνδεδεμένη λίστα του παραδείγματος, α) η αρχική της μορφή, β) η μορφή της μετά την μεταφορά του στοιχείου 8 στην αρχή της λίστας, από την $T(j,n_j)$. 62

Σχήμα 3.2: Η μορφή της συνδεδεμένης λίστας του παραδείγματος που η $T(i,n_i)$ γνωρίζει, α) πριν την ανάγνωση του στοιχείου 8 και β) μετά την ανάγνωσή του. 62

Σχήμα 4.1: Η μορφή ενός διαμοιραζόμενου αντικειμένου obj που περιγράφεται μέσω μιας t-μεταβλητής x, στον DSTM. 69

Σχήμα 4.2: Οι δομές που χρησιμοποιούνται από τον αλγόριθμο DSTM. 70

Σχήμα 4.3: Η μορφή μιας ταξινομημένης κατ' αύξουσα διάταξη συνδεδεμένης λίστας στον DSTM. 71

Σχήμα 4.4: Κώδικας της λειτουργίας BeginTransaction του DSTM. 72

Σχήμα 4.5: Κώδικας της λειτουργίας ReadTmVar του DSTM. 72

Σχήμα 4.6: Κώδικας της λειτουργίας AccessForUpdateTmVar του DSTM. 73

Σχήμα 4.7: Κώδικας της λειτουργίας OpenTmVar του DSTM. 75

Σχήμα 4.8: Εισαγωγή του στοιχείου 4, σε μια ταξινομημένη κατ' αύξουσα διάταξη συνδεδεμένη λίστα που αρχικά περιέχει τα στοιχεία 1,7,10. 77

Σχήμα 4.9: Κώδικας της συνάρτησης GetCurrentData του DSTM. 83

Σχήμα 4.10: Κώδικας της συνάρτησης Validate του DSTM. 84

Σχήμα 4.11: Κώδικας της λειτουργίας CreateNewTmVar του DSTM. 84

Σχήμα 4.12: Κώδικας της λειτουργίας CommitTransaction του DSTM. 85

Σχήμα 4.13: Κώδικας της συνάρτησης AbortTransaction του DSTM. 86

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Σχήμα 4.14: Εκτέλεση που περιγράφεται κατά την απόδειξη του Λήμματος 4.11. 98
Σχήμα 4.15: Εκτέλεση που περιγράφεται κατά την απόδειξη του Λήμματος 4.11. 99
Σχήμα 4.16: Εκτέλεση που περιγράφεται κατά την απόδειξη του Λήμματος 4.11. 100
Σχήμα 4.17: Εκτέλεση που περιγράφεται κατά την απόδειξη του Λήμματος 4.11. 100
Σχήμα 4.18: Εκτελέσεις που περιγράφονται κατά την απόδειξη του Λήμματος 4.11. 101
Σχήμα 5.1: Σχηματική αναπαράσταση συστήματος SSTM με M=6 και τη δοσοληψία $T(i,n_i)$ να κατέχει τις θέσεις 0,4,9 του πίνακα memory και την $T(j,n_j)$ τις 3,6,12,14. 115
Σχήμα 5.2: Κώδικας της λειτουργίας BeginTransaction του SSTM για την p_i. 117
Σχήμα 5.3: Κώδικας της συνάρτησης Initialize του SSTM για την p_i. 117
Σχήμα 5.4: Κώδικας της λειτουργίας AbortTransaction του SSTM για την p_i. 118
Σχήμα 5.5: Κώδικας της λειτουργίας CommitTransaction του SSTM για την p_i. 118
Σχήμα 5.6: Κώδικας της συνάρτησης PerformTrans του SSTM για την p_i. 120
Σχήμα 5.7: Κώδικας της λειτουργίας AcquireOwnerships του SSTM για την p_i. 122
Σχήμα 5.8: Κώδικας της λειτουργίας ReleaseOwnerships του SSTM για την p_i. 123
Σχήμα 5.9: Κώδικας της λειτουργίας AgreeOldValues του SSTM για την p_i. 123
Σχήμα 5.10: Κώδικας της λειτουργίας UpdateMemory του SSTM για την p_i. 124
Σχήμα 5.11: Εκτελέσεις που περιγράφονται κατά την απόδειξη του Λήμματος 5.7. 137
Σχήμα 5.12: Εκτέλεση που περιγράφεται κατά την απόδειξη του Λήμματος 5.11. 141
Σχήμα 5.13: Σχηματική αναπαράσταση του «κακού» σεναρίου που περιγράφεται κατά την απόδειξη του Λήμματος 5.17. 149
Σχήμα 5.14: Εκτέλεση που περιγράφεται κατά την απόδειξη του Λήμματος 5.20. 153
Σχήμα 6.1: Η μορφή ενός διαμοιραζόμενου αντικειμένου obj που περιγράφεται μέσω μιας t-μεταβλητής x, στον NBSTM. 177
Σχήμα 6.2: Οι δομές που χρησιμοποιούνται από τον αλγόριθμο NBSTM. 178
Σχήμα 6.3: Η μορφή μιας ταξινομημένης κατ' αύξουσα διάταξη συνδεδεμένης λίστας στον NBSTM. 180
Σχήμα 6.4: Κώδικας της λειτουργίας BeginTransaction του DSTM. 181
Σχήμα 6.5: Κώδικας της λειτουργίας ReadTmVar του NBSTM. 183
Σχήμα 6.6: Κώδικας της συνάρτησης InitializeNewLocator του NBSTM. 185
Σχήμα 6.7: Κώδικας της συνάρτησης GetCurrentData του NBSTM. 187
Σχήμα 6.8: Κώδικας της συνάρτησης TryHelpTransaction του NBSTM. 189
Σχήμα 6.9: Κώδικας της συνάρτησης Validate του NBSTM. 190
Σχήμα 6.10: Κώδικας της λειτουργίας WriteTmVar του NBSTM. 191
Σχήμα 6.11: Κώδικας της λειτουργίας CreateNewTmVar του NBSTM. 192
Σχήμα 6.12: Κώδικας της λειτουργίας AbortTransaction του NBSTM. 193
Σχήμα 6.13: Κώδικας της λειτουργίας CommitTransaction του NBSTM. 194
Σχήμα 6.14: Κώδικας της συνάρτησης PerformTrans του NBSTM. 195
Σχήμα 6.15: Κώδικας της συνάρτησης PerformTrans του NBSTM. 197
Σχήμα 7.1: Η μορφή ενός διαμοιραζόμενου αντικειμένου obj που περιγράφεται μέσω μιας t-μεταβλητής tv, στον FSTM. 227
Σχήμα 7.2: Η μορφή μιας ταξινομημένης κατ' αύξουσα διάταξη συνδεδεμένης λίστας στον FSTM. 228
Σχήμα 7.3: Η μορφή μιας ταξινομημένης κατ' αύξουσα διάταξη συνδεδεμένης λίστας στον FSTM. 230

ΠΕΡΙΛΗΨΗ

Ελευθέριος Κοσμάς του Κωνσταντίνου και της Θεώνης. MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Σεπτέμβριος, 2008. Συγχρονισμός Διεργασιών Μέσω Δοσοληψιών.

Επιβλέπουσα: Παναγιώτα Φατούρου.

Τα τελευταία χρόνια οι σχεδιαστές επεξεργαστών χρησιμοποιούν πολυ-επεξεργαστικές αρχιτεκτονικές για την κάλυψη των αυξανόμενων υπολογιστικών αναγκών των σύγχρονων εφαρμογών. Αυτή η σχεδιαστική κατεύθυνση ακολουθείται επειδή προέκυψαν φυσικοί περιορισμοί που δεν επιτρέπουν στην ταχύτητα ενός επεξεργαστή να αυξάνεται με ικανοποιητικό ρυθμό. Ένα βασικό εμπόδιο για την πλήρη εκμετάλλευση της υπολογιστικής ισχύος τέτοιων πολυ-επεξεργαστικών συστημάτων αποτελεί η δυσκολία παράλληλου προγραμματισμού των εφαρμογών που εκτελούνται σε τέτοια περιβάλλοντα. Η παραδοσιακή τεχνική χρήσης κλειδωμάτων είναι είτε μη-επεκτάσιμη ή δύσχρηστη, αφού επιφέρει πολλές δυσκολίες που μπορούν να αντιμετωπιστούν μόνο από έμπειρους προγραμματιστές.

Μια νέα τεχνική παράλληλου προγραμματισμού που αναγνωρίζεται από τους ερευνητές ως η επικρατέστερη εναλλακτική μέθοδος παράλληλου προγραμματισμού είναι η τεχνική ελέγχου συγχρονισμού μέσω δοσοληψιών (Software Transactional Memory ή STM). Η STM χρησιμοποιεί δοσοληψίες που επιτρέπουν την ατομική εκτέλεση κομματιών κώδικα που ο χρήστης καθορίζει. Αν και υπάρχουν αρκετές εργασίες που προτείνουν νέους αλγορίθμους STM, η θεωρητική μελέτη θεμελιωδών ιδιοτήτων των συστημάτων αυτών είναι ακόμη σε πρώιμο στάδιο.

Στην παρούσα εργασία, περιγράφουμε ένα νέο μοντέλο STM μέσω του οποίου μπορούν να περιγραφούν όλοι οι υπάρχοντες αλγόριθμοι STM και ταυτόχρονα επιτρέπει την απλοποίηση της διαδικασίας συγγραφής παράλληλου κώδικα, διότι ο κώδικας αυτός μοιάζει αρκετά με σειριακό κώδικα. Περιγράφουμε ιδιότητες ορθότητας και τερματισμού, οι οποίες ανταποκρίνονται στις επιθυμητές ιδιότητες των συστημάτων επίτευξης συγχρονισμού μέσω δοσοληψιών. Παρουσιάζουμε ψευδοκώδικα των βασικότερων υπαρχόντων αλγορίθμων, οι οποίοι είχαν περιγραφεί μη-φορμαλιστικά στο παρελθόν, και παραθέτουμε φορμαλιστικές αποδείξεις της ορθότητάς τους και των ιδιοτήτων τους. Τέλος, προτείνουμε ένα νέο αλγόριθμο STM που παρουσιάζει ισχυρές ιδιότητες τερματισμού.

EXTENDED ABSTRACT IN ENGLISH

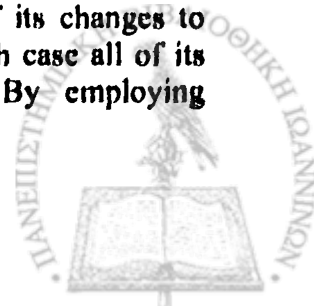
Kosmas, Eleftherios, K., T. MSc, Computer Science Department, University of Ioannina, Greece. September, 2008. Software Transactional Memory.

Thesis Supervisor: Faturu Panagiota.

Due to various architectural constraints, clock speeds are no longer rising. Instead, major chip manufacturers are producing multicore architectures, in which several processors reside on each chip, to achieve better performance. This trend is considered a revolution in modern computing by many computer specialists. Multiprocessor machines are more powerful and reliable, and they offer more capacity and better prices. Therefore, we will soon see multiprocessors in every computer, from desktops to cellular phones, and there is no doubt that multicore architectures will have tremendous impact on future research and technology.

Introducing as much parallelism as possible into software applications has become ever so urgent since this is the only way to take advantage of the new regime of multicore computers. A major obstacle in this direction comes from the difficulty of developing concurrent data structures and algorithms. This difficulty is inherent in achieving synchronization between processes that run concurrently. The current practice of using locks has several well-known shortcomings. Lock-based programming is widely considered to be too difficult for any but a few experts. Moreover, locks explicitly restrict parallelism, and hence are inherently non-scalable. Despite the hardness of concurrent programming, all modern distributed applications must rely on multithreading enhance performance.

Concurrent programming will become manageable only if alternative approaches to locking are developed to simplify the task of parallel programming. Software Transactional Memory (STM) [19] is currently one of the most promising programming models to take much of the pain out of writing highly concurrent programs. It supports simple an flexible transactional programming of synchronization operations. More specifically, a *transaction* is a piece of code executed by a single thread that may atomically execute a number of operations on distributed objects. A transaction may either commit making all of its changes to shared objects visible to other transactions at once, or abort in which case all of its modifications are discarded without ever becoming apparent. By employing



transactions, STM supplies the programmers with a powerful facility that eliminates the need for (1) tracking which locks should protect which objects, (2) discovering an appropriate trade-off between lock-granularity and concurrency, and (3) elaborating mechanisms for avoiding deadlock, priority inversion, convoying, low-level race conditions, etc., and ensuring atomicity and fault tolerance. Indeed, STM allows the programmer to barely think about concurrency at all; they write their programs as if they were sequential but they still gain the performance benefits of concurrency. So, the difficult problem of reasoning about a concurrent system is reduced to the simpler problem of reasoning about a sequential system.

Although transactions have been extensively used by the database community for a long time, it is only fairly recently that their benefits as concurrent programming primitives have started to be widely investigated by the parallel and distributed computing community. The last years have seen a flurry of research [] towards experimenting with several different STM implementation strategies. Unfortunately, this is not the case with research on fundamental theoretical aspects arising in the context of STM which is still at very premature level. It is surprising that there is so little formalization of the semantics and the guarantees that STM systems should provide. Precise definitions are badly missing and no consensus yet has emerged on specifying what guarantees should be provided by STM systems. Without such formalization, it is impossible to check the correctness of current implementations, capture all the details of their performance, discover their inherent limitations and establish optimality results, or determine whether any STM design tradeoffs are indeed fundamental or simply artifacts of certain environments. Most of the proposed STM implementations are described in a very informal way without providing even a pseudo-code, and in any case, without any formal proof of their correctness. As a result, the behavior of many of these algorithms is simply unpredictable.

In this thesis, we propose a new STM model which provides a set of operations in order to encapsulate and describe all the existing and oncoming STM algorithms, which is not an easy task. This model is simple enough to achieve the desirable resemblance of the concurrent code to the corresponding serial program which is apparently easier to program. On the other hand, the set of provided operations is rich enough to provide all promised functionality. Moreover, we redefine the linearizability property (correctness property used in atomic objects) to ensure the desirable properties of STM. We use this definition of linearizability and the proposed STM model, to describe many existing STM algorithms, proving that our model is general enough, and for the first time we give formal proofs for two of them, which considered as the state-of-the-arts. Finally, we propose a new non-blocking STM algorithm with competitive performance, and we introduce his code and a formal proof. This new algorithm is the first ever algorithm introduced together with a full formal proof.

ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ

1.1 Γενικά

1.2 Προγραμματισμός Εφαρμογών για Πολυεπεξεργαστικά Συστήματα

1.3 Στόχοι Εργασίας

1.4 Διάρθρωση Εργασίας

1.1. Γενικά

Παραδοσιακά η σχεδίαση των επεξεργαστών βασιζόταν στο *σειριακό μοντέλο* ή αλλιώς *μοντέλο von Neumann*. Στο μοντέλο αυτό υπάρχει μία μονάδα επεξεργασίας που επικοινωνεί με τη μνήμη προσκομίζοντας και εκτελώντας μία προς μία τις εντολές ενός προγράμματος που εμπεριέχουν την ανάγνωση και ενημέρωση δεδομένων. Στο μοντέλο αυτό η ταχύτητα με την οποία εκτελείται ένα πρόγραμμα εξαρτάται από την ταχύτητα του επεξεργαστή. Όσο η τεχνολογία σχεδίασης και ανάπτυξης επεξεργαστών εξελισσόταν οι επεξεργαστές γίνονταν ταχύτεροι, μικρότεροι σε μέγεθος και φθηνότεροι. Σημειώνεται ότι η ταχύτητα των επεξεργαστών διπλασιάζονταν κάθε τρία χρόνια για το χρονικό διάστημα μεταξύ 1980 και 1996. Όμως προέκυψαν φυσικοί περιορισμοί που εδώ και κάποια χρόνια δεν επιτρέπουν στην ταχύτητα ενός επεξεργαστή να αυξάνεται με υψηλούς ρυθμούς. Ένας τέτοιος περιορισμός είναι η ταχύτητα με την οποία μπορούν να κινούνται τα ηλεκτρόνια μέσα στα ηλεκτρονικά κυκλώματα, η οποία φράσσεται άνω από την ταχύτητα του φωτός. Επίσης, όσο το μέγεθος των επεξεργαστών μειώνεται ανακύπτουν φυσικά όρια μικροηλεκτρονικής φύσης που σχετίζονται με τους ημιαγωγούς. Λόγω αυτών των φυσικών ορίων η ταχύτητα εκτέλεσης εντολών από έναν μεμονωμένο επεξεργαστή δεν αυξάνεται πλέον με τον ίδιο ρυθμό που αυξανόταν στο παρελθόν.

Οι σημερινές εφαρμογές έχουν ωστόσο όλο και περισσότερες υπολογιστικές απαιτήσεις. Στις μέρες μας, έχουν προκύψει πλήθος σοβαρών προβλημάτων που αφορούν την ατμόσφαιρα, το διάστημα και την επιστήμη, οι λύσεις των οποίων απαιτούν τρομερή υπολογιστική δύναμη. Επομένως, η απαίτηση αύξησης της υπολογιστικής ισχύος εξακολουθεί να είναι επιτακτική. Για το λόγο αυτό, οι σχεδιαστές επεξεργαστών έχουν στραφεί τα τελευταία χρόνια στη χρήση πολλαπλών επεξεργαστών οι οποίοι θα εκτελούν πολλαπλές εντολές κάθε χρονική στιγμή, αντί της μίας εντολής που εκτελείται στο σειριακό μοντέλο. Έτσι, προέκυψαν οι *πολυεπεξεργαστικές αρχιτεκτονικές* (multicore architectures), στις οποίες πολλαπλοί επεξεργαστές αποθηκεύονται στην ίδια πλακέτα και συνθέτουν ένα πολυεπεξεργαστικό σύστημα το οποίο έχει αυξημένη απόδοση. Αυτή η κατεύθυνση χαρακτηρίστηκε επαναστατική από πολλούς ερευνητές.

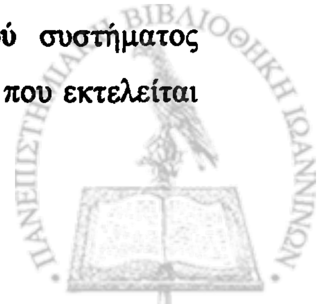
Ένα από τα πλεονεκτήματα των πολυεπεξεργαστικών συστημάτων είναι η δυνατότητα χρήσης φθηνών επεξεργαστών, για την υλοποίηση ενός συστήματος με υψηλή απόδοση. Επίσης, η αποθήκευση πολλαπλών επεξεργαστών στην ίδια πλακέτα μειώνει το κόστος κατασκευής, το οποίο σχετίζεται με τους κοινούς πόρους που αυτοί χρησιμοποιούν. Επομένως, τα πολυεπεξεργαστικά συστήματα είναι πιο φθηνά. Ακόμη, οι πολλαπλοί επεξεργαστές προσφέρουν υψηλότερη αξιοπιστία, διότι η αστοχία ενός από αυτούς δεν οδηγεί σε αστοχία όλων των προγραμμάτων που εκτελούνται, παρά μόνο του προγράμματος ή του τμήματος ενός προγράμματος που εκείνη τη στιγμή εκτελούνταν στο συγκεκριμένο επεξεργαστή. Επομένως τα πολυεπεξεργαστικά συστήματα είναι πιο ισχυρά, με μεγαλύτερη αξιοπιστία και με μειωμένο κόστος υλοποίησης. Για το λόγο αυτό, πολύ σύντομα θα δούμε την εμφάνιση πολυεπεξεργαστών σε κάθε υπολογιστή, από τον προσωπικό υπολογιστή έως τα κινητά τηλέφωνα, και δεν υπάρχει αμφιβολία ότι οι πολυεπεξεργαστικές αρχιτεκτονικές θα ασκήσουν μεγάλη επιρροή μελλοντικά στον τομέα της έρευνας και της τεχνολογίας.

1.2. Προγραμματισμός Εφαρμογών για Πολυεπεξεργαστικά Συστήματα

Με την ανάπτυξη των πολυεπεξεργαστικών συστημάτων, έγινε αναγκαία η ανάπτυξη των εφαρμογών με τέτοιο τρόπο ώστε να εκμεταλλεύονται με τον καλύτερο δυνατό τρόπο την αυξημένη υπολογιστική ισχύ των πολλαπλών επεξεργαστών. Οι εφαρμογές μπορούν να ευεργετηθούν από τις πολυεπεξεργαστικές αρχιτεκτονικές εάν ο κώδικας τους γραφτεί με τέτοιο τρόπο ώστε να εκτελείται παράλληλα σε διαφορετικούς επεξεργαστές. Επίσης, αυτό πρέπει να μπορεί να επιτευχθεί με τον απλούστερο δυνατό τρόπο, έτσι ώστε να διευκολύνεται ο μέσος προγραμματιστής στον προγραμματισμό της εφαρμογής του με παράλληλο τρόπο.

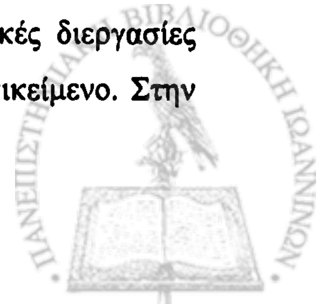
Στα περισσότερα λειτουργικά συστήματα η εκτέλεση ενός προγράμματος σε παράλληλο επεξεργαστικό περιβάλλον μπορεί να επιτευχθεί με την εκτέλεση του κώδικα του σε ξεχωριστές *διεργασίες* (ή *νήματα*), οι οποίες μπορούν να εκτελούνται παράλληλα. Κάθε εφαρμογή εκτελείται στη δική της διεργασία, έτσι πολλαπλές εφαρμογές ευεργετούνται από την πολυεπεξεργαστική αρχιτεκτονική. Επίσης, μία συγκεκριμένη εφαρμογή μπορεί να χρησιμοποιεί πολλαπλές διεργασίες. Σημειώνεται ότι οι προγραμματιστές είναι εξοικειωμένοι με τη χρήση νημάτων κατά την ανάπτυξη εφαρμογών, διότι ακόμα και στα συστήματα που αποτελούντο από έναν μόνο επεξεργαστή, οι προγραμματιστές έκαναν χρήση των νημάτων κατά την ανάπτυξη μιας εφαρμογής για λόγους ευκολίας, ώστε διαφορετικές λειτουργίες της εφαρμογής να εκτελούνται από διαφορετικά νήματα. Με την εμφάνιση των πολυεπεξεργαστικών συστημάτων η χρήση των νημάτων γίνεται με στόχο την αύξηση της απόδοσης της εφαρμογής, διότι διαφορετικές λειτουργίες του συστήματος μπορούν να εκτελούνται ταυτόχρονα. Κύριος στόχος των προγραμματιστών αποτελεί η εκμετάλλευση στο έπακρο των δυνατοτήτων των πολυεπεξεργαστικών συστημάτων με την κατά το δυνατό μεγιστοποίηση του αριθμού των λειτουργιών που μπορούν να εκτελεστούν ταυτόχρονα.

Διαισθητικά, σε ένα πολυεπεξεργαστικό σύστημα θα μπορούσαμε να πετύχουμε όσο υψηλή απόδοση επιθυμούμε αυξάνοντας το πλήθος των επεξεργαστών όσο χρειάζεται. Όμως κάτι τέτοιο δεν είναι αληθές. Ένα βασικό εμπόδιο για την πλήρη εκμετάλλευση της υπολογιστικής ισχύος ενός πολυεπεξεργαστικού συστήματος αποτελεί η δυσκολία παράλληλου προγραμματισμού μιας εφαρμογής που εκτελείται


ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

σε τέτοιο περιβάλλον. Ο τρόπος παράλληλου προγραμματισμού μιας εφαρμογής, εξαρτάται από την αρχιτεκτονική του πολυεπεξεργαστικού συστήματος. Μέχρι σήμερα, το κυρίαρχο προγραμματιστικό μοντέλο παράλληλων εφαρμογών είναι το *μοντέλο διαμοιραζόμενης μνήμης*, στο οποίο οι διεργασίες προσπελάζουν μια συλλογή από κοινές δομές δεδομένων και επικοινωνούν μεταξύ τους διαβάζοντας και ενημερώνοντας τις δομές αυτές. Το μοντέλο αυτό είναι το επικρατέστερο, διότι μοιάζει αρκετά με το σειριακό μοντέλο προγραμματισμού και άρα είναι πιο απλό. Το μοντέλο αυτό χρησιμοποιείται σε πολυεπεξεργαστικά συστήματα, στα οποία οι επεξεργαστές συνδέονται με μία κοινή μνήμη και η επικοινωνία μεταξύ τους γίνεται μέσω της τροποποίησης κοινών μεταβλητών στη μνήμη. Το δεύτερο μοντέλο προγραμματισμού παράλληλων συστημάτων, ονομάζεται *μοντέλο μεταβίβασης μηνυμάτων*. Στο μοντέλο αυτό, οι διεργασίες επικοινωνούν μεταξύ τους ανταλλάσσοντας δεδομένα, κάτι το οποίο απέχει αρκετά από τον τρόπο προγραμματισμού ενός σειριακού προγράμματος. Το μοντέλο αυτό χρησιμοποιείται σε πολυεπεξεργαστικά συστήματα, στα οποία κάθε επεξεργαστής έχει τη δική του ιδιωτική κοινή μνήμη που μπορεί να προσπελάζετε μόνο από τον ίδιο. Επίσης, υπάρχουν διασυνδέσεις μεταξύ των επεξεργαστών και η επικοινωνία μεταξύ των επεξεργαστών πραγματοποιείται με την αποστολή μηνυμάτων μέσω των διασυνδέσεων αυτών. Για να γίνει ευκολότερος ο προγραμματισμός των πολυεπεξεργαστών αυτής της κατηγορίας, οι σχεδιαστές υποστηρίζουν το μοντέλο προγραμματισμού διαμοιραζόμενης μνήμης μέσω οργάνωσης της μνήμης τους ως *κατανεμημένα κοινής μνήμης*, με την υλοποίηση αντίστοιχων πρωτοκόλλων.

Έτσι, στην παρούσα εργασία θα ασχοληθούμε με το μοντέλο προγραμματισμού της διαμοιραζόμενης μνήμης. Συνοπτικά, στο μοντέλο αυτό υποστηρίζονται *διαμοιραζόμενα αντικείμενα* (όπως π.χ. ένας καταχωρητής ή μια συνδεδεμένη λίστα) που έχουν κάποια κατάσταση και παρέχουν ένα σύνολο λειτουργιών μέσω των οποίων οι διεργασίες μπορούν να τα προσπελάσουν και να τροποποιήσουν την κατάστασή τους. Όμως ακόμα και με το μοντέλο της διαμοιραζόμενης μνήμης, ο παράλληλος προγραμματισμός είναι αρκετά δύσκολος, λόγω των συγκρούσεων που μπορεί να προκύψουν κατά την πρόσβαση κοινών διαμοιραζόμενων αντικειμένων από τις διεργασίες. Μια σύγκρουση παρουσιάζεται όταν διαφορετικές διεργασίες επιθυμούν να προσπελάσουν ταυτόχρονα το ίδιο διαμοιραζόμενο αντικείμενο. Στην


ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

περίπτωση αυτή, η υλοποίηση των λειτουργιών του αντικειμένου αυτού πρέπει να εξασφαλίζει ότι η προσπέλαση της κατάστασής του γίνεται με τέτοιο τρόπο, ώστε να αποφευχθούν προβλήματα ασυνέπειας της κατάστασής του. Επίσης, μια ακόμη επιθυμητή ιδιότητα της υλοποίησης των λειτουργιών αυτών είναι η υποστήριξη της ταυτόχρονης προσπέλασης του διαμοιραζόμενου αντικειμένου από διεργασίες που δεν συγκρούονται.

Σημειώνεται ότι μια *υλοποίηση* ενός διαμοιραζόμενου αντικειμένου παρέχει σε κάθε διεργασία έναν αλγόριθμο για την εκτέλεση των λειτουργιών που υποστηρίζει. Είναι σημαντικό ότι απλά διαμοιραζόμενα αντικείμενα (όπως π.χ. ο καταχωρητής) μπορούν να συνδυαστούν για την υλοποίηση των λειτουργιών ενός πολύπλοκου διαμοιραζόμενου αντικειμένου (π.χ. των λειτουργιών μιας λίστας). Με αυτό τον τρόπο δημιουργούνται κατανεμημένα συστήματα που είναι καλά δομημένα και η ορθότητά τους μπορεί να επαληθευθεί πιο εύκολα.

1.2.1. Παραδοσιακές Τεχνικές Εξασφάλισης Συνέπειας

Παραδοσιακά η συνέπεια ενός διαμοιραζόμενου αντικειμένου εξασφαλιζόταν με τη χρήση μιας τεχνικής που ονομάζεται *αμοιβαίος αποκλεισμός*, η οποία εγγυάται ότι ένα συγκεκριμένο τμήμα του κώδικα, το οποίο ονομάζεται *κρίσιμο τμήμα*, θα εκτελείται από μία μόνο διεργασία κάθε χρονική στιγμή. Έτσι, λέμε ότι το κρίσιμο τμήμα εκτελείται *ατομικά*. Έχει διεξαχθεί αρκετή έρευνα για την εύρεση αποδοτικών αλγορίθμων αμοιβαίου αποκλεισμού. Συγκεκριμένα έχουν σχεδιαστεί ένα πλήθος από αποδοτικές τεχνικές spin lock [1]. Οι προτεινόμενες τεχνικές αμοιβαίου αποκλεισμού χρησιμοποιούν την έννοια των *κλειδωμάτων*. Συγκεκριμένα αποδίδεται ένα κλείδωμα για κάθε κρίσιμο τμήμα του κώδικα και κάποια διεργασία πρέπει να αποκτήσει το συγκεκριμένο κλείδωμα πριν εκτελέσει τον κώδικα του κρίσιμου τμήματος και να το ελευθερώσει μετά την εκτέλεσή του. Σημειώνεται ότι μόνο μία διεργασία μπορεί να κατέχει το κλείδωμα κάθε χρονική στιγμή.

Ωστόσο ο προγραμματισμός με κλειδώματα επιφέρει αρκετά προβλήματα. Ένα ζήτημα είναι ο τρόπος ανάθεσης κλειδωμάτων. Η απλούστερη προσέγγιση είναι η ανάθεση ενός κλειδώματος σε κάθε διαμοιραζόμενο αντικείμενο, η οποία

αποκαλείται *ανάθεση κλειδωμάτων μεγάλης κλίμακας* (coarse-grained locking). Στην περίπτωση αυτή η υλοποίηση ενός διαμοιραζόμενου αντικειμένου είναι αρκετά απλή, όμως επιτυγχάνεται πολύ μικρός βαθμός ταυτοχρονισμού των λειτουργιών (ενώ θα έπρεπε να εξασφαλίζεται στο μέγιστο) διότι κάθε χρονική στιγμή μία μόνο διεργασία μπορεί να προσπελάζει το διαμοιραζόμενο αντικείμενο, ανεξάρτητα από το τμήμα των δεδομένων του αντικειμένου που επιθυμεί να προσπελάσει. Έτσι η προσέγγιση αυτή είναι εξ ορισμού μη επεκτάσιμη. Για την επίλυση του προβλήματος αυτού επινοήθηκε η χρήση «μικρότερων κλειδωμάτων», η οποία αποκαλείται ως *ανάθεση κλειδωμάτων μικρής κλίμακας* (fine-grained locking), όπου δεν ανατίθεται ένα κλείδωμα για όλο το διαμοιραζόμενο αντικείμενο, αλλά ανατίθεται ένα κλείδωμα σε μικρότερα τμήματά του, π.χ. σε κάθε στοιχείο μιας λίστας. Έτσι οι διεργασίες δεν θα κλειδώνουν όλο το διαμοιραζόμενο αντικείμενο όταν επιθυμούν να το προσπελάσουν, αλλά επιμέρους κομμάτια του. Όμως η απόδοση κλειδωμάτων μικρής κλίμακας σε κομμάτια του διαμοιραζόμενου αντικειμένου και πολύ περισσότερο ο σωστός προγραμματισμός των λειτουργιών που προσπελάζουν τη δομή δεδομένων, είναι μια δύσκολη διαδικασία η οποία πρέπει να γίνει προσεκτικά από έμπειρους προγραμματιστές για να αποφευχθούν προβλήματα όπως η κατάσταση αδιεξόδου (deadlock). Επίσης τα κλειδώματα αυξάνουν τη συμφόρηση στη μνήμη και κάνουν το σύστημα επιρρεπές σε χρονικές ανωμαλίες, όπως η αντιστροφή προτεραιότητας, και σε καταρρεύσεις διεργασιών (μια διεργασία μπορεί να *καταρρεύσει*, δηλαδή να σταματήσει να εκτελείται σε οποιοδήποτε στάδιο της εκτέλεσής της). Η αντιστροφή προτεραιότητας παρουσιάζεται όταν ένα νήμα υψηλότερης προτεραιότητας, αναμένει την ελευθέρωση κάποιου κλειδώματος που κατέχεται από κάποιο νήμα χαμηλότερης προτεραιότητας, το οποίο στη χειρότερη περίπτωση μπορεί να μην εκτελείται σε κάποιο επεξεργαστή. Επίσης, εάν μια διεργασία που κατέχει κάποιο κλείδωμα καταρρεύσει πριν την ελευθέρωση του κλειδώματος αυτού, τότε όλες οι υπόλοιπες δοσοληψίες που επιθυμούν να αποκτήσουν το κλείδωμα αυτό δε θα μπορούν να συνεχίσουν να εκτελούνται.

Για τους παραπάνω λόγους, οι αλγόριθμοι που βασίζονται σε κλειδώματα, οι οποίοι αποκαλούνται και ως *εμποδιστικοί* (blocking) αλγόριθμοι, θεωρούνται δύσκολοι να προγραμματιστούν, να γίνουν κατανοητοί, να επαληθευθούν και να διατηρηθούν. Για το λόγο αυτό σχεδιάστηκαν *μη-εμποδιστικοί* (non-blocking) αλγόριθμοι που

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

αποφεύγουν τη χρήση κλειδωμάτων και επομένως συνεχίζουν να εκτελούνται κατάλληλα ανεξάρτητα από τις καταρρεύσεις διεργασιών. Έχουν προταθεί αρκετοί τέτοιοι αλγόριθμοι για παράδειγμα για λίστες [21], [5], πίνακες κατακερματισμού [16] και λίστες αποφυγής (skip lists) [5]. Ωστόσο, η σχεδίαση μη-εμποδιστικών αλγορίθμων είναι μερικές φορές το ίδιο δύσκολη όσο και η αντιμετώπιση των προβλημάτων των κλειδωμάτων, και επομένως πρέπει να πραγματοποιείται από έμπειρους προγραμματιστές.

Παρά τη δυσκολία του παράλληλου προγραμματισμού, όλες οι σύγχρονες κατανεμημένες εφαρμογές πρέπει να βασιστούν στον προγραμματισμό μέσω νημάτων για την επίτευξη υψηλής απόδοσης. Ο παράλληλος προγραμματισμός θα γίνει εύχρηστος μόνο εάν αναπτυχθούν εναλλακτικές μέθοδοι που θα απλοποιούν την διαδικασία προγραμματισμού.

1.2.2. Software Transactional Μνήμη

Μια νέα τεχνική παράλληλου προγραμματισμού που αναγνωρίζεται από τους ερευνητές ως η επικρατέστερη εναλλακτική μέθοδος παράλληλου προγραμματισμού είναι η Software Transactional Μνήμη (STM) [19]. Η STM είναι ένα πολλά υποσχόμενο προγραμματιστικό μοντέλο που εξασφαλίζει την απλοποίηση της διαδικασίας ανάπτυξης παράλληλου κώδικα, απαλλάσσοντας τον προγραμματιστή από τη δυσκολία χρήσης των κλειδωμάτων και τη δυσκολία προγραμματισμού μη-εμποδιστικών αλγορίθμων. Για να το επιτύχει αυτό η μνήμη STM υποστηρίζει την εκτέλεση *δοσοληψιών* από τις διεργασίες, οι οποίες περιέχουν λειτουργίες που ο χρήστης θέλει να εκτελέσει στα διαμοιραζόμενα αντικείμενα. Η STM εγγυάται την ατομική εκτέλεση των λειτουργιών που περιέχονται σε μια δοσοληψία, διευκολύνοντας έτσι το έργο του προγραμματιστή. Έτσι, μια δοσοληψία είναι ένα κομμάτι κώδικα το οποίο η STM εξασφαλίζει ότι θα εκτελεστεί ατομικά. Σημειώνεται ότι οι συγγραφείς της εργασίας [19] εμπνεύστηκαν αυτή την τεχνική παράλληλου προγραμματισμού (σε επίπεδο λογισμικού) βάσει μίας παρόμοιας τεχνικής που προτάθηκε από τους συγγραφείς της εργασίας [9] και η οποία πρότεινε την υλοποίηση δοσοληψιών στο επίπεδο υλικού ενός συστήματος.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Πιο συγκεκριμένα, εάν μία δοσοληψία ολοκληρωθεί επιτυχώς, η STM εγγυάται ότι όλες οι λειτουργίες που περιέχει θα εκτελεστούν ατομικά. Από την άλλη εάν η δοσοληψία ολοκληρωθεί μη-επιτυχώς τότε οι πιθανές αλλαγές της δεν είναι ορατές από άλλες δοσοληψίες. Έτσι, ο χρήστης δεν αναλαμβάνει την επίλυση οποιωνδήποτε προβλημάτων προκύπτουν από τον ταυτοχρονισμό. Αρκεί να γράψει τον κώδικα του προγράμματός του σαν να εκτελούνταν σειριακά. Αυτό επιτρέπει στο χρήστη να επαληθεύσει την ορθότητα του προγράμματός του αρκετά εύκολα, διότι αρκεί να επαληθεύσει ότι η σειριακή του εκτέλεση είναι ορθή, ενώ ταυτόχρονα το πρόγραμμά του έχει αυξημένη απόδοση, διότι εκτελείται σε ένα πολυεπεξεργαστικό σύστημα.

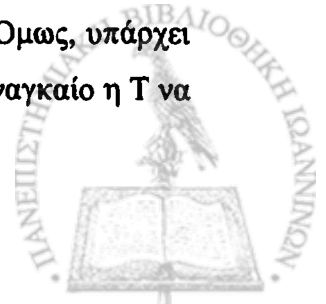
Υπάρχουν αρκετοί μη-εμποδιστικοί αλγόριθμοι που υλοποιούν αποδοτικά και με σωστό τρόπο τις λειτουργίες που υποστηρίζουν δομές όπως οι στοίβες, ουρές και λίστες. Οι αλγόριθμοι αυτοί εξασφαλίζουν ότι κάθε λειτουργία που καλεί ο χρήστης, θα εκτελείται ατομικά. Όμως δεν μπορούν να εξασφαλίσουν ότι θα εκτελεστούν ατομικά δύο διαφορετικές λειτουργίες που καλεί ο χρήστης. Για παράδειγμα, μπορεί να υπάρχει μια μη-εμποδιστική υλοποίηση της λειτουργίας εισαγωγής Insert σε μια λίστα, η οποία να σου επιτρέπει την ατομική εκτέλεση της εισαγωγής του στοιχείου x (Insert(x)) και την ατομική εκτέλεση της εισαγωγής του στοιχείου y (Insert(y)). Εάν, είναι επιθυμητή η υλοποίηση μιας δεύτερης μη-εμποδιστικής λειτουργίας εισαγωγής InsertDual δύο στοιχείων στη λίστα. Ένας αλγόριθμος που θα υλοποιούσε την InsertDual με την κλήση δύο λειτουργιών της λειτουργίας Insert (Insert(x) και Insert(y) για την εισαγωγή των στοιχείων x και y), δε θα εξασφάλιζε την ατομική εκτέλεση της λειτουργίας InsertDual. Έτσι λέμε ότι οι υλοποιήσεις των λειτουργιών των διαμοιραζόμενων αντικειμένων που παρέχουν οι υπάρχοντες μη-εμποδιστικοί αλγόριθμοι, δεν μπορούν να *συντεθούν*. Στο προγραμματιστικό μοντέλο που παρέχει η μνήμη STM, ο χρήστης μπορεί πολύ εύκολα να εκτελέσει ατομικά τις δύο ή περισσότερες λειτουργίες, απλά γράφοντας τον αντίστοιχο σειριακό κώδικα για τις λειτουργίες αυτές και εκτελώντας τον στα πλαίσια της ίδιας δοσοληψίας, η οποία εγγυάται την ατομική τους εκτέλεση. Έτσι λέμε ότι η μνήμη STM επιτρέπει τη *σύνθεση* διαφορετικών λειτουργιών ενός ή πολλαπλών διαμοιραζόμενων αντικειμένων, το οποίο αποτελεί ένα ακόμη πλεονέκτημα του προγραμματιστικού μοντέλου της STM.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Γενικά τα συστήματα STM δουλεύουν αντιστοιχίζοντας κάποιου είδους *επιπλέον πληροφορίες* (metadata) σε κάθε δοσοληψία, π.χ. για τη διατήρηση της κατάστασης μιας δοσοληψίας, η οποία μπορεί να είναι ενεργή ή ολοκληρωμένη επιτυχώς ή μη-επιτυχώς, και σε κάθε διαμοιραζόμενο αντικείμενο που προσπελάζεται μέσω δοσοληψιών, π.χ. για την περιγραφή της δοσοληψίας που κατέχει δικαίωμα προσπέλασης (*ιδιοκτησία*) του αντικειμένου αυτού. Όταν μια δοσοληψία θέλει να ενημερώσει κάποια διαμοιραζόμενα δεδομένα πρέπει να το κάνει με τέτοιο τρόπο, έτσι ώστε εάν η δοσοληψία ολοκληρωθεί μη επιτυχώς, οι ενημερώσεις της να μην είναι ορατές στις υπόλοιπες δοσοληψίες.

Στους περισσότερους αλγορίθμους STM μία δοσοληψία πριν την ενημέρωση ενός διαμοιραζόμενου αντικειμένου, αποκτά την ιδιοκτησία του αντικειμένου αυτού τροποποιώντας τις επιπλέον πληροφορίες του. Η συγκεκριμένη ιδιοκτησία μπορεί να είναι μη-αντιστρέψιμη, δηλαδή να είναι ένα κλείδωμα, κάτι το οποίο οδηγεί σε εμποδιστική υλοποίηση ή να είναι αντιστρέψιμη κάτι το οποίο οδηγεί σε μη-εμποδιστική υλοποίηση. Ένα παράδειγμα αντιστρέψιμης ιδιοκτησίας, είναι η δυνατότητα που παρέχουν αρκετοί αλγόριθμοι STM σε μια δοσοληψία T_1 να μεταβάλλει τις επιπλέον πληροφορίες μιας άλλης δοσοληψίας T_2, με τέτοιο τρόπο ώστε η T_2 να ολοκληρωθεί μη επιτυχώς και να καταργήσει τις ιδιοκτησίες που κατέχει (π.χ. αλλάζοντας την τιμή της κατάσταση της δοσοληψίας T_2). Στην περίπτωση αυτή λέμε ότι η T_2 *τερματίζεται βίαια ως μη επιτυχημένη* από την T_1, κάτι το οποίο επιτρέπει την υλοποίηση μη-εμποδιστικών αλγορίθμων. Με αντίστοιχο τρόπο η T_1 θα μπορούσε να *βοηθήσει* την T_2 να ολοκληρωθεί είτε ως επιτυχημένη είτε ως μη επιτυχημένη, κάτι το οποίο επίσης επιτρέπει την υλοποίηση μη-εμποδιστικών αλγορίθμων.

Ακόμη, όταν μια δοσοληψία T διαβάζει τα δεδομένα ενός διαμοιραζόμενου αντικειμένου x, πρέπει να εξασφαλίζει ότι τα δεδομένα αυτά δεν τροποποιούνται από κάποια άλλη δοσοληψία. Αυτό μπορεί να επιτευχθεί εάν η T τροποποιήσει τις επιπλέον πληροφορίες του x, με τέτοιο τρόπο ώστε κάποια άλλη δοσοληψία που θα θέλει να ενημερώσει το x ενόσω η T δεν έχει ολοκληρωθεί, να μπορεί να το γνωρίζει. Στην περίπτωση αυτή λέμε ότι χρησιμοποιούνται *ορατές αναγνώσεις*. Όμως, υπάρχει περίπτωση να μην χρησιμοποιούνται ορατές αναγνώσεις. Τότε, είναι αναγκαίο η T να


ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

εκτελεί έναν μηχανισμό ελέγχου της συνέπειας των δεδομένων που έχει διαβάσει, ο οποίος θα εξασφαλίζει ότι τα δεδομένα δεν έχουν ενημερωθεί μετά την ανάγνωσή τους από την Τ. Στους περισσότερους αλγορίθμους STM, ο έλεγχος αυτός πρέπει να εκτελείται κάθε φορά που μια δοσοληψία διαβάζει τα δεδομένα ενός διαμοιραζόμενου αντικειμένου, ώστε να αποφευχθούν προβλήματα λόγω ασυνέπειας των δεδομένων που διάβασε, όπως άπειροι βρόχοι ή μη επιτρεπτές προσπελάσεις μνήμης.

Η σχεδίαση ενός συστήματος μνήμης STM πραγματοποιείται από έναν έμπειρο προγραμματιστή, ο οποίος έχει ως κύριο στόχο του να αποκρύψει τις λεπτομέρειες υλοποίησης της STM από τον χρήστη (δηλαδή τον προγραμματιστή που χρησιμοποιεί το μοντέλο προγραμματισμού της μνήμης STM). Επίσης με βάση τα παραπάνω, γίνεται κατανοητό ότι η σχεδίαση ενός τέτοιου συστήματος δεν είναι προφανώς μια εύκολη διαδικασία. Υπάρχουν πολλά ζητήματα που πρέπει να ληφθούν υπόψιν και απαιτείται να αναπτυχθούν καινοτόμες τεχνικές για την αντιμετώπισή τους.

1.3. Στόχοι Εργασίας

Η χρησιμότητα των δοσοληψιών για τον προγραμματισμό των πολυεπεξεργαστικών συστημάτων έγινε κατανοητή μόλις πριν από λίγα χρόνια από την κοινότητα ερευνητών της παράλληλης και κατανεμημένης επεξεργασίας. Τα τελευταία χρόνια παρουσιάστηκαν αρκετές εργασίες που προτείνουν αλγορίθμους για την υλοποίηση μνήμης STM και πειραματίζονται με διαφορετικές προσεγγίσεις. Δυστυχώς δε συμβαίνει το ίδιο και με τη μελέτη θεμελιωδών θεωρητικών θεμάτων που ανακύπτουν σχετικά με τους αλγορίθμους μνήμης STM, η μελέτη των οποίων είναι ακόμα σε πρώιμο στάδιο. Είναι εντυπωσιακό ότι υπάρχουν πολύ λίγοι αυστηροί ορισμοί που αφορούν τη σημασιολογία και τις εγγυήσεις που ένα σύστημα STM πρέπει να παρέχει. Χωρίς αυτούς τους αυστηρούς ορισμούς είναι αδύνατο να ελεγχθεί η ορθότητα των τρεχόντων αλγορίθμων STM, να γίνουν κατανοητές όλες οι λεπτομέρειες της απόδοσής τους, να ανακαλυφθούν οι περιορισμοί τους και να εξακριβωθεί εάν κάποιοι σχεδιαστικοί συμβιβασμοί (tradeoffs) των σημερινών STM αλγορίθμων είναι πράγματι θεμελιώδεις ή είναι απλά τεχνουργήματα συγκεκριμένων περιβαλλόντων. Οι περισσότεροι από τους αλγορίθμους STM περιγράφονται χωρίς

αυστηρό τρόπο και πολλές φορές χωρίς καν την παρουσίαση του ψευδοκώδικά τους και σε κάθε περίπτωση χωρίς καμία απόδειξη της ορθότητάς τους. Αυτό έχει ως αποτέλεσμα, η συμπεριφορά αρκετών αλγορίθμων να είναι απρόβλεπτη.

Στην παρούσα εργασία αρχικά προτείνουμε ένα μοντέλο για τη μνήμη STM, το οποίο περιγράφει τη μνήμη STM ως ένα ατομικό αντικείμενο και περιέχει τις απαιτούμενες λειτουργίες που κάθε αλγόριθμος STM πρέπει να παρέχει στο χρήστη. Η σχεδίαση ενός τέτοιου μοντέλου δεν είναι εύκολη. Ένας βασικός στόχος ενός τέτοιου μοντέλου αποτελεί η υποστήριξη όσο το δυνατόν λιγότερων και απλούστερων λειτουργιών που θα επιτρέπουν την εύκολη μετατροπή του κώδικα του χρήστη από σειριακό σε παράλληλο. Στην ιδανική περίπτωση αυτό πρέπει να μπορεί να επιτευχθεί με πολύ μικρές τροποποιήσεις του σειριακού κώδικα. Ακόμη, η διαδικασία αυτή πρέπει να πραγματοποιείται χωρίς ο χρήστης να γνωρίζει την εσωτερική υλοποίηση ενός συστήματος STM. Ταυτόχρονα το σύνολο των λειτουργιών του μοντέλου αυτού πρέπει να είναι αρκετά πλούσιο ώστε να παρέχει την απαιτούμενη λειτουργικότητα, για παράδειγμα τη δημιουργία, αναφορά και διαχείριση δεδομένων που προσπελάζονται μέσω δοσοληψιών, και την εκκίνηση και τον τερματισμό ως επιτυχημένης ή αποτυχημένης μιας δοσοληψίας. Αξίζει να σημειωθεί ότι οι ήδη υπάρχοντες αλγόριθμοι δεν έχουν σχεδιαστεί βάσει κάποιου κοινού μοντέλου STM, αλλά ακολουθούν τη διαίσθηση που οι σχεδιαστές τους έχουν για το ποιες πρέπει να είναι οι απαραίτητες λειτουργίες τους. Επομένως ένα μοντέλο STM πρέπει να λάβει υπόψιν του αυτή τη διαίσθηση που υπάρχει στην ερευνητική κοινότητα, ώστε να είναι σε θέση να περιγράψει τη λειτουργικότητα των λειτουργιών που παρέχουν οι ήδη υπάρχοντες αλγόριθμοι STM.

Με βάση το προτεινόμενο μοντέλο STM, παρουσιάζουμε την λειτουργικότητα και την υλοποίηση των λειτουργιών που παρέχουν οι κυριότεροι από τους υπάρχοντες αλγορίθμους STM. Με τον τρόπο αυτό αποδεικνύεται η επίτευξη των απαιτούμενων στόχων που τέθηκαν κατά τη σχεδίαση του μοντέλου αυτού. Επίσης κατά την παρουσίαση των ήδη υπαρχόντων αλγορίθμων, γίνεται κατηγοριοποίησή τους βάσει των σχεδιαστικών επιλογών του καθενός, κάτι το οποίο μελλοντικά θα επιτρέψει την περιγραφή των ιδιοτήτων τους και την εύκολη σύγκρισή τους.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Είναι σημαντικό να οριστούν κατάλληλα οι απαιτούμενες ιδιότητες ορθότητας και τερματισμού των δοσοληψιών ενός συστήματος STM. Στην παρούσα εργασία, επανα-ορίζουμε την ιδιότητα της σειριοποιησιμότητας που ορίζεται για τα ατομικά αντικείμενα, ώστε να καλύπτει τις εγγυήσεις που ένας αλγόριθμος STM πρέπει να παρέχει ώστε να είναι ορθός. Επίσης, εξετάζονται διάφορες ιδιότητες τερματισμού.

Έπειτα, με βάση το μοντέλο STM και την ιδιότητα της σειριοποιησιμότητας, περιγράφουμε δύο σημαντικούς υπάρχοντες αλγορίθμους STM, παρουσιάζουμε ψευδοκώδικα και αποδεικνύουμε την ορθότητά τους. Είναι σημαντικό, ότι για πρώτη φορά παρουσιάζεται κώδικας για έναν από τους δύο αλγορίθμους, ενώ ο κώδικας του άλλου αλγορίθμου είχε κάποια μικρά προβλήματα, τα οποία διορθώθηκαν.

Τέλος, στην παρούσα εργασία προτείνουμε έναν νέο μη-εμποδιστικό αλγόριθμο για την υλοποίηση μνήμης STM. Ο αλγόριθμος αυτός είναι ένας από τους ελάχιστους αλγορίθμους STM που ικανοποιούν κάποια μη-εμποδιστική ιδιότητα. Επίσης στον αλγόριθμο αυτό προτείνουμε νέες ιδέες και συνδυάζουμε ιδέες από προηγούμενους αλγορίθμους που του επιτρέπουν να εξασφαλίζει αυξημένη απόδοση. Για τον αλγόριθμο αυτό παρουσιάζεται ο κώδικάς του και αποδεικνύεται φορμαλιστικά η απόδειξή του κάτι το οποίο δεν πραγματοποιείται από την πλειοψηφία των υπαρχόντων αλγορίθμων μνήμης STM.

1.4. Διάρθρωση Εργασίας

Η παρούσα εργασία οργανώνεται ως ακολούθως. Αρχικά, στο Κεφάλαιο 2 παρουσιάζεται το μοντέλο STM το οποίο εισάγουμε και επανα-ορίζεται η ιδιότητα της σειριοποιησιμότητας. Στο Κεφάλαιο 3 συζητούνται οι κατηγορίες σχεδιαστικών επιλογών με βάση τις οποίες διαφοροποιούνται οι ήδη υπάρχοντες αλγόριθμοι STM και περιγράφονται συνοπτικά οι κυριότεροι από αυτούς βάσει του προτεινόμενου μοντέλου STM. Στα Κεφάλαια 4 και 5 περιγράφονται δύο ήδη υπάρχοντες αλγόριθμοι STM, παρουσιάζοντας τους κώδικές τους και τη φορμαλιστική απόδειξη της ορθότητάς τους. Στο Κεφάλαιο 6 παρουσιάζεται ένας νέος μη-εμποδιστικός αλγόριθμος STM που προτείνουμε στην παρούσα εργασία και αποδεικνύεται με φορμαλιστικό τρόπο η ορθότητά του. Στο Κεφάλαιο 7 περιγράφονται συνοπτικά οι

κυριότεροι από τους υπάρχοντες αλγορίθμους STM. Τέλος, στο Κεφάλαιο 8 εξάγουμε συμπεράσματα από την παρούσα εργασία και συζητούμε μελλοντικές ερευνητικές κατευθύνσεις.

ΚΕΦΑΛΑΙΟ 2. ΜΟΝΤΕΛΟ

2.1 Ασύγχρονο Σύστημα Διαμοιραζόμενης Μνήμης με Αποτυχίες Κατάρρευσης

2.2 Ατομικά Αντικείμενα

0 Εάν θεωρήσουμε ότι αρχικά η ουρά είναι κενή, τότε η εκτέλεση που παρουσιάζεται στο Σχήμα 2.3 δεν είναι σειριοποιήσιμη. Αυτό ισχύει γιατί δεν είναι εφικτό για κάθε λειτουργία π, να εισαχθούν σημεία σειριοποίησης *π, κάπου μεταξύ της κλήσης και της απάντησής της, τέτοια ώστε εάν οι λειτουργίες εκτελεστούν σειριακά, με βάση τη σειρά που ορίζεται από τα σημεία σειριοποίησης, να επιστραφούν οι ίδιες απαντήσεις με αυτές της εκτέλεσης του παραδείγματος. Αφού οι λειτουργίες $deq_1()$ και $deq_2()$ επιστρέφουν 5, τότε η λειτουργία enq(5) σειριοποιείται πρώτη. Τότε όμως, όποια από τις $deq_1()$, $deq_2()$ σειριοποιείται πρώτη επιστρέφει 5, και η άλλη null και όχι 5. Επομένως, δεν είναι δυνατό να επιλεγούν κατάλληλα σημεία σειριοποίησης για όλες τις λειτουργίες και άρα η εκτέλεση που παρουσιάζεται στο Σχήμα 2.3 δεν είναι σειριοποιήσιμη.

Software Transactional Μνήμη

Αρχικά στην Ενότητα 2.1 παρουσιάζεται το ασύγχρονο σύστημα διαμοιραζόμενης μνήμης με αποτυχίες κατάρρευσης. Στη συνέχεια, στην Ενότητα 2.2 θα οριστούν τα ατομικά αντικείμενα, θα παρουσιαστούν οι ιδιότητές τους και θα δοθούν κάποιοι ορισμοί. Τέλος, στην Ενότητα 2.3 θα οριστεί το ακριβές μοντέλο για το STM που μελετάται στην παρούσα εργασία.

2.1. Ασύγχρονο Σύστημα Διαμοιραζόμενης Μνήμης με Αποτυχίες Κατάρρευσης

Θεωρούμε ένα *ασύγχρονο σύστημα διαμοιραζόμενης μνήμης* που περιέχει n *διεργασίες* $p_0, \ldots, p_{n-1}$. Οι διεργασίες μπορούν να επικοινωνούν μεταξύ τους προσπελάζοντας

διαμοιραζόμενους καταχωρητές που βρίσκονται στη διαμοιραζόμενη μνήμη. Κάθε καταχωρητής έχει έναν τύπο που καθορίζει το σύνολο των τιμών που μπορούν να αποθηκευτούν σε αυτόν καθώς και το σύνολο των λειτουργιών που μπορούν να εκτελεστούν (ατομικά) σε αυτόν (περισσότερες πληροφορίες για τους καταχωρητές παρουσιάζονται στην Ενότητα 2.2). Το σύστημα είναι ασύγχρονο που σημαίνει ότι δε μπορούν να υπάρξουν περιορισμοί στον τρόπο εκτέλεσης των διεργασιών του συστήματος, διότι αυτές εκτελούνται με αυθαίρετη σειρά και με αυθαίρετες ταχύτητες. Οι διεργασίες μπορούν χωρίς προειδοποίηση να *καταρρεύσουν*, δηλαδή να σταματήσουν να εκτελούνται από κάποιο σημείο και έπειτα.

Κάθε διεργασία μοντελοποιείται ως μια *μηχανή καταστάσεων*, όπου η *κατάσταση* κάθε διεργασίας αποτελείται από τις τοπικές της μεταβλητές. Μια *καθολική κατάσταση* C ενός συστήματος με m καταχωρητές είναι ένα διάνυσμα n+m τιμών ($q_0, \ldots, q_{n-1}, r_0, \ldots, r_{m-1}$), όπου q_i είναι η κατάσταση της p_i, $0 \leq i \leq n-1$, και r_j είναι η τιμή του καταχωρητή R_j, $0 \leq j \leq m-1$. Μια καθολική κατάσταση παρέχει μια καθολική εικόνα του συστήματος σε κάποια χρονική στιγμή. Στην αρχική καθολική κατάσταση, όλες οι διεργασίες βρίσκονται στην αρχική τους κατάσταση. Σε ένα ασύγχρονο σύστημα διαμοιραζόμενης μνήμης υπάρχουν μόνο *υπολογιστικά γεγονότα*, όπου σε κάθε υπολογιστικό γεγονός: i) η p_i επιλέγει, βάσει της τρέχουσας κατάστασής της, έναν από τους καταχωρητές για προσπέλαση, έστω τον R_j, ii) η p_i εκτελεί την καθορισμένη λειτουργία στον R_j, iii) η κατάσταση της p_i μεταβάλλεται σύμφωνα με τη συνάρτηση μετάβασής της. Η *μετάβαση* αυτή γίνεται σύμφωνα με την τρέχουσα κατάσταση της p_i και την τιμή που επιστράφηκε από την λειτουργία που εφαρμόστηκε στον R_j. Ένα *τμήμα εκτέλεσης* είναι μια πεπερασμένη ή άπειρη ακολουθία από καθολικές καταστάσεις και γεγονότα που εναλλάσσονται: $C_0, e_1, C_1, e_2, C_2, \ldots$, όπου κάθε e_i είναι ένα υπολογιστικό γεγονός και κάθε C_i είναι μία καθολική κατάσταση, $i \geq 0$. Λέμε ότι σε μια καθολική κατάσταση C εκτελείται κάποια ενέργεια k, αν η k είναι η ενέργεια που εκτελείται από το τελευταίο υπολογιστικό γεγονός που προηγείται της C. Μια *εκτέλεση* είναι ένα τμήμα εκτέλεσης που ξεκινά από μια αρχική καθολική κατάσταση. Το *χρονοδιάγραμμα* μιας εκτέλεσης είναι η ακολουθία από γεγονότα $e_1, e_2, \ldots$ της εκτέλεσης. Μια διεργασία που δεν καταρρέει σε μια συγκεκριμένη εκτέλεση ονομάζεται *ενεργή*.

2.2. Ατομικά Αντικείμενα

Ένα ατομικό αντικείμενο έχει μία κατάσταση και υποστηρίζει ένα σύνολο λειτουργιών μέσω των οποίων μπορεί να τροποποιηθεί η κατάστασή του ταυτόχρονα από πολλές διεργασίες. Στην Ενότητα 2.2.1 παρουσιάζονται κάποια βασικά ατομικά αντικείμενα και δίνονται οι απαραίτητοι ορισμοί. Στην Ενότητα 2.2.2 παρουσιάζονται οι ιδιότητες ορθότητας των ατομικών αντικειμένων.

2.2.1. Γενικά

Το πιο βασικό ατομικό αντικείμενο είναι ο *καταχωρητής ανάγνωσης-εγγραφής*, ο οποίος αποθηκεύει μια τιμή από κάποιο σύνολο και υποστηρίζει δύο λειτουργίες: i) read(R_i) που επιστρέφει την τιμή του καταχωρητή R_i χωρίς να την αλλάζει, και ii) write(R_i,v), που γράφει την τιμή v στον καταχωρητή R_i και επιστρέφει μια επιβεβαίωση (ack). Εάν το σύνολο τιμών που μπορεί να αποθηκευθεί σε έναν καταχωρητή είναι πεπερασμένο τότε ο καταχωρητής είναι πεπερασμένης χωρητικότητας. Σε αντίθετη περίπτωση είναι μη πεπερασμένης χωρητικότητας. Επίσης οι καταχωρητές διακρίνονται, με βάση τον τρόπο που μπορούν να γραφούν, σε *απλής εγγραφής* (single-writer) και σε *πολλαπλής εγγραφής* (multi-writer). Ένας καταχωρητής απλής εγγραφής μπορεί να γραφεί από μία μόνο συγκεκριμένη διεργασία ενώ ένας πολλαπλής εγγραφής από οποιαδήποτε διεργασία. Σημαντικό είναι ότι όλοι οι καταχωρητές μπορούν να διαβαστούν από οποιαδήποτε διεργασία.

Ένα βασικό ατομικό αντικείμενο ισχυρότερο, από τον καταχωρητή ανάγνωσης-εγγραφής, είναι ο *καταχωρητής Load-Link, Store-Conditional (καταχωρητής LL/SC)*, ο οποίος αποθηκεύει μια τιμή από κάποιο σύνολο και υποστηρίζει δύο λειτουργίες: i) LL(R_i) που επιστρέφει την τιμή του καταχωρητή R_i, και ii) SC(R_i , v) με την οποία μία διεργασία p αποθηκεύει την τιμή v στον καταχωρητή R_i εάν δεν έχει γίνει καμία αλλαγή στον R_i από την προηγούμενη ανάγνωσή του από την p μέσω της λειτουργίας LL. Όπως γίνεται κατανοητό η λειτουργία SC δεν μπορεί να καλεστεί μόνη της από κάποια διεργασία αλλά πρέπει να συνδυαστεί με μια προηγούμενη κλήση της λειτουργίας LL από την ίδια διεργασία.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Ένα ακόμη βασικό ατομικό αντικείμενο είναι ο *καταχωρητής CAS (Compare and Swap)*, ο οποίος αποθηκεύει μια τιμή από κάποιο σύνολο και υποστηρίζει δύο λειτουργίες: i) read(R_i) που επιστρέφει την τιμή του καταχωρητή R_i, και ii) CAS(R_i, u, v) όπου εάν η τιμή του R_i είναι u κατά την εκτέλεση της CAS, τότε αλλάζει σε v και επιστρέφεται TRUE. Διαφορετικά, η τιμή του R_i δε μεταβάλλεται και επιστρέφεται FALSE.

Πολλά υπολογιστικά συστήματα υποστηρίζουν μέσω υλικού κάποια από τα παραπάνω βασικά ατομικά αντικείμενα και αυτό σημαίνει ότι το υλικό εγγυάται την ατομική εκτέλεση των λειτουργιών τους. Το υλικό οποιουδήποτε συστήματος παρέχει καταχωρητές ανάγνωσης-εγγραφής, αλλά υπάρχουν συστήματα των οποίων το υλικό παρέχει και κάποιο πλήθος καταχωρητών CAS. Χρησιμοποιώντας βασικά ατομικά αντικείμενα που παρέχονται από το υλικό μπορούν να υλοποιηθούν άλλα πιο πολύπλοκα. Με αυτό τον τρόπο δημιουργούνται κατανεμημένα συστήματα που είναι καλά δομημένα και η ορθότητά τους μπορεί να επαληθευθεί πιο εύκολα.

Μια υλοποίηση ενός αντικειμένου από κάποια βασικά αντικείμενα χρησιμοποιεί τα βασικά αντικείμενα για την αποθήκευση των δεδομένων του υλοποιούμενου αντικειμένου και παρέχει αλγορίθμους για τις λειτουργίες που παρέχονται από αυτό. Λέμε ότι ένα αντικείμενο x *μπορεί να υλοποιήσει* ένα αντικείμενο y, εάν υπάρχει αλγόριθμος ο οποίος να υλοποιεί το αντικείμενο y χρησιμοποιώντας κάποιο αριθμό στιγμιοτύπων του x και καταχωρητές ανάγνωσης-εγγραφής. Επίσης, λέμε ότι τα αντικείμενα x και y είναι *ισοδύναμα* εάν το x μπορεί να υλοποιήσει το y και το y μπορεί να υλοποιήσει το x.

Η εκτέλεση μιας λειτουργίας σε ένα αντικείμενο αποτελείται από δύο γεγονότα: την *κλήση (call)* και την *απόκριση (response)* της λειτουργίας. Όταν μια διεργασία p_i εκτελεί μια λειτουργία OP σε ένα υλοποιούμενο αντικείμενο x, τότε η p_i βάσει της υλοποίησης της OP εκτελεί τις απαραίτητες λειτουργίες στα βασικά αντικείμενα που υλοποιούν το x, έτσι ώστε να παραχθεί η σωστή λειτουργικότητα της λειτουργίας OP στο x. Σημειώνεται ότι σε κάθε εκτέλεση ενώ στην πραγματικότητα η υλοποίηση των λειτουργιών ενός αντικειμένου x αποτελείται από λειτουργίες επί των βασικών αντικειμένων που η συγκεκριμένη υλοποίηση χρησιμοποιεί, είναι δυνατό να

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

καθοριστούν και οι λειτουργίες πάνω στο υλοποιούμενο αντικείμενο x, οι οποίες ονομάζονται λειτουργίες υψηλού επιπέδου. Στο εξής για ένα υλοποιούμενο αντικείμενο O, θα αποκαλούμε τις λειτουργίες του O απλά ως *λειτουργίες* και τις λειτουργίες επί των βασικών αντικειμένων που χρησιμοποιεί η υλοποίηση του O ως *εντολές*, εκτός και αν αναφερόμαστε ρητά στη μία ή στην άλλη κατηγορία λειτουργιών.

Δοθείσης μιας εκτέλεσης α, ονομάζουμε *ιστορικό της α* και το συμβολίζουμε με h_a την ακολουθία των κλήσεων και αποκρίσεων επί του υλοποιούμενου αντικειμένου που περιέχονται στην α. Αν h_a είναι ένα ιστορικό και p_i μια διεργασία, συμβολίζουμε με $h_a|p_i$ τον περιορισμό του h_a σε κλήσεις και αποκρίσεις που πραγματοποιήθηκαν από την p_i. Το $h_a|p_i$ ονομάζεται *ιστορικό της p_i*. Το ιστορικό της p_i είναι *καλά δομημένο* εάν αποτελείται από εναλλασσόμενες κλήσεις και αποκρίσεις ξεκινώντας με μία κλήση. Αυτό σημαίνει ότι δεν μπορεί μια διεργασία που έχει καλέσει κάποια λειτουργία ενός ατομικού αντικειμένου να καλέσει άλλη λειτουργία του ίδιου αντικειμένου πριν την ολοκλήρωση της πρώτης. Σημειώνεται ότι δοθέντος ενός ιστορικού h_a είναι πάντα δυνατό να καθοριστεί το ιστορικό κάθε διεργασίας p_i. Έτσι λοιπόν ένα ιστορικό h_a είναι *καλά δομημένο* εάν το ιστορικό κάθε διεργασίας p_i, $0 \leq i \leq n-1$, είναι καλά δομημένο.

2.2.2. Ιδιότητες Ορθότητας

Η πιο ευρέως διαδεδομένη ιδιότητα ορθότητας είναι η *σειριοποιησιμότητα* [13]. Έστω μια υλοποίηση Y ενός αντικειμένου O. Η σειριοποιησιμότητα εγγυάται ότι κάθε λειτουργία της Y σε μια εκτέλεση, εμφανίζεται σαν να εκτελείται στιγμιαία (ατομικά) σε κάποιο σημείο μεταξύ της κλήσης της και της απάντησής της, στη εκτέλεση αυτή. Στη συνέχεια παρουσιάζεται αυστηρός ορισμός της σειριοποιησιμότητας.

Έστω μια εκτέλεση α. Λέμε ότι η α είναι *σειριοποιήσιμη* εάν μπορούν να γίνουν τα εξής:

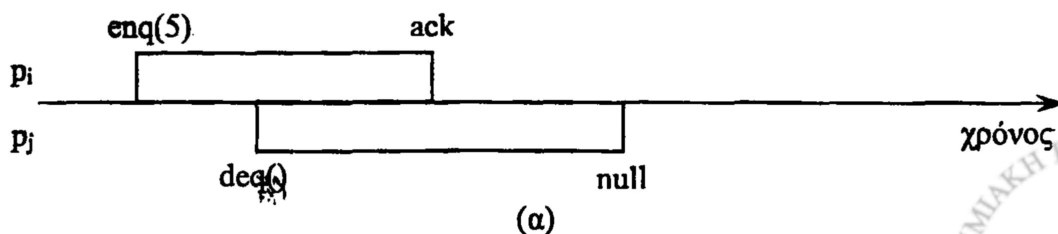
i) Για κάθε ολοκληρωμένη λειτουργία π στην α μπορεί να επιλεχθεί ένα σημείο σειριοποίησης *π κάπου μεταξύ της κλήσης της π και της απάντησης της στην α,

ii) Να επιλεχθεί ένα υποσύνολο Φ από μη ολοκληρωμένες λειτουργίες τέτοιο ώστε: για κάθε λειτουργία π στο σύνολο Φ, α) να μπορεί να εισαχθεί μία απόκριση, και β) να μπορεί να επιλεχθεί ένα σημείο σειριοποίησης *π κάπου μετά από την κλήση της π στην α, και

iii) αυτά τα σημεία σειριοποίησης πρέπει να επιλεχθούν με τέτοιο τρόπο ώστε, εάν οι λειτουργίες εκτελούνταν σειριακά, με βάση τη σειρά που ορίζεται από τα σημεία σειριοποίησης, να επιστρέφουν τις ίδιες αποκρίσεις με αυτές στην παράλληλη εκτέλεση α.

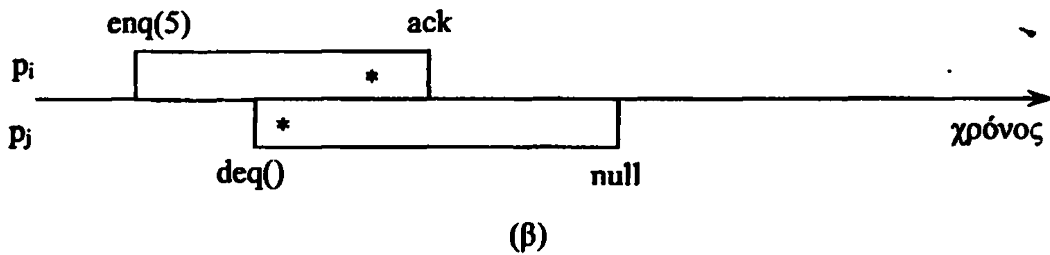
Ο προσδιορισμός *ατομικά* των αντικειμένων οφείλεται στη σειριοποιησιμότητα. Για να γίνει κατανοητός ο ορισμός της σειριοποιησιμότητας θα παρουσιαστούν παραδείγματα εκτελέσεων που είναι σειριοποιήσιμες και άλλων που δεν είναι. Ας θεωρήσουμε ως αντικείμενο μια ουρά Q, η οποία υποστηρίζει τις εξής λειτουργίες: i) enq(x), η οποία εισάγει το στοιχείο x στην ουρά και της επιστρέφεται μια επιβεβαίωση (ack), και ii) deq(), η οποία εξάγει το πρώτο στοιχείο της ουράς (FIFO-First In First Out Queue). Έστω ότι την ουρά προσπελαύνουν δύο διεργασίες, οι p_i και p_j.

Το πρώτο παράδειγμα σειριοποιήσιμης εκτέλεσης παρουσιάζεται στο Σχήμα 2.1(α). Η συγκεκριμένη εκτέλεση είναι σειριοποιήσιμη διότι σε κάθε λειτουργία π μπορούν να εισαχθούν σημεία σειριοποίησης *π, όπως παρουσιάζεται στο Σχήμα 2.1 (β), σε κάποιο σημείο μεταξύ της κλήσης και της απάντησής της, τέτοια ώστε εάν οι λειτουργίες εκτελεστούν σειριακά, με βάση την σειρά που ορίζεται από τα σημεία σειριοποίησης, να επιστρέφουν τις ίδιες αποκρίσεις με αυτές στην εκτέλεση (α). Είναι σημαντικό να παρατηρηθεί ότι το σημείο σειριοποίησης της enq(5) πρέπει απαραίτητα να επιλεγεί μετά από το σημείο σειριοποίησης της deq(), διότι στην παράλληλη εκτέλεση η deq() επιστρέφει null που σημαίνει ότι δεν πρόλαβε να δει την εισαγωγή του στοιχείου 5 από την p_i.



(α)

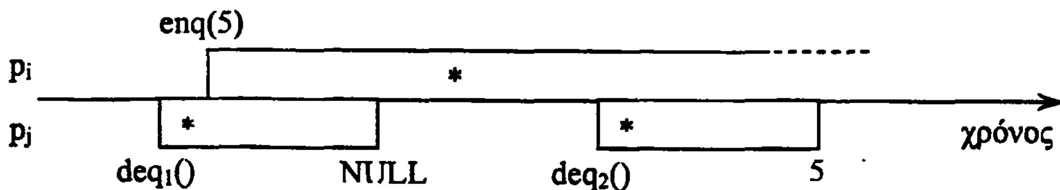
ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



(β)

Σχήμα 2.1: Πρώτο παράδειγμα σειριοποιήσιμης εκτέλεσης. α) Χωρίς τα σημεία σειριοποίησης, β) με τα σημεία σειριοποίησης

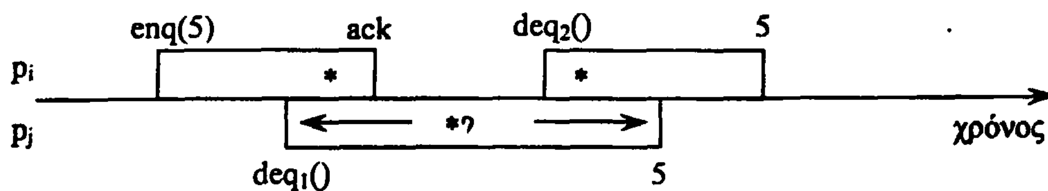
Στο Σχήμα 2.2 παρουσιάζεται ένα ακόμα παράδειγμα σειριοποιήσιμης εκτέλεσης. Στο συγκεκριμένο παράδειγμα η λειτουργία enq(5) της διεργασίας p_i δεν έχει λάβει απόκριση. Στην περίπτωση αυτή πρέπει πάλι να επιλεχθούν σημεία σειριοποίησης για όλες τις ολοκληρωμένες διεργασίες. Για την enq(5) θα πρέπει επίσης να επιλεχθεί σημείο σειριοποίησης αφού η $deq_2()$ επιστρέφει το 5 που εισάγει στην ουρά η enq(5). Η εκτέλεση του παραδείγματος αυτού είναι σειριοποιήσιμη διότι, όπως παρουσιάζεται στο Σχήμα 2.2, μπορούν να επιλεχθούν σημεία σειριοποίησης. Το σημείο σειριοποίησης της enq(5) πρέπει να επιλεγεί μετά από το σημείο σειριοποίησης της $deq_1()$ και πριν το αντίστοιχο σημείο σειριοποίησης της $deq_2()$. Αυτό ισχύει διότι η $deq_1()$ επιστρέφει NULL και η $deq_2()$ την τιμή 5, άρα η τιμή 5 πρέπει να έχει εισαχθεί μεταξύ των σημείων σειριοποίησης της $deq_1()$ και της $deq_2()$. Για την μη ολοκληρωμένη enq(5) το σημείο σειριοποίησής της θα είναι και σημείο απόκρισής της.



Σχήμα 2.2: Δεύτερο παράδειγμα σειριοποιήσιμης εκτέλεσης.

Στη συνέχεια, στο Σχήμα 2.3 παρουσιάζεται ένα παράδειγμα μη-σειριοποιήσιμης εκτέλεσης.

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ



Σχήμα 2.3: Παράδειγμα μη-σειριοποιήσιμης εκτέλεσης.

Εάν θεωρήσουμε ότι αρχικά η ουρά είναι κενή, τότε η εκτέλεση που παρουσιάζεται στο Σχήμα 2.3 δεν είναι σειριοποιήσιμη. Αυτό ισχύει γιατί δεν είναι εφικτό για κάθε λειτουργία π, να εισαχθούν σημεία σειριοποίησης *π, κάπου μεταξύ της κλήσης και της απάντησής της, τέτοια ώστε εάν οι λειτουργίες εκτελεστούν σειριακά, με βάση τη σειρά που ορίζεται από τα σημεία σειριοποίησης, να επιστραφούν οι ίδιες απαντήσεις με αυτές της εκτέλεσης του παραδείγματος. Αφού οι λειτουργίες $deq_1()$ και $deq_2()$ επιστρέφουν 5, τότε η λειτουργία enq(5) σειριοποιείται πρώτη. Τότε όμως, όποια από τις $deq_1()$, $deq_2()$ σειριοποιείται πρώτη επιστρέφει 5, και η άλλη null και όχι 5. Επομένως, δεν είναι δυνατό να επιλεγούν κατάλληλα σημεία σειριοποίησης για όλες τις λειτουργίες και άρα η εκτέλεση που παρουσιάζεται στο Σχήμα 2.3 δεν είναι σειριοποιήσιμη.

2.3. Software Transactional Μνήμη

Η *Software Transactional Μνήμη* (*STM*) αποτελείται από ένα σύνολο κελιών μνήμης, τα οποία ονομάζονται *transactional μεταβλητές*, ή *t-μεταβλητές* εν συντομία, και τα οποία επιτρέπεται να προσπελάζονται από τις διεργασίες μόνο μέσω των λειτουργιών που η STM παρέχει. Συγκεκριμένα η STM επιτρέπει στις διεργασίες να επικοινωνούν διαβάζοντας και ενημερώνοντας t-μεταβλητές με τη βοήθεια *δοσοληψιών*. Αυτό σημαίνει ότι οι διεργασίες περικλείουν τις λειτουργίες που επιθυμούν να εκτελέσουν στις t-μεταβλητές σε μια δοσοληψία και ζητούν από την STM να εκτελέσει τη συγκεκριμένη δοσοληψία. Εάν το μέγεθος της μνήμης αυτής είναι συγκεκριμένο τότε λέμε ότι χρησιμοποιείται *στατική STM*, ενώ εάν μπορεί να τροποποιηθεί λέμε ότι χρησιμοποιείται *δυναμική STM*. Σημειώνεται ότι για την εκτέλεση μιας δοσοληψίας που χρησιμοποιεί δυναμικά δεδομένα απαιτείται η χρήση δυναμικής STM.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Η STM εγγυάται την ατομική εκτέλεση των λειτουργιών που περιέχονται σε μια δοσοληψία, διευκολύνοντας έτσι το έργο του προγραμματιστή. Μία δοσοληψία μπορεί να ολοκληρωθεί επιτυχώς ή μη-επιτυχώς. Η STM εγγυάται ότι εάν μία δοσοληψία T ολοκληρωθεί *επιτυχώς*, όλες οι λειτουργίες που περιέχει θα εκτελεστούν *ατομικά*. Από την άλλη εάν η T ολοκληρωθεί μη-επιτυχώς τότε η STM εγγυάται ότι οι πιθανές αλλαγές της T στις t-μεταβλητές δεν είναι ορατές από άλλες δοσοληψίες.

Η συνέχεια της παρούσας ενότητας οργανώνεται ως ακολούθως. Στην Ενότητα 2.3.1 θα παρουσιαστεί το μοντέλο STM που εισαγάγουμε στην παρούσα εργασία. Στις Ενότητες 2.3.2, 2.3.3 και 2.3.4 θα παρουσιαστούν ορισμοί, ιδιότητες και μετρικά πολυπλοκότητας για την STM. Στην Ενότητα 2.3.5 ορίζονται οι συγκρούσεις που μπορούν να παρουσιαστούν μεταξύ λειτουργιών διαφορετικών δοσοληψιών. Στην Ενότητα 2.3.6 περιγράφονται οι απαραίτητες συμβάσεις χρήσης του μοντέλου από τον χρήστη και στην Ενότητα 2.3.7 παρουσιάζεται ένα παράδειγμα χρήσης του μοντέλου για την υλοποίηση της λειτουργίας εισαγωγής δεδομένων σε μια συνδεδεμένη λίστα.

2.3.1. Η Software Transactional Μνήμη ως διαμοιραζόμενο αντικείμενο

Η STM μπορεί να μοντελοποιηθεί ως ένα ατομικό αντικείμενο που παρέχει κάποιες λειτουργίες στις διεργασίες που επιθυμούν να το χρησιμοποιήσουν. Οι λειτουργίες αυτές αποτελούν τη διεπαφή μεταξύ του χρήστη της STM και της υλοποίησης της STM. Το μοντέλο STM πρέπει να είναι αρκετά γενικό ώστε μέσω αυτού να μπορούν να εκφραστούν όλοι οι υπάρχοντες αλγόριθμοι STM, μερικοί από τους οποίους περιγράφονται στα επόμενα Κεφάλαια της παρούσας εργασίας. Στη συνέχεια παρουσιάζεται το *βασικό μοντέλο STM* που εισαγάγουμε στην παρούσα εργασία.

Μια μνήμη STM ορίζεται να είναι ένα ατομικό αντικείμενο, το οποίο υποστηρίζει για κάθε διεργασία p_i τις εξής λειτουργίες:

i) *pointer BeginTransaction()*: Η λειτουργία αυτή χρησιμοποιείται από την p_i για να εκκινήσει μια νέα δοσοληψία. Η BeginTransaction επιστρέφει ένα δείκτη σε μια κατάλληλη δομή που περιγράφει τη δοσοληψία που μόλις εκκινήθηκε. Ο χρήστης

απαιτείται να χρησιμοποιήσει ως παράμετρο ένα τέτοιο δείκτη κατά την κλήση των υπολοίπων λειτουργιών που θα εκτελεστούν από αυτή τη δοσοληψία.

ii) *pointer CreateNewTmVar(t, d)*: Η λειτουργία αυτή χρησιμοποιείται από την p_i που εκτελεί τη δοσοληψία t για να δημιουργήσει μια νέα t-μεταβλητή tvar τα δεδομένα της οποίας αρχικοποιούνται ώστε να είναι ίδια με τα δεδομένα *d*. Η χρησιμότητα της λειτουργίας αυτής εξηγείται στη συνέχεια. Η CreateNewTmVar επιστρέφει στην p_i ένα δείκτη σε μια κατάλληλη δομή που περιγράφει την tvar. Ο χρήστης απαιτείται να χρησιμοποιήσει ως παράμετρο ένα τέτοιο δείκτη στην κλήση των λειτουργιών ReadTmVar και WriteTmVar που απευθύνονται στην tvar.

iii) *(Boolean, d) ReadTmVar(t, tVar)*: Η λειτουργία αυτή χρησιμοποιείται από την p_i που εκτελεί τη δοσοληψία *t* (o pointer που επιστράφηκε από την BeginTransaction) για να διαβάσει τα δεδομένα της t-μεταβλητής *tVar* (o pointer που επιστράφηκε από την CreateNewTmVar). Η ReadTmVar επιστρέφει στην p_i δύο τιμές. Η πρώτη είναι μία τιμή Boolean η οποία είναι TRUE εάν η λειτουργία ReadTmVar ολοκληρώθηκε επιτυχώς και FALSE σε αντίθετη περίπτωση. Εάν επιστραφεί TRUE, τότε στην δεύτερη μεταβλητή επιστρέφονται τα δεδομένα *d* της tVar.

iv) *Boolean WriteTmVar (t, tVar, d)*: Η λειτουργία αυτή χρησιμοποιείται από την p_i που εκτελεί τη δοσοληψία *t* για να ενημερώσει τα δεδομένα της t-μεταβλητής *tVar* με την τιμή *d*. Η WriteTmVar επιστρέφει στην p_i μία τιμή Boolean η οποία είναι TRUE εάν η λειτουργία WriteTmVar ολοκληρώθηκε επιτυχώς και FALSE σε αντίθετη περίπτωση.

v) *Boolean CommitTransaction(t)*: Με τη λειτουργία αυτή η p_i δηλώνει ότι ολοκλήρωσε την εκτέλεση της δοσοληψίας *t* και ζητά από την STM να ολοκληρώσει την *t* επιτυχώς. Η λειτουργία αυτή επιστρέφει TRUE σε περίπτωση επιτυχούς ολοκλήρωσης της t ως επιτυχημένης ή FALSE σε περίπτωση ολοκλήρωσης της t ως μη-επιτυχημένης.

vi) *AbortTransaction(t)*: Με τη λειτουργία αυτή η p_i δηλώνει ότι επιθυμεί τον τερματισμό της δοσοληψίας *t* ως μη-επιτυχημένη. Η λειτουργία αυτή εξασφαλίζει ότι η t θα τερματίσει μη-επιτυχώς και έτσι δεν επιστρέφει τίποτε.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Χάριν απλότητας, υποθέτουμε ότι κάθε διεργασία μπορεί να αποθηκεύει ως δεδομένα των t-μεταβλητών μόνο δείκτες. Αυτό σημαίνει ότι η διεργασία μπορεί για παράδειγμα να χρησιμοποιεί την t-μεταβλητή για την αποθήκευση μιας αριθμητικής τιμής x ή ενός πίνακα τιμών A, εισάγοντας ως δεδομένα της t-μεταβλητής έναν δείκτη στη x ή στον A, αντίστοιχα. Είναι σημαντικό ότι η t-μεταβλητή μπορεί να χρησιμοποιηθεί από τον χρήστη για την αποθήκευση ενός δείκτη που περιγράφει κάποια δομή που ο χρήστης έχει δημιουργήσει. Αυτό είναι χρήσιμο, για παράδειγμα για την αποθήκευση των κόμβων μιας συνδεδεμένης λίστας αντιστοιχίζοντας μια t-μεταβλητή σε κάθε κόμβο της λίστας, κάτι το οποίο πραγματοποιείται με την αποθήκευση του αντίστοιχου δείκτη κάθε κόμβου στην t-μεταβλητή.

Σημειώνεται ότι οι τρέχουσες υλοποιήσεις μνήμης STM χρησιμοποιούν διαφορετικές διεπαφές. Οι περισσότερες από αυτές μπορούν να υλοποιηθούν εύκολα, μέσω των λειτουργιών της διεπαφής που παρέχει το βασικό μοντέλο STM. Υπάρχουν όμως άλλες που παρέχουν μεγαλύτερη λειτουργικότητα στο χρήστη. Στην παρούσα εργασία θα μελετηθεί κάποιος αριθμός από αυτές. Αυτές θα παρουσιαστούν βάσει ενός *επεκτεταμένου μοντέλου STM* που περιγράφεται στη συνέχεια. Για να επιτευχθεί η παρουσίαση των περισσότερων αλγορίθμων που μελετώνται στην παρούσα εργασία με τον απλούστερο δυνατό τρόπο, θεωρήθηκε προτιμητέο να βασιστεί η παρουσίασή τους στο βασικό μοντέλο. Προφανώς, αυτό είναι δυνατό να επιτευχθεί και βάσει του επεκτεταμένου μοντέλου, αλλά θα επέφερε μεγαλύτερη πολυπλοκότητα στην παρουσίαση του κώδικά τους.

Μια μνήμη STM ορίζεται να είναι ένα ατομικό αντικείμενο, το οποίο βάσει του επεκτεταμένου μοντέλου STM υποστηρίζει για κάθε διεργασία p_i τις εξής λειτουργίες:

i) *pointer BeginTransaction()*: Η λειτουργία αυτή δουλεύει όπως και η αντίστοιχη του βασικού μοντέλου STM.
ii) *pointer CreateNewTmVar()*: Η λειτουργία αυτή δουλεύει όπως και η αντίστοιχη του βασικού μοντέλου STM.
iii) *(Boolean, d) ReadTmVar(t, tVar)*: Η λειτουργία αυτή δουλεύει όπως και η αντίστοιχη του βασικού μοντέλου STM.

iv) *(Boolean, pd) AccessForUpdateTmVar(t, tVar)*: Με τη λειτουργία αυτή η διεργασία p_i που εκτελεί τη δοσοληψία *t* (ο pointer που επιστράφηκε από την BeginTransaction) ζητά από το σύστημα STM να της επιστραφεί ένας δείκτης στα δεδομένα της t-μεταβλητής *tvar* (ο pointer που επιστράφηκε από την CreateNewTmVar), τα οποία στη συνέχεια θα μπορεί να ενημερώσει. Η λειτουργία AccessForUpdateTmVar επιστρέφει στην p_i δύο τιμές. Η πρώτη είναι μία τιμή Boolean η οποία είναι TRUE εάν η λειτουργία αυτή ολοκληρώθηκε επιτυχώς και FALSE σε αντίθετη περίπτωση. Εάν επιστραφεί TRUE, τότε στη δεύτερη μεταβλητή επιστρέφεται ένας δείκτης *pd* στα δεδομένα της tVar, τα οποία η p_i μπορεί εκτός από το να τα διαβάσει και να τα ενημερώσει μέσω του δείκτη αυτού.

v) *Boolean CommitTransaction(t)*: Η λειτουργία αυτή δουλεύει όπως και η αντίστοιχη του βασικού μοντέλου STM.

vi) *AbortTransaction(t)*: Η λειτουργία αυτή δουλεύει όπως και η αντίστοιχη του βασικού μοντέλου STM.

Σημειώνεται ότι τα δύο μοντέλα παρέχουν την ίδια διεπαφή για την εκκίνηση μιας δοσοληψίας, την ανάγνωση των δεδομένων των t-μεταβλητών και τον τερματισμό μιας δοσοληψίας, και διαφέρουν στη λειτουργία που παρέχουν για την ενημέρωση των t-μεταβλητών. Στο βασικό μοντέλο STM μια διεργασία p_i μπορεί να ενημερώσει τα δεδομένα μιας t-μεταβλητής tvar με την τιμή d, περνώντας ως όρισμα την tvar και την τιμή d στην αντίστοιχη κλήση της λειτουργίας WriteTmVar και η λειτουργία αυτή επιστρέφει επιβεβαίωση επιτυχίας ή μη-επιτυχίας. Αντίθετα, στο επεκτεταμένο μοντέλο STM για να ενημερώσει η p_i τα δεδομένα της tvar με την τιμή d, πρέπει να καλέσει τη λειτουργία AccessForUpdateTmVar με όρισμα μόνο την tvar και η λειτουργία αυτή θα επιστρέψει στην p_i έναν δείκτη μέσω του οποίου θα μπορεί να ενημερώσει τα δεδομένα που επιθυμεί. Επίσης με τον δείκτη αυτό η p_i μπορεί εκτός από το να ενημερώσει τα δεδομένα της tvar και να τα διαβάσει (χωρίς προηγούμενη κλήση της ReadTmVar), κάτι το οποίο δεν παρέχεται από τη λειτουργία WriteTmVar του βασικού μοντέλου STM.

Τα δύο μοντέλα που παρουσιάστηκαν παρέχουν τις απαραίτητες λειτουργίες που απαιτούνται από μία διεπαφή STM. Παρέχουν λειτουργίες για την εκκίνηση της

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

δοσοληψίας, την προσπέλαση των δεδομένων των t-μεταβλητών και τον τερματισμό της δοσοληψίας. Μια λειτουργικότητα που παρέχεται και η αναγκαιότητά της είναι ίσως λίγο δύσκολα ορατή, είναι η δημιουργία μιας νέας t-μεταβλητής με τη λειτουργία CreateNewTmVar. Κάθε υλοποίηση μνήμης STM αντιστοιχεί κάποιες επιπλέον πληροφορίες (metadata) σε κάθε μεταβλητή του χρήστη που πρόκειται να προσπελαστεί μέσω δοσοληψιών. Κάθε υλοποίηση μνήμης STM αντιστοιχεί κάποιες επιπλέον πληροφορίες (metadata) σε κάθε t-μεταβλητή που περιγράφει δεδομένα του χρήστη τα οποία πρόκειται να προσπελαστούν μέσω δοσοληψιών. Στη συνέχεια θα αναφερόμαστε στις απαραίτητες αυτές πληροφορίες ως *αναπαράσταση της t-μεταβλητής* στο σύστημα STM. Μέσω της CreateNewTmVar ο χρήστης ενημερώνει το σύστημα STM για την ύπαρξη κάποιων ακόμα δεδομένων που πρόκειται να προσπελάζονται μέσω δοσοληψιών, έτσι ώστε να δημιουργηθεί μια νέα t-μεταβλητή που θα αντιστοιχισθεί στα δεδομένα αυτά και να αρχικοποιηθεί κατάλληλα ο τρόπος αναπαράστασής της στο εκάστοτε σύστημα STM.

Στο σημείο αυτό αξίζει να σημειωθεί ότι εάν χρησιμοποιείται στατική μνήμη STM, τότε η απαιτούμενη αρχικοποίηση των t-μεταβλητών μπορεί να πραγματοποιηθεί κατά την εκκίνηση του συστήματος STM. Όμως για να μπορεί να χρησιμοποιηθεί δυναμική μνήμη STM, οι t-μεταβλητές πρέπει να μπορούν να δημιουργηθούν δυναμικά, ανάλογα με τις απαιτήσεις των διεργασιών που χρησιμοποιούν την STM. Για το λόγο αυτό παρέχεται η λειτουργία CreateNewTmVar.

Σημειώνεται ότι η κοινότητα των ερευνητών δεν έχει καταλήξει ακόμη σε κάποιο κοινά αποδεκτό μοντέλο STM. Υπάρχουν αρκετές εργασίες που προτείνουν κάποιο μοντέλο STM. Ενδεικτικά αναφέρουμε κάποιες από τις οποίες εμπνευστήκαμε το βασικό μοντέλο STM. Η εργασία [18] από τον Michael L. Scott, προτείνει ένα μοντέλο STM με αρκετά απλές λειτουργίες. Επίσης, η εργασία [7] των Rachid Guerraoui και Michal Kapalka προτείνει ένα πιο ισχυρό μοντέλο STM, το οποίο έχει αρκετές ομοιότητες με το προηγούμενο μοντέλο. Σημειώνεται ότι και τα δύο αυτά μοντέλα αποτυγχάνουν να περιγράψουν την πλήρη λειτουργικότητα που μια διεπαφή πρέπει να προσφέρει στο χρήστη της STM, δηλαδή στις διεργασίες. Η λειτουργικότητα που δεν παρέχουν στο χρήστη, είναι η δημιουργία και αρχικοποίηση μιας νέας t-μεταβλητής, η οποία όπως εξηγήθηκε είναι απαραίτητη ώστε να

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

υποστηριχθεί η εκτέλεση δοσοληψιών από διεργασίες που χρησιμοποιούν δυναμικά δεδομένα και άρα απαιτούν δυναμική μνήμη STM.

Οι απαραίτητες συμβάσεις χρήσης του μοντέλου STM από το χρήστη καθώς επίσης και ένα παράδειγμα χρήσης του μοντέλου STM για την υλοποίηση μιας ταξινομημένης συνδεδεμένης λίστας, παρουσιάζονται στις Ενότητες 2.3.6 και 2.3.7, αντίστοιχα.

2.3.2. Ορισμοί

Μια *δοσοληψία* είναι μια ακολουθία από λειτουργίες που πραγματοποιούνται από μία συγκεκριμένη διεργασία. Εάν χρησιμοποιείται το βασικό μοντέλο είναι της μορφής (BeginTransaction (ReadTmVar | WriteTmVar)* (CommitTransaction | AbortTransaction)), ενώ εάν χρησιμοποιείται το επεκτεταμένο μοντέλο είναι της μορφής (BeginTransaction (ReadTmVar | AccessForUpdateTmVar)* (CommitTransaction | AbortTransaction)). Σημειώνεται ότι μία διεργασία μπορεί να εκκινήσει πολλές δοσοληψίες (σειριακά, τη μία μετά την άλλη). Για το λόγο αυτό συμβολίζουμε με $T(i,n_i)$ την n_i-οστή δοσοληψία που εκκινείται από μια διεργασία p_i. Έτσι, κάθε δοσοληψία έχει ένα μοναδικό αναγνωριστικό. Η p_i ονομάζεται *δημιουργός* (*initiator*) της $T(i,n_i)$. Επίσης, κάθε δοσοληψία $T(i,n_i)$ διατηρεί μια δομή δεδομένων $Trec(i,n_i)$ στην οποία αποθηκεύει οποιαδήποτε πληροφορία επιθυμεί. Η δομή αυτή μπορεί να είναι διαμοιραζόμενη, δηλαδή προσπελάσιμη από άλλες δοσοληψίες, ή όχι. Χάριν απλούστευσης, πολλές φορές αναφέρουμε ότι μια δοσοληψία $T(i,n_i)$ εκτελεί κάποια λειτουργία ή εντολή, εννοώντας ότι η διεργασία που την εκκίνησε, δηλαδή η p_i, εκτελεί τη συγκεκριμένη λειτουργία ή εντολή κατά τη διάρκεια εκτέλεσης της $T(i,n_i)$.

Έστω $T(i,n_i)$ μια δοσοληψία που εκτελείται σε μια εκτέλεση α του συστήματος STM. Ο δημιουργός p_i της $T(i,n_i)$ εκκινεί την εκτέλεσή της καλώντας τη λειτουργία BeginTransaction. Συμβολίζουμε ως $Cs(i,n_i)$ την τελευταία καθολική κατάσταση της α που προηγείται της κλήσης της λειτουργίας BeginTransaction από την p_i. Συμβολίζουμε ως $Cf(i,n_i)$ τη πρώτη καθολική κατάσταση που έπεται της ολοκλήρωσης της εκτέλεσης μιας εκ των λειτουργιών CommitTransaction και

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

AbortTransaction από την p_i (εάν αυτό συμβαίνει). Η $Cf(i,n_i)$ ίσως δεν ορίζεται επειδή ο δημιουργός p_i της $T(i,n_i)$ δεν καλέσει ποτέ μία εκ των λειτουργιών CommitTransaction και AbortTransaction, επειδή για παράδειγμα η p_i έχει καταρρεύσει. Το *διάστημα εκτέλεσης* μιας δοσοληψίας $T(i,n_i)$ που συμβολίζεται ως $E(T(i,n_i))$, ορίζεται να είναι το τμήμα εκτέλεσης της α που ξεκινά από την $Cs(i,n_i)$ και καταλήγει στην $Cf(i,n_i)$ (εάν αυτή ορίζεται). Αν η $Cf(i,n_i)$ δεν ορίζεται το $E(T(i,n_i))$ είναι το επίθεμα της εκτέλεσης α που ξεκινά από την $Cs(i,n_i)$.

Έστω α μια εκτέλεση ενός αλγορίθμου STM. Λέμε ότι μια δοσοληψία $T(i,n_i)$ *ανήκει* στην α και γράφουμε $T(i,n_i) \in \alpha$, εάν υπάρχει κάποιο γεγονός της $T(i,n_i)$ που εκτελείται από την $T(i,n_i)$ στην α. Εάν η $T(i,n_i)$ καλεί τη λειτουργία CommitTransaction στην α και η λειτουργία αυτή επιστρέψει την τιμή TRUE, λέμε ότι η $T(i,n_i)$ *ολοκληρώνεται επιτυχώς* στην α. Εάν μια δοσοληψία $T(i,n_i)$ καλέσει τη λειτουργία AbortTransaction στην α, λέμε ότι η $T(i,n_i)$ *ολοκληρώνεται μη επιτυχώς* στην α. Σημειώνεται ότι η κλήση της λειτουργίας CommitTransaction από την $T(i,n_i)$ στην α, μπορεί να επιστρέψει την τιμή FALSE, στην περίπτωση αυτή λέμε ότι η $T(i,n_i)$ *ολοκληρώθηκε μη-επιτυχώς* στην α. Λέμε ότι η $T(i,n_i)$ *τερματίζεται βίαια ως αποτυχημένη* στην α εάν η $T(i,n_i)$ ολοκληρωθεί ως μη-επιτυχημένη στην α αλλά η $T(i,n_i)$ δεν εκτέλεσε τη λειτουργία AbortTransaction στην α. Μια δοσοληψία που έχει ολοκληρωθεί επιτυχώς ή μη-επιτυχώς λέμε ότι έχει *ολοκληρωθεί*, σε αντίθετη περίπτωση λέμε ότι είναι *εκκρεμής*. Μια δοσοληψία $T(i,n_i)$ *προηγείται* μιας άλλης δοσοληψίας $T(j,n_j)$ στην α, εάν η $T(i,n_i)$ έχει ολοκληρωθεί και το τελευταίο γεγονός της $T(i,n_i)$ προηγείται του πρώτου γεγονότος της $T(j,n_j)$ στην α. Λέμε ότι οι δοσοληψίες $T(i,n_i)$ και $T(j,n_j)$ *εκτελούνται παράλληλα* στην α εάν ούτε η $T(i,n_i)$ προηγείται της $T(j,n_j)$ ούτε η $T(j,n_j)$ προηγείται της $T(i,n_i)$. Υποθέτουμε ότι δοσοληψίες που εκκινούνται από την ίδια διεργασία δεν εκτελούνται ποτέ παράλληλα, ή αλλιώς κάθε διεργασία μπορεί να έχει το πολύ μία εκκρεμή δοσοληψία κάθε χρονική στιγμή. Αυτό σημαίνει ότι στην παρούσα εργασία δεν ασχολούμαστε με εμφωλευμένες δοσοληψίες. Λέμε ότι μια εκτέλεση α είναι *σειριακή* εάν για κάθε ζεύγος δοσοληψιών $(T(i,n_i),T(j,n_j))$ που ανήκουν στην α, είτε η $T(i,n_i)$ προηγείται της $T(j,n_j)$, είτε η $T(j,n_j)$ προηγείται της $T(i,n_i)$.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Λέμε ότι μια δοσοληψία $T(i,n_i)$ *προσπελάζει για ανάγνωση* ή *διαβάζει* μια t-μεταβλητή x, εάν η $T(i,n_i)$ καλέσει τη λειτουργία ReadTmVar για την x. Λέμε ότι μια δοσοληψία $T(i,n_i)$ *προσπελάζει για ενημέρωση* ή *ενημερώνει* μια t-μεταβλητή x, εάν η $T(i,n_i)$ καλέσει τη λειτουργία WriteTmVar για την x στο βασικό μοντέλο STM, ή την AccessForUpdateTmVar για την x στο επεκτεταμένο μοντέλο STM. Εάν πριν την ανάγνωση μιας t-μεταβλητής tv από μια δοσοληψία $T(i,n_i)$, δεν προηγείται κάποια ενημέρωση της tv από την $T(i,n_i)$, τότε η ανάγνωση αυτή ονομάζεται *καθολική ανάγνωση*. Ομοίως, εάν πριν την ενημέρωση μιας t-μεταβλητής tv από μια δοσοληψία $T(i,n_i)$, δεν προηγείται κάποια ανάγνωση της tv από την $T(i,n_i)$, τότε η ενημέρωση αυτή ονομάζεται *καθολική ενημέρωση*. Μια δοσοληψία ονομάζεται *στατική* εάν είναι γνωστό εκ των προτέρων (πριν την εκκίνησή της) το σύνολο των t-μεταβλητών το οποίο πρόκειται να προσπελάσει για ανάγνωση ή ενημέρωση. Αντίθετα, μια *δυναμική* δοσοληψία δεν γνωρίζει εκ των προτέρων το σύνολο των t-μεταβλητών που θα προσπελάσει κάποιες εκ των οποίων μπορεί να δημιουργούνται δυναμικά από την ίδια τη δοσοληψία. Σημειώνεται ότι στο επόμενο κεφάλαιο θα παρουσιαστούν υπάρχοντες αλγόριθμοι που υλοποιούν μνήμη STM, από τους οποίους ένας μόνο υλοποιεί στατικές δοσοληψίες. Μια δοσοληψία που προσπελάζει t-μεταβλητές μόνο για ανάγνωση ονομάζεται *δοσοληψία ανάγνωσης μόνο* και μία δοσοληψία που προσπελάζει t-μεταβλητές μόνο για ενημέρωση ονομάζεται *δοσοληψία ενημέρωσης μόνο*.

2.3.3. Ιδιότητες ατομικού αντικειμένου STM

Στην παρούσα ενότητα παρουσιάζεται αρχικά η ιδιότητα ορθότητας που πρέπει να ικανοποιεί μια υλοποίηση ενός αντικειμένου STM, ώστε να είναι ορθή, και στη συνέχεια παρουσιάζονται κάποιες ιδιότητες τερματισμού που μπορεί να ικανοποιεί η υλοποίηση αυτή.

Όπως αναφέρθηκε στην Ενότητα 2.2.2, η πιο ευρέως διαδεδομένη ιδιότητα ορθότητας για τα ατομικά αντικείμενα είναι η *σειριοποιησιμότητα* [13]. Έστω Y μια υλοποίηση ενός διαμοιραζόμενου αντικειμένου O. Διαισθητικά, η Y είναι σειριοποιήσιμη αν σε κάθε εκτέλεση που παράγει κάθε λειτουργία που εμπεριέχεται στην εκτέλεση εμφανίζεται σαν να εκτελείται στιγμιαία (ατομικά) σε κάποιο σημείο μεταξύ της

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

κλήσης της και της απάντησής της. Όμως, οι υλοποιήσεις STM απαιτούν μεγαλύτερες εγγυήσεις αφού *όλες* οι λειτουργίες (ReadTmVar, WriteTmVar ή AccessForUpdateTmVar) που εκτελεί μια δοσοληψία (και όχι η κάθε λειτουργία ξεχωριστά) πρέπει να εμφανίζονται σαν να εκτελούνται στιγμιαία σε κάποιο σημείο του διαστήματος εκτέλεσης της δοσοληψίας. Για το λόγο αυτό επανα-ορίζουμε την ιδιότητα της σειριοποιησιμότητας για να εκφράσει τις απαιτήσεις ενός συστήματος STM, όπως περιγράφεται στη συνέχεια.

Έστω μια εκτέλεση α μιας υλοποίησης STM. Λέμε ότι η α είναι σειριοποιήσιμη εάν μπορούν να γίνουν τα εξής:

i) Για κάθε δοσοληψία T που έχει ολοκληρωθεί επιτυχώς στην α μπορεί να επιλεχθεί ένα σημείο σειριοποίησης *T κάπου στο διάστημα εκτέλεσης της T στην α,
ii) Να επιλεχθεί ένα υποσύνολο Φ από εκκρεμείς δοσοληψίες τέτοιο ώστε: για κάθε δοσοληψία T στο σύνολο Φ, να μπορεί να επιλεχθεί ένα σημείο σειριοποίησης *T κάπου μετά από την εκκίνηση της T στην α, και
iii) αυτά τα σημεία σειριοποίησης πρέπει να επιλεχθούν με τέτοιο τρόπο ώστε, εάν οι δοσοληψίες εκτελούνταν σειριακά, με βάση τη σειρά που ορίζεται από τα σημεία σειριοποίησης, οι λειτουργίες που εκτελούνται σε αυτές να επιστρέφουν τις ίδιες αποκρίσεις με αυτές στην παράλληλη εκτέλεση α.

Στη συνέχεια παρουσιάζονται κάποιες ιδιότητες τερματισμού που μπορεί να ικανοποιεί η υλοποίηση ενός ατομικού αντικειμένου STM. Λέμε ότι μία υλοποίηση STM ικανοποιεί την ιδιότητα τερματισμού *ελευθερία ανταγωνισμού* (*obstruction-free*) [11] εάν κάθε διεργασία που δεν καταρρέει και εκτελείται χωρίς ανταγωνισμό ξεκινώντας από οποιοδήποτε σημείο της εκτέλεσης μιας δοσοληψίας, ολοκληρώνει τη δοσοληψία μετά από πεπερασμένο αριθμό βημάτων. Λέμε ότι μια υλοποίηση STM ικανοποιεί την ιδιότητα τερματισμού *ελευθερία κλειδωμάτων* (*lock-free*) [8] εάν σε κάθε σημείο της εκτέλεσης υπάρχει τουλάχιστον μία διεργασία από αυτές που δεν έχουν καταρρεύσει η οποία μπορεί να ολοκληρώσει τη δοσοληψία που εκτελεί μέσα σε πεπερασμένο αριθμό βημάτων (ανεξάρτητως του ανταγωνισμού που θα συναντήσει). Λέμε ότι μια υλοποίηση STM ικανοποιεί την ιδιότητα τερματισμού *ελευθερία αναμονής* (*wait-free*) [8] εάν κάθε διεργασία που δεν καταρρέει

ολοκληρώνει οποιαδήποτε δοσοληψία εκτελεί μέσα σε πεπερασμένο αριθμό βημάτων.

Μια υλοποίηση STM που εγγυάται οποιαδήποτε από τις παραπάνω ιδιότητες, ονομάζεται *μη-εμποδιστική*. Σε αντίθετη περίπτωση, ονομάζεται *εμποδιστική*. Μια υλοποίηση STM που ικανοποιεί την ιδιότητα τερματισμού ελευθερία κλειδωμάτων ικανοποιεί και την ιδιότητα ελευθερία ανταγωνισμού. Μια υλοποίηση STM που ικανοποιεί την ιδιότητα τερματισμού ελευθερία αναμονής ικανοποιεί και την ιδιότητα ελευθερία κλειδωμάτων. Εάν μια υλοποίηση STM είναι μη-εμποδιστική και δεν ικανοποιεί την ιδιότητα τερματισμού ελευθερία αναμονής, τότε μπορεί να οδηγήσει μια διεργασία p_i σε *κατάσταση παρατεταμένης στέρησης* (*starvation*). Αυτό σημαίνει ότι η p_i δεν καταφέρνει να ολοκληρώσει τη δοσοληψία που εκτελεί σε πεπερασμένο πλήθος βημάτων. Εάν μια υλοποίηση STM είναι μη-εμποδιστική και δεν ικανοποιεί την ιδιότητα τερματισμού ελευθερία κλειδωμάτων, τότε μπορεί να οδηγήσει το σύστημα σε *κατάσταση καθολικής παρατεταμένης στέρησης* (*livelock*). Αυτό σημαίνει ότι καμία δοσοληψία στο σύστημα δεν ολοκληρώνει την εκτέλεση της δοσοληψίας που εκτελεί σε πεπερασμένο πλήθος βημάτων.

2.3.4. Μετρικές Πολυπλοκότητας

Για την αξιολόγηση ενός αλγορίθμου STM χρησιμοποιούμε δύο θεωρητικές μετρικές, την πολυπλοκότητα χρόνου και την πολυπλοκότητα χώρου. Ορίζουμε ως *πολυπλοκότητα χρόνου* μιας λειτουργίας, το μέγιστο πλήθος βημάτων που μια διεργασία πρέπει να πραγματοποιήσει σε οποιαδήποτε εκτέλεση ώστε να εκτελέσει τη λειτουργία. Ορίζουμε την *πολυπλοκότητα χρόνου* ενός αλγορίθμου STM να είναι το μέγιστο των πολυπλοκοτήτων χρόνου των λειτουργιών του. Ορίζουμε ως *πολυπλοκότητα χώρου* ενός αλγορίθμου STM, το πλήθος και το μέγεθος των καταχωρητών που χρησιμοποιούνται από τον αλγόριθμο για την υλοποίηση των λειτουργιών του μοντέλου STM, καθώς και το *είδος* τους. Συχνά, μελετάμε τη χρονική πολυπλοκότητα ενός αλγορίθμου STM υπό ειδικές συνθήκες, για παράδειγμα όταν δεν υπάρχει ανταγωνισμός κατά την πρόσβαση στην κοινή μνήμη.

Επίσης για την αξιολόγηση της *απόδοσης* ενός αλγορίθμου STM συχνά πραγματοποιείται πειραματική μελέτη. Κάποιες από τις μετρικές που αναλύονται από τις υπάρχουσες πειραματικές μελέτες είναι η *ικανότητα διεκπεραίωσης* (*throughput*) και ο *μέσος χρόνος διεκπεραίωσης* των δοσοληψιών. Η ικανότητα διεκπεραίωσης αναφέρεται στο πλήθος των δοσοληψιών που ολοκληρώνονται επιτυχώς ανά μονάδα χρόνου, ενώ ο μέσος χρόνος διεκπεραίωσης αναφέρεται στο μέσο χρόνο διεκπεραίωσης μιας δοσοληψίας ως επιτυχημένης. Στην περίπτωση της μελέτης του μέσου χρόνου διεκπεραίωσης μιας δοσοληψίας, υποθέτουμε ότι μια δοσοληψία επανεκτελείται έως ότου ολοκληρωθεί επιτυχώς.

2.3.5. Συγκρούσεις

Λέμε ότι παρουσιάζεται *σύγκρουση* (conflict) μεταξύ δύο ταυτόχρονα εκκρεμών δοσοληψιών $T(i,n_i)$ και $T(j,n_j)$ εάν υπάρχουν λειτουργίες op_i που εκτελείται από την $T(i,n_i)$ και op_j που εκτελείται από την $T(j,n_j)$, οι οποίες εκτελούνται στην ίδια t-μεταβλητή tv και μια εκ των op_i και op_j είναι λειτουργία ενημέρωσης (ενώ η άλλη μπορεί να είναι είτε ανάγνωσης είτε ενημέρωσης). Έτσι υπάρχουν τρία είδη συγκρούσεων ανάλογα με το ποια εκ των op_i και op_j είναι λειτουργία ενημέρωσης και ανάλογα με το είδος της άλλης. Αν η op_i προηγείται της op_j, η op_i είναι λειτουργία ενημέρωσης της tv, η op_j λειτουργία ανάγνωσης της tv και η op_j προηγείται της ολοκλήρωσης της $T(i,n_i)$, η σύγκρουση ονομάζεται *σύγκρουση εγγραφής-ανάγνωσης* ή *σύγκρουση W-R* εν συντομία. Αν η op_i προηγείται της op_j, η op_i είναι λειτουργία ανάγνωσης της tv, η op_j λειτουργία ενημέρωσης της tv και η op_j προηγείται της ολοκλήρωσης της $T(i,n_i)$, η σύγκρουση ονομάζεται *σύγκρουση ανάγνωσης-εγγραφής* ή *σύγκρουση R-W* εν συντομία. Αν η op_i προηγείται της op_j, η op_i είναι λειτουργία ενημέρωσης της tv, η op_j είναι επίσης λειτουργία ενημέρωσης της tv και η op_j προηγείται της ολοκλήρωσης της $T(i,n_i)$, η σύγκρουση αυτή ονομάζεται *σύγκρουση εγγραφής-εγγραφής* ή *σύγκρουση W-W* εν συντομία. Στη συνέχεια, εξηγείται ο τρόπος με τον οποίο κάθε ένα από τα παραπάνω είδη συγκρούσεων μπορεί να προκαλέσει πρόβλημα εάν δεν αντιμετωπιστεί, το οποίο σημαίνει ότι μπορεί να καταστρατηγηθεί η ιδιότητα της σειριοποιησιμότητας που κάθε αλγόριθμος STM πρέπει να ικανοποιεί.

Είναι αξιοσημείωτο ότι η εμφάνιση ενός είδους σύγκρουσης δε συνεπάγεται πάντα την καταστρατήγηση της ιδιότητας της σειριοποιησιμότητας, παρά μόνο σε συγκεκριμένα σενάρια εκτελέσεων. Στη συνέχεια παρουσιάζονται κάποια παραδείγματα τέτοιων «κακών» σεναρίων. Στην περίπτωση της W-R σύγκρουσης, είναι πιθανό η $T(j,n_j)$ να διαβάσει μέσω της op_j ως δεδομένα της tv τα δεδομένα που γράφτηκαν από μια μη-ολοκληρωμένη δοσοληψία, την $T(i,n_i)$, μέσω της op_i. Εάν αυτό συμβαίνει και η $T(i,n_i)$ στη συνέχεια ολοκληρωθεί ως μη επιτυχημένη, σημαίνει ότι η $T(j,n_j)$ διάβασε ασυνεπή δεδομένα. Στην περίπτωση της R-W σύγκρουσης, τα δεδομένα d_i που η $T(i,n_i)$ διαβάζει για την tv μέσω της op_i ενημερώνονται στη συνέχεια από την $T(j,n_j)$ μέσω της op_j πριν η $T(i,n_i)$ ολοκληρωθεί. Εάν η $T(i,n_i)$ διαβάσει και πάλι τα δεδομένα d_i' της tv με τη λειτουργία op_i' μετά την εκτέλεση της op_j, τότε θα ισχύει $d_i \neq d_i'$, το οποίο συνεπάγεται ότι η $T(i,n_i)$ δε μπορεί να σειριοποιηθεί (διότι δε μπορεί να σειριοποιηθεί ούτε πριν την op_j, λόγω της op_i, ούτε μετά από αυτήν, λόγω της op_j). Στην περίπτωση της W-W σύγκρουσης, τα δεδομένα που η $T(i,n_i)$ έγραψε μέσω της op_i για την tv πανογράφονται στη συνέχεια από την $T(j,n_j)$ μέσω της op_j πριν η $T(i,n_i)$ ολοκληρωθεί. Με βάση τη σύγκρουση αυτή, μπορεί να προκληθεί πρόβλημα σειριοποιησιμότητας εάν το παραπάνω σενάριο επαναληφθεί και για μία ακόμα t-μεταβλητή, έστω tv', με την $T(i,n_i)$ να πανογράφει τα δεδομένα που η $T(j,n_j)$ γράφει στην tv, μέσω λειτουργιών ενημέρωσης op_i' και op_j', αντίστοιχα. Στην περίπτωση αυτή η μνήμη είναι σε ασυνεπή κατάσταση διότι τελικά η tv θα περιέχει την τιμή που έγραφε η $T(j,n_j)$ και η tv' την τιμή που έγραψε η $T(i,n_i)$, κάτι το οποίο δε μπορεί να πραγματοποιηθεί εάν οι δύο δοσοληψίες εκτελούνταν σειριακά. Το πρόβλημα σειριοποιησιμότητας παρουσιάζεται εάν μία άλλη δοσοληψία, έστω $T(k,n_k)$ προσπαθήσει να διαβάσει τις tv και tv'. Η $T(k,n_k)$ δε θα μπορεί να σειριοποιηθεί.

Έστω ένας αλγόριθμος STM και α μια οποιαδήποτε εκτέλεσή του. Για να ικανοποιεί ο αλγόριθμος αυτός την ιδιότητα της σειριοποιησιμότητας πρέπει να παρέχει μηχανισμούς αποτροπής ή ανίχνευσης και διαχείρισης των συγκρούσεων που αναφέρθηκαν παραπάνω, στην περίπτωση που αυτές οδηγούν σε ασυνέπειες. Όπως θα παρουσιαστεί στο Κεφάλαιο 3, υπάρχουν διαφορές στον τρόπο με τον οποίο αποτρέπονται, ανιχνεύονται και επιλύονται τα διαφορετικά είδη ασυνεπειών από τον εκάστοτε αλγόριθμο.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

2.3.6. Απαραίτητες συμβάσεις

Στην ενότητα αυτή παρουσιάζονται κάποιες συμβάσεις ως προς τον τρόπο χρήσης των λειτουργιών που υποστηρίζουν τα δύο μοντέλα STM που παρουσιάστηκαν. Οι συμβάσεις αυτές θα πρέπει να ακολουθούνται από τον κώδικα του χρήστη. Μια δοσοληψία εκκινείται με την εκτέλεση της λειτουργίας BeginTransaction, η οποία επιστρέφει ένα δείκτη σε μια κατάλληλη δομή που περιγράφει τη δοσοληψία που μόλις εκκινήθηκε. Ο χρήστης πρέπει να χρησιμοποιεί αυτό το δείκτη σε όλες τις λειτουργίες που θέλει να εκτελέσει μέσω τις συγκεκριμένης δοσοληψίας. Από το σημείο αυτό και έπειτα η δοσοληψία που εκτελεί ο χρήστης είναι εκκρεμής έως ότου εκτελεστεί από το χρήστη είτε η λειτουργία CommitTransaction (για την ολοκλήρωση της δοσοληψίας ως επιτυχημένης) είτε η λειτουργία AbortTransaction (για την ολοκλήρωση της δοσοληψίας ως αποτυχημένης). Σημειώνεται ότι ο χρήστης μπορεί να έχει το πολύ μία εκκρεμή δοσοληψία κάθε χρονική στιγμή, δηλαδή κάθε κλήση της λειτουργίας BeginTransaction πρέπει να ακολουθείται από μια κλήση μιας εκ των λειτουργιών CommitTransaction ή AbortTransaction, χωρίς να παρεμβάλλεται στο ενδιάμεσο κάποια άλλη κλήση της BeginTransaction.

Ο χρήστης μπορεί να προσπελάσει τις t-μεταβλητές που επιθυμεί ή να δημιουργήσει κάποια νέα t-μεταβλητή μέσω μιας δοσοληψίας μόνο ενόσω η δοσοληψία αυτή είναι εκκρεμής, και μέσω των αντίστοιχων λειτουργιών ReadTmVar, WriteTmVar, AccessForUpdateTmVar και CreateNewTmVar. Ο χρήστης είναι ελεύθερος να καλεί τις λειτουργίες προσπέλασης για την ίδια t-μεταβλητή όσες φορές επιθυμεί. Συγκεκριμένα εάν πρόκειται να διαβάσει τα δεδομένα μιας t-μεταβλητής για πρώτη φορά πρέπει να τα προσπελάσει με την ReadTmVar. Κάθε φορά που ο χρήστης θέλει να ενημερώσει κάποια t-μεταβλητή, στο βασικό μοντέλο STM πρέπει να εκτελεί τη λειτουργία WriteTmVar, ενώ στο επεκτεταμένο μοντέλο αρκεί να εκτελέσει τη λειτουργία AccessForUpdateTmVar την πρώτη μόνο φορά και στη συνέχεια να ενημερώνει απευθείας τα δεδομένα της t-μεταβλητής μέσω του δείκτη που του επιστράφηκε από αυτή. Επίσης, στην περίπτωση του επεκτεταμένου μοντέλου, εάν μια t-μεταβλητή tv έχει προσπελαστεί μέσω της λειτουργίας AccessForUpdateTmVar και στη συνέχεια ο χρήστης επιθυμεί να διαβάσει τα δεδομένα της tv, δεν είναι απαραίτητο να τα προσπελάσει και πάλι μέσω της ReadTmVar, αλλά μπορεί να τα

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

διαβάσει άμεσα μέσω του δείκτη που του επιστράφηκε από την προηγούμενη κλήση της AccessForUpdateTmVar.

Όπως αναφέρθηκε στην Ενότητα 2.3.1, έχει θεωρηθεί ότι τα δεδομένα των t-μεταβλητών είναι δείκτες. Επομένως, το σύστημα STM εξασφαλίζει την ατομική προσπέλαση των δεικτών αυτών από τις δοσοληψίες. Εάν οι δείκτες αυτοί περιγράφουν δεδομένα σε διαφορετικές θέσεις τις μνήμης, τότε τα δεδομένα αυτά θα προσπελάζονται επίσης ατομικά από τις δοσοληψίες. Επομένως ο χρήστης είναι υπεύθυνος να εγγυηθεί την ατομική (ή μη) προσπέλαση των δεδομένων στα οποία δείχνουν οι δείκτες που προφυλάσσονται από το σύστημα STM, ενημερώνοντάς τους με δείκτες που δείχνουν σε νέες (ή ήδη χρησιμοποιούμενες) θέσεις μνήμης.

Όταν ο χρήστης επιθυμεί να ολοκληρώσει μια δοσοληψία είναι υποχρεωμένος να καλέσει είτε τη λειτουργία CommitTransaction για να επιχειρήσει να ολοκληρώσει τη δοσοληψία ως επιτυχημένη ή τη λειτουργία AbortTransaction για να την ολοκληρώσει ως μη-επιτυχημένη. Εάν οποιαδήποτε λειτουργία προσπέλασης επιστρέψει στο χρήστη για την Boolean μεταβλητή την τιμή FALSE, δηλώνοντας ότι η συγκεκριμένη λειτουργία απέτυχε, ο χρήστης είναι υποχρεωμένος να καλέσει άμεσα μία εκ των λειτουργιών CommitTransaction ή AbortTransaction ώστε να ολοκληρώσει τη δοσοληψία που εκτελεί. Σημειώνεται ότι στην περίπτωση αυτή είναι βέβαιο, ανεξάρτητα με τη λειτουργία τερματισμού που καλείται, ότι η δοσοληψία θα ολοκληρωθεί ως μη-επιτυχημένη.

Ο χρήστης είναι υποχρεωμένος να χρησιμοποιεί τη λειτουργία CreateNewTmVar, κάθε φορά που επιθυμεί να δημιουργήσει και να αρχικοποιήσει μια νέα t-μεταβλητή κατά την εκτέλεση μιας δοσοληψίας. Για παράδειγμα, αν ο χρήστης επιθυμεί να υλοποιήσει μια συνδεδεμένη λίστα βάσει κάποιου ατομικού αντικειμένου STM πρέπει να αντιστοιχίσει μια t-μεταβλητή σε κάθε κόμβο της λίστας, η οποία αποθηκεύει ένα δείκτη στον κόμβο αυτό. Στην περίπτωση αυτή, εάν ο χρήστης θέλει να διατηρήσει έναν δείκτη p σε κάποιο κόμβο x προς κάποιο κόμβο y της λίστας, ο p δεν επιτρέπεται να δείχνει απευθείας στο y, αλλά πρέπει να δείχνει στην t-μεταβλητή που περιγράφει τον κόμβο y της λίστας. Αυτό σημαίνει ότι το μοντέλο STM δεν επιτρέπει στον χρήστη να διατηρεί δείκτες μεταξύ των ίδιων των διαμοιραζόμενων

δεδομένων, αλλά ένα διαμοιραζόμενο δεδομένο x μπορεί να περιέχει δείκτη σε ένα διαμοιραζόμενο δεδομένο y μόνο εάν ο δείκτης δείχνει στην αντίστοιχη t-μεταβλητή που περιγράφει το y.

2.3.7. Παράδειγμα χρήσης της Software Transactional Μνήμης

Στην παρούσα ενότητα θα δοθεί ένα παράδειγμα χρήσης του ατομικού αντικειμένου της μνήμης STM για την υλοποίηση της λειτουργίας εισαγωγής δεδομένων σε μια ταξινομημένη κατά αύξουσα διάταξη απλά συνδεδεμένη λίστα, η οποία μπορεί να προσπελάσετε ταυτόχρονα από πολλές διεργασίες. Έστω, λοιπόν, ότι έχουμε μια απλά συνδεδεμένη λίστα κάθε κόμβος της οποίας περιέχει έναν ακέραιο αριθμό *num*. Η μορφή που κάθε κόμβος *node* της λίστας θα είχε εάν η λίστα προσπελάζονταν από μία μόνο διεργασία παρουσιάζεται στο Σχήμα 2.4 (α).

```
struct node {
    int num;
    struct node *next;
};
```

(α)

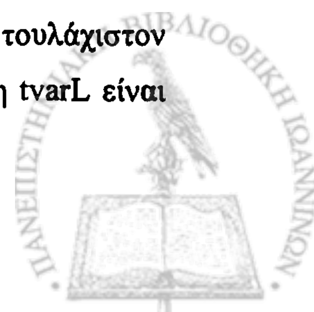
```
struct tmnode {
    int num;
    TmVar next;
};
```

(β)

Σχήμα 2.4: α) Η μορφή του κόμβου τύπου node της συνδεδεμένης λίστας. β) Η μορφή του κόμβου τύπου tmnode της συνδεδεμένης λίστας.

Για την υλοποίηση της λίστας με τη χρήση μιας βιβλιοθήκης που παρέχει ατομικά αντικείμενα STM πρέπει κάθε κόμβος της λίστας να περιγράφεται από μία t-μεταβλητή (αποθηκεύοντας στην t-μεταβλητή έναν δείκτη στον κόμβο). Η μορφή που κάθε κόμβος *tmnode* της λίστας θα έχει στο σύστημα STM παρουσιάζεται στο Σχήμα 2.4 (β). Όπως έχει αναφερθεί το μοντέλο STM ορίζει ότι ο δείκτης που κάθε κόμβος περιέχει για τον επόμενο κόμβο του πρέπει να δείχνει στην t-μεταβλητή που περιγράφει τον αντίστοιχο κόμβο και όχι στον ίδιο τον κόμβο. Για τον λόγο αυτό έχει τροποποιηθεί ο τύπος της μεταβλητής next του κόμβου τύπου tmnode από struct node * (που ήταν στον κόμβου τύπου node) σε TmVar.

Ας υποθέσουμε, χάριν απλότητας, ότι στην ταξινομημένη λίστα υπάρχει τουλάχιστον ένας κόμβος L ο οποίος περιγράφεται από την t-μεταβλητή tvarL και η tvarL είναι


ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

κατάλληλα αρχικοποιημένη. Επίσης για λόγους απλότητας θεωρούμε ότι ο κόμβος αυτός αποτελεί ταυτόχρονα και κόμβο φρουρό, δηλαδή έχει την μικρότερη δυνατή τιμή num και δεν μπορεί να διαγραφεί. Επομένως ο κόμβος L είναι το πρώτο στοιχείο της λίστας και παραμένει πρώτο μετά από κάθε εισαγωγή και διαγραφή.

Στη συνέχεια θα παρουσιαστεί ο κώδικας της συνάρτησης Insert για την εισαγωγή ενός κόμβου *x* στην ταξινομημένη αυτή λίστα. Η συνάρτηση αυτή επιστρέφει την τιμή TRUE εάν ο κόμβος εισήχθη στη λίστα, ή την τιμή FALSE εάν υπήρχε ήδη στη λίστα. Αρχικά, στο Σχήμα 2.5 παρουσιάζεται ο κώδικας της συνάρτησης Insert όταν η λίστα μπορεί να προσπελάζεται από μία μόνο διεργασία, την οποία ονομάζουμε *InsertS*. Ο κώδικας είναι αρκετά απλός. Με τον βρόχο των γραμμών 4-5 προσπαθούμε να εντοπίσουμε το σημείο στο οποίο πρέπει να μπει ο νέος κόμβος, διατηρώντας πάντα δείκτη στον κόμβο που θα χρειαστεί να τροποποιήσουμε (prevP). Εάν ο κόμβος υπάρχει ήδη επιστρέφουμε την τιμή FALSE, αλλιώς εισάγουμε τον νέο κόμβο μετά από τον κόμβο prevP. (Σημειώνεται ότι από υπόθεση ο νέος κόμβος δε μπορεί να εισαχθεί στην αρχή της λίστας, λόγω του κόμβου φρουρού L. Επομένως ο δείκτης prevP δε μπορεί να είναι ποτέ null).

```
1.  boolean InsertS (struct node *L, int x) {
2.      struct node *P, *prevP, *newNode;
3.      P = L;
4.      while ( P!=null && P->num < x)   // 1. Εντοπισμός του σημείου εισαγωγής του νέου στοιχείου
5.          prevP = P;
6.          P = P->next;
7.      if (P!=null && P->num==x)        // 2. Εάν το στοιχείο υπάρχει ήδη ...
8.          return (FALSE);
9.      else                             // 3. Εάν το στοιχείο δεν υπάρχει ήδη ...
10.         newNode = malloc(sizeof(struct node)); // 3.1. Αρχικοποίηση του νέου κόμβου
11.         newNode->num = x;
12.         newNode->next = P;
13.         prevP->next = newNode;       // 3.2 Ενημέρωση των δεδομένων του παλιού κόμβου
14.         return (TRUE);
```

Σχήμα 2.5: Κώδικας της συνάρτησης εισαγωγής ενός στοιχείου x σε μια ταξινομημένη κατά αύξουσα διάταξη απλά συνδεδεμένη λίστα, η οποία μπορεί να προσπελάζεται από μία μόνο διεργασία.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Στο Σχήμα 2.6 παρουσιάζεται ο κώδικας της συνάρτησης Insert όταν η λίστα μπορεί να προσπελάζεται ταυτόχρονα από πολλές διεργασίες, χρησιμοποιώντας το βασικό αντικείμενο STM, την οποία ονομάζουμε InsertSTM.

```
1.  boolean InsertSTM (TmVar tvarL, int x) {
2.      void *t;
3.      struct tmnode *P, *prevP, *newNode, *newprevNode;
4.      TmVar tvarP, tvarprevP, tvarnewNode;
5.      Boolean bool;
6.      while (1)
7.          t = BeginTransaction ();              // 1. Εκκίνηση μιας νέα δοσοληψίας
8.          tvarP = tvarL;
9.          (bool, P) = ReadTmVar (t, tvarP);     // 2. Ανάγνωση των δεδομένων του πρώτου στοιχείου
10.         Check(bool);
11.         while ( P!=null && P->num < x)        // 3. Εντοπισμός του σημείου εισαγωγής του νέου στοιχείου
12.             tvarprevP = tvarP;
13.             tvarP = P->next;
14.             prevP = P;
15.             (bool, P) = ReadTmVar (t, P->next);
16.             Check(bool);
17.         if (P!=null && P->num==x)             // 4. Εάν το στοιχείο υπάρχει ήδη ...
18.             AbortTransaction(t);
19.             return (FALSE);
20.         else                                  // 5. Εάν το στοιχείο δεν υπάρχει ήδη ...
21.             newNode = malloc(sizeof(struct tmnode));   // 5.1. Αρχικοποίηση του νέου κόμβου
22.             newNode->num = x;
23.             newNode->next = tvarP;
    // 5.2. Δημιουργία μιας νέας t-μεταβλητής και αντιστοίχιση του νέου κόμβου στη t-μεταβλητή αυτή
24.             tvarnewNode = CreateNewTmVar (t, newNode);
    // 5.3 Ενημέρωση των δεδομένων του παλιού κόμβου (γρ. 25 - 30)
25.             newprevNode = malloc(sizeof(struct tmnode));
26.             newprevNode->num = prevP->num;
27.             newprevNode->next = tvarnewNode;
28.             bool WriteTmVar (t, tvarprevP, newprevNode);
29.             Check(bool);
30.             if (CommitTransaction (t) == true)
31.                 return (TRUE);
```

Σχήμα 2.6: Κώδικας της συνάρτησης εισαγωγής ενός στοιχείου x σε μια ταξινομημένη κατά αύξουσα διάταξη απλά συνδεδεμένη λίστα, η οποία μπορεί να προσπελάσετε ταυτόχρονα από πολλές διεργασίες, με βάση το βασικό μοντέλο STM.

Από την πρώτη γραμμή του κώδικα των συναρτήσεων InsertS και InsertSTM παρατηρούμε ότι στην InsertS αναφερόμαστε απευθείας σε κάποιον κόμβο της λίστας

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

(στον πρώτο κόμβο στην προκειμένη περίπτωση, με το L), ενώ στην InsertSTM αναφερόμαστε στον ίδιο κόμβο μέσω της t-μεταβλητής που τον περιγράφει (tvarL). Αρχικά, στην γραμμή 7 της InsertSTM εκκινείται μία νέα δοσοληψία t καλώντας τη λειτουργία BeginTransaction. H t είναι εκκρεμής από το σημείο αυτό και έως ότου κληθεί μια εκ των CommitTransaction και AbortTransaction για την t. Έπειτα στη γραμμή 9, διαβάζουμε τα δεδομένα (P) του πρώτου στοιχείου της λίστας (tvarL) με τη λειτουργία ReadTmVar του αντικειμένου STM και στη γραμμή 10 ελέγχουμε με τη μακροεντολή *Check* εάν η λειτουργία αυτή ολοκληρώθηκε επιτυχώς, δηλαδή εάν πράγματι επέστρεψε bool == TRUE. Εάν αυτό δεν ισχύει, τότε η δοσοληψία t ολοκληρώνεται ως μη-επιτυχημένη καλώντας την αντίστοιχη λειτουργία AbortTransaction. O κώδικας της Check παρουσιάζεται στο Σχήμα 2.7. Χάριν απλότητας του κώδικα της συνάρτησης InsertSTM, επιλέχθηκε να αποφευχθεί η ενσωμάτωση του κώδικα της Check στην InsertSTM. Σημειώνεται ότι η ανάγνωση των δεδομένων του πρώτου κόμβου της λίστας μέσω της εκτέλεσης της συνάρτησης ReadTmVar στη γραμμή 9 και η εκτέλεση της Check στην γραμμή 10 της InsertSTM, αντικατέστησαν την αντίστοιχη ανάγνωση της γραμμής 3 της InsertS. Αυτός είναι και ο τρόπος ανάγνωσης των δεδομένων των κόμβων της λίστας με τη χρήση του αντικειμένου STM.

```
1. #define Check (bool):
2.        if ( bool == false)
3.           AbortTransaction (t);
4.           goto 6;
```

Σχήμα 2.7: Κώδικας της μακροεντολής Check.

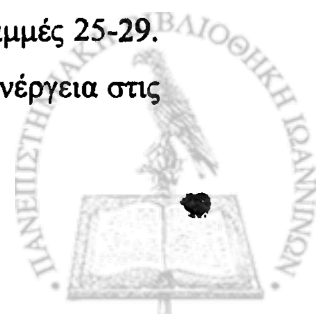
Στη συνέχεια, ο βρόχος των γραμμών 11-16 της InsertSTM αντικαθιστά τον βρόχο 4-5 της InsertS, και εντοπίζει το σημείο στο οποίο πρέπει να εισαχθεί το νέο στοιχείο. Σημειώνεται ότι στην περίπτωση της InsertSTM, εκτός από τα δεδομένα (prevP) του κόμβου που θα τροποποιηθεί πρέπει να διατηρείται και η t-μεταβλητή που περιγράφει το στοιχείο αυτό (tvarprevP). Υπενθυμίζεται ότι ο δείκτης P→next που εισάγεται ως όρισμα στη ReadTmVar της γραμμής 15, είναι ο δείκτης που δείχνει στον επόμενο κόμβο της λίστας και επομένως δείχνει στην αντίστοιχη t-μεταβλητή που περιγράφει τον κόμβο αυτό, όπως ορίζει το αντικείμενο STM.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Αφού εντοπιστεί το σημείο στο οποίο πρέπει να εισαχθεί το στοιχείο x, ελέγχεται εάν υπάρχει ήδη στη λίστα (γραμμή 17) και εάν αυτό συμβαίνει η συνάρτηση InsertSTM ολοκληρώνεται και επιστρέφει FALSE (γραμμή 19). Πριν την ολοκλήρωσή της πρέπει να τερματίσει και την εκκρεμή δοσοληψία t που εκτελεί, καλώντας την AbortTransaction (γραμμή 18). Εάν το στοιχείο x δεν υπάρχει ήδη στην λίστα τότε θα εισαχθεί μετά από τον κόμβο prevP, ο οποίος περιγράφεται από την t-μεταβλητή tvarprevP. Για την εισαγωγή του στοιχείου x δημιουργείται αρχικά ένας κόμβος newNode της λίστας που το περιέχει (γραμμές 21-22). Ο δείκτης next του νέου αυτού κόμβου (newNode→next) αρχικοποιείται ώστε να δείχνει στην t-μεταβλητή που περιγράφει τον επόμενο κόμβο της λίστας (tvarP). Στη συνέχεια ο νέος αυτός κόμβος newNode πρέπει να περιγραφεί από κάποια καινούργια t-μεταβλητή και το λόγο αυτό εκτελείται η αντίστοιχη λειτουργία CreateNewTmVar, στη γραμμή 24. Στη συνέχεια δημιουργούνται και αρχικοποιούνται τα νέα δεδομένα του κόμβου που πρέπει να ενημερωθεί (tvarprevP) στις γραμμές 25-27 και στη συνέχεια πραγματοποιείται η ενημέρωση του κόμβου αυτού μέσω της εκτέλεσης της λειτουργίας WriteTmVar της γραμμής 28. Σημειώνεται ότι η ενημέρωση των δεδομένων του επιθυμητού κόμβου της λίστας μέσω της δημιουργίας των νέων δεδομένων του κόμβου αυτού (γραμμές 25-27) και της εκτέλεσης της συνάρτησης WriteTmVar στη γραμμή 28, και η εκτέλεση της Check στην γραμμή 29 της InsertSTM, αντικατέστησαν την αντίστοιχη ενημέρωση της γραμμής 12 της InsertS. Αυτός είναι και ο τρόπος ενημέρωσης των δεδομένων των κόμβων της λίστας με τη χρήση του βασικού αντικειμένου STM.

Απομένει η εκκρεμής δοσοληψία t να ολοκληρωθεί επιτυχώς καλώντας τη λειτουργία CommitTransaction (γραμμή 30). Εάν αυτό επιτευχθεί (η λειτουργία CommitTransaction επιστρέψει TRUE) ολοκληρώνεται η συνάρτηση InsertSTM και επιστρέφει TRUE, αλλιώς η t έχει ολοκληρωθεί μη-επιτυχώς και η InsertSTM ξαναπροσπαθεί να εισάγει το στοιχείο x, επαναλαμβάνοντας τη διαδικασία από την αρχή (ο κώδικας επιστρέφει στη γραμμή 6).

Όπως αναφέρθηκε η διαδικασία ενημέρωσης των δεδομένων μιας t-μεταβλητής στο βασικό αντικείμενο STM πραγματοποιείται με τρόπο ανάλογο με τις γραμμές 25-29. Στο σημείο αυτό θα παρουσιαστεί η μορφή που θα είχε η συγκεκριμένη ενέργεια στις



γραμμές αυτές εάν χρησιμοποιόταν το επεκτεταμένο αντικείμενο STM. Οι γραμμές κώδικα για την ενημέρωση των δεδομένων μιας t-μεταβλητής με βάση το επεκτεταμένο αντικείμενο STM που θα αντικαθιστούσαν τις γραμμές 25-29 της InsertSTM παρουσιάζονται στο Σχήμα 2.8. Όπως παρουσιάζεται, η προσπέλαση των δεδομένων της t-μεταβλητής tvarprevP μέσω της αντίστοιχης λειτουργίας AccessForUpdateTmVar, επιστρέφει ένα δείκτη p στα δεδομένα αυτά (τύπου tmnode **), μέσω του οποίου η αντίστοιχη διεργασία μπορεί να ενημερώσει τα δεδομένα της t-μεταβλητής εισάγοντας τον δείκτη προς τα ενημερωμένα δεδομένα του κόμβου prevP (γραμμή 6).

```
1.        newprevNode = malloc(sizeof(node));
2.        newprevNode->num = prevP->num;
3.        newprevNode->next = tvarnewNode;
4.        (bool, p) AccessForUpdateTmVar (t, tvarprevP);
5.        Check(bool);
6.        *p = newprevNode;
```

Σχήμα 2.8: Ενημέρωση των δεδομένων που περιγράφει η t-μεταβλητή tvarprevP, στο επεκτεταμένο αντικείμενο STM.

ΚΕΦΑΛΑΙΟ 3. ΚΑΤΗΓΟΡΙΕΣ ΣΧΕΔΙΑΣΤΙΚΩΝ ΕΠΙΛΟΓΩΝ

3.1 Τρόπος Ανάθεσης, Απόκτησης και Κατάργησης Ιδιοκτησιών
3.2 Χρόνος Απόκτησης Ιδιοκτησιών
3.3 Τρόποι Αποτροπής ή Ανίχνευσης και Επίλυσης Συγκρούσεων και Μηχανισμός Ελέγχου Συνέπειας
3.4 Πλήθος Ενδιάμεσων Επιπέδων
3.5 Μοντέλο Μνήμης

Στο παρών κεφάλαιο θα παρουσιαστούν οι κατηγορίες των σχεδιαστικών επιλογών που θα μελετήσουμε κατά την παρουσίαση των υπαρχόντων αλγορίθμων STM. Αρχικά, συζητείται μία σχεδιαστική επιλογή που ακολουθείται από όλους τους αλγορίθμους που θα μελετηθούν στην παρούσα εργασία. Όπως αναφέρθηκε, κάθε αλγόριθμος STM για να είναι ορθός πρέπει να εξασφαλίζει την ιδιότητα της σειριοποιησιμότητας δοσοληψιών. Η συνηθέστερη μέθοδος για την εξασφάλιση της ιδιότητας αυτής, η οποία ακολουθείται από όλους τους αλγορίθμους που θα παρουσιαστούν, είναι η *μέθοδος απόκτησης ιδιοκτησιών* επί των t-μεταβλητών.

Οι αλγόριθμοι διαφέρουν στη μέθοδο διαχείρισης ιδιοκτησιών. Συγκεκριμένα υπάρχουν διαφορές στον τρόπο ανάθεσης, απόκτησης και κατάργησης των ιδιοκτησιών που θα μελετηθούν στην Ενότητα 3.1, και στον χρόνο απόκτησης των ιδιοκτησιών που μελετάται στην Ενότητα 3.2. Σημειώνεται ότι σε κάθε περίπτωση, όλοι οι αλγόριθμοι που θα παρουσιαστούν στην παρούσα εργασία υποχρεώνουν μια δοσοληψία που θέλει να ενημερώσει κάποια t-μεταβλητή tv, να αποκτήσει την αντίστοιχη ιδιοκτησία της tv, πριν την ενημερώσει. Ονομάζουμε την ιδιοκτησία αυτή

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ιδιοκτησία ενημέρωσης. Μία δοσοληψία που επιθυμεί να διαβάσει κάποια t-μεταβλητή tv μπορεί να το κάνει μόνο εάν καμία δοσοληψία δεν κατέχει την αντίστοιχη ιδιοκτησία ενημέρωσης. Επίσης, ένας αλγόριθμος μπορεί να υποχρεώνει κάθε δοσοληψία να αποκτά την ιδιοκτησία των t-μεταβλητών που επιθυμεί να προσπελάσει για ανάγνωση. Ονομάζουμε την ιδιοκτησία αυτή *ιδιοκτησία ανάγνωσης*. Εάν ένας αλγόριθμος απαιτεί οι δοσοληψίες να αποκτούν ιδιοκτησίες ανάγνωσης, λέμε ότι χρησιμοποιεί *ορατές αναγνώσεις* t-μεταβλητών. Σε αντίθετη περίπτωση ο αλγόριθμος χρησιμοποιεί *μη-ορατές αναγνώσεις*.

Όταν χρησιμοποιούνται ορατές αναγνώσεις αρκεί να αποκτηθούν ιδιοκτησίες ανάγνωσης μόνο στις t-μεταβλητές που μια δοσοληψία προσπελάζει με καθολική ανάγνωση, όπως εξηγείται στη συνέχεια. Έστω α μια εκτέλεση ενός αλγορίθμου STM. Εάν μια δοσοληψία προσπελάσει με καθολική ενημέρωση μια t-μεταβλητή tv, έστω στην καθολική κατάσταση C της α, τότε οποιαδήποτε ανάγνωση της tv από την $T(i,n_i)$ μετά τη C θα επιστρέφει την τιμή που η ίδια η $T(i,n_i)$ έχει εγγράψει στη C. Επομένως, στην περίπτωση αυτή, εάν χρησιμοποιούνται ορατές αναγνώσεις, η $T(i,n_i)$ δεν χρειάζεται εάν προσπελάσει για ανάγνωση την tv μετά την C, να αποκτήσει την ιδιοκτησία ανάγνωσης της tv.

Σημειώνεται ότι όλοι οι υπάρχοντες αλγόριθμοι STM που χρησιμοποιούν ορατές αναγνώσεις υλοποιούν με τον ίδιο τρόπο το μηχανισμός διαχείρισης ιδιοκτησιών για τις ιδιοκτησίες ανάγνωσης και ενημέρωσης. Επομένως, εάν μια δοσοληψία $T(i,n_i)$ αποκτήσει ιδιοκτησία ανάγνωσης σε μια t-μεταβλητή tv, τότε καμία δοσοληψία δε μπορεί να προσπελάσει για ανάγνωση ή ενημέρωση την tv πριν την κατάργηση της ιδιοκτησίας της από την $T(i,n_i)$. Αυτό οδηγεί σε μειωμένη απόδοση του συστήματος, διότι κάποια δοσοληψία που επιθυμεί να διαβάσει την tv θα έπρεπε να μπορεί να το κάνει χωρίς να περιμένει την $T(i,n_i)$ να καταργήσει την ιδιοκτησία ανάγνωσης της tv, δεδομένου ότι η $T(i,n_i)$ δεν προτίθεται να ενημερώσει τα δεδομένα της tv. Είναι αξιοσημείωτο ότι, ο νέος αλγόριθμος STM που θα παρουσιαστεί στην παρούσα εργασία κάνει την διάκριση μεταξύ των δύο αυτών ιδιοκτησιών και θεωρητικά επιτυγχάνει αυξημένη απόδοση.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Η ιδιοκτησία μιας t-μεταβλητής επιτυγχάνεται διαφορετικά στους εμποδιστικούς και στους μη-εμποδιστικούς αλγορίθμους STM. Στους εμποδιστικούς αλγορίθμους μια διεργασία p_i που κατέχει την ιδιοκτησία σε μια t-μεταβλητή tv επιτρέπεται να καθυστερεί όλες τις υπόλοιπες διεργασίες στο να προσπελάσουν τη tv, για οσοδήποτε μεγάλο χρονικό διάστημα, ακόμα και άπειρο, σε περίπτωση που η p_i καταρρεύσει. Αυτό συμβαίνει διότι στους εμποδιστικούς αλγορίθμους, μόνο η p_i μπορεί να καταργήσει τις ιδιοκτησίες που η ίδια κατέχει. Αντίθετα, στους μη-εμποδιστικούς αλγορίθμους η p_i δεν μπορεί να καθυστερεί όλες τις διεργασίες για οσοδήποτε μεγάλο χρονικό διάστημα. Για να εξασφαλιστεί η επιθυμητή ιδιότητα των μη-εμποδιστικών αλγορίθμων STM, ο εκάστοτε μη-εμποδιστικός αλγόριθμος STM υλοποιεί ένα *μηχανισμό βοήθειας* με βάση τον οποίο μία δοσοληψία p_j θα μπορούσε να βοηθήσει την p_i να ολοκληρωθεί επιτυχώς ή μη-επιτυχώς, έτσι ώστε να καταργηθεί η ιδιοκτησία της tv από την p_i ακόμη και εάν η p_i έχει καταρρεύσει. Επομένως, στους εμποδιστικούς αλγορίθμους η ιδιοκτησία έχει την έννοια τής *μόνιμης ιδιοκτησίας*, δηλαδή την έννοια του *κλειδώματος*, ενώ στους μη εμποδιστικούς αλγορίθμους την έννοια της *προσωρινής ιδιοκτησίας*.

Στη συνέχεια θα μελετήσουμε πέντε κατηγορίες σχεδιαστικών επιλογών των αλγορίθμων STM. Κατά την παρουσίαση των κατηγοριών αυτών θα παρουσιαστούν οι εναλλακτικές προσεγγίσεις που ακολουθούνται σε κάθε κατηγορία και παράλληλα θα γίνει σχετική αναφορά στην επιρροή που έχει κάθε προσέγγιση στις ιδιότητες του αντίστοιχου αλγορίθμου STM. Συγκεκριμένα στην Ενότητα 3.1 θα παρουσιαστεί ο τρόπος με τον οποία ανατίθενται, αποκτώνται και καταργούνται οι ιδιοκτησίες και στην Ενότητα 3.2 μελετάται ο χρόνος απόκτησης των ιδιοκτησιών. Στην Ενότητα 3.3 παρουσιάζονται οι διαφορετικοί τρόποι αποτροπής ή ανίχνευσης και επίλυσης των συγκρούσεων και ο μηχανισμός ελέγχου συνέπειας. Τέλος, στις Ενότητες 3.4 και 3.5 παρουσιάζεται ο ορισμός του πλήθους ενδιάμεσων επιπέδων και μελετώνται διαφορετικά μοντέλα μνήμης, αντίστοιχα.

3.1. Τρόπος Ανάθεσης, Απόκτησης και Κατάργησης Ιδιοκτησιών

Στην παρούσα Ενότητα θα μελετηθούν οι διαφορετικές προσεγγίσεις ως προς τον τρόπο ανάθεσης, απόκτησης και κατάργησης των ιδιοκτησιών. Ο απλούστερος

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

αλγόριθμος STM είναι ένας εμποδιστικός αλγόριθμος ο οποίος ορίζει ότι υπάρχει μία μόνο ιδιοκτησία *lock* για όλες τις t-μεταβλητές και κάθε δοσοληψία που επιθυμεί να προσπελάσει τις t-μεταβλητές πρέπει αρχικά να αποκτά την lock, στη συνέχεια να προσπελάσει τις t-μεταβλητές που επιθυμεί (για ανάγνωση ή ενημέρωση) και να καταργεί τη lock μόλις ολοκληρωθεί. Ο αλγόριθμος αυτός ικανοποιεί την ιδιότητα της σειριοποιησιμότητας, διότι εάν μια δοσοληψία καταφέρει να αποκτήσει την lock, τότε μπορεί να προσπελάσει ατομικά τις t-μεταβλητές. Όμως, ο αλγόριθμος αυτός οδηγεί στο μικρότερο δυνατό παραλληλισμό, διότι αποτυγχάνει να υποστηρίξει την ταυτόχρονη εκτέλεση δοσοληψιών που προσπελάζουν διαφορετικές t-μεταβλητές, μειώνοντας έτσι δραματικά την απόδοση του συστήματος STM.

Για τη μεγιστοποίηση της απόδοσης του συστήματος STM, οι περισσότεροι από τους αλγορίθμους STM που θα παρουσιαστούν αναθέτουν μια διαφορετική ιδιοκτησία σε κάθε t-μεταβλητή. Η προσέγγιση αυτή ονομάζεται *ανά t-μεταβλητή* ανάθεση ιδιοκτησιών. Μια ακόμη προσέγγιση είναι η *ανα σύνολο* ανάθεση ιδιοκτησιών, στην οποία ορίζεται ένας συγκεκριμένος αριθμός από ιδιοκτησίες και χρησιμοποιείται μια συνάρτηση κατακερματισμού, η οποία αντιστοιχεί ξένα σύνολα t-μεταβλητών σε συγκεκριμένες ιδιοκτησίες (δηλαδή, ένα σύνολο από t-μεταβλητές θα ανατίθεται σε μια συγκεκριμένη ιδιοκτησία). Υπενθυμίζεται ότι τα δεδομένα μιας t-μεταβλητής καθορίζονται από το χρήστη του συστήματος STM και μπορεί να είναι οτιδήποτε από μία αριθμητική τιμή έως κάποιο δείκτη που δείχνει σε κάποιο διαμοιραζόμενο αντικείμενο, π.χ. στον κόμβο μιας λίστας. Στη βιβλιογραφία πολλές φορές όταν τα δεδομένα της t-μεταβλητής είναι κάποιος δείκτης σε διαμοιραζόμενο αντικείμενο (ή το ίδιο το διαμοιραζόμενο αντικείμενο) και χρησιμοποιείται η ανα t-μεταβλητή ανάθεση ιδιοκτησιών αναφέρεται ότι χρησιμοποιείται ανά *διαμοιραζόμενο αντικείμενο* ανάθεση ιδιοκτησιών.

Στους αλγορίθμους που πρόκειται να παρουσιαστούν στην παρούσα εργασία, διαφέρει αρκετά η μορφή της πληροφορίας ιδιοκτησίας καθώς και ο τρόπος απόκτησης και κατάργησης της ιδιοκτησίας. Η μορφή της πληροφορίας ιδιοκτησίας έχει συνήθως ιδιαίτερη σημασία στους μη-εμποδιστικούς αλγορίθμους, διότι οι αλγόριθμοι συχνά απαιτείται να αυτοί αποθηκεύουν απαραίτητες πληροφορίες που αφορούν τη δοσοληψία που κατέχει μια ιδιοκτησία ώστε να μπορεί να υλοποιηθεί ο

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

μηχανισμός βοήθειας που χρησιμοποιούν. Όταν συμβαίνει αυτό λέμε ότι ο αλγόριθμος χρησιμοποιεί *επώνυμη* πληροφορία ιδιοκτησιών. Αντίθετα, στους εμποδιστικούς αλγορίθμους η πληροφορία ιδιοκτησίας συνήθως δεν αναφέρεται στο ποια δοσοληψία κατέχει την ιδιοκτησία. Σε αυτή την περίπτωση λέμε ότι ο αλγόριθμος χρησιμοποιεί ανώνυμη πληροφορία ιδιοκτησιών. Όταν χρησιμοποιείται η τεχνική της επώνυμης πληροφορίας ιδιοκτησιών, μια πληροφορία που συνήθως διατηρείται στις πληροφορίες που σχετίζονται με την ιδιοκτησία μιας t-μεταβλητής tv, είναι η δοσοληψία $T(i,n_i)$ που κατέχει την ιδιοκτησία της tv. Συγκεκριμένα, συνήθως αποθηκεύεται ένας δείκτης στη δομή δεδομένων $Trec(i,n_i)$ που περιγράφει την $T(i,n_i)$.

Όταν χρησιμοποιείται η τεχνική της ανώνυμης πληροφορίας ιδιοκτησιών, είναι επαρκές η πληροφορία που αντιστοιχίζεται σε κάθε t-μεταβλητή να έχει δύο καταστάσεις: «υπό κατοχή» και «ελεύθερη». Τότε, μια δοσοληψία μπορεί να αποκτήσει (αντίστοιχα καταργήσει) την ιδιοκτησία μιας t-μεταβλητής αλλάζοντας την κατάσταση της αντίστοιχης ιδιοκτησίας σε «υπό κατοχή» (αντίστοιχα «ελεύθερη»). Διαφορετική προσέγγιση ακολουθείται από τους υπάρχοντες αλγορίθμους STM που χρησιμοποιούν επώνυμη πληροφορία ιδιοκτησίας. Κάθε δοσοληψία διατηρεί συνήθως μια διαμοιραζόμενη μεταβλητή *status* η οποία μπορεί να παίρνει τουλάχιστον δύο τιμές, *ενεργή* και *ολοκληρωμένη* (ή κάποιες αντίστοιχες αυτών). Μια δοσοληψία μπορεί να αποκτήσει την ιδιοκτησία μιας t-μεταβλητής, γράφοντας στις πληροφορίες της συγκεκριμένης ιδιοκτησίας ένα δείκτη προς τη μεταβλητή status. Όλες οι ιδιοκτησίες που κατέχει μια δοσοληψία καταργούνται μόλις η δοσοληψία αλλάξει το status της στην τιμή *ολοκληρωμένη*. Με αντίστοιχο τρόπο μια δοσοληψία μπορεί να εξακριβώσει εάν κατέχεται ή όχι η ιδιοκτησία μιας t-μεταβλητής, από κάποια δοσοληψία. Παρατηρούμε ότι η πρώτη προσέγγιση απαιτεί την εκτέλεση k εντολών αλλαγής της κατάστασης σε «ελεύθερη» για την κατάργηση των ιδιοκτησιών k t-μεταβλητών, ενώ η δεύτερη προσέγγιση απαιτεί την εκτέλεση μόνο μίας εντολής αλλαγής του status της δοσοληψίας στην τιμή *ολοκληρωμένη*.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

3.2. Χρόνος Απόκτησης Ιδιοκτησιών

Στην ενότητα αυτή θα μελετηθεί πότε ο κάθε αλγόριθμος STM αποκτά τις ιδιοκτησίες των t-μεταβλητών που επιθυμεί να προσπελάσει. Σημειώνεται ότι οι t-μεταβλητές στις οποίες θα αποκτήσει ιδιοκτησίες μια δοσοληψία εξαρτώνται από την πολιτική απόκτησης ιδιοκτησιών που ακολουθείται από τον εκάστοτε αλγόριθμο STM. Τα δύο είδη χρονικής απόκτησης ιδιοκτησιών είναι η *εκ των προτέρων* απόκτηση και η *εκ των υστέρων* απόκτηση. Στην εκ των προτέρων απόκτηση μία δοσοληψία αποκτά την ιδιοκτησία μιας t-μεταβλητής την πρώτη φορά που την προσπελάζει. Στην εκ των υστέρων απόκτηση μία δοσοληψία αποκτά τις ιδιοκτησίες των t-μεταβλητών που επιθυμεί στη φάση ολοκλήρωσής της, κατά την εκτέλεση της λειτουργίας CommitTransaction.

Συνήθως, στους αλγόριθμους που χρησιμοποιούν την εκ των προτέρων απόκτηση ιδιοκτησιών οι δοσοληψίες ενημερώνουν άμεσα τα δεδομένα της t-μεταβλητής που προσπελάζουν για ενημέρωση, αμέσως μετά την απόκτηση της αντίστοιχης ιδιοκτησίας, και διατηρούν μια *δομή επαναφοράς* (*undo log*) με τα παλιά δεδομένα των t-μεταβλητών που έχουν αποκτήσει, έτσι ώστε εάν η δοσοληψία ολοκληρωθεί μη-επιτυχώς να είναι δυνατή η επαναφορά της μνήμης στην κατάσταση που βρισκόταν πριν την εκκίνηση της δοσοληψίας αυτής. Στην περίπτωση αυτή οι ενημερώσεις της συγκεκριμένης δοσοληψίας είναι εμφανείς στις άλλες δοσοληψίες, πριν την ολοκλήρωσή της. Ονομάζουμε *άμεση* (*direct*) αυτή τη μέθοδο ενημέρωσης των t-μεταβλητών. Εναλλακτικά, μια δοσοληψία θα μπορούσε να εφαρμόσει τις ενημερώσεις στις t-μεταβλητές μόνο κατά την εκτέλεση της λειτουργίας CommitTransaction και όποτε θα ήταν βέβαιο ότι θα ολοκληρωνόταν επιτυχώς. Στην περίπτωση αυτή οι δοσοληψίες καταγράφουν σε μια *δομή επανεκτέλεσης* (*redo log*) τα νέα δεδομένα που οι διεργασίες εισάγουν στις t-μεταβλητές που προσπελάζουν για ενημέρωση, ώστε να μπορούν στη συνέχεια να τις εφαρμόσουν κατά την ολοκλήρωσή τους. Ονομάζουμε *ετεροχρονισμένη* (*deferred*) αυτή τη μέθοδο ενημέρωσης των t-μεταβλητών.

Κάθε δοσοληψία διατηρεί δύο *λίστες t-μεταβλητών* που περιέχουν πληροφορίες για τις t-μεταβλητές που αυτή διάβασε και ενημέρωσε. Ονομάζουμε *λίστα ενημερώσεων* και *λίστα αναγνώσεων* τις λίστες αυτές, αντίστοιχα. Οι λίστες αυτές, μεταξύ άλλων, είναι

απαραίτητες ώστε να μπορούν οι δοσοληψίες να εφαρμόσουν την εκ των υστέρων απόκτηση ιδιοκτησιών (διατηρώντας τις t-μεταβλητές των οποίων οι ιδιοκτησίες πρέπει να αποκτηθούν), να εφαρμόσουν την άμεση μέθοδο ενημέρωσης των t-μεταβλητών (διατηρώντας στη λίστα ενημερώσεων τις παλιές τιμές των t-μεταβλητών που ενημερώθηκαν, δηλαδή υλοποιώντας έτσι τη δομή επαναφοράς) και να εφαρμόσουν την ετεροχρονισμένη μέθοδο ενημέρωσης (διατηρώντας στη λίστα ενημερώσεων τις νέες τιμές των t-μεταβλητών που ενημερώθηκαν, δηλαδή υλοποιώντας έτσι τη δομή επανεκτέλεσης).

3.3. Τρόποι Αποτροπής ή Ανίχνευσης και Επίλυσης Συγκρούσεων και Μηχανισμός Ελέγχου Συνέπειας

Στην παρούσα ενότητα θα παρουσιαστούν οι διαφορετικοί τρόποι με τους οποίους ένας αλγόριθμος που χρησιμοποιεί τη μέθοδο απόκτησης ιδιοκτησιών μπορεί να αποτρέψει ή να ανιχνεύσει και να επιλύσει τις συγκρούσεις που παρουσιάζονται μεταξύ των δοσοληψιών. Έστω ότι η δοσοληψία $T(i,n_i)$ που εκτελεί τη λειτουργία op_i συγκρούεται με τη δοσοληψία $T(j,n_j)$ που εκτελεί τη λειτουργία op_j, επειδή οι λειτουργίες op_i και op_j εκτελούνται στην ίδια t-μεταβλητή tv και η μία εκ των δύο είναι ενημέρωση.

Οι αλγόριθμοι STM που χρησιμοποιούν τη μέθοδο απόκτησης ιδιοκτησιών μπορούν να ανιχνεύσουν τη σύγκρουση W-W μεταξύ των λειτουργιών op_i και op_j, και να αποτρέψουν τα προβλήματα που προκύπτουν από αυτή, διότι ορίζουν ότι μια δοσοληψία είναι υποχρεωμένη να αποκτά τις ιδιοκτησίες των t-μεταβλητών που επιθυμεί να ενημερώσει, πριν εφαρμόσει τις αλλαγές σε αυτές. Συγκεκριμένα, μία από τις δύο δοσοληψίες, έστω η $T(i,n_i)$, αποκτά την ιδιοκτησία της tv πριν την άλλη. Έτσι η $T(j,n_j)$ δε θα μπορεί να αποκτήσει την ιδιοκτησία της tv πριν η $T(i,n_i)$ την καταργήσει. Επομένως με τον τρόπο αυτό αποτρέπεται η εμφάνιση της σύγκρουσης W-W. Επίσης, για τον ίδιο λόγο, οι αλγόριθμοι STM που χρησιμοποιούν τη μέθοδο απόκτησης ιδιοκτησιών, μπορούν να ανιχνεύουν τη σύγκρουση W-R. Υπενθυμίζεται ότι ένας αλγόριθμος STM που ακολουθεί τη μέθοδο απόκτησης ιδιοκτησιών δεν απαιτεί από μια διεργασία που εκτελεί μια δοσοληψία να αποκτά τις ιδιοκτησίες των t-μεταβλητών που επιθυμεί να διαβάσει. Εάν χρησιμοποιούνται ορατές αναγνώσεις t-

μεταβλητών και η εκ των προτέρων απόκτηση ιδιοκτησιών, είναι δυνατή η ανίχνευση και αποτροπή μιας σύγκρουσης R-W, με τον τρόπο που περιγράφηκε παραπάνω για συγκρούσεις W-W και W-R. Ωστόσο εάν δεν χρησιμοποιούνται ορατές αναγνώσεις t-μεταβλητών ή χρησιμοποιούνται σε συνδυασμό με την εκ των υστέρων απόκτηση ιδιοκτησιών, τότε μπορεί να προκύψει σύγκρουση R-W. Στην περίπτωση αυτή, τα δεδομένα d_i που η $T(i,n_i)$ διαβάζει για την tv (μέσω της op_i) ενημερώνονται στη συνέχεια από την $T(j,n_j)$ (μέσω της op_j) πριν η $T(i,n_i)$ ολοκληρωθεί, το οποίο σημαίνει ότι τα δεδομένα d_i είναι πιθανό να είναι *ασυνεπή* σε συγκεκριμένες εκτελέσεις, όπως συζητήθηκε στην Ενότητα 2.3.5.

Συμπεραίνουμε ότι οι αλγόριθμοι STM που χρησιμοποιούν ορατές αναγνώσεις t-μεταβλητών και εκ των προτέρων απόκτηση ιδιοκτησιών μπορούν να αποτρέψουν την εμφάνιση όλων των πιθανών συγκρούσεων. Επίσης, οι αλγόριθμοι STM που είτε δεν χρησιμοποιούν ορατές αναγνώσεις t-μεταβλητών είτε τις χρησιμοποιούν σε συνδυασμό με την εκ των υστέρων απόκτηση ιδιοκτησιών, μπορούν να αποτρέψουν την εμφάνιση όλων των πιθανών συγκρούσεων, εκτός των συγκρούσεων R-W. Έτσι οι αλγόριθμοι αυτοί, για να μπορούν να ικανοποιούν την ιδιότητα της σειριοποιησιμότητας σε όλες τις περιπτώσεις, υλοποιούν έναν *μηχανισμό ελέγχου* της *συνέπειας* των δεδομένων των t-μεταβλητών που κάποια δοσοληψία έχει προσπελάσει για ανάγνωση, ο οποίος είναι υπεύθυνος για την ανίχνευση και επίλυση συγκρούσεων R-W που καταστρατηγούν την ιδιότητα της σειριοποιησιμότητας.

Για την υλοποίηση του μηχανισμού ελέγχου συνέπειας οι υπάρχοντες αλγόριθμοι που θα παρουσιαστούν στην παρούσα εργασία (στα Κεφάλαια 4, 5 και 7), ακολουθούν αρκετά διαφορετικές προσεγγίσεις. Παρόλ' αυτά, όλες βασίζονται στην ίδια λογική που περιγράφεται στη συνέχεια. Για να μπορούν να υλοποιήσουν τον μηχανισμό αυτό, εισαγάγουν την έννοια της *έκδοσης* (*version*) μιας t-μεταβλητής. Μια δοσοληψία που ενημερώνει μια t-μεταβλητή tv, ενημερώνει και την έκδοσή της. Η έκδοση μιας t-μεταβλητής tv επιτρέπει σε μια δοσοληψία να εξακριβώσει εάν τα δεδομένα της tv έχουν τροποποιηθεί. Σε κάποιους αλγορίθμους μέσω της έκδοσης μιας t-μεταβλητής tv μπορούν ακόμα και να προσδιοριστούν μοναδικά τα δεδομένα της tv στο χρόνο (με βάση τις ενημερώσεις που εφαρμόστηκαν στη tv). Μια δοσοληψία $T(i,n_i)$ υποχρεώνεται να διατηρεί για κάθε t-μεταβλητή tv που διαβάζει,

την έκδοσή της. Σημειώνεται ότι η πληροφορία αυτή διατηρείται στη λίστα ανάγνωσης που έχει περιγραφεί στην Ενότητα 3.2. Έτσι, η $T(i,n_i)$ μπορεί να ελέγξει οποιαδήποτε χρονική στιγμή εάν έχει προκύψει κάποια R-W σύγκρουση που αφορά την tv, ελέγχοντας την έκδοση των δεδομένων της tv και παρατηρώντας εάν αυτή έχει αλλάξει μετά την πρώτη προσπέλασή τους από την $T(i,n_i)$. Ο μηχανισμός ελέγχου συνέπειας ελέγχει με βάση την παραπάνω μέθοδο όλες τις t-μεταβλητές που η $T(i,n_i)$ προσπέλασε για ανάγνωση (που υπάρχουν καταχωρημένες στη αντίστοιχη λίστα αναγνώσεων) και επιστρέφει την τιμή ΑΛΗΘΕΣ εάν καμία από αυτές δεν έχει τροποποιηθεί ή την τιμή ΨΕΥΔΕΣ σε αντίθετη περίπτωση.

Για να μπορεί μια δοσοληψία να εγγυηθεί τη συνέπεια των δεδομένων των t-μεταβλητών που έχει προσπελάσει για ανάγνωση, θα αρκούσε ο αντίστοιχος μηχανισμός ελέγχου συνέπειας να εκτελούνταν μία φορά κατά την εκτέλεση της λειτουργίας CommitTransaction από τη δοσοληψία αυτή. Συγκεκριμένα, εάν ο μηχανισμός εκτελεστεί μετά την απόκτηση των ιδιοκτησιών επί των t-μεταβλητών που η δοσοληψία προσπέλασε για ενημέρωση, και επιστρέψει την τιμή ΑΛΗΘΕΣ, σημαίνει ότι τα δεδομένα που η δοσοληψία προσπέλασε για ανάγνωση είναι συνεπή, στο σημείο που εκκινήθηκε η εκτέλεση της λειτουργίας CommitTransaction. Όμως, η εκτέλεση του μηχανισμού ελέγχου συνέπειας είναι απαραίτητη και κατά τη διάρκεια εκτέλεσης της δοσοληψίας, έτσι ώστε να αποφευχθούν προβλήματα που μπορούν να εμφανιστούν λόγω της ασυνέπειας των δεδομένων των t-μεταβλητών που διαβάστηκαν, όπως ο εγκλωβισμός του κώδικά του χρήστη σε άπειρο βρόχο ή η προσπέλαση κάποιου null δείκτη (στη συνέχεια παρουσιάζεται ένα παράδειγμα εμφάνισης άπειρου βρόχου λόγω ασυνέπειας των δεδομένων των t-μεταβλητών). Για να αποφευχθούν τα προβλήματα αυτά πρέπει ο μηχανισμός ελέγχου συνέπειας να εκτελείται μετά από κάθε λειτουργία ανάγνωσης.

Στο σημείο αυτό παρουσιάζεται ένα παράδειγμα που δείχνει ότι μπορεί να παρουσιαστεί κάποιος ανεπιθύμητος άπειρος βρόχος λόγω ασυνέπειας των δεδομένων των t-μεταβλητών που μια δοσοληψία $T(i,n_i)$ διαβάζει, όταν δε χρησιμοποιείται ο μηχανισμός ελέγχου συνέπειας των t-μεταβλητών κατά τη διάρκεια εκτέλεσης της $T(i,n_i)$. Υποθέτουμε ότι έχουμε τη συνδεδεμένη λίστα που παρουσιάζεται στο Σχήμα 2.3 (α), η οποία περιέχει τα στοιχεία 2,1,8,4 και

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

υποστηρίζει δύο λειτουργίες, την αναζήτηση ενός στοιχείου και την μεταφορά ενός στοιχείου στην αρχή της λίστας. Σημειώνεται ότι η αναζήτηση ενός στοιχείου απαιτεί την προσπέλαση όλων των στοιχείων της λίστας, διότι αυτή δεν είναι ταξινομημένη, και επίσης η μεταφορά ενός στοιχείου στην αρχή της λίστας πραγματοποιείται με κατάλληλη τροποποίηση των αντίστοιχων δεικτών και όχι απλά με την ενημέρωση των τιμών των κόμβων. Έστω ότι η δοσοληψία $T(i,n_i)$ εκτελεί τη λειτουργία αναζήτησης του στοιχείου 100 και έχει διαβάσει τα στοιχεία 2 και 1. Η μορφή της λίστας που η $T(i,n_i)$ γνωρίζει τη συγκεκριμένη χρονική στιγμή παρουσιάζεται στο Σχήμα 3.2 (α). Στη συνέχεια μια άλλη δοσοληψία $T(j,n_j)$ εκτελεί τη λειτουργία μεταφοράς στην αρχή της λίστας του στοιχείου 8, η οποία απαιτεί την τροποποίηση των δεικτών των στοιχείων 2 και 8, και ολοκληρώνεται επιτυχώς. Η μορφή της λίστας την τρέχουσα χρονική στιγμή παρουσιάζεται στο Σχήμα 2.3 (β). Σημειώνεται ότι τα δεδομένα του στοιχείου 1 που η $T(i,n_i)$ έχει διαβάσει έχουν τροποποιηθεί, όμως η $T(i,n_i)$ δε μπορεί να το γνωρίζει διότι δεν εκτελεί τον μηχανισμό ελέγχου συνέπειας κατά τη διάρκεια εκτέλεσής της. Έτσι, όταν στη συνέχεια η $T(i,n_i)$ θα διαβάσει το επόμενο στοιχείο του 1, θα διαβάσει τα δεδομένα του στοιχείου 8, το οποίο θα εγκλωβίσει την $T(i,n_i)$ στον άπειρο βρόχο που σχηματίζουν τα στοιχεία 2,1,8, όπως παρουσιάζεται στο Σχήμα 3.2 (β) (όπου φαίνεται η μορφή της λίστας που η $T(i,n_i)$ γνωρίζει τη συγκεκριμένη χρονική στιγμή).

(α) (β)

Σχήμα 3.1: Η συνδεδεμένη λίστα του παραδείγματος, α) η αρχική της μορφή, β) η μορφή της μετά την μεταφορά του στοιχείου 8 στην αρχή της λίστας, από την $T(j,n_j)$.

(α) (β)

Σχήμα 3.2: Η μορφή της συνδεδεμένης λίστας του παραδείγματος που η $T(i,n_i)$ γνωρίζει, α) πριν την ανάγνωση του στοιχείου 8 και β) μετά την ανάγνωσή του.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Σημειώνεται ότι οι αλγόριθμοι STM που χρησιμοποιούν το μηχανισμό ελέγχου συνέπειας, διαφέρουν ως προς το χρόνο εκτέλεσης του μηχανισμού αυτού. Υπάρχουν αλγόριθμοι που εκτελούν τον έλεγχο αυτό κάθε φορά που ο χρήστης προσπελάζει για ανάγνωση μια t-μεταβλητή. Ονομάζουμε τη μέθοδο αυτή *αυξητικό έλεγχο συνέπειας*, διότι κάθε φορά που θα εκτελείται ο μηχανισμός ελέγχου θα πρέπει να ελέγχει τα δεδομένα μιας επιπλέον t-μεταβλητής. Άλλοι αλγόριθμοι εγγυώνται την εκτέλεση του μηχανισμού ελέγχου συνέπειας μόνο μετά από συγκεκριμένο πλήθος λειτουργιών ανάγνωσης. Ακόμη, υπάρχουν αλγόριθμοι που απαιτούν από το χρήστη να εκτελεί τον έλεγχο συνέπειας σε οποιοδήποτε σημείο του προγράμματός του κρίνει απαραίτητο (κάτι το οποίο προϋποθέτει ότι η διεπαφή του μοντέλου STM πρέπει να επεκταθεί κατάλληλα ώστε να προσφέρει τη λειτουργικότητα αυτή). Έτσι διαχωρίζουμε τους μηχανισμούς ελέγχου συνέπειας σε *αυτόματους* εάν δεν απαιτούν την παρέμβαση του χρήστη του αλγορίθμου STM και σε *μη-αυτόματους* στην αντίθετη περίπτωση. Σημειώνεται ότι μόνο ο αυξητικός έλεγχος συνέπειας (ο οποίος μπορεί να εκτελείται αυτόματα ή μη-αυτόματα) εξασφαλίζει την πλήρη αποφυγή προβλημάτων λόγω ασυνέπειας των δεδομένων που διαβάζονται.

Είναι σημαντικό ότι για να εξασφαλιστεί η συνέπεια των δεδομένων των t-μεταβλητών που έχουν προσπελαστεί για ανάγνωση, αρκεί να ελεγχθούν ως προς τη συνέπεια μόνο τα δεδομένα των t-μεταβλητών που η δοσοληψία προσπέλασε με καθολική ανάγνωση, όπως εξηγείται στη συνέχεια. Εάν μια δοσοληψία ενημερώσει μια t-μεταβλητή tv, έστω στην καθολική κατάσταση C μιας εκτέλεσης α, τότε οποιαδήποτε ανάγνωση της t-μεταβλητής tv από την $T(i,n_i)$ μετά την C θα επιστρέφει την τιμή που η ίδια η $T(i,n_i)$ έγραψε στη C (εάν στο μεταξύ δεν ενημέρωσε και πάλι η $T(i,n_i)$ την tv), όπως εξηγείται στη συνέχεια. Η σειριοποιησιμότητα εγγυάται ότι η $T(i,n_i)$ εκτελείται με τον ίδιο τρόπο όπως αν είχε εκτελεστεί ατομικά σε κάποιο σημείο του διαστήματος εκτέλεσής της. Έτσι, οποιαδήποτε ανάγνωση μιας t-μεταβλητής tv από την $T(i,n_i)$ ακολουθεί μια ενημέρωση της tv από την ίδια, θα πρέπει να επιστρέψει την τιμή με την οποία ενημερώθηκε η tv από την $T(i,n_i)$ προκειμένου αυτή να είναι συνεπής.

Συμπεραίνουμε ότι όλοι οι αλγόριθμοι STM μπορούν να αποτρέψουν την εμφάνιση όλων των πιθανών συγκρούσεων, με αυτούς που χρησιμοποιούν μη-ορατές

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

αναγνώσεις t-μεταβλητών, να απαιτούν την υλοποίηση του μηχανισμού ελέγχου συνέπειας για να είναι σε θέση να ανιχνεύσουν τις συγκρούσεις R-W. Μετά την αποτροπή ή ανίχνευση μιας σύγκρουσης, οι αλγόριθμοι STM διαφοροποιούνται ως προς τις ενέργειες που πρέπει να ακολουθήσουν οι δοσοληψίες που συγκρούονται. Έστω ότι μια δοσοληψία $T(i,n_i)$ συγκρούεται με μια άλλη δοσοληψία $T(j,n_j)$. Η $T(i,n_i)$ μπορεί να περιμένει για κάποιο χρονικό διάστημα την $T(j,n_j)$ να ολοκληρωθεί. Στους εμποδιστικούς αλγορίθμους STM, η $T(i,n_i)$ έχει επίσης της επιλογή να ολοκληρωθεί η ίδια ως μη-επιτυχημένη. Στους μη-εμποδιστικούς αλγορίθμους, η $T(i,n_i)$ μπορεί είτε να τερματίσει βίαια την $T(j,n_j)$ ως μη επιτυχημένη είτε να βοηθήσει την $T(j,n_j)$ να ολοκληρωθεί (επιτυχώς ή μη-επιτυχώς). Συγκεκριμένα, οι μη-εμποδιστικοί αλγόριθμοι που ικανοποιούν την ασθενέστερη ιδιότητα τερματισμού της ελευθερίας ανταγωνισμού επιτρέπουν στην $T(i,n_i)$ είτε να τερματίσει βίαια την $T(j,n_j)$ είτε να την περιμένει να ολοκληρωθεί.

Ανάλογα με το χρόνο απόκτησης των ιδιοκτησιών, η αποτροπή ή ανίχνευση μιας σύγκρουσης ονομάζεται *εκ των προτέρων ή εκ των υστέρων αποτροπή-ανίχνευση*, εάν χρησιμοποιείται η εκ των προτέρων ή εκ των υστέρων απόκτηση ιδιοκτησιών, αντίστοιχα. Είναι σημαντικό ότι, στην εκ των προτέρων αποτροπή (ή ανίχνευση) υπάρχει δυνατότητα αποτροπής (ή ανίχνευσης) μιας σύγκρουσης ακριβώς στο σημείο της εκτέλεσης στο οποίο η σύγκρουση αυτή εμφανίζεται. Ενώ, στην εκ των υστέρων μια σύγκρουση αποτροπή (ανίχνευση), μια σύγκρουση αποτρέπεται (ή ανιχνεύεται) αργότερα. Η παρατήρηση αυτή σημαντική αφού αυτό μπορεί να επηρεάσει την απόδοση των συστημάτων STM, όπως εξηγείται στη συνέχεια.

Κατά την αποτροπή μιας σύγκρουσης από μια δοσοληψία $T(i,n_i)$ με κάποια άλλη δοσοληψία $T(j,n_j)$, η $T(i,n_i)$ θα επιβαρυνθεί είτε με το χρονικό κόστος αναμονής, είτε με το κόστος βοήθειας της $T(j,n_j)$ είτε στην χειρότερη περίπτωση με το κόστος της μη-επιτυχούς ολοκλήρωσής της. Το κόστος αυτό θα «πληρωθεί» άσκοπα από την $T(i,n_i)$ εάν η $T(j,n_j)$ στη συνέχεια ολοκληρωθεί μη-επιτυχώς. Ονομάζουμε την περίπτωση αυτή, *άσκοπη πληρωμή*. Σημειώνεται ότι με βάση τη μέθοδο απόκτησης ιδιοκτησιών, μια δοσοληψία αποκτά τις ιδιοκτησίες των t-μεταβλητών που επιθυμεί και τις διατηρεί έως ότου ολοκληρωθεί. Έτσι, επειδή στην εκ των προτέρων αποτροπή ή ανίχνευση των συγκρούσεων, χρησιμοποιείται η εκ των προτέρων

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

απόκτηση ιδιοκτησιών, το διάστημα διατήρησης των ιδιοκτησιών των t-μεταβλητών από τις δοσοληψίες είναι μεγαλύτερο από ότι στην εκ των υστέρων αποτροπή ή ανίχνευση συγκρούσεων. Αυτό σημαίνει ότι η πιθανότητα εμφάνισης συγκρούσεων αυξάνεται και επομένως αυξάνεται η πιθανότητα εμφανίσεως καταστάσεων άσκοπης πληρωμής.

Επίσης, κατά την ανίχνευση μιας σύγκρουσης R-W μεταξύ δύο δοσοληψιών, της $T(i,n_i)$ (που κάνει την ανάγνωση) και της $T(j,n_j)$ (που κάνει την ενημέρωση), σε συγκεκριμένες εκτελέσεις (όπως συζητήθηκε στην Ενότητα 2.3.5) η $T(i,n_i)$ είναι καταδικασμένη να ολοκληρωθεί μη-επιτυχώς εάν η $T(j,n_j)$ ολοκληρωθεί επιτυχώς. Στην περίπτωση αυτή, ονομάζουμε τη δοσοληψία $T(i,n_i)$ *εν δυνάμει καταδικασμένη* και εάν η $T(j,n_j)$ πρόκειται να ολοκληρωθεί επιτυχώς και επίσης πρόκειται να εμφανιστεί κάποιο «κακό» σενάριο, ονομάζεται *καταδικασμένη*. Η εκ των προτέρων ανίχνευση συγκρούσεων R-W επιτρέπει την γρήγορη ανίχνευση μιας καταδικασμένης δοσοληψίας. Από την άλλη, η εκ των υστέρων ανίχνευση συγκρούσεων R-W αυξάνει την πιθανότητα μια εν δυνάμει καταδικασμένη δοσοληψία (η $T(i,n_i)$) να ολοκληρωθεί επιτυχώς (επειδή η $T(j,n_j)$ ολοκληρώθηκε ως μη-επιτυχημένη).

Με βάση τα παραπάνω, προκύπτει ένας συμβιβασμός (trade-off) μεταξύ της εκ των προτέρων και της εκ των υστέρων αποτροπής ή ανίχνευσης συγκρούσεων, με την εκ των προτέρων να επιτρέπει τη γρήγορη ανίχνευση των καταδικασμένων δοσοληψιών και την εκ των υστέρων να μειώνει την πιθανότητα εμφάνισης καταστάσεων άσκοπης πληρωμής και να αυξάνει την πιθανότητα οι εν δυνάμει καταδικασμένες δοσοληψίες να ολοκληρωθούν ως επιτυχημένες.

3.4. Πλήθος Ενδιάμεσων Επιπέδων

Ορίζουμε ως *πλήθος ενδιάμεσων επιπέδων* το μέγιστο πλήθος δεικτών μνήμης που πρέπει να ακολουθήσει μια λειτουργία ανάγνωσης οποιασδήποτε t-μεταβλητής tv σε οποιαδήποτε εκτέλεση, βάσει της αναπαράστασης της tv από τον εκάστοτε αλγόριθμο STM, ώστε να εντοπίσει και να επιστρέψει τα δεδομένα της tv στη δοσοληψία που κάλεσε τη λειτουργία αυτή, όταν η ιδιοκτησία της tv δεν έχει αποκτηθεί από κάποια

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

άλλη δοσοληψία. Το πλήθος των ενδιάμεσων επιπέδων εξαρτάται από την μορφή της πληροφορίας ιδιοκτησίας που ο εκάστοτε αλγόριθμος STM χρησιμοποιεί.

3.5. Μοντέλο Μνήμης

Οι υπάρχοντες αλγόριθμοι STM υποθέτουν δύο διαφορετικά είδη μοντέλου μνήμης. Το πρώτο ονομάζεται *ανοιχτό μοντέλο μνήμης* και εκφράζει το συμβατικό μοντέλο μνήμης στο οποίο οι διεργασίες δεσμεύουν και αποδεσμεύουν τη μνήμη υπ' ευθύνη τους, όπως ακριβώς συμβαίνει και στο μοντέλο προγραμματισμού της γλώσσας C μέσω των αντίστοιχων λειτουργιών malloc και free. Το δεύτερο μοντέλο ονομάζεται *κλειστό μοντέλο μνήμης*. Στο μοντέλο αυτό, το σύστημα είναι υπεύθυνο για τη δέσμευση και αποδέσμευση της μνήμης του εκτελούμενου προγράμματος. Ένα τέτοιο σύστημα χρησιμοποιεί η γλώσσα προγραμματισμού JAVA. Συγκεκριμένα, το πρόγραμμα διαχείρισης μνήμης της JAVA καταγράφει όλες τις θέσεις μνήμης που χρησιμοποιούνται από οποιαδήποτε διεργασία. Η αποδέσμευση μνήμης, πραγματοποιείται από ένα ειδικό πρόγραμμα *αποδέσμευσης* της *μη χρησιμοποιούμενης μνήμης* (garbage collector). Το πρόγραμμα αυτό εκτελείται περιοδικά, εντοπίζει τις διευθύνσεις μνήμης που δεν χρησιμοποιούνται πλέον και τις αποδεσμεύσει.

Η επιλογή μεταξύ του ανοιχτού και του κλειστού μοντέλου μνήμης κατά τη σχεδίαση ενός αλγορίθμου STM είναι σημαντική, διότι στο ανοιχτό μοντέλο ο χρήστης είναι υπεύθυνος να διαχειρίζεται κατάλληλα τις δεσμεύσεις και αποδεσμεύσεις μνήμης, ώστε να μην εμφανίζονται προβλήματα όπως αναφορά σε null δείκτες, ενώ στο κλειστό μοντέλο μνήμης είναι υπεύθυνο το ίδιο το σύστημα για την αποδέσμευση της μνήμης όποτε αυτή δε χρησιμοποιείται. Η διαφορά αυτή είναι σημαντική και επιτρέπει στους αλγορίθμους STM που χρησιμοποιούν το κλειστό μοντέλο μνήμης να ορίζουν με έναν αρκετά εύκολο τρόπο την έκδοση των δεδομένων μιας t-μεταβλητής. Υπενθυμίζεται ότι η έννοια της έκδοσης είναι απαραίτητη στους αλγορίθμους STM που υλοποιούν κάποιον μηχανισμό ελέγχου συνέπειας. Συγκεκριμένα, ένας αλγόριθμος STM μπορεί να διατηρεί ως έκδοση μιας t-μεταβλητής tv τη διεύθυνση στην οποία είναι αποθηκευμένα τα δεδομένα d της x. Κάθε φορά που μια διεργασία επιθυμεί να ενημερώσει την tv με τα δεδομένα d', δεσμεύει μνήμη για την

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

αποθήκευση των νέων δεδομένων d'. Και τα δύο μοντέλα μνήμης εξασφαλίζουν ότι η μνήμη που θα δεσμευθεί για τα δεδομένα d' δεν θα επικαλύπτεται με την αντίστοιχη μνήμη του d. Επομένως, μια δοσοληψία μπορεί να ελέγξει εάν τα δεδομένα της tv έχουν τροποποιηθεί, ελέγχοντας εάν έχει αλλάξει η διεύθυνση στην οποία είναι αποθηκευμένα τα δεδομένα της. Έτσι υλοποιείται η έννοια της έκδοσης με αρκετά απλό τρόπο. Τα δεδομένα μιας t-μεταβλητής πρέπει να αποδεσμεύονται όταν πλέον δε δεικτοδοτούνται από καμία διεργασία. Το ανοιχτό μοντέλο μνήμης επιβάλει στον αλγόριθμο STM να γνωρίζει πότε ένα δεδομένο δε χρησιμοποιείται για να το αποδεσμεύει. Αντίθετα, το κλειστό μοντέλο μνήμης παρέχει μηχανισμούς για την αποδέσμευση των μη χρησιμοποιούμενων θέσεων μνήμης χωρίς να απαιτείται η εκτέλεση οποιονδήποτε ενεργειών από τον αλγόριθμο STM.

ΚΕΦΑΛΑΙΟ 4. ΑΛΓΟΡΙΘΜΟΣ DSTM

4.1 Περιγραφή αλγορίθμου DSTM

4.2 Απόδειξη ορθότητας αλγορίθμου DSTM

4.3 Πολυπλοκότητα

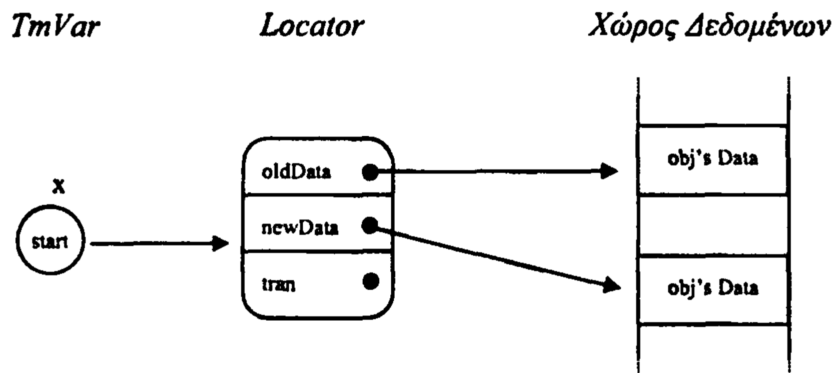
Στο κεφάλαιο αυτό περιγράφεται ο αλγόριθμος Dynamic STM ή DSTM, ο οποίος παρουσιάστηκε στην εργασία [12] των M.Herlihy, M.Moir, V.Lunchango και W.N.Scherer, που δημοσιεύθηκε στο συνέδριο Principles Of Distributed Computing (PODC) τον Ιούλιο του 2003. Στην Ενότητα 4.1 περιγράφεται ο κώδικας του αλγορίθμου, στην Ενότητα 4.2 παρουσιάζεται η απόδειξή του και στην Ενότητα 4.3 παρουσιάζεται η πολυπλοκότητά του.

4.1. Περιγραφή αλγορίθμου DSTM

Ο αλγόριθμος DSTM είναι ο πρώτος αλγόριθμος STM που υποστηρίζει δυναμικές δοσοληψίες. Ο DSTM ικανοποιεί την ασθενέστερη ιδιότητα τερματισμού ελευθερία ανταγωνισμού και έχει σχεδιαστεί να λειτουργεί σε κλειστό μοντέλο μνήμης. Η ιδιότητα τερματισμού ελευθερία ανταγωνισμού επιτρέπει την εμφάνιση καταστάσεων καθολικής παρατεταμένης στέρησης.

Στον DSTM κάθε t-μεταβλητή x υλοποιείται από έναν καταχωρητή CAS που περιέχεται σε μια δομή η οποία ονομάζεται *TmVar*. Ο καταχωρητής αυτός αποθηκεύει έναν δείκτη *start* που δείχνει σε κάποιο διαμοιραζόμενο αντικείμενο, που ονομάζεται *Locator* και το οποίο διατηρεί i) έναν καταχωρητή ανάγνωσης-εγγραφής *tran*, ο οποίος περιέχει ένα δείκτη σε μια δομή που περιγράφει τη δοσοληψία που κατέχει την προσωρινή ιδιοκτησία της x, ii) έναν καταχωρητή ανάγνωσης-εγγραφής

oldData που περιέχει έναν δείκτη προς τα παλιά δεδομένα της x και iii) έναν καταχωρητή ανάγνωσης-εγγραφής *newData* που περιέχει έναν δείκτη προς τα νέα δεδομένα της x. Η μορφή μιας t-μεταβλητής x που αναπαριστά ένα διαμοιραζόμενο αντικείμενο obj παρουσιάζεται στο Σχήμα 2.2.



Σχήμα 4.1: Η μορφή ενός διαμοιραζόμενου αντικειμένου obj που περιγράφεται μέσω μιας t-μεταβλητής x, στον DSTM.

Για κάθε δοσοληψία $T(i,n_i)$ ο αλγόριθμος DSTM διατηρεί τις ακόλουθες πληροφορίες στη δομή $Trec(i,n_i)$: i) έναν καταχωρητή CAS που ονομάζεται *status* και περιγράφει την κατάσταση της δοσοληψίας και ii) μια λίστα αναγνώσεων *readList* που περιέχει πληροφορίες για τις t-μεταβλητές που η δοσοληψία έχει προσπελάσει με καθολική ανάγνωση. Συγκεκριμένα κάθε στοιχείο item της readList περιγράφει κάποια t-μεταβλητή x και περιέχει i) έναν καταχωρητή ανάγνωσης εγγραφής *tvar* τύπου TmVar που περιγράφει τη x και ii) έναν καταχωρητή ανάγνωσης-εγγραφής *tvarVersion* που περιέχει ένα δείκτη προς τα δεδομένα της x που αναγνώσθηκαν (ο οποίος αναπαριστά την έκδοση της x τη χρονική στιγμή της καθολικής ανάγνωσης).

Στο Σχήμα 4.2 παρουσιάζονται οι δομές που χρησιμοποιούνται από τον αλγόριθμο DSTM. Σημειώνεται ότι το status μιας δοσοληψίας μπορεί να παίρνει τιμές από το σύνολο {ACTIVE, COMMITTED, ABORTED}, όπου κάθε μία χαρακτηρίζει την $T(i,n_i)$ ως ενεργή, επιτυχώς ολοκληρωμένη και μη επιτυχώς ολοκληρωμένη, αντίστοιχα. Επίσης, για τη λίστα αναγνώσεων readList υπάρχουν οι συναρτήσεις: i) *Read_List *NewReadList()* που αρχικοποιεί μία νέα λίστα αναγνώσεων και την επιστρέφει, ii) *Read_List *InsertInReadList(Read_List *readList, TmVar tvar, DATA *tvarVersion)* που εισάγει το στοιχείο tvar με έκδοση tvarVersion στη λίστα

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

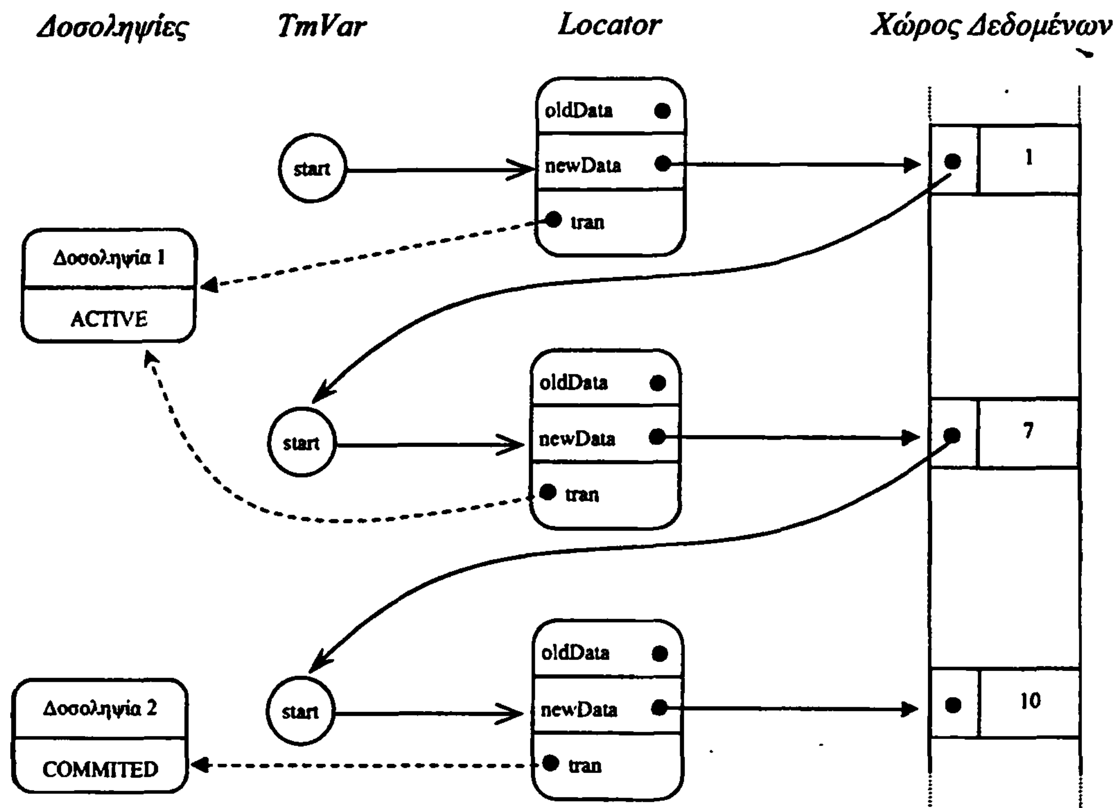
αναγνώσεων readList, iii) *DeleteItemFromReadList (Read_List *readList, Read_List *item)* που διαγράφει από τη λίστα αναγνώσεων readList το στοιχείο item, iv) *Read_List *SearchInReadList(Read_List readList, TmVar tvar)* που ερευνά εάν η λίστα readList διατηρεί κάποια καταχώρηση για την tvar και αν αυτό ισχύει επιστρέφει την καταχώρηση αυτή, ενώ σε αντίθετη περίπτωση επιστρέφει null, v) *Read_List *GetNextItemFromReadList(Read_List *readList)* που σε κάθε κλήση της επιστρέφει με τη σειρά όλα τα στοιχεία της λίστας readList και όταν αυτά τελειώσουν επιστρέφει null. Οι παραπάνω λειτουργίες υποστηρίζονται από οποιαδήποτε συμβατή υλοποίηση λίστας και έτσι οι κώδικές τους δεν παρουσιάζονται.

```
Read_List {
    TmVar *tvar;
    DATA *tvarVersion;
    Read_List *next;
}

Locator {
    Transaction *tran;
    DATA *newData;
    DATA *oldData;
}

Transaction {
    int status;
    Read_List *readList;
}

TmVar {
    Locator *start;
}
```

Σχήμα 4.2: Οι δομές που χρησιμοποιούνται από τον αλγόριθμο DSTM.

Ο αλγόριθμος DSTM χρησιμοποιεί τον δείκτη tran του Locator μιας t-μεταβλητής x για την περιγραφή της δοσοληψίας που κατέχει την προσωρινή ιδιοκτησία στη x. Συγκεκριμένα, εάν ο δείκτης tran δείχνει σε κάποια δομή $Trec(i,n_i)$ μιας δοσοληψίας $T(i,n_i)$ της οποίας το status είναι ACTIVE τότε η $T(i,n_i)$ κατέχει την προσωρινή ιδιοκτησία της x. Αν για την $T(i,n_i)$ ισχύει ότι status≠ACTIVE, τότε καμία δοσοληψία δεν κατέχει την προσωρινή ιδιοκτησία της x. Για το λόγο αυτό, λέμε ότι ο DSTM χρησιμοποιεί ανά t-μεταβλητή ανάθεση προσωρινών ιδιοκτησιών. Παρατηρούμε ότι η πληροφορία κατοχής προσωρινής ιδιοκτησίας σχετίζεται με τη δοσοληψία που κατέχει την προσωρινή ιδιοκτησία, είναι δηλαδή επώνυμη, ένα χαρακτηριστικό που έχουν οι μη-εμποδιστικοί αλγόριθμοι STM.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 4.3: Η μορφή μιας ταξινομημένης κατ' αύξουσα διάταξη συνδεδεμένης λίστας στον DSTM.

Στο Σχήμα 4.3 παρουσιάζεται η μορφή που θα είχε μια ταξινομημένη κατ' αύξουσα διάταξη συνδεδεμένη λίστα στον DSTM, που περιέχει τα στοιχεία 1,7,10. Σημειώνεται ότι όπως απαιτεί το αντικείμενο STM, ο DSTM δεν επιτρέπει να υπάρχουν δείκτες απευθείας μεταξύ των δεδομένων, αλλά ένα δεδομένο d_1 μπορεί να περιέχει δείκτη p σε ένα δεδομένο d_2 μόνο εάν ο p δείχνει στην αντίστοιχη t-μεταβλητή που περιγράφει το d_2. Επίσης, χάριν ευκολίας παρουσίασης του συγκεκριμένου παραδείγματος, υποθέτουμε ότι τα δεδομένα των t-μεταβλητών είναι οι κόμβοι της λίστας και όχι δείκτες προς τους κόμβους αυτούς (όπως συμβαίνει στην πράξη). Στο συγκεκριμένο σχήμα οι προσωρινές ιδιοκτησίες των t-μεταβλητών που περιγράφουν τα στοιχεία 1 και 7 της λίστας κατέχονται από τη Δοσοληψία 1 και η προσωρινή ιδιοκτησία της t-μεταβλητής που περιγράφει το στοιχείο 10 της λίστας έχει αποκτηθεί τελευταία φορά από την επιτυχημένη Δοσοληψία 2 και αυτή τη στιγμή δεν έχει αποκτηθεί από καμία δοσοληψία.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Ο αλγόριθμος DSTM υλοποιείται από ένα επεκτεταμένο αντικείμενο STM. Ο DSTM απαιτεί από τον χρήστη να εκκινήσει μία δοσοληψία εκτελώντας τη λειτουργία BeginTransaction χωρίς κανένα όρισμα. Η λειτουργία BeginTransaction επιστρέφει στο χρήστη ένα δείκτη στη δομή Trec της δοσοληψίας που εκκινήθηκε, τον οποίο ο χρήστης θα πρέπει να περνά ως όρισμα στις υπόλοιπες λειτουργίες που θα εκτελέσει μέσω της συγκεκριμένης δοσοληψίας. Ο κώδικας της λειτουργίας BeginTransaction παρουσιάζεται στο Σχήμα 4.4. Η λειτουργία αυτή δημιουργεί μια νέα δομή Trec (γραμμή 2) για τη δοσοληψία που πρόκειται να εκτελεστεί, δηλώνει ότι το status της είναι ACTIVE (γραμμή 3), αρχικοποιεί τη λίστα αναγνώσεών της εκτελώντας τη συνάρτηση NewReadList (γραμμή 4) και επιστρέφει τη νέα αυτή δομή στο χρήστη (γραμμή 5).

```
1    Transaction *BeginTransaction ()
2        Transaction *newTrec = allocMemory(Transaction);
3        newTrec→status = ACTIVE;
4        newTrec→readList = NewReadList();
5        return newTrec;
```

Σχήμα 4.4: Κώδικας της λειτουργίας BeginTransaction του DSTM.

```
6    (Boolean, DATA) ReadTmVar (Transaction *Trec, TmVar *tvar)
7        Boolean bool;
8        DATA *curData;
9        (bool, curData) = OpenTmVar(Trec, tvar, READ);
10       if (bool == TRUE) return (TRUE, *curData);
11       else return (FALSE, null);
```

Σχήμα 4.5: Κώδικας της λειτουργίας ReadTmVar του DSTM.

Ο κώδικας της λειτουργίας ReadTmVar παρουσιάζεται στο Σχήμα 4.5 και της AccessForUpdateTmVar στο Σχήμα 4.6. Μετα την εκκίνηση μιας δοσοληψίας $T(i,n_i)$ ο χρήστης είναι υπεύθυνος να προσπελάσει μέσω της λειτουργίας ReadTmVar και της λειτουργίας AccessForUdateTmVar τα δεδομένα κάθε t-μεταβλητής x που επιθυμεί να αναγνώσει και να ενημερώσει, αντίστοιχα, περνώντας ως παράμετρο στις λειτουργίες αυτές την αντίστοιχη TmVar *tvar* της x. Σημειώνεται ότι ο τρόπος υλοποίησης των λειτουργιών αυτών είναι αρκετά όμοιος. Για το λόγο αυτό

επιλέχθηκε η υλοποίησή τους μέσω της ίδιας συνάρτησης *OpenTmVar*, η οποία παραμετροποιείται κατάλληλα όπως περιγράφεται στη συνέχεια. Παρατηρούμε ότι οι λειτουργίες αυτές απλά καλούν την OpenTmVar και επιστρέφουν στο χρήστη ότι αυτή επέστρεψε.

```
12    (Boolean, DATA *) AccessForUpdateTmVar (Transaction *Trec, TmVar *tvar)
13        return (OpenTmVar(Trec, tvar, WRITE));
```

Σχήμα 4.6: Κώδικας της λειτουργίας AccessForUpdateTmVar του DSTM.

Ο κώδικας της λειτουργίας OpenTmVar του DSTM παρουσιάζεται στο Σχήμα 4.7. Η λειτουργία αυτή παίρνει ως όρισμα την tvar και επειδή μπορεί να καλείται είτε από την ReadTmVar (γραμμή 9) είτε από την AccessForUdateTmVar (γραμμή 13) δέχεται το όρισμα *mode*, το οποίο έχει τιμή READ εάν η tvar πρέπει να προσπελαστεί μόνο για ανάγνωση και WRITE αν πρέπει να προσπελαστεί για ενημέρωση. Στην περίπτωση που η προσπέλαση των δεδομένων της tvar γίνει επιτυχώς, η λειτουργία OpenTmVar επιστρέφει την τιμή TRUE και ένα δείκτη στα δεδομένα της tvar. Στην περίπτωση αυτή, η λειτουργία AccessForUdateTmVar επιστρέφει τις ίδιες αυτές τιμές στο χρήστη (γραμμή 13), ενώ η ReadTmVar επιστρέφει την τιμή TRUE και τα ίδια τα δεδομένα της tvar (γραμμή 10) και όχι τον δείκτη στα δεδομένα της. Σε αντίθετη περίπτωση, εάν η προσπέλαση των δεδομένων της tvar γίνει μη-επιτυχώς, η λειτουργία OpenTmVar επιστρέφει την τιμή FALSE και έναν null δείκτη. Στην περίπτωση αυτή, η ReadTmVar (γραμμή 11) και AccessForUdateTmVar επιστρέφουν την τιμή FALSE στο χρήστη και ο χρήστης είναι υποχρεωμένος να ολοκληρώσει τη δοσοληψία που εκτελεί ως μη-επιτυχημένη, καλώντας μία εκ των λειτουργιών CommitTransaction και AbortTransaction. Σημειώνεται ότι στην περίπτωση αυτή, ακόμα και αν κληθεί η λειτουργία CommitTransaction, ο DSTM εξασφαλίζει ότι η δοσοληψία θα ολοκληρωθεί ως μη επιτυχημένη.

Σημειώνεται ότι κάθε δοσοληψία $T(i,n_i)$ του αλγορίθμου DSTM καταγράφει τις απαραίτητες πληροφορίες για τις t-μεταβλητές που ο χρήστης επιθυμεί να διαβάσει στη λίστα αναγνώσεων που διατηρεί στη δομή της. Σημειώνεται ότι όπως θα

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

παρουσιαστεί στη συνέχεια, ο DSTM χρησιμοποιεί μη ορατές αναγνώσεις t-μεταβλητών και έτσι παρέχει έναν μηχανισμό ελέγχου συνέπειας των δεδομένων των t-μεταβλητών που αναγνώσθηκαν. Υπενθυμίζεται ότι ο μηχανισμός ελέγχου συνέπειας απαιτεί την ύπαρξη έκδοσης για κάθε δεδομένο. Ο DSTM εκμεταλλεύεται το κλειστό μοντέλο μνήμης στο οποίο εκτελείται για να ορίσει την έκδοση ενός δεδομένου. Το κλειστό μοντέλο μνήμης επιτρέπει στον αλγόριθμο DSTM να χρησιμοποιεί ως έκδοση ενός δεδομένου τη διεύθυνση στην οποία ξεκινά η μνήμη στην οποία είναι αποθηκευμένο το δεδομένο αυτό, όπως συζητήθηκε στην Ενότητα 3.5. Συμπεραίνουμε ότι απαιτείται η $T(i,n_i)$ να διατηρεί στη λίστα ανάγνωσης την τρέχουσα έκδοση των δεδομένων κάθε t-μεταβλητής που ο χρήστης διαβάζει, ώστε να είναι δυνατή η εκτέλεση του μηχανισμού ελέγχου συνέπειας.

Στη συνέχεια περιγράφεται η συνάρτηση OpenTmVar. Ανεξάρτητα από το εάν η OpenTmVar καλείται με παράμετρο READ ή WRITE, αρχικά προσπαθεί να ανακτήσει την έκδοση των τρεχόντων δεδομένων της tvar, χρησιμοποιώντας τη συνάρτηση GetCurrentData (γραμμή 20), η οποία επιστρέφει τα δεδομένα αυτά, καθώς επίσης και τον αντίστοιχο Locator του διαμοιραζόμενου αντικειμένου (η χρησιμότητα του οποίου εξηγείται παρακάτω). Όπως αναφέρθηκε, ο Locator που περιέχει κάθε t-μεταβλητή x, περιέχει δείκτες προς τα παλιά και τα καινούργια δεδομένα της x. Η GetCurrentData επιλέγει τα τρέχοντα δεδομένα της x βάσει του status της δοσοληψία που απέκτησε τελευταία την προσωρινή ιδιοκτησία της x. Εάν το status της δοσοληψίας αυτής είναι COMMITTED, η GetCurrentData επιλέγει τα νέα δεδομένα, ενώ εάν το status της είναι ABORTED, επιλέγει τα παλιά δεδομένα. (Περισσότερες λεπτομέρειες για τη συνάρτηση GetCurrentData παρουσιάζονται στη συνέχεια της παρούσας Ενότητας.)

Μετά την ανάκτηση των τρεχόντων δεδομένων της tvar η συνάρτηση OpenTmVar ενεργεί ανάλογα με την τιμή του ορίσματος mode. Σημειώνεται ότι παραβλέπουμε προσωρινά τις γραμμές 21 έως 23, η χρησιμότητα των οποίων παρουσιάζεται στη συνέχεια της παρούσας Ενότητας. Εάν, η τιμή του mode είναι WRITE, τότε η εν λόγω δοσοληψία, έστω $T(i,n_i)$, πρέπει να αποκτήσει την προσωρινή ιδιοκτησία της tvar, να δημιουργήσει ένα αντίγραφο των δεδομένων της και να επιστρέψει στον χρήστη το αντίγραφο αυτό, ώστε ο χρήστης να το ενημερώσει. Η διαδικασία

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

δημιουργίας αντιγράφου πραγματοποιείται μαζί με τη διαδικασία απόκτησης ιδιοκτησίας που περιγράφεται στη συνέχεια. Σημειώνεται ότι η απόκτηση της προσωρινή ιδιοκτησίας μιας t-μεταβλητής x που περιγράφεται από την TmVar $tvar_x$, από την $T(i,n_i)$, γίνεται ενημερώνοντας ατομικά (με τη βοήθεια μιας λειτουργίας CAS) το δείκτη $tvar_x$.start ώστε να δείχνει σε κάποιον νέο Locator, έστω loc, ο οποίος θα περιγράφει ως προσωρινό ιδιοκτήτη της x τη δοσοληψία αυτή, δηλαδή θα ισχύει loc.Tran == $Trec(i,n_i)$ και $Trec(i,n_i)$.status == ACTIVE.

```
14 (Boolean, DATA *) OpenTmVar (Transaction *Trec, TmVar *tvar, int mode)
15    DATA *returnData, *curData;
16    Locator *loc;
17    Read_List *readList = Trec→readList;
18    Read_List *listItem = SearchInReadList(readList, tvar)
19    while (1)
20        (loc, curData) = GetCurrentData (Trec, tvar)
21        if (listItem!=null && curData!=listItem→tvarVersion)
22            CAS (Trec→status, ACTIVE, ABORTED);
23            return (FALSE, null);
24        if (mode == WRITE && loc→tran != Trec)
25            newLoc = allocMemory(Locator);
26            newLoc→tran = Trec;
27            newLoc→oldData = curData;
28            newLoc→newData = allocMemory(DATA);
29            *(newLoc→newData) = *curData;
30            if (CAS(tvar.start, loc, newLoc)==TRUE)
31                 returnData = newLoc.newData;
32                 if (listItem != null)
33                        DeleteItemFromReadList (readList, listItem);
34            else continue;
35        else if (mode == WRITE && loc→tran == Trec)
36            returnData = curData;
37        else
38            returnData = curData;
39            if (listItem == null && loc→tran != Trec)
40                 InsertInReadList (readList, tvar, curData);
41        break;
42    if (Validate(Trec)==TRUE)
43        return (TRUE, returnData);
44    else
45        return (FALSE, null);
```

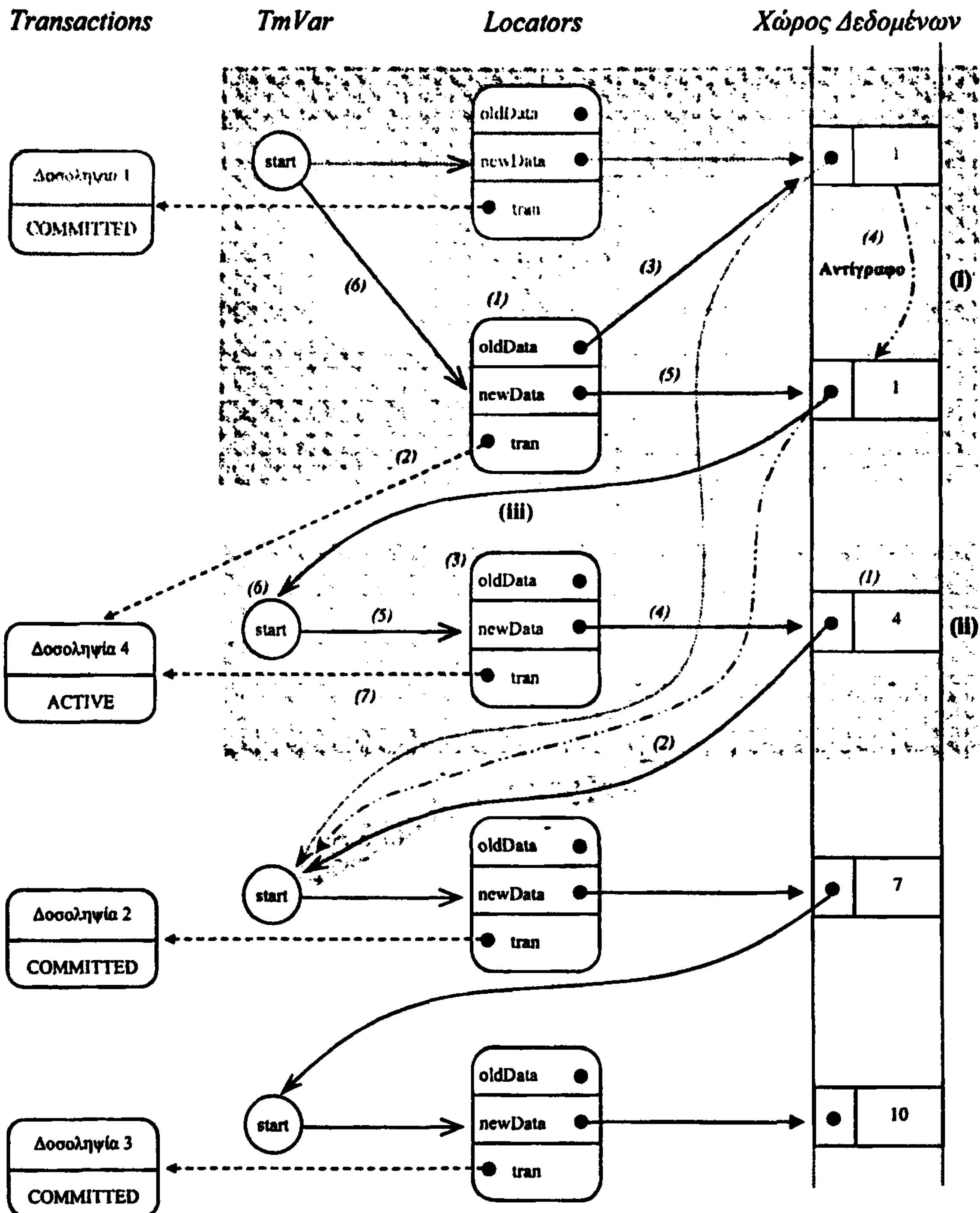
Σχήμα 4.7: Κώδικας της λειτουργίας OpenTmVar του DSTM.

Έστω x η t-μεταβλητή που περιγράφεται από την tvar. Εάν το mode έχει τιμή WRITE και η δοσοληψία που περιγράφεται από τον Locator που επιστρέφει η συνάρτηση GetCurrentData για τη x, δεν είναι ίδια με την $T(i,n_i)$ (γραμμή 24) (το οποίο σημαίνει ότι η $T(i,n_i)$ δεν έχει αποκτήσει ήδη την προσωρινή ιδιοκτησία της x), τότε για να αποκτήσει η OpenTmVar που εκτελείται από την $T(i,n_i)$ την προσωρινή ιδιοκτησία της x, εκτελεί τα παρακάτω. Αρχικά, δημιουργεί έναν νέο Locator (γραμμή 25), έστω newLoc, και i) αρχικοποιεί τον δείκτη newLoc→tran ώστε να δείχνει στη δική της δομή, την $Trec(i,n_i)$ (γραμμή 26) για την οποία ισχύει $Trec(i,n_i)$.status == ACTIVE, ii) αρχικοποιεί τον δείκτη newLoc→oldData ώστε να δείχνει στα τρέχοντα δεδομένα της x (γραμμή 27), iii) δημιουργεί μια καινούργια έκδοση των δεδομένων της x αποκτώντας νέα μνήμη για τα δεδομένα αυτά (γραμμή 28), αντιγράφει τα τρέχοντα δεδομένα της x στην έκδοση αυτή (γραμμή 29) και ταυτόχρονα διατηρεί έναν δείκτη προς την έκδοση αυτή στον καταχωρητή newLoc→newData (γραμμή 29). Στη συνέχεια προσπαθεί να ενημερώσει το δείκτη start της tvar ώστε να δείχνει στον newLoc, εκτελώντας την κατάλληλη εντολή CAS (γραμμή 30). Σημειώνεται ότι για την εκτέλεση της εντολής αυτής απαιτείται ο Locator που επιστρέφεται από τη συνάρτηση GetCurrentData. Εάν καταφέρει, να ενημερώσει το δείκτη αυτό, τότε λέμε ότι η $T(i,n_i)$ κατάφερε να αποκτήσει την προσωρινή ιδιοκτησία της x και δηλώνει ότι τα δεδομένα που πρόκειται να επιστραφούν στο χρήστη είναι αυτά του αντιγράφου που δημιούργησε (γραμμή 31), ενώ εάν δεν τα καταφέρει ξαναπροσπαθεί από την αρχή (γραμμή 34). Σημειώνεται ότι η $T(i,n_i)$ διατηρεί κάποια προσωρινή ιδιοκτησία μέχρι το status της να γίνει COMMITED ή ABORTED. Προσωρινά παραβλέπουμε τις γραμμές 32 και 33 η χρησιμότητα των οποίων παρουσιάζεται στη συνέχεια της παρούσας Ενότητας.

Για την καλύτερη κατανόηση της διαδικασίας απόκτησης της ιδιοκτησίας, δημιουργίας αντιγράφου και ενημέρωσης μιας t-μεταβλητής x παρουσιάζεται στο Σχήμα 4.8 ένα παράδειγμα εισαγωγής του στοιχείου 4 στην ταξινομημένη κατ' αύξουσα διάταξη συνδεδεμένη λίστα του παραδείγματος που παρουσιάστηκε στο Σχήμα 6.3, η οποία περιέχει αρχικά τα στοιχεία 1,7,10. Θεωρούμε ότι οι προσωρινές ιδιοκτησίες σε αυτά τα στοιχεία έχουν αποκτηθεί για τελευταία φορά από δοσοληψίες που έχουν ολοκληρωθεί επιτυχώς. Επίσης, θεωρούμε ότι ο χρήστης επιθυμεί να εισάγει το στοιχείο 4, κατά την εκτέλεση της *Δοσοληψίας 4* (σημειώνεται ότι η

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

δοσοληψία αυτή εκκινήθηκε από τον χρήστη με την εκτέλεση της λειτουργίας BeginTransaction).



Σχήμα 4.8: Εισαγωγή του στοιχείου 4, σε μια ταξινομημένη κατ' αύξουσα διάταξη συνδεδεμένη λίστα που αρχικά περιέχει τα στοιχεία 1,7,10.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Κατά την εκτέλεση του κώδικα του χρήστη εντοπίζεται ότι για την εισαγωγή του στοιχείου 4 στη συνδεδεμένη λίστα, απαιτείται η τροποποίηση των δεδομένων του στοιχείου 1. Για το λόγο αυτό ο χρήστης καλεί τη λειτουργία AccessForUpdateTmVar με όρισμα την TmVar που περιγράφει το στοιχείο 1, η οποία καλεί τη συνάρτηση OpenTmVar με mode == WRITE. Σημειώνεται ότι η συγκεκριμένη συνάρτηση OpenTmVar εκτελείται από τη *Δοσοληψία 4* και πρέπει να αποκτήσει την προσωρινή ιδιοκτησία του στοιχείου 1. Αυτό είναι το στάδιο (i) και παρουσιάζεται στο Σχήμα 4.8 μαζί με τις αλλαγές που προκαλεί στη λίστα. Η μορφή του στοιχείου 1 πριν την απόκτηση της προσωρινής ιδιοκτησίας του από τη Δοσοληψία 4 παρουσιάζεται με γκρίζο χρώμα, ενώ η μορφή του μετά την απόκτησή της με μαύρο χρώμα. Η απόκτηση της προσωρινής ιδιοκτησίας του στοιχείου 1 πραγματοποιείται με βάση τη διαδικασία που περιγράφηκε παραπάνω, κάθε βήμα της οποίας αριθμείται στο Σχήμα 4.8. Συνοπτικά αναφέρουμε ότι, στο βήμα 1 δημιουργείται ο νέος Locator newLoc για το στοιχείο 1, στα βήματα 2,3,5 ο newloc αρχικοποιείται κατάλληλα και στο βήμα 6 αλλάζει ο δείκτης που περιέχεται στο TmVar ώστε να δείχνει στον νέο Locator. Σημειώνεται ότι ο newLoc περιγράφει ως ιδιοκτήτη του στοιχείου 1 την εκκρεμή Δοσοληψία 4 (βήμα 2), ως παλιά δεδομένα του τα νέα δεδομένα του παλιού Locator (διότι ο παλιός Locator, περιγράφει ως ιδιοκτήτη την επιτυχώς ολοκληρωμένη Δοσοληψία 1) και ως νέα δεδομένα του ένα αντίγραφο των δεδομένων d (βήμα 5), το οποίο παράχθηκε με την εκτέλεση των γραμμών 28 και 29 στο βήμα 4. Σημειώνεται ότι το συγκεκριμένο αντίγραφο πραγματοποιείται όχι επί των δεδομένων του κόμβου 1, αλλά επί του δείκτη που δείχνει στον κόμβο αυτό, όμως χάριν απλότητας στην παρουσίαση του σχήματος επιλέγουμε να μην παρουσιάζουμε τους δείκτες αυτούς.

Κατά τον τερματισμό της, η συνάρτηση OpenTmVar επιστρέφει τα νέα δεδομένα, έστω returnData, που δεικτοδοτούνται από τον newLoc, διότι κατάφερε να αποκτήσει την ιδιοκτησία του στοιχείου 1. Σημειώνεται ότι τα δεδομένα returnData περιέχουν την τιμή 1 και έναν δείκτη προς το TMobject του στοιχείου 7, δηλαδή ότι ακριβώς περιέχουν και τα παλιά δεδομένα που δεικτοδοτούνται από τον newLoc. Είναι ευθύνη του χρήστη να δημιουργήσει τα νέα δεδομένα του στοιχείου 4 που επιθυμεί να εισάγει στο στάδιο (ii) και να ενημερώσει τα returnData στο στάδιο (iii). Τα στάδια αυτά παρουσιάζονται στο Σχήμα 4.8. Συγκεκριμένα, κατά το στάδιο (ii) ο χρήστης

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

δημιουργεί ένα καινούργιο στοιχείο της λίστας newListItem (βήμα 1), στο οποίο αποθηκεύει την τιμή 4 και αρχικοποιεί τον δείκτη του ώστε να δείχνει στο TmVar του στοιχείου 7 (βήμα 2). Στη συνέχεια, ο χρήστης καλεί τη λειτουργία CreateNewTmVar για να αντιστοιχίσει αυτό το νέο στοιχείο σε μια t-μεταβλητή. Το σύστημα STM δημιουργεί για το νέο αυτό στοιχείο έναν καινούργιο Locator loc (βήμα 3), αρχικοποιώντας το δείκτη loc.newData ώστε να δείχνει στο newListItem (βήμα 4) και τον δείκτη tran ώστε να δείχνει στη δομή της Δοσοληψίας 4 (βήμα 5), και δημιουργεί ένα καινούργιο TmVar για το στοιχείο 4 (βήμα 6) αρχικοποιώντας το δείκτη του ώστε να δείχνει στον loc (βήμα 7). Κατά το στάδιο (iii) ο χρήστης ενημερώνει τον δείκτη που περιέχεται στα δεδομένα returnData ώστε να δείχνει στην TmVar του νέου στοιχείου 4.

Στο σημείο αυτό η κατάσταση στην μνήμη είναι αυτή που παρουσιάζεται στο Σχήμα 4.8. Παρατηρούμε ότι η Δοσοληψία 4 διατηρεί προσωρινές ιδιοκτησίες στα στοιχεία 1 και 4 της λίστας. Σημειώνεται ότι το στοιχείο 4 της λίστας θα γίνει εμφανές στις υπόλοιπες δοσοληψίες εάν η Δοσοληψία 4 ολοκληρωθεί επιτυχώς, δηλαδή εάν το status της λάβει την τιμή COMMITED. Εάν αυτό συμβεί, μια δοσοληψία που θα προσπελάσει το TMobject του στοιχείου 1 (μέσω της OpenTmVar), θα διαβάζει τον Locator newLoc και μέσω αυτού θα διαβάσει τα δεδομένα newLoc.newData διότι ο newLoc περιγράφει ότι η προσωρινή ιδιοκτησία του στοιχείου 1 έχει αποκτηθεί για τελευταία φορά από μια δοσοληψία που ολοκληρώθηκε επιτυχώς (τη Δοσοληψία 4). Έτσι η δοσοληψία μπορεί να εντοπίσει το TmVar του στοιχείου 4 και να διαβάσει να δεδομένα του. Εάν στο status της Δοσοληψίας 4 γραφεί η τιμή ABORTED τότε το νέο στοιχείο 4 δεν θα είναι εμφανές σε καμία δοσοληψία (αφού τα τρέχοντα δεδομένα του στοιχείου 1 θα καθορίζονται σε αυτή την περίπτωση από τον δείκτη olddata του αντίστοιχου Locator για το στοιχείο 1). Στη συνέχεια της παρούσας ενότητας, περιγράφεται τι συμβαίνει στην περίπτωση που το status της Δοσοληψίας 4 παραμένει ACTIVE κατά την προσπέλαση του στοιχείου 1 από κάποια άλλη δοσοληψία.

Είναι σημαντικό ότι, εάν το mode έχει τιμή WRITE και η δοσοληψία που περιγράφεται από τον Locator που επιστρέφει η συνάρτηση GetCurrentData είναι η ίδια η $T(i,n_i)$ (γραμμή 35), τότε η $T(i,n_i)$ προσπαθεί να αποκτήσει την προσωρινή

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ιδιοκτησία μιας t-μεταβλητής που ήδη κατέχει. Έτσι, στην περίπτωση αυτή αρκεί να επιστραφούν στο χρήστη τα τρέχοντα δεδομένα που έχουν ήδη ανακτηθεί (γραμμή 36) μέσω της GetCurrentData.

Από την άλλη, εάν ο χρήστης δηλώσει ότι θέλει να προσπελάσει μια t-μεταβλητή x ώστε να τη διαβάσει, εκτελώντας τη λειτουργία ReadTmVar, η οποία εκτελεί τη συνάρτηση OpenTmVar με mode == READ, τότε η OpenTmVar δεν αποκτά την προσωρινή ιδιοκτησία της x, αλλά i) δηλώνει ότι τα δεδομένα που πρόκειται να επιστραφούν στο χρήστη είναι τα τρέχοντα δεδομένα του αντικειμένου (γραμμή 38) που έχει ήδη ανακτήσει, και ii) ενημερώνει τη λίστα readList (γραμμές 39 και 40). Σημειώνεται ότι η $T(i,n_i)$ έχει ήδη ψάξει στην readList εάν υπάρχει κάποια καταχώρηση $listItem_x$ για την TmVar tvar της x (γραμμή 18), με τη βοήθεια της συνάρτησης SearchInReadList. Αν δεν υπάρχει τέτοια καταχώρηση η συνάρτηση αυτή επιστρέφει την τιμή null. Σε αυτή την περίπτωση (γραμμή 39) εισάγεται η tvar στη readList και η έκδοσή της (δηλαδή ένας δείκτης στα τρέχοντα δεδομένα της) (γραμμή 40), με τη βοήθεια της συνάρτησης InsertInReadList. Αυτό εξασφαλίζει ότι η συνέπεια της x θα εξετάζεται κάθε φορά που εκτελείται ο μηχανισμός ελέγχου συνέπειας.

Στο σημείο αυτό εξηγείται η χρησιμότητα των γραμμών 21 έως 23 και 32 έως 33. Έστω μια οποιαδήποτε κλήση της λειτουργίας ReadTmVar για κάποια t-μεταβλητή x κατά τη διάρκεια εκτέλεσης μιας δοσοληψίας $T(i,n_i)$. Διακρίνω περιπτώσεις ανάλογα με το εάν η λειτουργία αυτή είναι καθολική προσπέλαση για ανάγνωση ή όχι. Εάν είναι καθολική ανάγνωση, θα έχει ως αποτέλεσμα να δημιουργηθεί μια καταχώρηση $listItem_x$ για τη x στην $Trec(i,n_i)$→readList. Στη συνέχεια, υπάρχει περίπτωση να κληθεί και πάλι για τη x μια εκ των λειτουργιών ReadTmVar και AccessForUdateTmVar από την $T(i,n_i)$. Στην περίπτωση αυτή, πριν την απόκτηση της προσωρινής ιδιοκτησίας της x (εάν έχει κληθεί η AccessForUdateTmVar) ή πριν την ενημέρωση της λίστας αναγνώσεων $Trec(i,n_i)$.readList (εάν έχει κληθεί η ReadTmVar), πρέπει να ελεγχθεί ότι τα δεδομένα που περιέχονται στο $listItem_x$→tvarVersion είναι συνεπή, δηλαδή δεν έχουν τροποποιηθεί, ώστε να είναι εγγυημένο ότι η $T(i,n_i)$ είναι σε κάθε περίπτωση σειριοποιήσιμη. Για το λόγο αυτό, ελέγχεται αν τα αποθηκευμένα δεδομένα listItem.tvarVersion για τη x στη

Trec(i,n_i).readList είναι διαφορετικά από τα τρέχοντα δεδομένα της x (γραμμή 21) και εάν αυτό ισχύει, η T(i,n_i) χαρακτηρίζεται ως μη-επιτυχημένη (γραμμή 22) και επιστρέφεται η τιμή FALSE στο χρήστη (γραμμή 23). Στην περίπτωση που έχει κληθεί η AccessForUdateTmVar, εάν τα δεδομένα αυτά είναι όμοια, μετά την απόκτηση της προσωρινής ιδιοκτησίας της x απομακρύνεται από την Trec(i,n_i).readList το $listItem_x$ (γραμμή 33) με την βοήθεια της συνάρτησης DeleteItemFromReadList, ώστε να εξαιρεθεί από τους επόμενους ελέγχους συνέπειας των δεδομένων της readList που πραγματοποιούνται με τη συνάρτηση Validate (η οποία περιγράφεται παρακάτω στην παρούσα Ενότητα). Σημειώνεται ότι εάν το στοιχείο $listItem_x$ δεν απομακρυνόταν από την readList, η συνάρτηση Validate θα αποφαινόταν λανθασμένα ότι τα δεδομένα του δεν είναι συνεπή.

Εάν η λειτουργία ReadTmVar δεν είναι καθολική ανάγνωση, σημαίνει ότι προηγείται κάποια λειτουργία ενημέρωσης, δηλαδή κάποια AccessForUdateTmVar, της x που θα έχει ως αποτέλεσμα η T(i,n_i) να αποκτήσει την προσωρινή ιδιοκτησία της x. Στην περίπτωση αυτή για λόγους απόδοσης αποφεύγεται η εισαγωγή του στοιχείου αυτού στη readList, με τη βοήθεια του δεύτερου ελέγχου της γραμμής 39.

Στο τέλος της εκτέλεσης της OpenTmVar, εξετάζεται εάν τα δεδομένα που η δοσοληψία T(i,n_i) έχει προσπελάσει για ανάγνωση και περιέχονται στη readList της T(i,n_i) είναι συνεπή, εκτελώντας τη συνάρτηση Validate (γραμμή 42). Η συνάρτηση αυτή επιστρέφει TRUE εάν τα δεδομένα που διάβασε η T(i,n_i) είναι ακόμα συνεπή και FALSE σε αντίθετη περίπτωση. Εάν η Validate επιστρέψει TRUE, τότε η OpenTmVar επιστρέφει την τιμή επιτυχίας TRUE και τα δεδομένα που έχουν ήδη οριστεί ότι πρόκειται να επιστραφούν στον χρήστη (γραμμή 43), ενώ εάν η Validate επιστρέψει FALSE, η ReadTmVar επιστρέφει στο χρήστη την τιμή αποτυχίας FALSE (γραμμή 45). Σημειώνεται ότι η συνάρτηση Validate υλοποιεί τον μηχανισμό ελέγχου συνέπειας των δεδομένων που έχουν προσπελαστεί για ανάγνωση μέχρι την τρέχουσα χρονική στιγμή από την T(i,n_i). Σημειώνεται επίσης, ότι ο μηχανισμός ελέγχου συνέπειας εκτελείται κάθε φορά που ο χρήστης προσπελάζει για ανάγνωση ή ενημέρωση μια t-μεταβλητή. Επομένως ο αλγόριθμος DSTM χρησιμοποιεί αυτόματο και αυξητικό έλεγχο συνέπειας.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Επειδή η απόκτηση των προσωρινών ιδιοκτησιών γίνεται κατά την εκτέλεση της λειτουργίας AccessForUdateTmVar, προκύπτει ότι ο DSTM χρησιμοποιεί εκ των προτέρων απόκτηση προσωρινών ιδιοκτησιών. Επίσης ο αλγόριθμος DSTM ενημερώνει τα δεδομένα των t-μεταβλητών τη στιγμή που τις προσπελάζει για πρώτη φορά, για το λόγο αυτό λέμε ότι χρησιμοποιεί τη μέθοδο άμεσης ενημέρωσης των t-μεταβλητών. Ακόμη, επειδή ο DSTM αποκτά ιδιοκτησίες μόνο στα διαμοιραζόμενα αντικείμενα που ο χρήστης επιθυμεί να προσπελάσει για να ενημερώσει, λέμε ότι ο DSTM χρησιμοποιεί μη-ορατές αναγνώσεις t-μεταβλητών.

Στη συνέχεια περιγράφεται η συνάρτηση GetCurrentData, ο κώδικας της οποίας παρουσιάζεται στο Σχήμα 4.9. Η συνάρτηση αυτή χρησιμοποιείται για την ανάκτηση των τρεχόντων δεδομένων μιας t-μεταβλητής. Όπως αναφέρθηκε, ο Locator που περιγράφει κάθε t-μεταβλητή x, περιέχει δείκτες προς τα παλιά και τα νέα δεδομένα της x. Τα τρέχοντα δεδομένα της x καθορίζονται με βάση το status της δοσοληψίας που απέκτησε τελευταία την προσωρινή ιδιοκτησία της x, έστω $T(j,n_j)$ η δοσοληψία αυτή. Ας υποθέσουμε ότι μια δοσοληψία $T(i,n_i)$ θέλει να προσπελάσει τα τρέχοντα δεδομένα της x. Αν το status της $T(j,n_j)$ είναι COMMITTED, τότε τα τρέχοντα δεδομένα της x είναι τα δεδομένα στα οποία δείχνει ο δείκτης newdata του Locator της x (γραμμή 53), ενώ εάν το status της $T(j,n_j)$ είναι ABORTED τότε τα τρέχοντα δεδομένα της x είναι τα δεδομένα στα οποία δείχνει ο δείκτης olddata του Locator της x (γραμμή 54). Σημειώνεται ότι και στις δύο αυτές περιπτώσεις σημαίνει ότι η $T(j,n_j)$ έχει καταργήσει την προσωρινή ιδιοκτησία της x. Αν η κατάστασή της $T(j,n_j)$ είναι ACTIVE (γραμμή 55) τότε δεν μπορεί να γίνει αποτίμηση των τρεχόντων δεδομένων της x πριν την ολοκλήρωση της $T(j,n_j)$ ως επιτυχημένης ή μη-επιτυχημένης. Στον DSTM μία δοσοληψία μπορεί να ολοκληρώσει «βίαια» κάποια άλλη, γράφοντας την τιμή ABORTED στο status της. Έτσι, η $T(i,n_i)$ προσπαθεί να ολοκληρώσει «βίαια» ως μη-επιτυχημένη την $T(j,n_j)$ (γραμμή 56) και αν τα καταφέρει, τα τρέχοντα δεδομένα του x είναι τα παλιά δεδομένα που περιγράφονται από τον Locator του (γραμμή 57). Εάν δεν τα καταφέρει, τότε το $Trec(j,n_j)$.status αποδεικνύεται (στην Ενότητα 4.2) ότι θα έχει εν τω μεταξύ αλλάξει στην τιμή COMMITED ή ABORTED και έτσι αφού η $T(i,n_i)$ διαβάσει το status (γραμμή 58), πραγματοποιεί ενέργειες αντίστοιχες με πριν (γραμμές 58 έως 60). Αν η προσωρινή ιδιοκτησία της x κατέχεται

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

από την ίδια τη δοσοληψία T(i,n$_i$) (γραμμή 50) επιστρέφονται τα νέα δεδομένα της x (γραμμή 51).

```
46   (Locator*,DATA*)GetCurrentData(Transaction *Trec,TMObject tmobj)
47       Locator *loc;
48       int status;
49       loc = tmobj.start;
50       if (loc→trans == Trec)
51           return (loc, loc→newData);
52       status = loc→tran.status;
53       if (status==COMMITTED) return (loc, loc→newData);
54       else if (status==ABORTED) return (loc, loc→oldData);
55       else if (status==ACTIVE)
56           if (CAS(loc→trans.status, ACTIVE, ABORTED)==TRUE)
57               return (loc, loc→oldData);
58           else if (loc→trans→status == COMMITTED)
59               return (loc, loc→newData);
60           else return (loc, loc→oldData);
```

Σχήμα 4.9: Κώδικας της συνάρτησης GetCurrentData του DSTM.

Σημειώνεται ότι η δυνατότητα «βίαιου» τερματισμού μιας δοσοληψίας από κάποια άλλη είναι απαραίτητη προκειμένου ο DSTM να εγγυάται την ιδιότητα τερματισμού ελευθερία ανταγωνισμού.

Στη συνέχεια περιγράφεται η συνάρτηση Validate, ο κώδικας της οποίας παρουσιάζεται στο Σχήμα 4.10. Η συνάρτηση Validate υλοποιεί το μηχανισμό ελέγχου συνέπειας. Αρχικά ελέγχεται η τιμή του Trec(i,n$_i$).status (γραμμή 65) και εάν η τιμή του είναι ABORTED, η Validate επιστρέφει την τιμή αποτυχίας FALSE (γραμμή 66). Σε αντίθετη περίπτωση, διατρέχονται όλες οι t-μεταβλητές που περιέχονται στη λίστα Trec(i,n$_i$).readList (γραμμή 67), με τη βοήθεια της συνάρτησης GetNextItemFromReadList. Για κάθε στοιχείο item της λίστας αυτής, i) ανακτάται η τρέχουσα έκδοση (curVersion) της t-μεταβλητής item→tvar (γραμμή 68), με τη βοήθεια της συνάρτησης GetCurrentData, και ii) ελέγχεται εάν η τρέχουσα έκδοσή της είναι ίδια με αυτή που είναι αποθηκευμένη στη readList (item→tvarVersion) (γραμμή 69). Η συνάρτηση Validate επιστρέφει TRUE εάν όλα τα στοιχεία της readList είναι συνεπή, και FALSE σε αντίθετη περίπτωση.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
61 Boolean Validate (Transaction *Trec)
62   Locator *loc;
63   DATA *curVersion;
64   Read_List *item, *readList = Trec→readList;
65   if (Trec→status == ABORTED)
66     return (FALSE);
67   while (item = GetNextItemFromReadList(readList))
68     (loc, curVersion) = GetCurrentData(Trec, item→obj);
69     if (curVersion != item→tvarVersion)
70       return (FALSE);
71   return (TRUE);
```

Σχήμα 4.10: Κώδικας της συνάρτησης Validate του DSTM.

Στη συνέχεια περιγράφεται η λειτουργία CreateNewTmVar του DSTM, ο κώδικας της οποίας παρουσιάζεται στο Σχήμα 4.11. Υπενθυμίζεται ότι ο χρήστης εκτελεί τη συγκεκριμένη λειτουργία κάθε φορά που επιθυμεί να δημιουργήσει μια νέα t-μεταβλητή και να την αρχικοποιήσει με τα δεδομένα *data*, κατά τη διάρκεια εκτέλεσης μιας δοσοληψίας $T(i,n_i)$. Η CreateNewTmVar δημιουργεί μια καινούργια t-μεταβλητή *newTvar* (γραμμή 73) και έναν καινούργιο Locator *newLoc* (74). Στη συνέχεια αρχικοποιεί τον newLoc έτσι ώστε ο δείκτης των παλιών δεδομένων του να είναι null (γραμμή 75), ο δείκτης των νέων δεδομένων του (αφού δεσμευθεί η απαραίτητη μνήμη στη γραμμή 76) να δείχνει στα δεδομένα data (γραμμή 77) και ο κάτοχος της ιδιοκτησίας της newtvar να είναι η $T(i,n_i)$ (γραμμή 78). Έπειτα, αρχικοποιείται ο δείκτης start της newTvar ώστε να δείχνει στον newLoc (γραμμή 79) και επιστρέφεται (γραμμή 80).

```
72   TmVar *CreateNewTmVar (Transaction *Trec, DATA data)
73     TmVar *newTvar = allocMemory(TmVar);
74     Locator *newLoc = allocMemory (Locator);
75     newLoc→oldData = null;
76     newLoc→newData = allocMemory(DATA);
77     *(newLoc→newData) = data;
78     newLoc→tran = Trec;
79     newTvar→start = newLoc;
80     return (newTvar);
```

Σχήμα 4.11: Κώδικας της λειτουργίας CreateNewTmVar του DSTM.

Μόλις ο χρήστης επιθυμεί να ολοκληρώσει τη δοσοληψία, πρέπει να καλέσει τη λειτουργία CommitTransaction ή την AbortTransaction για να γίνει προσπάθεια ολοκλήρωσης της δοσοληψίας ως επιτυχημένης ή μη-επιτυχημένης, αντίστοιχα. Ο κώδικας της λειτουργίας CommitTransaction παρουσιάζεται στο Σχήμα 4.12. Η λειτουργία αυτή, ελέγχει αρχικά εάν τα δεδομένα των t-μεταβλητών που προσπέλασε η δοσοληψία $T(i,n_i)$ για ανάγνωση είναι συνεπή, εκτελώντας τη συνάρτηση Validate (γραμμή 82). Εάν αυτό ισχύει, τότε γίνεται προσπάθεια να αλλαχθεί το status της $T(i,n_i)$ σε COMMITED, υποδηλώνοντας ότι η $T(i,n_i)$ ολοκληρώθηκε επιτυχώς, εκτελώντας την κατάλληλη εντολή CAS (γραμμή 83). Εάν αυτή η εντολή CAS ολοκληρωθεί επιτυχώς τότε η CommitTransaction επιστρέφει την τιμή TRUE (γραμμή 84). Σε οποιαδήποτε άλλη περίπτωση, εάν η τιμή του status της $T(i,n_i)$ είναι ακόμη ACTIVE (γραμμή 85) γράφεται στο status της $T(i,n_i)$ η τιμή ABORTED, υποδηλώνοντας ότι η $T(i,n_i)$ ολοκληρώθηκε μη-επιτυχώς, εκτελώντας την κατάλληλη λειτουργία CAS (γραμμή 86), και η λειτουργία CommitTransaction επιστρέφει τη τιμή FALSE (γραμμή 87). Σημειώνεται ότι εάν αποτύχει η εκτέλεση οποιαδήποτε εντολής CAS των γραμμών 83 και 86, τότε κατά την εκτέλεση της εντολής αυτής το status της $T(i,n_i)$ δεν έχει τιμή ACTIVE. Αυτό σημαίνει ότι κάποια άλλη δοσοληψία τερμάτισε «βίαια» την $T(i,n_i)$ αλλάζοντας το status της στην τιμή ABORTED.

```
81 Boolean CommitTransaction (Transaction *Trec)
82    if (Validate(Trec)==TRUE)
83       if (CAS(Trec→status, ACTIVE, COMMITTED))
84          return (TRUE);
85    if (Trec→status == ACTIVE)
86       CAS(Trec→status, ACTIVE, ABORTED);
87    return (FALSE);
```

Σχήμα 4.12: Κώδικας της λειτουργίας CommitTransaction του DSTM.

Σημειώνεται ότι στην περίπτωση που κάποια λειτουργία ReadTmVar ή AccessForUdateTmVar επιστρέψει στο χρήστη την τιμή FALSE, τότε η αντίστοιχη δοσοληψία θα ολοκληρωθεί σίγουρα μη-επιτυχώς. Όμως, αυτό θα πραγματοποιηθεί όταν κληθεί η αντίστοιχη λειτουργία CommitTransaction ή AbortTransaction.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Στο Σχήμα 4.13 παρουσιάζεται ο κώδικας της λειτουργίας AbortTransaction. Η λειτουργία αυτή καλείται από τον χρήστη όταν θέλει να ολοκληρώσει μια δοσοληψία $T(i,n_i)$ ως μη επιτυχημένη. Η λειτουργία αυτή, απλά αλλάζει το status της $T(i,n_i)$ από ACTIVE σε ABORTED, εκτελώντας την κατάλληλη λειτουργία CAS (γραμμή 89). Σημειώνεται ότι μόνο η ίδια η $T(i,n_i)$ μπορεί να αλλάξει το status της από ACTIVE σε COMMITTED. Έτσι, εάν αποτύχει η εκτέλεση της λειτουργίας CAS της γραμμής 89 σημαίνει ότι το status της $T(i,n_i)$ δεν είχε την τιμή ACTIVE, επομένως κάποια άλλη δοσοληψία χαρακτήρισε «βίαια» την $T(i,n_i)$ ως μη-επιτυχημένη, γράφοντας την τιμή ABORTED σε αυτό. Στην περίπτωση αυτή εκπληρώνεται ο στόχος της συνάρτησης AbortTransaction.

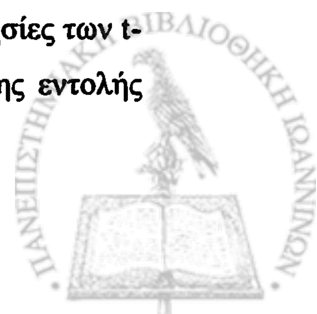
```
88    AbortTransaction (Transaction *Trec)
89        CAS(Trec→status, ACTIVE, ABORTED);
```

Σχήμα 4.13: Κώδικας της συνάρτησης AbortTransaction του DSTM.

4.2. Απόδειξη ορθότητας αλγορίθμου DSTM

Υπενθυμίζεται ότι η n_i-οστή δοσοληψία που εκκινείται από μια διεργασία p_i συμβολίζεται με $T(i,n_i)$ και η διεργασία p_i είναι ο *δημιουργός* της $T(i,n_i)$. Επίσης, το *διάστημα εκτέλεσης* μιας δοσοληψίας $T(i,n_i)$ συμβολίζεται ως $E(T(i,n_i))$. Όπως αναφέρθηκε, κάθε δοσοληψία $T(i,n_i)$ διατηρεί μια διαμοιραζόμενη δομή δεδομένων *Trec(i,nᵢ)* στην οποία περιέχεται ένας καταχωρητής CAS που ονομάζεται *status* και μια λίστα αναγνώσεων *readList*.

Έστω ότι μια δοσοληψία έχει εκτελέσει τις γραμμές 25 έως 29 του κώδικα, που σημαίνει ότι έχει δημιουργήσει έναν καινούργιο Locator newLoc για κάποια t-μεταβλητή x και ισχύει newLoc→tran == $Trec(i,n_i)$. Κάθε φορά που η δοσοληψία εκτελεί επιτυχώς την εντολή CAS της γραμμής 30 του κώδικα για τη x με το τρίτο όρισμα να είναι ο newLoc, λέμε ότι η δοσοληψία *γίνεται ιδιοκτήτης* της x ή *αποκτά την προσωρινή ιδιοκτησία* της x *βάσει του Locator newLoc*. Κάθε φορά που μια δοσοληψία εκτελεί επιτυχώς την εντολή CAS μιας εκ των γραμμών 22, 83, 86 και 89 στο status της, λέμε ότι η δοσοληψία *καταργεί* όλες τις προσωρινές ιδιοκτησίες των t-μεταβλητών που αυτή κατείχε. Η επιτυχημένη εκτέλεση της γραμμής της εντολής


ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

CAS της γραμμής 83 έχει σαν αποτέλεσμα και την *εφαρμογή* των *ενημερώσεων* της δοσοληψίας στις t-μεταβλητές των οποίων την ιδιοκτησία κατείχε. Κάθε φορά που μια δοσοληψία $T(i,n_i)$ εκτελεί επιτυχώς την εντολή CAS της γραμμής 56 στο status κάποιας δοσοληψίας $T(j,n_j)$, όπου $i \neq j$, λέμε ότι η $T(i,n_i)$ *καταργεί* όλες τις προσωρινές ιδιοκτησίες των t-μεταβλητών που η $T(j,n_j)$ κατείχε.

Για να αποδείξουμε την ορθότητα του αλγορίθμου DSTM πρέπει να δείξουμε ότι ικανοποιεί την ιδιότητα της σειριοποιησιμότητας και την ιδιότητα τερματισμού ελευθερία ανταγωνισμού.

4.2.1. Σειριοποιησιμότητα

Έστω α μια οποιαδήποτε εκτέλεση του DSTM και έστω μια δοσοληψία $T(i,n_i)$. Υπενθυμίζεται ότι εάν η $T(i,n_i)$ τερματίζει στην α, συμβολίζουμε με $Cf(i,n_i)$ την τελευταία καθολική κατάσταση του $E(T(i,n_i))$.

Λήμμα 4.1:Εάν το Trec(i,n_i).status μιας δοσοληψίας T(i,n_i) έχει τιμή διαφορετική από ACTIVE σε κάποια καθολική κατάσταση C της α, τότε καμία επιτυχημένη εντολή CAS στο Trec(i,n_i).status δεν πραγματοποιείται μετά την C.

Απόδειξη: Στη C το $Trec(i,n_i)$.status έχει τιμή v, όπου $v \in \{COMMITTED, ABORTED\}$. Ας υποθέσουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι πραγματοποιείται μια επιτυχημένη CAS στο $Trec(i,n_i)$.status μετά την C. Έστω CS_1 η *πρώτη* λειτουργία CAS που αλλάζει μετά την C το $Trec(i,n_i)$.status από v σε v', όπου $v' \in \{ACTIVE, COMMITTED, ABORTED\}$. Σημειώνεται ότι $Trec(i,n_i) \neq Trec(i,n_j)$ αν $n_i \neq n_j$, δηλαδή η δομή που περιγράφει μια δοσοληψία είναι μοναδική, ακόμη και για δοσοληψίες που εκκινήθηκαν από την ίδια διεργασία. Από τον κώδικα προκύπτει ότι το $Trec(i,n_i)$.status μπορεί να αλλάξει κατά την εκτέλεση μιας εκ των γραμμών 22, 56, 83, 86 ή 89. Σε όλες τις περιπτώσεις, η εντολή CAS εκτελείται έχοντας ως δεύτερο όρισμα την τιμή ACTIVE που σημαίνει ότι πρέπει να ισχύει $Trec(i,n_i)$.status==ACTIVE για να επιτύχει. Ωστόσο, όταν εκτελείται η CS_1 το $Trec(i,n_i)$.status≠ACTIVE. Άρα, η CS_1 αποτυγχάνει, το οποίο είναι άτοπο! ■

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

Από τον κώδικα (γραμμή 3) προκύπτει ότι κατά την εκκίνηση της $T(i,n_i)$ ισχύει $Trec(i,n_i)$.status==ACTIVE. Από το Λήμμα 4.1 προκύπτει ότι το $Trec(i,n_i)$.status της $T(i,n_i)$ μπορεί να ενημερωθεί σε COMMITED ή ABORTED μία μόνο φορά. Από τον κώδικα προκύπτει ότι το $Trec(i,n_i)$.status μπορεί να λάβει την τιμή COMMITED μόνο μέσω της εκτέλεσης της εντολής CAS της γραμμής 83 από την ίδια την $T(i,n_i)$. Σε αυτή την περίπτωση συμβολίζουμε με $Cc(i,n_i)$ την καθολική κατάσταση στην οποία αυτό συμβαίνει και χαρακτηρίζουμε την $T(i,n_i)$ ως *επιτυχημένη*. Το $Trec(i,n_i)$.status μπορεί να λάβει την τιμή ABORTED μέσω της εκτέλεσης της κατάλληλης εντολής CAS από οποιαδήποτε δοσοληψία. Σε αυτή την περίπτωση συμβολίζουμε με $Ca(i,n_i)$ την καθολική κατάσταση στην οποία αυτό συμβαίνει και χαρακτηρίζουμε την $T(i,n_i)$ ως *μη-επιτυχημένη*. Σημειώνεται ότι από το Λήμμα 4.1 προκύπτει ότι θα ορίζεται μία από τις $Cc(i,n_i)$ και $Ca(i,n_i)$.

Από τον κώδικα προκύπτει ότι η $T(i,n_i)$ μπορεί να ενημερώσει το $Trec(i,n_i)$.status, με την εκτέλεση της εντολής CAS μιας εκ των γραμμών 22, 83, 86 και 89. Έστω CS μια οποιαδήποτε τέτοια CAS που εκτελείται στην καθολική κατάσταση C. Εάν η CS αποτύχει, σημαίνει ότι το $Trec(i,n_i)$.status είχε τιμή διαφορετική από ACTIVE στη C. Από τον κώδικα (γραμμή 83) προκύπτει ότι η $Cc(i,n_i)$ μπορεί να οριστεί μόνο με την επιτυχή εκτέλεση της εντολής CAS της γραμμής 83. Επομένως, εάν η CS αποτύχει, τότε η $Ca(i,n_i)$ είναι καλά ορισμένη και προηγείται της C. Έτσι, προκύπτει το παρακάτω πόρισμα.

Πόρισμα 4.2: *Αν η $T(i,n_i)$ εκτελέσει μια εντολή CAS στο $Trec(i,n_i)$.status, σε κάποια καθολική κατάσταση C και αποτύχει, τότε η $Ca(i,n_i)$ είναι καλά ορισμένη και προηγείται της C.*

Από τον κώδικα προκύπτει ότι η $T(i,n_i)$ μπορεί να ενημερώσει το status μιας δοσοληψίας, έστω $T(j,n_j)$, $i \neq j$, από ACTIVE σε ABORTED, με την εκτέλεση της εντολής CAS της γραμμής 56. Έστω ότι αυτή εκτελείται σε κάποια καθολική κατάσταση C. Εάν η εντολή αυτή αποτύχει, σημαίνει ότι το $Trec(j,n_j)$.status είχε τιμή διαφορετική από ACTIVE κατά την εκτέλεσή της. Επομένως είτε η $Cc(j,n_j)$ είτε η $Ca(j,n_j)$ είναι καλά ορισμένη και προηγείται της C. Έτσι προκύπτει το παρακάτω Πόρισμα.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Πόρισμα 4.3: Έστω C η καθολική κατάσταση στην οποία μια δοσοληψία $T(i,n_i)$ εκτελεί την εντολή CAS της γραμμής 56 στο status μιας δοσοληψίας $T(j,n_j)$ και αποτυγχάνει. Τότε μία εκ των $Cc(j,n_j)$ και $Ca(j,n_j)$ είναι καλά ορισμένη και προηγείται της C.

Λήμμα 4.4: Έστω μια δοσοληψία $T(i,n_i)$ για την οποία ορίζεται η $Cf(i,n_i)$. Τότε:
i) εάν ορίζεται η $Cc(i,n_i)$ προηγείται της $Cf(i,n_i)$,
ii) εάν ορίζεται η $Ca(i,n_i)$ προηγείται της $Cf(i,n_i)$.

Απόδειξη: Πριν τον τερματισμό της, η $T(i,n_i)$ καλεί είτε τη λειτουργία CommitTransaction είτε τη λειτουργία AbortTransaction. Επομένως η $Cf(i,n_i)$ δε μπορεί να προηγείται της ολοκλήρωσης μιας εξ αυτών των λειτουργιών. Εάν η εκτέλεση της εντολής CAS, της γραμμής 83 ή μιας εκ των γραμμών 86 και 89 από την $T(i,n_i)$ επιτύχει, τότε στην αμέσως επόμενη καθολική κατάσταση ορίζεται η $Cc(i,n_i)$ ή η $Ca(i,n_i)$, αντίστοιχα, και προηγείται της $Cf(i,n_i)$. Εάν η εκτέλεση της εντολής CAS, μιας εκ των γραμμών 83, 86 και 89 από την $T(i,n_i)$, έστω στην καθολική κατάσταση C, αποτύχει, τότε με βάση το Πόρισμα 4.2, η $Ca(i,n_i)$ είναι καλά ορισμένη και προηγείται της C. Στην περίπτωση αυτή, επειδή η C προηγείται της $Cf(i,n_i)$, προκύπτει ότι η $Ca(i,n_i)$ θα προηγείται της $Cf(i,n_i)$. ■

Έστω ότι κάποια δοσοληψία $T(i,n_i)$ εκτελεί τη λειτουργία AccessForUpdateTmVar για κάποια t-μεταβλητή x και καταφέρνει να γίνει ιδιοκτήτης της x. Συμβολίζουμε με $C_x(i,n_i)$ την πρώτη καθολική κατάσταση στην οποία αυτό συμβαίνει. Εάν το $C_x(i,n_i)$ ορίζεται και αν προηγείται[4.1] της $Cc(i,n_i)$ ή της $Ca(i,n_i)$ (ανάλογα με το ποια ορίζεται), συμβολίζουμε με $a_x(i,n_i)$ το διάστημα εκτέλεσης της a που ξεκινά από την $C_x(i,n_i)$ και

[4.1] Σημειώνεται ότι η $C_x(i,n_i)$ δεν μπορεί να έπεται της $Cc(i,n_i)$ διότι από τον κώδικα (γραμμές 85 και 86) προκύπτει ότι η $Cc(i,n_i)$ μπορεί να οριστεί μόνο από την $T(i,n_i)$ και αυτό συμβαίνει αμέσως πριν την ολοκλήρωση της $T(i,n_i)$. Αντίθετα, η $C_x(i,n_i)$ μπορεί να έπεται της $Ca(i,n_i)$, διότι η $Ca(i,n_i)$ μπορεί να οριστεί κατά τον «βίαιο» τερματισμό της $T(i,n_i)$ από κάποια άλλη δοσοληψία (γραμμή 56) και στη συνέχεια η $T(i,n_i)$ να αποκτήσει την ιδιοκτησία κάποιου διαμοιραζόμενου αντικειμένου x κατά την εκτέλεση της αντίστοιχης λειτουργίας OpenTmVar.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

καταλήγει στην τελευταία καθολική κατάσταση που προηγείται είτε της $Cc(i,n_i)$ ή της $Ca(i,n_i)$. Επομένως κατά τη διάρκεια του $\alpha_x(i,n_i)$ ισχύει $Trec(i,n_i)$.status==ACTIVE. Σημειώνεται ότι το $\alpha_x(i,n_i)$ δεν ορίζεται, εάν είτε το $C_x(i,n_i)$ δεν ορίζεται, είτε το $C_x(i,n_i)$ δεν προηγείται της $Cc(i,n_i)$ ή της $Ca(i,n_i)$. Σημειώνεται ότι χρησιμοποιούμε το συμβολισμό C_x αντί του $C_x(i,n_i)$ και τον α_x αντί του $\alpha_x(i,n_i)$, όταν η δοσοληψία στην οποία αναφέρεται ο συμβολισμός εξάγεται εύκολα από τα συμφραζόμενα.

Λήμμα 4.5: Έστω μια οποιαδήποτε δοσοληψία $T(i,n_i)$ και έστω ένα οποιοδήποτε διαμοιραζόμενο αντικείμενο x για το οποίο ορίζεται η $\alpha_x(i,n_i)$. Καμία δοσοληψία δε μπορεί να αποκτήσει την προσωρινή ιδιοκτησίας της x, στην $\alpha_x(i,n_i)$, δηλαδή δε μπορεί να εκτελέσει επιτυχώς την εντολή CAS της γραμμής 30 για το x στην $\alpha_x(i,n_i)$.

Απόδειξη: Έστω, δια της μεθόδου της εις άτοπο απαγωγής, ότι στην $\alpha_x(i,n_i)$ υπάρχει τουλάχιστον μία δοσοληψία που καταφέρνει να αποκτήσει την προσωρινή ιδιοκτησία της x. Έστω, $T(j,n_j)$ η *πρώτη* από αυτές και έστω C η καθολική κατάσταση στην οποία αυτή αποκτά την προσωρινή ιδιοκτησία της x. Εφόσον η $T(j,n_j)$ αποκτά την προσωρινή ιδιοκτησία της x, από τον κώδικα προκύπτει ότι καλεί τη συνάρτηση OpenTmVar με ορίσματα $Trec(j,n_j)$, την TmVar $tvar_x$ που αντιστοιχεί στο x και mode == WRITE. Επιπρόσθετα η $T(j,n_j)$ εκτελεί επιτυχώς την εντολή CAS της γραμμής 30, έστω CS_x, για την $tvar_x$. Από τον κώδικα προκύπτει (γραμμή 24), ότι εάν ισχύει j == i, η γραμμή 30 δεν εκτελείται. Επομένως πρέπει να ισχύει, $j \neq i$.

Πριν η $T(j,n_j)$ εκτελέσει τη CS_x, εκτελεί στη γραμμή 20 τη συνάρτηση GetCurrentData για την $tvar_x$, η οποία επιστρέφει τα δεδομένα της x και τον αντίστοιχο Locator, έστω loc_x. Κατά την εκτέλεση της συνάρτησης GetCurrentData από την $T(j,n_j)$, γίνεται ανάγνωση της τιμής του loc_x στη γραμμή 49. Εάν η γραμμή 49 εκτελεστεί μετά την $C_x(i,n_i)$, τότε το loc_x→tran θα έχει τιμή $Trec(i,n_i)$ και το loc_x→tran→status θα έχει τιμή ACTIVE (εξαιτίας του ορισμού της CS_x και του ότι η CS_x προηγείται του τέλους της $\alpha_x(i,n_i)$). Από τον κώδικα (γραμμές 53 έως 60) προκύπτει ότι για να ολοκληρωθεί η εκτέλεση της συνάρτησης GetCurrentData πρέπει να εκτελεστεί η εντολή CAS της γραμμής 56. Εάν η συγκεκριμένη εντολή εκτελεστεί επιτυχώς θα οριστεί η $Ca(i,n_i)$, ενώ εάν εκτελεστεί μη-επιτυχώς, τότε από το Πόρισμα 4.3, προκύπτει ότι έχει οριστεί είτε η $Cc(i,n_i)$ ή η $Ca(i,n_i)$. Σε κάθε περίπτωση πριν το τέλος της GetCurrentData θα έχει οριστεί είτε η $Cc(i,n_i)$ ή η

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

$Ca(i,n_i)$. Αφού η εκτέλεση της CS_x έπεται της GetCurrentData, σημαίνει ότι η CS_x εκτελείται μετά το τέλος της $\alpha_x(i,n_i)$, το οποίο είναι άτοπο.

Συμπεραίνουμε ότι η γραμμή 49 εκτελείται πριν την $C_x(i,n_i)$. Στην $C_x(i,n_i)$, η $T(i,n_i)$ θα αποκτήσει την προσωρινή ιδιοκτησία της x, το οποίο σημαίνει ότι θα ορίσει έναν νέο Locator για τη x, διαφορετικό του loc_x. Έτσι, η εκτέλεση της CS_x μετά την $C_x(i,n_i)$ που καλείται ως CAS($tvar_x$, loc_x, newLoc), θα αποτύχει διότι ο Locator της $tvar_x$ δεν θα είναι ο loc_x[4.2]. Άτοπο. ■

Από τον κώδικα (γραμμές 19, 23, 24, 30, 34 και 41) προκύπτει ότι εάν η $T(i,n_i)$ επιθυμεί να αποκτήσει την προσωρινή ιδιοκτησία μιας t-μεταβλητής x, τότε η αντίστοιχη κλήση της συνάρτησης OpenTmVar επιστρέφει μόνο είτε εάν αυτό συμβεί (γραμμή 30), είτε εάν η OpenTmVar επιστρέψει την τιμή FALSE (γραμμή 23) που σημαίνει ότι δεν κατέστη δυνατό να αποκτηθεί η προσωρινή ιδιοκτησία της x. Στη δεύτερη περίπτωση η $T(i,n_i)$ θα καλέσει απαραίτητα μία εκ των λειτουργιών CommitTransaction ή AbortTransaction και θα ολοκληρωθεί. Επομένως, εάν η $T(i,n_i)$ δεν καταφέρει να αποκτήσει την προσωρινή ιδιοκτησία της x τότε δε μπορεί να εκτελέσει στη συνέχεια καμία λειτουργία ReadTmVar ή AccessForUdateTmVar σε κάποια άλλη t-μεταβλητή.

Έστω *l*, η τελευταία t-μεταβλητή που επιθυμεί να αποκτήσει η $T(i,n_i)$. Έστω ότι η $T(i,n_i)$ αποκτά την *l* στην καθολική κατάσταση C_l και έστω ότι ορίζεται το α_l. Από τον ορισμό της *l* προκύπτει ότι η $T(i,n_i)$ έχει αποκτήσει την προσωρινή ιδιοκτησία σε κάθε t-μεταβλητή x που επιθυμεί πριν αποκτήσει την ιδιοκτησία της *l*. Επομένως, για κάθε t-μεταβλητή x που η $T(i,n_i)$ επιθυμεί ορίζεται η C_x και προηγείται της C_l. Επειδή ορίζεται η α_l, η C_x προηγείται της $Cc(i,n_i)$ ή της $Ca(i,n_i)$. Συμπεραίνουμε ότι ορίζονται όλα τα διαστήματα α_x. Επειδή η κατάργηση των προσωρινών ιδιοκτησιών συμβαίνει κατά την αλλαγή του status της $T(i,n_i)$ σε COMMITTED ή ABORTED, από τον

[4.2] Σημειώνεται ότι δε θα μπορούσε να προκύψει το πρόβλημα ABA διότι, όπως έχει αναφερθεί, ο αλγόριθμος DSTM εκτελείται σε κλειστό μοντέλο μνήμης και η μεταβλητή loc_x είναι μια από τις ενεργές τοπικές μεταβλητές της p_j.

ορισμό του a_l προκύπτει ότι καμία δοσοληψία δεν μπορεί να καταργήσει τις προσωρινές ιδιοκτησίες που κατέχει η T(i,n_i) στις t-μεταβλητές στην a_l. Έτσι, με βάση τα παραπάνω και το Λήμμα 4.5 προκύπτει ότι στο διάστημα a_l η T(i,n_i) κατέχει τις προσωρινές ιδιοκτησίες σε όλες τις t-μεταβλητές που επιθυμεί και καμία άλλη δοσοληψία δε κατέχει κάποια από τις συγκεκριμένες ιδιοκτησίες. Επειδή ο χρήστης καλεί τη λειτουργία CommitTransaction αφού έχει ήδη αποκτήσει τις προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που επιθυμεί και επειδή η Cc(i,n_i) μπορεί να οριστεί μόνο με την εκτέλεση της εντολής CAS της γραμμής 83, προκύπτει ότι εάν η Cc(i,n_i) ορίζεται τότε θα έπεται της C_l. Έτσι προκύπτει άμεσα το παρακάτω πόρισμα.

Πόρισμα 4.6: *Εάν η Cc(i,n_i) ορίζεται, τότε από την κλήση της CommitTransaction από την T(i,n_i) και μέχρι την Cc(i,n_i), η T(i,n_i) κατέχει τις προσωρινές ιδιοκτησίες σε όλες τις t-μεταβλητές που επιθυμεί και καμία άλλη δοσοληψία δε μπορεί να κατέχει κάποια από τις συγκεκριμένες ιδιοκτησίες.*

Λήμμα 4.7: Έστω GCD μια οποιαδήποτε κλήση της λειτουργίας GetCurrentData από την T(i,n_i) για κάποια t-μεταβλητή x. Έστω ότι η GCD ολοκληρώνεται στην καθολική κατάσταση C και επιστρέφει έναν δείκτη σε δεδομένα, τον οποίο έχει διαβάσει από έναν Locator loc. Έστω T(j,n_j), i≠j, η δοσοληψία που δημιούργησε το loc ώστε να αποκτήσει βάσει αυτού την προσωρινή ιδιοκτησία της x. Πριν την C θα έχει οριστεί είτε η Cc(j,n_j) είτε η Ca(j,n_j).

Απόδειξη: Η GCD διαβάζει στην γραμμή 49 την τιμή του loc, έστω στην καθολική κατάσταση C_{49}. Σημειώνεται ότι ο έλεγχος της γραμμής 50 δε μπορεί να αποτιμηθεί σε TRUE διότι από υπόθεση i≠j, επομένως η GCD δε μπορεί να επιστρέψει στη γραμμή 51. Στη συνέχεια η GCD διαβάζει στη γραμμή 52 την τιμή του Trec(j,n_j).status, έστω στην καθολική κατάσταση C_{52}. Εάν ο έλεγχος μιας εκ των γραμμών 53 και 54 αποτιμηθεί ως αληθής, η GCD θα επιστρέψει στις γραμμές 53 και 54 αντίστοιχα, και είτε η Cc(j,n_j) είτε η Ca(j,n_j) θα έχει οριστεί πριν την C_{52}, άρα και πριν την C. Σε αντίθετη περίπτωση, στην C_{52}, θα ισχύει Trec(j,n_j).status == ACTIVE και η GCD θα εκτελέσει την εντολή CAS της γραμμής 56, έστω CS_{56} η εντολή αυτή. Εάν η CS_{56} εκτελεστεί επιτυχώς, θα οριστεί η Ca(j,n_j), ενώ εάν αποτύχει, από το

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Πόρισμα 4.3, προκύπτει ότι είτε η $Cc(j,n_j)$ είτε η $Ca(j,n_j)$ έχει οριστεί πριν την κλήση της CS_{56}. ■

Στο σημείο αυτό αποδίδουμε σημεία σειριοποίησης στις δοσοληψίες. Σημειώνεται ότι από το Λήμμα 4.1 προκύπτει ότι μόνο μία εκ των $Cc(i,n_i)$ και $Ca(i,n_i)$ μπορεί να οριστεί για μια δοσοληψία $T(i,n_i)$. Για μια μη επιτυχημένη δοσοληψία $T(i,n_i)$ (δηλαδή μια δοσοληψία για την οποία έχει οριστεί η $Ca(i,n_i)$) δε χρειάζεται να αποδοθεί σημείο σειριοποίησης, διότι αυτή δεν μπορεί να εκτελέσει επιτυχώς την εντολή CAS της γραμμής 83 και να ορίσει την $Cc(i,n_i)$, ενημερώνοντας έτσι κάποιο διαμοιραζόμενο αντικείμενο. Έτσι αποδίδεται σημείο σειριοποίησης σε κάποια δοσοληψία $T(i,n_i)$ μόνο εάν αυτή είναι επιτυχημένη, δηλαδή αν έχει εκτελέσει επιτυχώς την εντολή CAS της γραμμής 83. Στην περίπτωση αυτή η εκτέλεση της συνάρτησης Validate της γραμμής 82 από την $T(i,n_i)$ επιστρέφει TRUE. Έστω *$Cvl(i,n_i)$* η τελευταία καθολική κατάσταση που προηγείται της κλήσης της Validate. Τοποθετούμε το σημείο σειριοποίησης της $T(i,n_i)$ στην $Cvl(i,n_i)$.

Λήμμα 4.8:Έστω α μια οποιαδήποτε εκτέλεση του DSTM. Έστω δύο διαφορετικές επιτυχημένες δοσοληψίες $T(i,n_i)$ και $T(j,n_j)$ στην α που αποκτούν την προσωρινή ιδιοκτησία της ίδιας t-μεταβλητής x. Έστω ότι η $T(i,n_i)$ σειριοποιείται πριν από την $T(j,n_j)$. Τότε:

i. Η $Cc(i,n_i)$ προηγείται της $Cc(j,n_j)$,

ii. Εάν η $T(j,n_j)$ αποκτά την προσωρινή ιδιοκτησία της x στην καθολική κατάσταση $C_x(j,n_j)$, τότε η $Cc(i,n_i)$ προηγείται της $C_x(j,n_j)$,

iii. Η $Cc(i,n_i)$ προηγείται της $Cvl(j,n_j)$.

Απόδειξη: Επειδή οι δοσοληψίες $T(i,n_i)$ και $T(j,n_j)$ είναι επιτυχημένες ορίζονται οι καθολικές καταστάσεις $Cc(i,n_i)$ και $Cc(j,n_j)$. Επίσης οι $T(i,n_i)$ και $T(j,n_j)$ σειριοποιούνται στις καθολικές καταστάσεις $Cvl(i,n_i)$ και $Cvl(j,n_j)$, αντίστοιχα. Από υπόθεση η $Cvl(i,n_i)$ προηγείται της $Cvl(j,n_j)$. Έστω, δια της μεθόδου της εις άτοπο απαγωγής, ότι η $Cc(i,n_i)$ έπεται της $Cc(j,n_j)$. Επειδή η $Cvl(i,n_i)$ και η $Cvl(j,n_j)$ ορίζονται κατά την εκτέλεση της λειτουργίας CommitTransaction από την $T(i,n_i)$ και την $T(j,n_j)$, αντίστοιχα, από το Πόρισμα 4.6 προκύπτει ότι στην $Cvl(i,n_i)$ και στην $Cvl(j,n_j)$, η $T(i,n_i)$ και η $T(j,n_j)$, αντίστοιχα έχουν αποκτήσει τις προσωρινές

ιδιοκτησίες όλως των t-μεταβλητών που επιθυμούν, άρα και της x. Επομένως στο διάστημα Cvl(j,n_j) έως Cc(j,n_j), δύο διαφορετικές δοσοληψίες κατέχουν την προσωρινή ιδιοκτησία της x. Αυτό όμως αντιτίθεται στο Πόρισμα 4.6. Επομένως η Cc(i,n_i) προηγείται της Cc(j,n_j).

Έστω, δια της μεθόδου της εις άτοπο απαγωγής, ότι η C_x(j,n_j) προηγείται της Cc(i,n_i). Έστω ότι η T(i,n_i) αποκτά την προσωρινή ιδιοκτησία της x στην καθολική κατάσταση C_x(i,n_i), η οποία προηγείται της Cvl(i,n_i). Από το Λήμμα 4.5 προκύπτει ότι η C_x(j,n_j) δε μπορεί να ορίζεται στο διάστημα [C_x(i,n_i), Cc(i,n_i)]. Επομένως η C_x(j,n_j) πρέπει να ορίζεται πριν την C_x(i,n_i). Εφόσον από τον Ισχυρισμό i) του Λήμματος η Cc(j,n_j) έπεται της Cc(i,n_i), η Cc(j,n_j) έπεται της Cvl(i,n_i). Με βάση τα παραπάνω προκύπτει ότι στο διάστημα [C_x(i,n_i), Cc(i,n_i)] δύο διαφορετικές δοσοληψίες κατέχουν την προσωρινή ιδιοκτησία της x, το οποίο με βάση το Πόρισμα 4.6 είναι άτοπο. Επομένως η C_x(j,n_j) έπεται της Cc(i,n_i).

Όπως αναφέρθηκε, στην Cvl(j,n_j) η T(j,n_j) έχει αποκτήσει τις προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που επιθυμεί. Επομένως η Cvl(j,n_j) έπεται της C_x(j,n_j), το οποίο σημαίνει ότι η Cvl(j,n_j) έπεται της Cc(i,n_i). ■

Λήμμα 4.9: Έστω μια οποιαδήποτε επιτυχημένη δοσοληψία T(i,n_i). Έστω ότι η T(i,n_i) ενημερώνει κάποια t-μεταβλητή x, βάσει ενός Locator loc για τον οποίο ο δείκτης loc.newData έχει τιμή d. Έστω T(j,n_j) μια άλλη δοσοληψία, i≠j, που εκτελεί τη μια κλήση GCD της συνάρτησης GetCurrentData για τη x. Έστω C_R η καθολική κατάσταση στην οποία η GCD εκτελεί τη γραμμή 49. Εάν η C_R βρίσκεται εντός του διαστήματος α_x(i,n_i), τότε η GCD επιστρέφει (loc, d).

Απόδειξη: Έστω loc_x ο Locator της x που η GCD διαβάζει στη C_R. Εφόσον η C_R εκτελείται εντός του διαστήματος α_x από το Λήμμα 4.5 προκύπτει ότι στη C_R θα ισχύει loc_x = loc. Επομένως, από τον κώδικα της GCD προκύπτει ότι αυτή θα επιστρέψει τον loc ως Locator της x.

Έστω, δια της μεθόδου της εις άτοπο απαγωγής ότι η GCD επιστρέφει (loc, d'), όπου d'≠d. Από τον κώδικα (γραμμές 27 έως 29) προκύπτει ότι η τιμή του δείκτη loc.oldData είναι πάντα διαφορετική από την τιμή του δείκτη loc.newData. Εφόσον η GCD διαβάζει loc στην C_R, για να μπορεί να ισχύει d'≠d, η GCD πρέπει να επιστρέψει την τιμή του δείκτη loc.oldData. Από τον κώδικα προκύπτει ότι η GCD

επιστρέφει την τιμή του δείκτη loc.oldData, εάν επιστρέψει εκτελώντας μία από τις γραμμές, 54, 57 και 60. Έστω ότι η GCD επιστρέφει στην καθολική κατάσταση C. Εάν η GCD επιστρέψει εκτελώντας τη γραμμή 54, λόγω του ελέγχου που προηγείται προκύπτει ότι πριν την C θα έχει οριστεί η $Ca(i,n_i)$. Εάν επιστρέψει στη γραμμή 57, έχει επιτύχει η εντολή CAS της γραμμής 56 και έτσι η $Ca(i,n_i)$ προηγείται της C. Εάν επιστρέψει στη γραμμή 60, η εκτέλεση της εντολής CAS της γραμμής 56 έχει αποτύχει. Από το Πόρισμα 4.3 προκύπτει ότι πριν την εκτέλεση της εντολής αυτής θα έχει οριστεί μία εκ των $Cc(i,n_i)$ και $Ca(i,n_i)$. Εφόσον η GCD επιστρέφει στη γραμμή 60 σημαίνει ότι η $Ca(i,n_i)$ θα προηγείται της C. Έτσι, σε κάθε περίπτωση πρέπει να έχει οριστεί η $Ca(i,n_i)$ και από το Λήμμα 4.1 προκύπτει ότι στη συνέχεια δε μπορεί να οριστεί η $Cc(i,n_i)$, το οποίο αντιτίθεται στην υπόθεση ότι η $T(i,n_i)$ είναι επιτυχημένη δοσοληψία. ■

Έστω μια εκτέλεση α και έστω C μια οποιαδήποτε καθολική κατάσταση της α στην οποία κάποια δοσοληψία $T(i,n_i)$ είναι εκκρεμής. Συμβολίζουμε με $O_C = \{x_1, x_2, \ldots, x_k\}$ το σύνολο των t-μεταβλητών που περιέχονται στη readList της $T(i,n_i)$ στη C. Σημειώνεται ότι το πλήθος k των στοιχείων του συνόλου O_C εξαρτάται από την καθολική κατάσταση C. Συμβολίζουμε με $D(O_C) = \{d_1', d_2', \ldots, d_k'\}$ το σύνολο των δεικτών στα δεδομένα των αντικειμένων του O_C που περιέχονται στη readList της $T(i,n_i)$ στην C (υπενθυμίζεται ότι κάθε δείκτης d_b', $1 \leq b \leq k$, αναπαριστά την έκδοση της x_b τη χρονική στιγμή της καθολικής ανάγνωσής της). Υπενθυμίζουμε ότι κάθε επιτυχημένη δοσοληψία σειριοποιείται βάσει μιας καθολικής κατάστασης της α και πιο συγκεκριμένα της τελευταίας που προηγείται της τελευταίας Validate που εκτελεί η δοσοληψία. Έτσι, δεδομένης μιας καθολικής κατάστασης C και των σημείων σειριοποίησης των επιτυχημένων δοσοληψιών στην α, μπορούμε να καθορίσουμε την επακολουθία των επιτυχημένων δοσοληψιών της α που σειριοποιούνται σε καθολικές καταστάσεις που προηγούνται της C. Λέμε ότι το σύνολο $D(O_C)$ είναι συνεπές αν κάθε στοιχείο d_b', $1 \leq b \leq k$, του συνόλου αυτού δείχνει στα δεδομένα που εγγράφονται από την τελευταία επιτυχημένη δοσοληψία που σειριοποιείται πριν τη C και ενημερώνει την αντίστοιχη x_b.

Για κάθε $x_b \in O_C$, όπου $1 \leq b \leq k$, η εισαγωγή του x_b στην readList της $T(i,n_i)$ πραγματοποιείται με την εκτέλεση της συνάρτησης InsertInReadList της γραμμής 40.

Υποθέτουμε χάριν απλούστευσης ότι η εκτέλεση της συνάρτησης InsertInReadList πραγματοποιείται με μία μόνο εντολή. Συμβολίζουμε με $Crlx_b(i,n_i)$ την πρώτη καθολική κατάσταση που έπεται της εκτέλεσης της InsertInReadList από την $T(i,n_i)$ για το x_b. Η συνάρτηση InsertInReadList αποθηκεύει στη readList της $T(i,n_i)$ την TmVar του x_b και το δείκτη $data_b$, όπως αυτά επιστράφηκαν από την εκτέλεση της συνάρτησης GetCurrentData της γραμμής 20, έστω $GCDx_b$ η κλήση της συνάρτησης αυτής. Από τον κώδικα της GetCurrentData προκύπτει ότι ο δείκτης $data_b$ που επιστράφηκε, διαβάστηκε μέσω ενός Locator $locx_b$, ο οποίος διαβάστηκε από την $GCDx_b$ στη γραμμή 49, έστω στην καθολική κατάσταση $Crlocx_b(i,n_i)$.

Σημειώνεται ότι πριν την απόκτηση της προσωρινής ιδιοκτησίας μιας t-μεταβλητής x στη γραμμή 30 από κάποια δοσοληψία $T(i,n_i)$, η $T(i,n_i)$ δημιουργεί έναν καινούργιο Locator στη γραμμή 25 και στη συνέχεια στη γραμμή 28 δεσμεύει νέο χώρο για το δείκτη newData του Locator αυτού. Επειδή ο αλγόριθμος DSTM λειτουργεί σε κλειστό μοντέλο μνήμης, αυτός ο χώρος δε μπορεί να χρησιμοποιείται ήδη.

Λήμμα 4.10: Έστω δύο διαφορετικές επιτυχημένες δοσοληψίες $T(i,n_i)$ και $T(j,n_j)$ που επιθυμούν να αποκτήσουν την ιδιοκτησία της ίδιας t-μεταβλητής $x_b \in O_C$, όπου $1 \leq b \leq k$. Ας υποθέσουμε ότι η $T(i,n_i)$ σειριοποιείται πριν την $T(j,n_j)$. Έστω ότι η $T(i,n_i)$ ενημερώνει την x_b βάσει του Locator loc_i για τον οποίο ο δείκτης loc_i.newData έχει τιμή d και ότι η $T(j,n_j)$ ενημερώνει το x_b βάσει του Locator loc_j για τον οποίο ο δείκτης loc_j.newData έχει επίσης την τιμή d. Έστω C_j η καθολική κατάσταση στην οποία πραγματοποιείται η δέσμευση μνήμης μέσω της συνάρτησης allocMemory της γραμμής 28 που επιστρέφει την τιμή d για το loc_j.newData, από την $T(j,n_j)$. Έστω ότι κάποια άλλη δοσοληψία $T(m,n_m)$ εισάγει την τιμή d για την x_b στην readList της, με την $Crlocx_b(m,n_m)$ να προηγείται της $Cvl(j,n_j)$, και διατηρεί την τιμή αυτή στη λίστα της τουλάχιστον έως την $Cvl(j,n_j)$. Η C_j έπεται της $Cc(i,n_i)$ και προηγείται της $Crlocx_b(m,n_m)$.

Απόδειξη: Έστω, δια της μεθόδου της εις άτοπο απαγωγής, ότι η καθολική κατάσταση C_j προηγείται της $Cc(i,n_i)$. Έστω ότι η $T(i,n_i)$ αποκτά την προσωρινή ιδιοκτησία της x_b στην καθολική κατάσταση $Cx_b(i,n_i)$ και η $T(j,n_j)$ στην καθολική κατάσταση $Cx_b(j,n_j)$. Έστω C_i η καθολική κατάσταση στην οποία πραγματοποιείται η

δέσμευση μνήμης μέσω της συνάρτησης allocMemory της γραμμής 28 που επιστρέφει την τιμή d για το loc_i.newData, από την $T(i,n_i)$. Από τον κώδικα προκύπτει ότι η C_i προηγείται της $C_{xb}(i,n_i)$. Σημειώνεται ότι στο διάστημα $[C_i, Cc(i,n_i)]$ η θέση μνήμης d δεικτοδοτείται από την $T(i,n_i)$. Αφού η C_j προηγείται της $Cc(i,n_i)$ και το σύστημα μνήμης είναι κλειστό, από τον ορισμό της C_j προκύπτει ότι η C_j προηγείται της C_i.

Επίσης, η C_i προηγείται της $Cx_b(i,n_i)$ και άρα προηγείται της $Cvl(i,n_i)$. Επομένως η C_j προηγείται της $Cvl(i,n_i)$. Επειδή η $T(j,n_j)$ σειριοποιείται μετά την $T(i,n_i)$ από τον Ισχυρισμό ii) του Λήμματος 4.8 προκύπτει ότι η $Cx_b(j,n_j)$ έπεται της $Cc(i,n_i)$, επομένως έπεται της $Cvl(i,n_i)$. Επομένως, η C_i βρίσκεται εντός του διαστήματος $[C_j, Cx_b(j,n_j)]$, κατά τη διάρκεια του οποίου η θέση μνήμης d δεικτοδοτείται από την $T(j,n_j)$. Επειδή ο αλγόριθμος DSTM λειτουργεί σε κλειστό μοντέλο μνήμης δεν είναι δυνατό στην C_i να επαναποδοθεί η θέση μνήμης d. Συμπεραίνουμε ότι δε μπορεί να ισχύει loc_i.newData == d. Άτοπο. Επομένως η C_j έπεται της $Cc(i,n_i)$.

Σημειώνεται ότι στην $Crlocx_b(m,n_m)$ η $T(m,n_m)$ διάβασε για πρώτη φορά την τιμή d για το x_b (μέσω του $locx_b$) εκτελώντας την $GCDx_b$, ώστε στη συνέχεια να εισάγει την τιμή αυτή στη readList της. Έστω, δια της μεθόδου της εις άτοπο απαγωγής, ότι η C_j έπεται της $Crlocx_b(m,n_m)$. Η τιμή d που διατηρείται στη λίστα readList της $T(i,n_i)$, έχει διαβαστεί στην $Crlocx_b(m,n_m)$ (μέσω του $locx_b$) και διατηρείται στη readList της $T(i,n_i)$ τουλάχιστον μέχρι την $Cvl(j,n_j)$. Έτσι η C_j εκτελείται εντός του διαστήματος $[Crlocx_b(m,n_m), Cvl(j,n_j)]$. Επειδή ο αλγόριθμος DSTM λειτουργεί σε κλειστό μοντέλο μνήμης δεν είναι δυνατό στη C_j να επαναποδοθεί η θέση μνήμης d. Συμπεραίνουμε ότι δε μπορεί να ισχύει loc_j.newData == d, το οποίο αντιτίθεται στην υπόθεση. Επομένως η C_j προηγείται της $Crlocx_b(m,n_m)$.

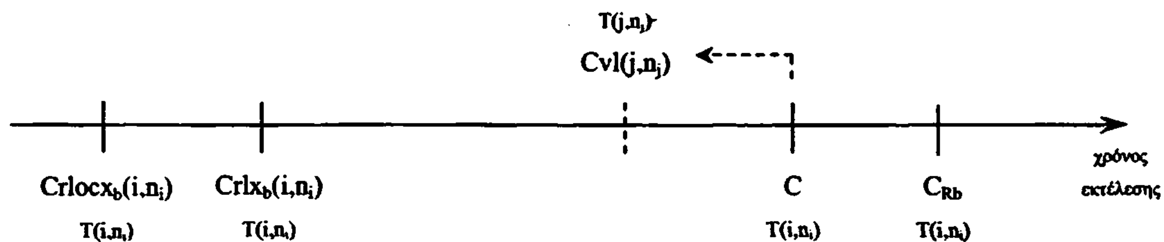
■

Έστω μια οποιαδήποτε εκτέλεση της συνάρτησης Validate από την $T(i,n_i)$ σε μια εκτέλεση α, η οποία επιστρέφει TRUE. Έστω V η κλήση της συγκεκριμένη Validate και έστω C η τελευταία καθολική κατάσταση που προηγείται της κλήσης της V. Από τον κώδικα της V, προκύπτει ότι η συνάρτηση GetCurrentData της γραμμής 68 εκτελείται για όλες τις t-μεταβλητές $x_b \in O_C$, όπου $1 \le b \le k$. Έστω $GCD_C(O_C) = \{d_1, d_2, \ldots, d_k\}$ το σύνολο των δεικτών που δείχνουν στα δεδομένα των t-μεταβλητών που περιέχονται στο O_C και επιστρέφονται από τις κλήσεις της GetCurrentData στη

γραμμή 68 από την V. Λέμε ότι το σύνολο $GCD_C(O_C)$ είναι συνεπές στη C, εάν κάθε στοιχείο d_b, $1 \leq b \leq k$, του συνόλου αυτού δείχνει στα δεδομένα που εγγράφονται από την τελευταία επιτυχημένη δοσοληψία που σειριοποιείται πριν την C και ενημερώνει το αντίστοιχο x_b του O_C.

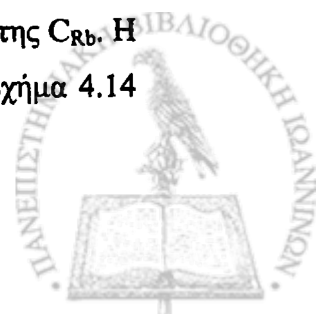
Λήμμα 4.11: Έστω μια οποιαδήποτε εκτέλεση της συνάρτησης Validate από μια δοσοληψία $T(i,n_i)$ σε μια εκτέλεση α, η οποία επιστρέφει TRUE. Έστω V η συγκεκριμένη Validate και έστω C η τελευταία καθολική κατάσταση που προηγείται της κλήσης της V. Τότε, το σύνολο $GCD_C(O_C)$ είναι συνεπές στη C.

Απόδειξη: Έστω $GCD_C(O_C) = \{d_1, d_2, \ldots, d_k\}$. Θα δειχθεί ότι για κάθε b, $1 \leq b \leq k$, το d_b εγγράφεται από την τελευταία επιτυχημένη δοσοληψία που σειριοποιείται πριν την C και ενημερώνει το x_b. Σημειώνεται ότι το d_b διαβάζεται από την $T(i,n_i)$ μέσω ενός Locator, ο οποίος διαβάζεται κατά την εκτέλεση της $GCDx_b$ στην $Crlocx_b(i,n_i)$ και εγγράφεται στη readList της $T(i,n_i)$ στην $Crlx_b(i,n_i)$. Έστω, δια της μεθόδου της εις άτοπο απαγωγής, ότι υπάρχει κάποιο διαμοιραζόμενο αντικείμενο $x_b \in O_C$ τέτοιο ώστε η τελευταία επιτυχημένη δοσοληψία, έστω $T(j,n_j)$, που ενημερώνει το x_b και σειριοποιείται πριν την C γράφει την τιμή $d_b' \neq d_b$ στο x_b. Σημειώνεται ότι η $T(j,n_j)$ σειριοποιείται στην $Cvl(j,n_j)$, η οποία από υπόθεση προηγείται της C.



Σχήμα 4.14: Εκτέλεση που περιγράφεται κατά την απόδειξη του Λήμματος 4.11.

Η $T(i,n_i)$ κατά την εκτέλεση της V εκτελεί τη συνάρτηση GetCurrentData για το x_b. Έστω GCD η κλήση αυτή. Εξ ορισμού της C, η GCD εκτελείται μετά τη C. Από τον κώδικα της GetCurrentData προκύπτει ότι η $T(i,n_i)$ διαβάζει την τιμή d_b για το x_b μέσω ενός Locator loc, τον οποίο έχει διαβάσει στη γραμμή 49, έστω στην καθολική κατάσταση C_{Rb}, κατά την εκτέλεση της GCD. Έτσι, η $Cvl(j,n_j)$ προηγείται της C_{Rb}. Η μορφή της εκτέλεσης την τρέχουσα χρονική στιγμή παρουσιάζεται στο Σχήμα 4.14

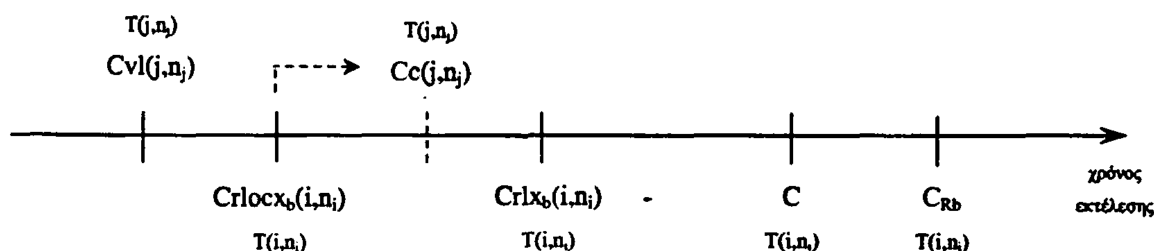

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

(το διακεκομμένο βελάκι υποδηλώνει ότι η Cvl(j,n_j) προηγείται της C και η διακεκομμένη γραμμή υποδηλώνει ότι η θέση της Cvl(j,n_j) δεν έχει καθοριστεί ακόμη).

Διακρίνω περιπτώσεις ανάλογα με το εάν η Crlocx$_b$(i,n_i) προηγείται ή έπεται της Cvl(j,n_j).

1) Έστω ότι η Crlocx$_b$(i,n_i) έπεται της Cvl(j,n_j). Διακρίνω περιπτώσεις ανάλογα με το εάν η Cc(j,n_j) προηγείται ή έπεται της Crlocx$_b$(i,n_i).

1.α) Έστω ότι η Cc(j,n_j) έπεται της Crlocx$_b$(i,n_i). Η μορφή της εκτέλεσης την τρέχουσα χρονική στιγμή παρουσιάζεται στο Σχήμα 4.15. Επειδή, η Crlocx$_b$(i,n_i) έπεται της εκκίνησης της CommitTransaction λειτουργίας που εκτελεί η T(j,n_j) (εφόσον έπεται της Cvl(j,n_j)) και προηγείται της Cc(j,n_j), από το Πόρισμα 4.6 προκύπτει ότι στην Crlocx$_b$(i,n_i) η T(j,n_j) κατέχει την προσωρινή ιδιοκτησία της x_b, δηλαδή η Crlocx$_b$(i,n_i) εκτελείται εντός του διαστήματος a_{xb}(j,n_j). Επειδή η T(j,n_j) ολοκληρώνεται επιτυχώς, από το Λήμμα 4.9 προκύπτει ότι η GCDx$_b$, θα επιστρέψει την τιμή του δείκτη locx$_b$.newData όπως αυτός αρχικοποιήθηκε από την T(j,n_j), δηλαδή την τιμή d_b', το οποίο είναι άτοπο.

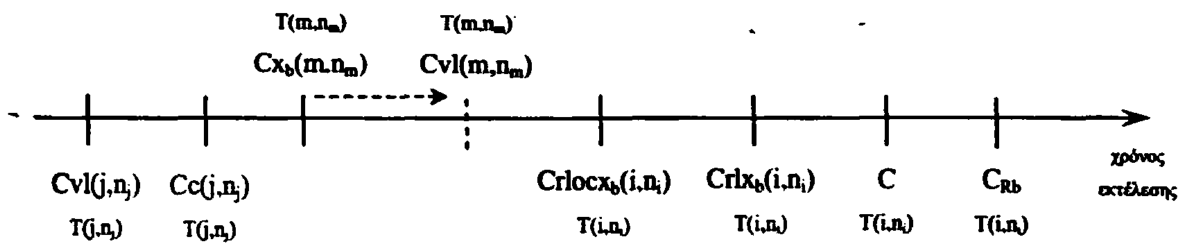


Σχήμα 4.15: Εκτέλεση που περιγράφεται κατά την απόδειξη του Λήμματος 4.11.

1.β) Επομένως πρέπει η Cc(j,n_j) να προηγείται της Crlocx$_b$(i,n_i). Στην περίπτωση αυτή, η T(i,n_i) διαβάζει στην Crlocx$_b$(i,n_i) την τιμή που γράφει η T(j,n_j) για το x_b, δηλαδή την d_b'. Για να μπορεί η T(i,n_i) να διαβάσει την τιμή d_b για το x_b στην Crlocx$_b$(i,n_i) πρέπει να υπάρχει κάποια επιτυχημένη δοσοληψία T(m,n_m), $m \neq i \neq j$, η οποία να αποκτά την ιδιοκτησία στο x_b και ταυτόχρονα να γράφει την τιμή d_b, έστω στην καθολική κατάσταση Cx$_b$(m,n_m), μετά την Cc(j,n_j), και πριν την Crlocx$_b$(i,n_i). Σημειώνεται ότι επειδή η Cvl(m,n_m) έπεται της Cx$_b$(m,n_m), η Cvl(m,n_m) θα έπεται της Cvl(j,n_j). Η μορφή της εκτέλεσης την τρέχουσα χρονική στιγμή παρουσιάζεται στο Σχήμα 4.16.

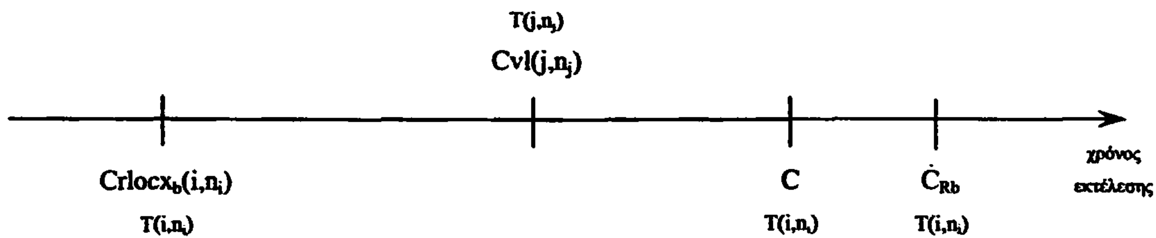
ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Ακόμη, η $Cc(m,n_m)$ μπορεί να προηγείται ή να έπεται της $Crlocx_b(i,n_i)$. Εάν η $Cc(m,n_m)$ προηγείται της $Crlocx_b(i,n_i)$, η $Cvl(m,n_m)$ θα προηγείται της C. Εάν η $Cc(m,n_m)$ έπεται της $Crlocx_b(i,n_i)$, από το Λήμμα 4.9 προκύπτει ότι η $GCDx_b$, θα επιστρέψει την τιμή d_b. Σημειώνεται ότι από τον ορισμό της $Crlx_b(i,n_i)$ προκύπτει ότι πριν την $Crlx_b(i,n_i)$ έχει ολοκληρωθεί η εκτέλεση της $GCDx_b$. Από το Λήμμα 5.7 προκύπτει ότι η $T(m,n_m)$ έχει ολοκληρωθεί πριν το τέλος της $GCDx_b$, δηλαδή πριν την $Crlx_b(i,n_i)$. Επειδή η $T(m,n_m)$ ολοκληρώνεται ως επιτυχημένη, προκύπτει ότι η $Cc(m,n_m)$ προηγείται της $Crlx_b(i,n_i)$, και άρα προηγείται της C (εφόσον η $Crlx_b(i,n_i)$ προηγείται της C). Άρα, σε κάθε περίπτωση η $Cvl(m,n_m)$ προηγείται της C. Επομένως σε κάθε περίπτωση η $Cvl(m,n_m)$ έπεται της $Cvl(j,n_j)$ και προηγείται της C. Άρα, η $T(j,n_j)$ δεν είναι η τελευταία δοσοληψία που σειριοποιείται πριν την C, το οποίο είναι άτοπο.



Σχήμα 4.16: Εκτέλεση που περιγράφεται κατά την απόδειξη του Λήμματος 4.11.

2) Επομένως η $Crlocx_b(i,n_i)$ προηγείται της $Cvl(j,n_j)$. Η μορφή της εκτέλεσης την τρέχουσα χρονική στιγμή παρουσιάζεται στο Σχήμα 4.17.

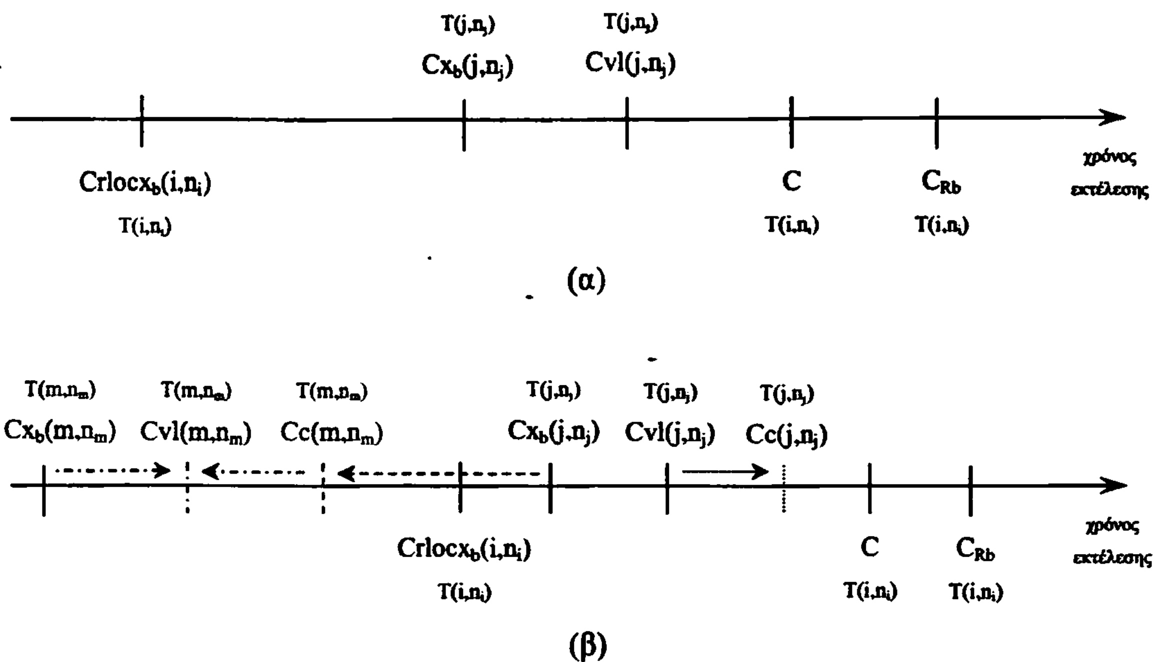


Σχήμα 4.17: Εκτέλεση που περιγράφεται κατά την απόδειξη του Λήμματος 4.11.

Έστω ότι η $T(j,n_j)$ αποκτά την προσωρινή ιδιοκτησία της x, στην καθολική κατάσταση $Cx_b(j,n_j)$, η οποία προηγείται της $Cvl(j,n_j)$. Σημειώνεται ότι η $Crlocx_b(i,n_i)$

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

δεν μπορεί να ορίζεται εντός του διαστήματος $[Cx_b(j,n_j), Cvl(j,n_j)]$, διότι στην περίπτωση αυτή από το Λήμμα 4.9 προκύπτει ότι η $GCDx_b$ θα επιστρέψει την τιμή d_b', το οποίο είναι άτοπο. Επομένως η $Crlocx_b(i,n_i)$ προηγείται της $Cx_b(j,n_j)$. Η μορφή της εκτέλεσης την τρέχουσα χρονική στιγμή παρουσιάζεται στο Σχήμα 4.18 (α). Για να μπορεί η $GCDx_b$ να επιστρέψει την τιμή d_b θα πρέπει να υπάρχει κάποια επιτυχημένη δοσοληψία $T(m,n_m)$, $m \neq i \neq j$, η οποία να αποκτά την ιδιοκτησία στο x_b και ταυτόχρονα να γράφει την τιμή d_b, έστω στην καθολική κατάσταση $Cx_b(m,n_m)$, πριν την $Crlocx_b(i,n_i)$. Έτσι η $Cx_b(m,n_m)$ προηγείται της $Cx_b(j,n_j)$. Από το Λήμμα 4.5 προκύπτει ότι τα διαστήματα $\alpha_{xb}(j,n_j)$ και $\alpha_{xb}(m,n_m)$ δεν μπορούν να αλληλοεπικαλύπτονται, και επειδή η $T(m,n_m)$ ολοκληρώνεται επιτυχώς προκύπτει ότι η $Cc(m,n_m)$ πρέπει να προηγείται της $Cx_b(j,n_j)$. Έτσι προκύπτει ότι η $Cc(m,n_m)$ προηγείται της $Cc(j,n_j)$ και η $Cvl(m,n_m)$ προηγείται της $Cvl(j,n_j)$. Η μορφή της εκτέλεσης την τρέχουσα χρονική στιγμή παρουσιάζεται στο Σχήμα 4.18 (β).



Σχήμα 4.18: Εκτελέσεις που περιγράφονται κατά την απόδειξη του Λήμματος 4.11.

Επομένως η $T(i,n_i)$ δε μπορεί να διαβάσει στην C_{Rb} την τιμή που γράφει η $T(m,n_m)$ για το x_b (d_b) διότι αυτή πανογράφεται από την τιμή που γράφει η $T(j,n_j)$ για το x_b (d_b'). Για να μπορεί η $T(i,n_i)$ να διαβάσει την τιμή d_b για την x_b πρέπει να υπάρχει κάποια άλλη επιτυχημένη δοσοληψία $T(k,n_k)$, $k \neq i \neq j \neq m$, η οποία να αποκτά

την ιδιοκτησία στο x_b και ταυτόχρονα να γράφει την τιμή d_b, έστω στην καθολική κατάσταση $Cx_b(k,n_k)$, μετά την $Cc(j,n_j)$, και η $Cx_b(k,n_k)$ να ορίζεται πριν την C_{Rb}. Επειδή η $Cvl(k,n_k)$ έπεται της $Cx_b(k,n_k)$, προκύπτει ότι η $Cvl(k,n_k)$ έπεται της $Cc(j,n_j)$ και της $Cvl(j,n_j)$. Επειδή η $Cvl(j,n_j)$ έπεται της $Cvl(m,n_m)$ προκύπτει ότι η $Cvl(k,n_k)$ έπεται της $Cvl(m,n_m)$.

Έστω ότι η $T(k,n_k)$ ενημερώνει το x_b βάσει του Locator loc_k, για τον οποίο ο δείκτης loc_k.newData έχει τιμή d_b. Έστω C_k η καθολική κατάσταση στην οποία πραγματοποιείται η δέσμευση μνήμης μέσω της συνάρτησης allocMemory της γραμμής 28 που επιστρέφει την τιμή d για το loc_k.newData, από την $T(k,n_k)$. Επειδή η $Cvl(k,n_k)$ έπεται της $Cvl(m,n_m)$, από το Λήμμα 4.10 προκύπτει ότι η C_k μπορεί να ορίζεται στο διάστημα $[Cc(m,n_m), Crlocx_b(i,n_i)]$, μόνο εάν η $Crlocx_b(i,n_i)$ έπεται της $Cc(m,n_m)$. Επομένως η $T(k,n_k)$ έχει δημιουργήσει και αρχικοποιήσει τον loc_k πριν την $Crlocx_b(i,n_i)$, και επομένως έχει διαβάσει ως Locator της x_b τον loc' με την εκτέλεση της συνάρτηση GetCurrentData της γραμμής 20, πριν την $Crlocx_b(i,n_i)$. Επομένως η $T(k,n_k)$ έχει διαβάσει τον loc' πριν την $Crlocx_b(i,n_i)$, άρα πριν την $Cx_b(j,n_j)$ και πριν την $Cvl(j,n_j)$, και στη συνέχεια προσπαθεί να αποκτήσει την ιδιοκτησία στο x με βάση τον loc_k, μετά την $Cvl(j,n_j)$, εκτελώντας της αντίστοιχη εντολή CAS, έστω CS_k, στην οποία περνά ως όρισμα τον loc'. Σημειώνεται ότι εάν η CS_k επιτύχει, ορίζεται η $Cx_b(k,n_k)$. Όμως, μετά την $Crlocx_b(i,n_i)$ και πριν την $Cvl(j,n_j)$ η $T(j,n_j)$ δημιουργεί έναν νέο Locator loc'', loc''≠loc', με βάση τον οποίο αποκτά την ιδιοκτησία του x, στην $Cx_b(j,n_j)$ η οποία προηγείται της εκτέλεσης της CS_k. Έτσι, από τον ορισμό της εντολής CAS προκύπτει ότι η εκτέλεση της CS_k μετά την $Cvl(j,n_j)$ θα αποτύχει επειδή ο Locator του x θα έχει αλλάξει από loc' σε loc''. Άτοπο. ∎

Σημειώνεται ότι εάν το σύνολο $GCD_C(O_C)$ είναι συνεπές στη C, τότε εάν ισχύει $D(O_C)==GCD_C(O_C)$, το $D(O_C)$ είναι και αυτό συνεπές στη C. Από τον κώδικα της συνάρτησης Validate (γραμμές 67 έως 71) προκύπτει ότι η V επιστρέφει την τιμή TRUE, εάν και μόνο εάν $D(O_C) == GCD_C(O_C)$. Έτσι με βάση το Λήμμα 4.11 προκύπτει το παρακάτω Πόρισμα.

Πόρισμα 4.11: *Έστω μια οποιαδήποτε εκτέλεση της συνάρτησης Validate από την $T(i,n_i)$ στην α, η οποία επιστρέφει TRUE. Έστω V η συγκεκριμένη Validate και έστω C*

η τελευταία καθολική κατάσταση που προηγείται της κλήσης της V. Τα στοιχεία του συνόλου $D(O_C)$ είναι συνεπή στη C.

Για κάθε καθολική κατάσταση C στο E(T(i,n_i)) συμβολίζουμε με $R_C = \{x_1, x_2, \ldots, x_{k'}\}$ το σύνολο των t-μεταβλητών που η T(i,n_i) έχει προσπελάσει για ανάγνωση, μέσω της λειτουργίας ReadTmVar, και έχει λάβει απάντηση με τιμή TRUE και τον δείκτη στα επιθυμητά δεδομένα από τη λειτουργία αυτή, μέχρι την C. Σημειώνεται ότι το πλήθος k' των στοιχείων του συνόλου R_C εξαρτάται από την καθολική κατάσταση C. Στο σύνολο R_C περιέχονται οι t-μεταβλητές που περιέχονται στη readList της T(i,n_i) στην C, δηλαδή τα στοιχεία του O_C, και όσες περιέχονταν στην readList της T(i,n_i) σε κάποια καθολική κατάσταση που προηγείται της C, οι οποίες αφαιρέθηκαν από την readList με την εκτέλεση της συνάρτησης DeleteItemFromReadList της γραμμής 33. Επομένως, το σύνολο O_C είναι υποσύνολο του συνόλου R_C. Επίσης, ισχύει $R_C = \bigcup_{\forall C' \in E(T(i,n_i)),\ C' < C} O_{C'}$, όπου C' κάθε καθολική κατάσταση του E(T(i,n_i)) που προηγείται της C. Συμβολίζουμε με $D(R_C) = \{d_1, d_2, \ldots, d_{k'}\}$ το σύνολο των τιμών των δεικτών στα δεδομένα των αντικειμένων του R_C. Συμβολίζουμε ως DIF_C το σύνολο $R_C - O_C$ που περιέχει τα στοιχεία που δεν ανήκουν στο O_C και ανήκουν στο R_C. Συμβολίζουμε με $D(DIF_C) = \{d_1', d_2', \ldots, d_{k'-k}'\}$ το σύνολο των τιμών των δεικτών στα δεδομένα των αντικειμένων του DIF_C. Έστω $Cdrx_b(i,n_i)$, $1 \leq b \leq k'-k$, η τελευταία καθολική κατάσταση στην οποία η τιμή d_b' για το αντίστοιχο διαμοιραζόμενο αντικείμενο x_b περιέχονταν στη readList της T(i,n_i). Σημειώνεται ότι η $Cdrx_b(i,n_i)$ προηγείται της C. Εάν υποθέσουμε χάριν απλούστευσης ότι η εκτέλεση της συνάρτησης DeleteItemFromReadList της γραμμής 33, πραγματοποιείται με μία μόνο εντολή, προκύπτει ότι η $Cdrx_b(i,n_i)$ είναι η τελευταία καθολική κατάσταση που προηγείται της κλήσης της συνάρτησης αυτής για το x_b από την T(i,n_i). Λέμε ότι τα στοιχεία του συνόλου $D(DIF_C)$ είναι συνεπή στην C, εάν κάθε στοιχείο d_b', $1 \leq b \leq k'-k$, του συνόλου αυτού δείχνει στα δεδομένα που εγγράφονται από την τελευταία επιτυχημένη δοσοληψία που σειριοποιείται πριν την C και ενημερώνει το αντίστοιχο x_b.

Λήμμα 4.13: Έστω μια οποιαδήποτε εκτέλεση της συνάρτησης Validate από την T(i,n_i) στην α, η οποία επιστρέφει TRUE. Έστω V η συγκεκριμένη Validate και έστω Cv η

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

τελευταία καθολική κατάσταση που προηγείται της κλήσης της V. Τα στοιχεία του συνόλου $D(DIF_{Cv})$ είναι συνεπή στη Cv.

Απόδειξη: Έστω $D(DIF_{Cv}) = \{d_1, d_2, \ldots, d_k\}$. Θα δειχθεί ότι για κάθε b, $1 \leq b \leq k$, το d_b εγγράφεται από την τελευταία επιτυχημένη δοσοληψία που σειριοποιείται πριν την Cv και ενημερώνει το x_b. Έστω, δια της μεθόδου της εις άτοπο απαγωγής, ότι υπάρχει κάποιο διαμοιραζόμενο αντικείμενο $x_b \in DIF_C$ τέτοιο ώστε η τελευταία επιτυχημένη δοσοληψία, έστω $T(j,n_j)$, που ενημερώνει το x_b και σειριοποιείται πριν την Cv γράφει την τιμή $d_b' \neq d_b$ στο x_b. Σημειώνεται ότι η $T(j,n_j)$ σειριοποιείται στην $Cvl(j,n_j)$, η οποία από υπόθεση προηγείται της Cv.

Σημειώνεται ότι η $Cdrx_b(i,n_i)$ είναι η τελευταία καθολική κατάσταση στην οποία η τιμή d_b για το αντίστοιχο διαμοιραζόμενο αντικείμενο x_b περιέχονταν στη readList της $T(i,n_i)$, και η $Cdrx_b(i,n_i)$ προηγείται της Cv. Έστω V' η τελευταία εκτέλεση της συνάρτησης Validate από την $T(i,n_i)$ στην α που πραγματοποιείται πριν την $Cdrx_b$, έστω στην καθολική κατάσταση Cv'. Σημειώνεται ότι η V' επιστρέφει TRUE και το x_b περιέχεται στο σύνολο $O_{V'}$. Από το Πόρισμα 4.12 προκύπτει ότι στη Cv' το στοιχείο d_b είναι συνεπές. Επομένως πρέπει η $Cvl(j,n_j)$ να έπεται της Cv'.

Σημειώνεται ότι το d_b διαβάζεται από την $T(i,n_i)$ μέσω ενός Locator, ο οποίος διαβάζεται από την $GCDx_b$ στην $Crlocx_b(i,n_i)$ και εγγράφεται στη readList της $T(i,n_i)$ στην $Crlx_b(i,n_i)$. Οι $Crlocx_b(i,n_i)$ και $Crlx_b(i,n_i)$ προηγούνται της Cv'. Για να μπορεί η $T(i,n_i)$ να διαβάσει την τιμή d_b για το x_b στην $Crlocx_b(i,n_i)$ πρέπει να υπάρχει κάποια επιτυχημένη δοσοληψία $T(m,n_m)$, $m \neq i \neq j$, η οποία να αποκτά την ιδιοκτησία στο x_b και ταυτόχρονα να γράφει την τιμή d_b, έστω στην καθολική κατάσταση $Cx_b(m,n_m)$, πριν την $Crlocx_b(i,n_i)$. Επίσης, επειδή η $Crlx_b(i,n_i)$ ορίζεται μετά την επιστροφή της $GCDx_b$ και από το Λήμμα 4.7 προκύπτει ότι πριν την επιστροφή της $GCDx_b$ θα έχει οριστεί η $Cc(m,n_m)$, προκύπτει ότι η $Cc(m,n_m)$ προηγείται της $Crlx_b(i,n_i)$. Επομένως η $Cc(m,n_m)$ προηγείται της Cv'. Σημειώνεται ότι επειδή η $Cc(j,n_j)$ έπεται της $Cvl(j,n_j)$ και η $Cvl(j,n_j)$ έπεται της Cv', προκύπτει ότι η $Cc(j,n_j)$ έπεται της $Cc(m,n_m)$.

Η $Cdrx_b(i,n_i)$ ορίζεται κατά την εκτέλεση της λειτουργίας ReadTmVar για το x_b. Από τον κώδικα (γραμμές 21, 24 και 30) προκύπτει ότι για να οριστεί η $Cdrx_b(i,n_i)$ είναι απαραίτητο να αποτιμηθεί ως αληθής ο έλεγχος της γραμμής 21 για το x_b, έστω $CHKx_b(i,n_i)$ ο έλεγχος αυτός, και να εκτελεστεί επιτυχώς η εντολή CAS της γραμμής 30 για το x_b, έστω $CSx_b(i,n_i)$ η εντολή αυτή, από την $T(i,n_i)$. Πριν την εκτέλεση του

$CHKx_b(i,n_i)$, εκτελείται η συνάρτηση GetCurrentData της γραμμής 20 για το x_b, έστω $GCD(i,n_i)$ η κλήση αυτή. Από τον κώδικα της $GCD(i,n_i)$ προκύπτει ότι η $T(i,n_i)$ διαβάζει την τιμή των δεδομένων του x_b μέσω ενός Locator loc, τον οποίο έχει διαβάσει στην γραμμή 49, έστω στην καθολική κατάσταση C_{Rb}, κατά την εκτέλεση της $GCD(i,n_i)$.

Έστω ότι η $T(j,n_j)$ αποκτά την ιδιοκτησία στο x, στην καθολική κατάσταση $Cx_b(j,n_j)$, η οποία προηγείται της $Cvl(j,n_j)$, με την αντίστοιχη εκτέλεση της εντολής CAS της γραμμής 30, έστω $CSx_b(j,n_j)$ η εντολή αυτή. Διακρίνω περιπτώσεις ανάλογα με το εάν η $Cx_b(j,n_j)$ προηγείται ή έπεται της C_{Rb}.

1) Έστω ότι η $Cx_b(j,n_j)$ προηγείται της C_{Rb}. Διακρίνω περιπτώσεις ανάλογα με το εάν η $Cc(j,n_j)$ προηγείται ή έπεται της C_{Rb}.

1.α) Έστω ότι η $Cc(j,n_j)$ έπεται της C_{Rb}. Επειδή, η C_{Rb} έπεται της $Cx_b(j,n_j)$ και προηγείται της $Cc(j,n_j)$ προκύπτει ότι στην C_{Rb} η $T(j,n_j)$ κατέχει την προσωρινή ιδιοκτησία της x_b, δηλαδή η C_{Rb} εκτελείται εντός του διαστήματος $\alpha x_b(j,n_j)$. Επειδή η $T(j,n_j)$ ολοκληρώνεται επιτυχώς, από το Λήμμα 4.9 προκύπτει ότι η $GCD(i,n_i)$, θα επιστρέψει την τιμή του δείκτη loc.newData όπως αυτός αρχικοποιήθηκε από την $T(j,n_j)$, δηλαδή την τιμή d_b' η οποία είναι διαφορετική της d_b. Στην περίπτωση αυτή ο έλεγχος $CHKx_b(i,n_i)$ θα αποτιμηθεί ως μη-αληθής και επομένως δε θα μπορεί να οριστεί η $Cdrx_b(i,n_i)$, το οποίο είναι άτοπο.

1.β) Επομένως πρέπει η $Cc(j,n_j)$ να προηγείται της $C_{Rb}(i,n_i)$. Στην περίπτωση αυτή, επειδή η $Cc(j,n_j)$ έπεται της $Cc(m,n_m)$, η $T(i,n_i)$ διαβάζει στην C_{Rb} την τιμή που γράφει η $T(j,n_j)$ για το x_b, δηλαδή την d_b'. Για να μπορεί η $T(i,n_i)$ να διαβάσει την τιμή d_b για το x_b στην C_{Rb} πρέπει να υπάρχει κάποια άλλη επιτυχημένη δοσοληψία $T(k,n_k)$, $k \neq i \neq j \neq m$, η οποία να αποκτά την ιδιοκτησία στο x_b και ταυτόχρονα να γράφει την τιμή d_b, έστω στην καθολική κατάσταση $Cx_b(k,n_k)$, μετά την $Cc(j,n_j)$, και πριν την C_{Rb}.

Έστω ότι η $T(k,n_k)$ ενημερώνει το x_b βάσει του Locator loc_k, για τον οποίο ο δείκτης loc_k.newData έχει τιμή d_b. Έστω C_k η καθολική κατάσταση στην οποία πραγματοποιείται η δέσμευση μνήμης μέσω της συνάρτησης allocMemory της γραμμής 28 που επιστρέφει την τιμή d για το loc_k.newData, από την $T(k,n_k)$. Επειδή η $Cvl(k,n_k)$ έπεται της $Cvl(m,n_m)$, από το Λήμμα 4.10 προκύπτει ότι η C_k μπορεί να ορίζεται στο διάστημα $[Cc(m,n_m), Crlocx_b(i,n_i)]$, μόνο εάν η $Crlocx_b(i,n_i)$ έπεται της $Cc(m,n_m)$. Επομένως η $T(k,n_k)$ έχει δημιουργήσει και αρχικοποιήσει τον loc_k πριν την

$Crlocx_b(i,n_i)$, και επομένως έχει διαβάσει ως Locator της x_b τον loc' με την εκτέλεση της συνάρτηση GetCurrentData της γραμμής 20, πριν την $Crlocx_b(i,n_i)$. Επομένως η $T(k,n_k)$ έχει διαβάσει τον loc' πριν την $Crlocx_b(i,n_i)$, άρα πριν την $Cx_b(j,n_j)$ και πριν την $Cvl(j,n_j)$, και στη συνέχεια προσπαθεί να αποκτήσει την ιδιοκτησία στο x με βάση τον loc_k, μετά την $Cvl(j,n_j)$, εκτελώντας της αντίστοιχη εντολή CAS, έστω CS_k, στην οποία περνά ως όρισμα τον loc'. Σημειώνεται ότι εάν η CS_k επιτύχει, ορίζεται η $Cx_b(k,n_k)$. Όμως, μετά την $Crlocx_b(i,n_i)$ και πριν την $Cvl(j,n_j)$ η $T(j,n_j)$ δημιουργεί έναν νέο Locator loc'', loc''≠loc', με βάση τον οποίο αποκτά την ιδιοκτησία του x, στην $Cx_b(j,n_j)$ η οποία προηγείται της εκτέλεσης της CS_k. Έτσι, από τον ορισμό της εντολής CAS προκύπτει ότι η εκτέλεση της CS_k μετά την $Cvl(j,n_j)$ θα αποτύχει επειδή ο Locator του x θα έχει αλλάξει από loc' σε loc''. Άτοπο.

2) Επομένως πρέπει η $Cx_b(j,n_j)$ να έπεται της C_{Rb}. Στην περίπτωση αυτή, ο loc που διαβάζεται στην C_{Rb} από την $T(i,n_i)$ και περιγράφει το x_b δεν έχει δημιουργηθεί από την $T(j,n_j)$. Διακρίνω περιπτώσεις ανάλογα με το εάν η $Cx_b(j,n_j)$ προηγείται ή έπεται της εκτέλεσης της $CSx_b(i,n_i)$ από την $T(i,n_i)$.

2.α) Έστω ότι η $Cx_b(j,n_j)$ προηγείται της $CSx_b(i,n_i)$. Στην περίπτωση αυτή, η $T(j,n_j)$ στην $Cx_b(j,n_j)$ θα αποκτήσει την ιδιοκτησία που αντιστοιχεί στο x_b, κάτι που σημαίνει ότι θα ορίσει έναν νέο Locator για το x_b, διαφορετικό του loc. Έτσι, η εκτέλεση της $CSx_b(i,n_i)$ μετά την $Cx_b(j,n_j)$, θα αποτύχει διότι ο Locator του x_b δεν θα είναι ο loc[4.3]. Άτοπο.

2.β) Έστω ότι η $Cx_b(j,n_j)$ έπεται της $CSx_b(i,n_i)$. Στην περίπτωση αυτή η $CSx_b(i,n_i)$ θα εκτελεστεί επιτυχώς, και στην αμέσως επόμενη καθολική κατάσταση θα οριστεί η $Cx_b(i,n_i)$. Επομένως η $Cx_b(j,n_j)$ έπεται της $Cx_b(i,n_i)$. Ακόμη επειδή η $Cx_b(j,n_j)$ έπεται της $Cvl(j,n_j)$ και η $Cvl(j,n_j)$ έπεται της Cv, προκύπτει ότι η $Cx_b(j,n_j)$ ορίζεται εντός του διαστήματος [$Cx_b(i,n_i)$, Cv]. Η $T(j,n_j)$ για να αποκτήσει την ιδιοκτησία στο x_b πρέπει να εκτελεστεί επιτυχώς την εντολή CAS της γραμμής 30 για το x_b, έστω $CSx_b(j,n_j)$ η εντολή αυτή. Επίσης, πριν την $CSx_b(j,n_j)$ η $T(j,n_j)$, αντίστοιχα με την $T(i,n_i)$, διαβάζει

[4.3] Σημειώνεται ότι δε θα μπορούσε να προκύψει το πρόβλημα ABA διότι, όπως έχει αναφερθεί, ο αλγόριθμος DSTM εκτελείται σε κλειστό μοντέλο μνήμης και η μεταβλητή loc είναι μια από τις ενεργές τοπικές μεταβλητές της p_i.

τον Locator του x_b, έστω loc', στην καθολική κατάσταση $C_{Rb'}$, κατά την εκτέλεση της συνάρτησης GetCurrentData για το x_b, έστω $GCD(j,n_j)$ η συνάρτηση αυτή. Διακρίνω περιπτώσεις ανάλογα με το εάν η $Cx_b(i,n_i)$ προηγείται ή έπεται της $C_{Rb'}$.

2.β.i) Έστω ότι η $Cx_b(i,n_i)$ έπεται της $C_{Rb'}$. Στην περίπτωση αυτή, η $T(i,n_i)$ στην $Cx_b(i,n_i)$ θα αποκτήσει την ιδιοκτησία που αντιστοιχεί στο x_b, κάτι που σημαίνει ότι θα ορίσει έναν νέο Locator για το x_b, διαφορετικό του loc'. Έτσι, η εκτέλεση της $CSx_b(j,n_j)$ μετά την $Cx_b(i,n_i)$, θα αποτύχει διότι ο Locator του x_b δεν θα είναι ο loc'[4.4]. Άτοπο.

2.β.ii) Επομένως πρέπει η $Cx_b(i,n_i)$ να προηγείται της $C_{Rb'}$. Σημειώνεται ότι εάν ορίζεται η $Cc(i,n_i)$, επειδή μπορεί να οριστεί μόνο από την $T(i,n_i)$ δε μπορεί να προηγείται της Cv. Κατά την εκτέλεσή της η V ελέγχει στην γραμμή 65 εάν έχει οριστεί η $Ca(i,n_i)$ και αν αυτό ισχύει επιστρέφει FALSE. Επειδή, από υπόθεση η V επιστρέφει TRUE, προκύπτει ότι εάν ορίζεται η $Ca(i,n_i)$ δεν μπορεί να προηγείται της Cv. Έτσι, στην περίπτωση αυτή, επειδή η $C_{Rb'}$ έπεται της $Cx_b(i,n_i)$ και προηγείται της Cv, δηλαδή προηγείται της $Cc(i,n_i)$ ή της $Ca(i,n_i)$ (ανάλογα με το ποια ορίζεται) προκύπτει ότι στην $C_{Rb'}$ η $T(i,n_i)$ κατέχει την προσωρινή ιδιοκτησία της x_b, δηλαδή η $C_{Rb'}$ εκτελείται εντός του διαστήματος $\alpha_{xb}(i,n_i)$. Επειδή η $T(j,n_j)$ ολοκληρώνεται επιτυχώς, από το Λήμμα 4.9 προκύπτει ότι η $GCD(j,n_j)$, θα επιστρέψει ως Locator της x_b αυτόν που αρχικοποιήθηκε από την $T(i,n_i)$, δηλαδή τον loc. Από το Λήμμα 4.7, προκύπτει ότι πριν την επιστροφή της $GCD(j,n_j)$ πρέπει να έχει οριστεί μία εκ των $Cc(i,n_i)$ και $Ca(i,n_i)$. Σημειώνεται ότι επειδή η $Cx_b(j,n_j)$ προηγείται της Cv και η $GCD(j,n_j)$ πρέπει να ολοκληρωθεί πριν την $Cx_b(j,n_j)$, προκύπτει ότι μία εκ των $Cc(i,n_i)$ και $Ca(i,n_i)$ πρέπει να προηγείται της Cv, το οποίο είναι άτοπο. ■

Για κάθε καθολική κατάσταση C στο $E(T(i,n_i))$ συμβολίζουμε με $W_C = \{x_1, x_2, \ldots, x_k\}$ το σύνολο των διαμοιραζόμενων αντικειμένων που η $T(i,n_i)$ έχει προσπελάσει για ανάγνωση μέσω της λειτουργίας AccessForUpdateTmVar και έχει λάβει απάντηση με

[4.4] Σημειώνεται ότι δε θα μπορούσε να προκύψει το πρόβλημα ABA διότι, όπως έχει αναφερθεί, ο αλγόριθμος DSTM εκτελείται σε κλειστό μοντέλο μνήμης και η μεταβλητή loc είναι μια από τις ενεργές τοπικές μεταβλητές της p_i.

τιμή TRUE και τον δείκτη στα επιθυμητά δεδομένα από τη λειτουργία αυτή, μέχρι τη C. Συμβολίζουμε με $D(W_C) = \{d_1, d_2, \ldots, d_k\}$ το σύνολο των τιμών των δεικτών στα δεδομένα των t-μεταβλητών του W_C. Λέμε ότι τα στοιχεία του συνόλου $D(W_C)$ είναι συνεπή στη C, εάν κάθε στοιχείο d_b, $1 \leq b \leq k'-k$, του συνόλου αυτού δείχνει στα δεδομένα που εγγράφονται από την τελευταία επιτυχημένη δοσοληψία που σειριοποιείται πριν τη C και ενημερώνει το αντίστοιχο x_b.

Λήμμα 4.14: Έστω μια οποιαδήποτε δοσοληψία $T(i,n_i)$ και C μια οποιαδήποτε καθολική κατάσταση του $E(T(i,n_i))$, η οποία προηγείται της $Cc(i,n_i)$ ή της $Ca(i,n_i)$ (ανάλογα με το ποια ορίζεται). Τα στοιχεία του συνόλου $D(W_C)$ είναι συνεπή στη C.

Απόδειξη: Έστω $D(W_C) = \{d_1, d_2, \ldots, d_k\}$. Θα δειχθεί ότι για κάθε b, $1 \leq b \leq k$, το d_b εγγράφεται από την τελευταία επιτυχημένη δοσοληψία που σειριοποιείται πριν την C και ενημερώνει το x_b. Σημειώνεται ότι επειδή το x_b περιέχεται στο W_C σημαίνει ότι η $T(i,n_i)$ έχει αποκτήσει την προσωρινή ιδιοκτησία του x_b, έστω στην καθολική κατάσταση $Cx_b(i,n_i)$, πριν τη C, εκτελώντας επιτυχώς την αντίστοιχη εντολή CAS της γραμμής 30 για το x_b, έστω $CSx_b(i,n_i)$ αυτή η εντολή. Επίσης επειδή τα δεδομένα του x_b είναι d_b σημαίνει ότι η δοσοληψία $T(i,n_i)$ πριν να αποκτήσει την προσωρινή ιδιοκτησία της x, διάβασε την τιμή d_b μέσω ενός Locator loc που διάβασε, έστω στην καθολική κατάσταση C_{Rb}, κατά την εκτέλεση της συνάρτησης GetCurrentData της γραμμής 20 για το x_b, έστω GCD η κλήση της συνάρτησης αυτής.

Έστω, δια της μεθόδου της εις άτοπο απαγωγής, ότι υπάρχει κάποιο διαμοιραζόμενο αντικείμενο $x_b \in W_C$ τέτοιο ώστε η τελευταία επιτυχημένη δοσοληψία, έστω $T(j,n_j)$, που ενημερώνει το x_b και σειριοποιείται πριν την C γράφει την τιμή $d_b' \neq d_b$ στο x_b. Σημειώνεται ότι η $T(j,n_j)$ σειριοποιείται στη $Cvl(j,n_j)$, η οποία από υπόθεση προηγείται της C. Έστω ότι η $T(j,n_j)$ αποκτά την ιδιοκτησία στο x_b στην καθολική κατάσταση $Cx_b(j,n_j)$, η οποία προηγείται της $Cvl(j,n_j)$ και άρα προηγείται της C. Επειδή η C, προηγείται της $Cc(i,n_i)$ ή της $Ca(i,n_i)$ (ανάλογα με το ποια ορίζεται), από το Λήμμα 4.5 προκύπτει ότι η $Cx_b(j,n_j)$ δε μπορεί να έπεται της $Cx_b(i,n_i)$. Επομένως η $Cx_b(j,n_j)$ πρέπει να προηγείται της $Cx_b(i,n_i)$. Σημειώνεται ότι η $Cx_b(i,n_i)$ ορίζεται μετά την επιτυχή εκτέλεση της $CSx_b(i,n_i)$. Επομένως η $Cx_b(j,n_j)$ προηγείται της $CSx_b(i,n_i)$.

Διακρίνω περιπτώσεις ανάλογα με το εάν η $Cx_b(j,n_j)$ προηγείται ή έπεται της C_{Rb}. Έστω ότι η $Cx_b(j,n_j)$ έπεται της C_{Rb}. Στην περίπτωση αυτή, ο loc που διαβάζεται στη C_{Rb} από την $T(i,n_i)$ και περιγράφει το x_b δεν έχει δημιουργηθεί από την $T(j,n_j)$. Η $T(j,n_j)$ στην $Cx_b(j,n_j)$ θα αποκτήσει την ιδιοκτησία που αντιστοιχεί στο x_b, κάτι που σημαίνει ότι θα ορίσει έναν νέο Locator για το x_b, διαφορετικό του loc. Έτσι, η εκτέλεση της $CSx_b(i,n_i)$ μετά την $Cx_b(j,n_j)$, θα αποτύχει διότι ο Locator του x_b δεν θα είναι ο loc[4.5]. Άτοπο. Επομένως η $Cx_b(j,n_j)$ πρέπει να προηγείται της C_{Rb}.

Διακρίνω περιπτώσεις ανάλογα με το εάν η $Cc(j,n_j)$ προηγείται ή έπεται της C_{Rb}. Έστω ότι η $Cc(j,n_j)$ έπεται της C_{Rb}. Επειδή, η C_{Rb} έπεται της $Cx_b(j,n_j)$ και προηγείται της $Cc(j,n_j)$ προκύπτει ότι στην C_{Rb} η $T(j,n_j)$ κατέχει την ιδιοκτησία που αντιστοιχεί στο x_b, δηλαδή η C_{Rb} εκτελείται εντός του διαστήματος $\alpha_{xb}(j,n_j)$. Επειδή η $T(j,n_j)$ ολοκληρώνεται επιτυχώς, από το Λήμμα 4.9 προκύπτει ότι η GCD, θα επιστρέψει την τιμή του δείκτη loc.newData όπως αυτός αρχικοποιήθηκε από την $T(j,n_j)$, δηλαδή την τιμή d_b' η οποία είναι διαφορετική της d_b, το οποίο είναι άτοπο. Επομένως, η $Cc(j,n_j)$ πρέπει να προηγείται της C_{Rb}.

Στην περίπτωση αυτή, η $T(i,n_i)$ διαβάζει στην C_{Rb} την τιμή που γράφει η $T(j,n_j)$ για το x_b, δηλαδή την d_b'. Για να μπορεί η $T(i,n_i)$ να διαβάσει την τιμή d_b για το x_b στην C_{Rb} πρέπει να υπάρχει κάποια άλλη επιτυχημένη δοσοληψία $T(k,n_k)$, $k \neq i \neq j$, η οποία να αποκτά την ιδιοκτησία στο x_b και ταυτόχρονα να γράφει την τιμή d_b, έστω στην καθολική κατάσταση $Cx_b(k,n_k)$, μετά την $Cc(j,n_j)$, και πριν την C_{Rb}.

Η $Cc(k,n_k)$ μπορεί να προηγείται ή να έπεται της C_{Rb}. Εάν η $Cc(k,n_k)$ προηγείται της C_{Rb}, η $Cvl(k,n_k)$ θα προηγείται της C. Εάν η $Cc(k,n_k)$ έπεται της C_{Rb}, από το Λήμμα 4.9 προκύπτει ότι η GCD, θα επιστρέψει την τιμή d_b. Σημειώνεται ότι από τον ορισμό της $Cx_b(i,n_i)$ προκύπτει ότι πριν την $Cx_b(i,n_i)$ έχει ολοκληρωθεί η εκτέλεση της GCD. Από το Λήμμα 4.7 προκύπτει ότι η $T(k,n_k)$ έχει ολοκληρωθεί πριν το τέλος της GCD, δηλαδή πριν την $Cx_b(i,n_i)$. Επειδή η $T(k,n_k)$ ολοκληρώνεται ως επιτυχημένη, προκύπτει ότι η $Cc(k,n_k)$ προηγείται της $Cx_b(i,n_i)$, και άρα προηγείται

[4.5] *Σημειώνεται ότι δε θα μπορούσε να προκύψει το πρόβλημα ABA διότι, όπως έχει αναφερθεί, ο αλγόριθμος DSTM εκτελείται σε κλειστό μοντέλο μνήμης και η μεταβλητή loc είναι μια από τις ενεργές τοπικές μεταβλητές της p.*

της C. Άρα η $Cvl(k,n_k)$, προηγείται της C. Επομένως σε κάθε περίπτωση η $Cvl(k,n_k)$ προηγείται της C. Άρα, η $T(j,n_j)$ δεν είναι η τελευταία δοσοληψία που σειριοποιείται πριν την C, το οποίο είναι άτοπο. ■

Για κάθε καθολική κατάσταση C στο $E(T(i,n_i))$ συμβολίζουμε με $RW_C = \{x_1, x_2, \ldots, x_{k'}\}$ το σύνολο των t-μεταβλητών που η $T(i,n_i)$ έχει προσπελάσει για ανάγνωση, μέσω μιας εκ των λειτουργιών ReadTmVar και AccessForUdateTmVar και έχει λάβει απάντηση με τιμή TRUE και τα επιθυμητά δεδομένα ή δείκτη στα επιθυμητά δεδομένα, αντίστοιχα, μέχρι την C. Σημειώνεται ότι ισχύει $RW_C = R_C \cup W_C$. Συμβολίζουμε με $D(RW_C) = \{d_1, d_2, \ldots, d_{k'}\}$ το σύνολο των τιμών των δεικτών στα δεδομένα των αντικειμένων του RW_C. Σημειώνεται ότι ισχύει $D(RW_C) = D(R_C) \cup D(W_C)$. Επίσης, ισχύει $R_C = O_C \cup DIF_C$ και $D(R_C) = D(O_C) \cup D(DIF_C)$.

Έστω μια οποιαδήποτε εκτέλεση της συνάρτησης Validate από την $T(i,n_i)$ στην α, η οποία επιστρέφει TRUE. Έστω V η συγκεκριμένη Validate και έστω C η τελευταία καθολική κατάσταση που προηγείται της κλήσης της V. Σημειώνεται ότι η $Cc(i,n_i)$ ορίζεται μόνο από την $T(i,n_i)$ και επομένως δε μπορεί να προηγείται της C. Επίσης η $Ca(i,n_i)$ δε μπορεί να προηγείται της C, διότι εάν αυτό συμβαίνει ο έλεγχος της γραμμής 65 της V θα αποτιμηθεί ως αληθής και η V θα επιστρέψει την τιμή FALSE, αντί για την τιμή TRUE. Έτσι, με βάση το Πόρισμα 4.12, το Λήμμα 4.13 και το Λήμμα 4.14 προκύπτει άμεσα το παρακάτω Πόρισμα.

Πόρισμα 4.15: *Έστω μια οποιαδήποτε εκτέλεση της συνάρτησης Validate από την $T(i,n_i)$ στην α, η οποία επιστρέφει TRUE. Έστω V η συγκεκριμένη Validate και έστω C η τελευταία καθολική κατάσταση που προηγείται της κλήσης της V. Τα στοιχεία του συνόλου $D(RW_C)$ είναι συνεπή στη C.*

Από τον κώδικα της CommitTransaction (γραμμές 82 και 83) προκύπτει ότι για την $T(i,n_i)$ η $Cc(i,n_i)$ μπορεί να οριστεί μόνο εάν η συνάρτηση Validation που ξεκίνησε να εκτελείται στην $Cvl(i,n_i)$ επιστρέψει TRUE. Επομένως, από το Πόρισμα 4.15 προκύπτει ότι τα στοιχεία του συνόλου $D(RW_{Cvl(i,ni)})$ είναι συνεπή στη $Cvl(i,n_i)$. Επίσης η $Cvl(i,n_i)$ προηγείται της $Cc(i,n_i)$, έτσι από το Λήμμα 4.4 προκύπτει ότι η $Cvl(i,n_i)$ είναι καλά ορισμένη. Έτσι προκύπτει το παρακάτω Θεώρημα.

Θεώρημα 4.16: Ο αλγόριθμος DSTM είναι σειριοποιήσιμος.

4.2.2. Ελευθερία Ανταγωνισμού

Στην παρούσα Ενότητα αποδεικνύεται ότι ο αλγόριθμος DSTM ικανοποιεί την ιδιότητα τερματισμού της ελευθερίας ανταγωνισμού.

Έστω ότι μια δοσοληψία $T(i,n_i)$ περιέχεται σε μια εκτέλεση α του αλγορίθμου DSTM. Σημειώνεται ότι το σύνολο $D(RW_C)$ περιέχει τα δεδομένα των διαμοιραζόμενων αντικειμένων που ο χρήστης χρησιμοποιεί στον κώδικά του. Ο αλγόριθμος DSTM ορίζει ότι ο χρήστης που εκτελεί τη δοσοληψία $T(i,n_i)$ είναι υποχρεωμένος την πρώτη φορά που προσπελάζει κάποια t-μεταβλητή x να την προσπελάσει μέσω της εκτέλεσης μιας εκ των λειτουργιών ReadTmVar και AccessForUdateTmVar, έστω Op η κλήση μιας εκ των λειτουργιών αυτών. Ο χρήστης χρησιμοποιεί τα δεδομένα της x μόνο εάν η Op επιστρέψει TRUE, διότι σε αντίθετη περίπτωση είναι υποχρεωμένος να ολοκληρώσει την $T(i,n_i)$ καλώντας μια εκ των CommitTransaction και AbortTransaction. Εάν η Op επιστρέψει TRUE σημαίνει ότι η συνάρτηση Validation που εκτελέστηκε στη γραμμή 42 επέστρεψε TRUE. Έστω C η τελευταία καθολική κατάσταση που προηγείται της εκκίνησης αυτής της συνάρτησης Validation. Από το Πόρισμα 4.15 προκύπτει ότι τα στοιχεία του συνόλου $D(RW_C)$ είναι συνεπή στη C. Επομένως, τα δεδομένα που ο χρήστης χρησιμοποιεί είναι συνεπή. Έτσι προκύπτει το παρακάτω Πόρισμα.

Πόρισμα 4.17: Ο κώδικας του χρήστη δε μπορεί να εκτελεί επ' άπειρο κάποιο βρόχο λόγω ασυνέπειας των δεδομένων που χρησιμοποιεί.

Θεώρημα 4.18: Ο αλγόριθμος DSTM ικανοποιεί την ιδιότητα τερματισμού της ελευθερίας ανταγωνισμού.

Απόδειξη: Έστω μια οποιαδήποτε πεπερασμένη εκτέλεση α του αλγορίθμου DSTM και έστω μια οποιαδήποτε δοσοληψία $T(i,n_i)$ που είναι ενεργή στην τελευταία καθολική κατάσταση C της α. Θα δείξω ότι αν η $T(i,n_i)$ συνεχίσει να εκτελείται χωρίς

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ανταγωνισμό ξεκινώντας από την C, τότε η $T(i,n_i)$ θα τερματίσει είτε ως επιτυχημένη είτε ως μη-επιτυχημένη.

Έστω, δια της μεθόδου της εις άτοπο απαγωγής, ότι η $T(i,n_i)$ δεν τερματίζει. Από το Πόρισμα 4.17 προκύπτει ότι ο κώδικας του χρήστη δε μπορεί ευθύνεται για τον μη τερματισμό της $T(i,n_i)$. Από τον κώδικα του DSTM προκύπτει ότι ο μόνος τρόπος για να μην τερματίσει η $T(i,n_i)$ είναι να εκτελεί συνεχώς το βρόχο των γραμμών 19 – 32, της συνάρτησης OpenTmVar, η οποία έχει εκτελεστεί για κάποια t-μεταβλητή x. Για να μην μπορεί να αποχωρήσει η $T(i,n_i)$ από το συγκεκριμένο βρόχο πρέπει να εκτελεί συνεχώς τη γραμμή 34. Η $T(i,n_i)$ εκτελεί τη γραμμή 34 μόνο εάν αποτύχει η εκτέλεση της εντολής CAS της γραμμής 30. Για να αποτύχει η εκτέλεση αυτής της εντολής CAS, πρέπει κάποια δοσοληψία διαφορετική της $T(i,n_i)$ να άλλαξε την τιμή του Locator της x. Άτοπο, διότι υποθέσαμε ότι η $T(i,n_i)$ δε συναντά ανταγωνισμό. ■

4.3. Πολυπλοκότητα αλγορίθμου DSTM

Στην παρούσα Ενότητα συζητείται η πολυπλοκότητα του αλγορίθμου DSTM. Ο DSTM απαιτεί για την αναπαράσταση κάθε t-μεταβλητής έναν καταχωρητή CAS πεπερασμένου μεγέθους και 3 καταχωρητές ανάγνωσης εγγραφής πεπερασμένου μεγέθους. Για την περιγραφή της δομής κάθε δοσοληψίας, απαιτεί έναν καταχωρητή CAS πεπερασμένου μεγέθους και 2 καταχωρητές ανάγνωσης εγγραφής για την περιγραφή κάθε στοιχείου της readList.

Κάθε επιτυχημένη δοσοληψία στον DSTM που προσπελάσει R t-μεταβλητές για ανάγνωση και W t-μεταβλητές για ενημέρωση ελλείψει ανταγωνισμού, απαιτεί τουλάχιστον W εντολές CAS για την απόκτηση των προσωρινών ιδιοκτησιών των t-μεταβλητών και το κόστος για τη δημιουργία των W αντιγράφων, μία εντολή CAS για την ενημέρωση των δεδομένων των t-μεταβλητών και την κατάργηση των προσωρινών ιδιοκτησιών των t-μεταβλητών και $O((R+W)^2)$ κόστος για την εκτέλεση του μηχανισμού ελέγχου συνέπειας.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ΚΕΦΑΛΑΙΟ 5. ΑΛΓΟΡΙΘΜΟΣ SSTM

5.1 Περιγραφή Αλγορίθμου SSTM
5.2 Απόδειξη Ορθότητας Αλγορίθμου SSTM
5.3 Πολυπλοκότητα

Στο κεφάλαιο αυτό περιγράφεται ο αλγόριθμος SSTM, ο οποίος παρουσιάστηκε στην εργασία [19] των N.Shavit και D.Touitou που δημοσιεύθηκε στο συνέδριο Principles Of Distributed Computing (PODC) το 1995. Στην Ενότητα 5.1 περιγράφεται ο κώδικας του αλγορίθμου, στην Ενότητα 5.2 παρουσιάζεται η απόδειξή του και στην Ενότητα 5.3 παρουσιάζεται η πολυπλοκότητά του.

5.1. Περιγραφή Αλγορίθμου SSTM

Η πρώτη εργασία σε μνήμη STM παρουσιάστηκε το 1995 από τους N.Shavit και D.Touitou [19], οι οποίοι είναι αυτοί που εισαγάγουν την έννοια της Software Transactional μνήμης. Στην εργασία αυτή παρουσιάζεται ένας αλγόριθμος που υλοποιεί στατική μνήμη STM και υποστηρίζει στατικές δοσοληψίες. Για το λόγο αυτό ο αλγόριθμος ονομάζεται *Static STM* ή *SSTM*. Ο SSTM ικανοποιεί την ιδιότητα τερματισμού ελευθερία κλειδωμάτων. Για την επίτευξη της ιδιότητας αυτής, ο SSTM χρησιμοποιεί έναν *μηχανισμό μη-επαναλαμβανόμενης βοήθειας* που θα περιγραφεί στη συνέχεια. Ακόμη ο SSTM έχει σχεδιαστεί να λειτουργεί σε ανοιχτό μοντέλο μνήμης.

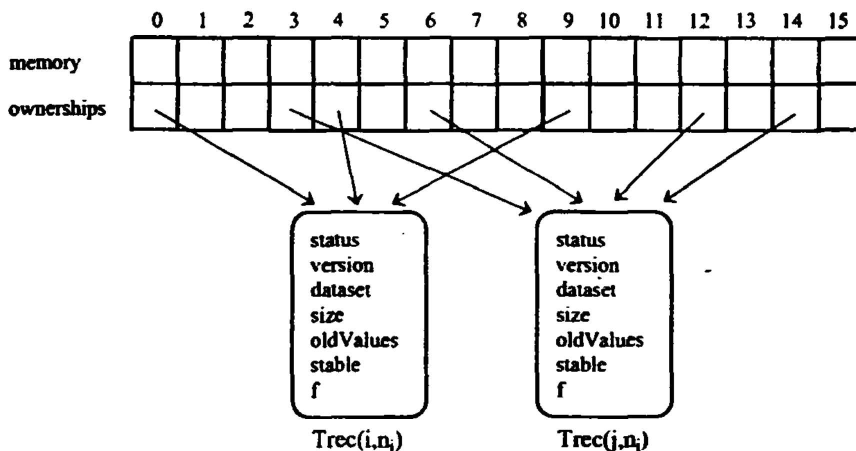
Ο αλγόριθμος SSTM υλοποιεί στατική μνήμη STM, δηλαδή θεωρεί ότι οι t-μεταβλητές που προσπελάζονται από όλες τις δοσοληψίες είναι συνολικά M στο πλήθος. Επίσης θεωρεί ότι οι t-μεταβλητές είναι συνεχόμενες στη μνήμη και

περιγράφονται από έναν πίνακα *memory* μεγέθους M. Για την επίτευξη της ιδιότητας της σειριοποιησιμότητας, ο SSTM χρησιμοποιεί τη μέθοδο απόκτησης προσωρινών ιδιοκτησιών επί των t-μεταβλητών. Οι προσωρινές ιδιοκτησίες περιγράφονται από τον πίνακα *ownerships* μεγέθους M, όπου η θέση i, $0 \leq i \leq M-1$, του πίνακα ownerhips περιγράφει την προσωρινή ιδιοκτησία της t-μεταβλητής που περιγράφεται από τη θέση i του πίνακα memory. Επομένως, ο SSTM χρησιμοποιεί ανα t-μεταβλητή ανάθεση ιδιοκτησιών. Η ιδιοκτησία που αντιστοιχίζεται σε κάποια t-μεταβλητή υλοποιείται με έναν ατομικό καταχωρητή LL/SC, ο οποίος κατά την εκκίνηση του συστήματος έχει την τιμή null, που σημαίνει ότι καμία δοσοληψία δεν κατέχει την αντίστοιχη ιδιοκτησία. Σημειώνεται ότι ο SSTM χρησιμοποιεί ορατές αναγνώσεις t-μεταβλητών, όπως θα περιγραφεί στη συνέχεια, χωρίς να κάνει διάκριση μεταξύ των ιδιοκτησιών ανάγνωσης και ενημέρωσης.

Για την απόκτηση της προσωρινής ιδιοκτησίας μιας t-μεταβλητής x από μια δοσοληψία $T(i,n_i)$, αρκεί η δοσοληψία να καταφέρει να γράψει στην ιδιοκτησία της x την τιμή ενός δείκτη προς τη δομή της $Trec(i,n_i)$, μέσω της κατάλληλης εντολής LL/SC και ενόσω καμία δοσοληψία δεν κατέχει την αντίστοιχη ιδιοκτησία. Επομένως ο SSTM χρησιμοποιεί επώνυμη πληροφορία ιδιοκτησίας. Επίσης, για την κατάργηση της ιδιοκτησίας της x αρκεί η $T(i,n_i)$ να γράψει την ειδική τιμή NOBODY στην αντίστοιχη θέση στην αντίστοιχη θέση του πίνακα ownerships εκτελώντας την κατάλληλη εντολή LL/SC. Η μορφή του συστήματος SSTM παρουσιάζεται σχηματικά στο Σχήμα 5.1. Στο συγκεκριμένο σχήμα οι t-μεταβλητές στις θέσεις 0,4,9 του πίνακα memory κατέχονται από τη δοσοληψία $T(i,n_i)$ και οι θέσεις 3,6,12,14 από την $T(j,n_j)$.

Για κάθε δοσοληψία $T(i,n_i)$, ο αλγόριθμος SSTM διατηρεί τις ακόλουθες πληροφορίες στη δομή $Trec(i,n_i)$: i) έναν καταχωρητή ανάγνωσης-εγγραφής *status* που περιγράφει την κατάσταση της δοσοληψίας, ii) έναν καταχωρητή ανάγνωσης-εγγραφής *version* που περιγράφει τον αύξοντα αριθμό της τρέχουσας δοσοληψίας που εκκίνησε η p_i (δηλαδή για την $T(i,n_i)$ θα έχει την τιμή n_i), iii) έναν πίνακα *dataset* μεγέθους M που περιγράφει τις θέσεις (στον πίνακα memory) των t-μεταβλητών της STM που η $T(i,n_i)$ θέλει να προσπελάσει, iv) έναν καταχωρητή ανάγνωσης-εγγραφής *size* που περιγράφει το πλήθος των θέσεων του πίνακα dataset που περιγράφουν τις t-

μεταβλητές που η $T(i,n_i)$ θέλει να προσπελάσει, v) έναν πίνακα *oldValues* μεγέθους size που θα περιέχει τις παλιές τιμές των t-μεταβλητών που η $T(i,n_i)$ θα τροποποιήσει εάν ολοκληρωθεί ως επιτυχημένη, vi) έναν καταχωρητή ανάγνωσης-εγγραφής *stable* που περιγράφει εάν η $Trec(i,n_i)$ διατηρεί την τρέχουσα χρονική στιγμή συνεπή δεδομένα, και vii) έναν καταχωρητή ανάγνωσης-εγγραφής *func* που διατηρεί έναν δείκτη σε μια συνάρτηση f, η χρησιμότητα της οποίας θα εξηγηθεί στη συνέχεια. Κατά την παρουσίαση του κώδικα των λειτουργιών του SSTM, υποθέτουμε ότι η δομή $Trec(i,n_i)$ είναι τύπου Transaction.



Σχήμα 5.1: Σχηματική αναπαράσταση συστήματος SSTM με M=6 και τη δοσοληψία $T(i,n_i)$ να κατέχει τις θέσεις 0,4,9 του πίνακα memory και την $T(j,n_j)$ τις 3,6,12,14.

Το status μπορεί να παίρνει τιμές από το σύνολο {ACTIVE, COMMITED, ABORTED}, όπου κάθε μία από τις τιμές έχει προφανή σημασία για την $T(i,n_i)$. Στον SSTM, η δομή δεδομένων $Trec(i,n_i)$ είναι διαμοιραζόμενη και κοινή για όλες τις δοσοληψίες που εκκινεί η διεργασία i. Έτσι, αν η p_i εκκίνησε k δοσοληψίες οι ίδιοι καταχωρητές που αποτελούν την $Trec(i,n_i)$ περιγράφουν όλες τις δοσοληψίες $T(i,n_j)$, όπου $1 \leq n_j \leq k$. Η τρέχουσα δοσοληψία που εκτελεί κάθε διεργασία p_i καθορίζεται από τη χρονοσφραγίδα που περιέχεται στον καταχωρητή version. Στη συνέχεια θα αναφερόμαστε στη δομή αυτή ως $Trec_i$.

Ο αλγόριθμος SSTM υποστηρίζει στατικές δοσοληψίες, το οποίο σημαίνει ότι το σύνολο των t-μεταβλητών που θα προσπελάσει μια δοσοληψία για ανάγνωση ή ενημέρωση, είναι γνωστό εκ των προτέρων. Ονομάζουμε το σύνολο αυτό *Dataset* (όπως θα παρουσιαστεί στη συνέχεια, τα στοιχεία του συνόλου αυτού διατηρούνται

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

στο σύνολο dataset που περιέχεται στη δομή μιας δοσοληψίας). Ο περιορισμός σε στατικές δοσοληψίες του αλγορίθμου SSTM αποτελεί και το βασικότερο μειονέκτημά του διότι περιορίζει αρκετά τη χρηστικότητά του. Ακόμη, η ιδιότητά του αυτή καθιστά τις περισσότερες από τις λειτουργίες που παρέχει το αντικείμενο STM μη απαραίτητες. Επειδή το σύνολο Dataset είναι γνωστό εκ των προτέρων, ο χρήστης που χρησιμοποιεί τον SSTM περνά ως όρισμα στη λειτουργία BeginTransaction το σύνολο αυτό. Επιπρόσθετα ο αλγόριθμος SSTM υποθέτει ότι ο χρήστης περνά ως όρισμα σε μια δοσοληψία τη συνάρτηση που θέλει να εφαρμόσει στα δεδομένα των προκαθορισμένων t-μεταβλητών. Ονομάζουμε f τη συνάρτηση αυτή. Επειδή ο αλγόριθμος SSTM υποστηρίζει στατικές δοσοληψίες, προκύπτει ότι δεν είναι απαραίτητες οι λειτουργίες ανάγνωσης και εγγραφής του ατομικού αντικειμένου STM. Ακόμη, επειδή ο SSTM υλοποιεί στατική μνήμη STM, η λειτουργία CreateNewTmVar του αντικειμένου STM δεν είναι απαραίτητη στον SSTM, διότι η δημιουργία και η κατάλληλη αρχικοποίηση μιας t-μεταβλητής μπορεί να πραγματοποιηθεί καθολικά για όλες τις M t-μεταβλητές κατά την εκκίνηση του συστήματος STM. Έτσι, οι μόνες απαραίτητες λειτουργίες του αντικειμένου STM για τον αλγόριθμο SSTM είναι οι λειτουργίες εκκίνησης (BeginTransaction) και τερματισμού μιας δοσοληψίας (CommitTransaction και AbortTransaction).

Επιλέγουμε να περιγράψουμε τον αλγόριθμο SSTM βάσει των λειτουργιών του βασικού αντικειμένου STM. Στο σημείο αυτό θα γίνει μια συνοπτική περιγραφή των λειτουργιών του αλγορίθμου SSTM, ενώ περισσότερες λεπτομέρειες θα δοθούν στη συνέχεια. Ο αλγόριθμος SSTM απαιτεί από τον χρήστη να εκκινήσει μία δοσοληψία $T(i,n_i)$ εκτελώντας τη λειτουργία BeginTransaction, περνώντας ως όρισμα σε αυτή το προκαθορισμένο σύνολο των t-μεταβλητών (Dataset) το οποίο θα προσπελάσει η δοσοληψία και τη συνάρτηση f την οποία ο αλγόριθμος SSTM θα μπορεί να εκτελέσει στα δεδομένα των t-μεταβλητών του Dataset, ώστε να λάβει τα νέα τους δεδομένα. Ο χρήστης δίνει τις πληροφορίες αυτές κατά την εκκίνηση της δοσοληψίας $T(i,n_i)$, ως παραμέτρους στη λειτουργία BeginTransaction, η οποία παρουσιάζεται στο Σχήμα 5.2. Όπως παρουσιάζεται στον κώδικα αυτό, αρχικοποιείται κατάλληλα η δομή $Trec_i$ (γραμμή 2) με την εκτέλεση της συνάρτησης Initialize, στη συνέχεια δηλώνεται ότι τα δεδομένα της είναι συνεπή (γραμμή 3) και επιστρέφεται η δομή αυτή στο χρήστη (γραμμή 4). Σημειώνεται ότι η p_i διατηρεί την ίδια δομή $Trec_i$ για

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

όλες τις δοσοληψίες που εκκινεί, για το λόγο αυτό δε δεσμεύεται εκ νέου μνήμη για την $Trec_i$ κατά την εκτέλεση της λειτουργίας BeginTransaction.

```
1    Transaction *BeginTransaction(DATA Dataset[], int size, void *f)
2        Initialize (Trec_i, Dataset, size, f);
3        Trec_i→stable = true;
4        return (Trec_i);
```

Σχήμα 5.2: Κώδικας της λειτουργίας BeginTransaction του SSTM για την p_i.

Ο κώδικας της συνάρτησης Initialize παρουσιάζεται στο Σχήμα 5.3. Αρχικά δηλώνεται ότι η δοσοληψία είναι εκκρεμής θέτοντας το status της σε ACTIVE (γραμμή 6), έπειτα αρχικοποιούνται οι καταχωρητές size, dataset και func βάσει των ορισμάτων που ο χρήστης έδωσε κατά την BeginTransaction (γραμμές 7 έως 10), και τέλος αρχικοποιούνται τα στοιχεία του πίνακα oldValues σε null (γραμμές 11 και 12). Παρατηρούμε ότι το σύνολο Dataset και η συνάρτηση f, διατηρούνται στη δομή της $T(i,n_i)$, δηλαδή στην $Trec(i,n_i)$, στις θέσεις dataset και func, αντίστοιχα. Σημειώνεται ότι στο status της $Trec_i$, διατηρείται η τιμή του status και μία ακόμα αριθμητική τιμή, η χρησιμότητα της οποίας εξηγείται στη συνέχεια. Η αριθμητική αυτή τιμή αρχικοποιείται σε μηδέν.

```
5    Initialize (Transaction *Trec_i, DATA Dataset[], int size, void *f)
6        Trec_i→status = (ACTIVE, 0);
7        Trec_i→size = size;
8        for (j=1; j<=size)
9            Trec_i→dataset[j] = Dataset[j];
10       Trec_i→func = f;
11       for (j=1; j<=size; j++)
12           Trec_i→oldValues[j] = (null,0);
```

Σχήμα 5.3: Κώδικας της συνάρτησης Initialize του SSTM για την p_i.

Έπειτα, ο χρήστης πρέπει να εκτελέσει τη λειτουργία CommitTransaction ώστε να ολοκληρωθεί η δοσοληψία ως επιτυχημένη ή την AbortTransaction για να ολοκληρωθεί η δοσοληψία ως μη-επιτυχημένη. Σημειώνεται ότι ο χρήστης δεν χρειάζεται να εκτελέσει κάποια λειτουργία ReadTmVar ή WriteTmVar ή CreateNewTmVar, η υλοποίηση των οποίων στον SSTM είναι κενή. Η υλοποίηση της

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

λειτουργίας AbortTransaction περιέχει μόνο τη δήλωση των δεδομένων της $Trec_i$ ως μη-συνεπή, όπως παρουσιάζεται στο Σχήμα 5.4.

```
13  AbortTransaction (Transaction *Trec_i)
14      Trec_i→stable = false;
```

Σχήμα 5.4: Κώδικας της λειτουργίας AbortTransaction του SSTM για την p_i.

```
15  (Boolean, DATA []) CommitTransaction (Transaction *Trec_i)
16      PerformsTrans (Trec_i, Trec_i→version, TRUE);
17      Trec_i→stable = FALSE;
18      Trec_i→version++;
19      if (Trec_i→status == COMMITED) then
20          return (TRUE, Trec_i→oldValues);
21      else
22          return FALSE;
```

Σχήμα 5.5: Κώδικας της λειτουργίας CommitTransaction του SSTM για την p_i.

Κατά την εκτέλεση της λειτουργίας CommitTransaction για μια δοσοληψία $T(i,n_i)$ από τον χρήστη, εκτελείται ο κώδικας που παρουσιάζεται στο Σχήμα 5.5. Ο SSTM προσπαθεί να ολοκληρώσει την $T(i,n_i)$ ως επιτυχημένη, εκτελώντας τη συνάρτηση PerformTrans (γραμμή 16). Σημειώνεται ότι η συνάρτηση PerformTrans, θα ορίσει το $Trec_i$→status στην τιμή COMMITED εάν η δοσοληψία $T(i,n_i)$ ολοκληρωθεί επιτυχώς και στην τιμή ABORTED σε αντίθετη περίπτωση. Έτσι, μετά την εκτέλεσή της, δηλώνεται ότι τα δεδομένα της $Trec_i$ δεν είναι πλέον συνεπή, αυξάνεται το $Trec_i$→version κατά 1 (που σημαίνει ότι η επόμενη δοσοληψία που θα εκκινηθεί από την p_i θα είναι η $T(i,n_i+1)$) και στη συνέχεια στις γραμμές 19 έως 22, γίνεται η επιστροφή δεδομένων στο χρήστη ανάλογα με το αποτέλεσμα της εκτέλεσης της δοσοληψίας, μέσω της PerformTrans. Εάν η δοσοληψία ολοκληρώθηκε επιτυχώς, επιστρέφονται στον χρήστη η τιμή επιτυχίας (TRUE) και ο πίνακας με τις προηγούμενες τιμές των t-μεταβλητών που προσπελάστηκαν, ενώ εάν η δοσοληψία ολοκληρωθεί μη επιτυχώς επιστρέφεται η τιμή μη επιτυχίας (FALSE). Υπενθυμίζεται ότι ο αλγόριθμος SSTM υλοποιεί υποστηρίζει μόνο στατικές δοσοληψίες και ο χρήστης περνά ως όρισμα τη συνάρτηση f που θα εφαρμοστεί στα επιθυμητά δεδομένα. Επομένως ο χρήστης δε γνωρίζει ούτε τα δεδομένα των t-μεταβλητών που

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

αναγνώσθηκαν από τον SSTM, ούτε τα νέα τους δεδομένα μετά την εκτέλεση της f και την ενημέρωσή τους από τον SSTM. Για το λόγο αυτό ο SSTM ορίζει ότι εάν μια δοσοληψία ολοκληρωθεί επιτυχώς, θα επιστρέφονται στο χρήστη τα παλιά δεδομένα των t-μεταβλητών που προσπελάστηκαν, κάτι το οποίο παραβιάζει τον ορισμό της λειτουργίας CommitTransaction του βασικού αντικειμένου STM.

Ο κώδικας της συνάρτησης PerformTrans παρουσιάζεται στο Σχήμα 5.6. Η προσπάθεια ολοκλήρωσης της $T(i,n_i)$ ως επιτυχημένης μέσω της PerformTrans γίνεται σε δύο στάδια. Κατά το πρώτο στάδιο ο SSTM προσπαθεί να αποκτήσει τις προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που επιθυμεί ο χρήστης, εκτελώντας τη συνάρτηση AcquireOwnerships (γραμμή 27). Όταν η συνάρτηση AcquireOwnerships επιστρέψει είτε θα έχουν αποκτηθεί οι προσωρινές ιδιοκτησίες όλων των t-μεταβλητών ή όχι, και η AcquireOwnerships εγγυάται ότι το status της $T(i,n_i)$ θα έχει τιμή ACTIVE ή ABORTED, αντίστοιχα. Έτσι, εάν κατά το πρώτο στάδιο η $T(i,n_i)$ έχει καταφέρει να αποκτήσει τις προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που επιθυμεί ο χρήστης, τότε σημαίνει ότι η $T(i,n_i)$ έχει αποκτήσει αποκλειστική πρόσβαση σε αυτές και κατά το δεύτερο στάδιο της εκτέλεσής της χαρακτηρίζεται ως επιτυχημένη, δηλαδή η κατάστασή της ορίζεται σε COMMITED (γραμμή 31). Στη συνέχεια, αποθηκεύει τις παλιές τιμές των t-αντικειμένων μέσω της συνάρτησης AgreeOldValues (γραμμή 35) και με βάση αυτές υπολογίζει τις νέες τους τιμές καλώντας τη συνάρτηση f που όρισε ο χρήστης (γραμμή 36). Τέλος, η συνάρτηση PerformTrans αποθηκεύει τις νέες τιμές στις t-μεταβλητές με τη βοήθεια της συνάρτησης UpdateMemory (γραμμή 37), καταργεί τις προσωρινές ιδιοκτησίες που κατέχει μέσω της συνάρτησης ReleaseOwnerships (γραμμή 38) και ολοκληρώνεται.

Αντίθετα, εάν κατά το πρώτο στάδιο η $T(i,n_i)$ δεν έχει καταφέρει να αποκτήσει τις προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που επιθυμεί, σημαίνει ότι δεν έχει καταφέρει να αποκτήσει την ιδιοκτησία μιας t-μεταβλητής x επειδή τη συγκεκριμένη χρονική στιγμή την κατείχε κάποια άλλη δοσοληψία $T(j,n_j)$. Στην περίπτωση αυτή, η συνάρτηση AcquireOwnerships έχει γράψει την τιμή ABORTED στο $Trec_i$.status, μαζί με τον αριθμό της θέσης του πίνακα Ownerhips, έστω *failnum*, στην οποία διατηρείται η ιδιοκτησία της x. Στη συνέχεια η $T(i,n_i)$ καταργεί όλες τις προσωρινές

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ιδιοκτησίες που πιθανώς είχε αποκτήσει (γραμμή 40) εκτελώντας τη συνάρτηση ReleaseOwnerships, *βοηθά* την T(j,n_j) να ολοκληρωθεί εκτελώντας τη συνάρτηση PerformTrans για τη συγκεκριμένη δοσοληψία (γραμμή 48) περνώντας ως όρισμα το $Trec_j$, και μετά την επιστροφή της συγκεκριμένης PerformTrans ολοκληρώνεται.

```
23  PerformTrans (Transaction *Trec_i, int version, Boolean isInitiator)
24      int status, failadd;
25      Transaction *failtran;
26      DATA newValues[Trec_i→size];
27      AcquireOwnerships (Trec_i, version);
28      (status, failadd) = LL (Trec_i→status);
29      while (status == ACTIVE)
30          if (version ≠ Trec_i→version) then return;
31          SC (Trec_i→status, (COMMITED, 0));
32          (status, failadd) = LL (Trec_i→status);
33      (status, failadd) = LL (Trec_i→status);
34      if (status == SUCCESS) then
35          AgreeOldValues (Trec_i, version);
36          newValues = Trec_i→func (Trec_i→oldValues);
37          UpdateMemory (Trec_i, version, newValues);
38          ReleaseOwnerships (Trec_i, version);
39      else
40          ReleaseOwnerships (Trec_i, version);
41          if (isInitiator == TRUE) then
42              failtran = ownerships[failadd];
43              if (failtran == NOBODY) then
44                  return;
45              else
46                  failversion = failtran→version;
47                  if (failtran→stable == TRUE) then
48                      PerformTrans (failtran, failversion, FALSE);
```

Σχήμα 5.6: Κώδικας της συνάρτησης PerformTrans του SSTM για την p_i.

Σημειώνεται ότι η T(i,n_i) είναι σε θέση να βοηθήσει την T(j,n_j), διότι μέσω της τιμής της προσωρινής ιδιοκτησίας της x μπορεί να μάθει το $Trec_j$, το οποίο περιέχει όλες τις απαιτούμενες πληροφορίες για την T(j,n_j). Συγκεκριμένα η T(i,n_i) μαθαίνει την τιμή $Trec_j$ με την εκτέλεση της γραμμής 42, όπου το failadd είναι ο αριθμός failnum που συζητήθηκε παραπάνω. Αυτός είναι και ο *μηχανισμός βοήθειας* που προσδίδει την ιδιότητα τερματισμού ελευθερία κλειδωμάτων στον αλγόριθμο SSTM. Σημειώνεται ότι η T(i,n_i) βοηθά το πολύ μία δοσοληψία να ολοκληρωθεί και όχι επαναληπτικά όσες χρειαστούν βοήθεια. Αυτό σημαίνει ότι εάν η T(i,n_i) βοηθήσει την T(j,n_j) να

ολοκληρωθεί και η $T(j,n_j)$ αποφασίσει με τη σειρά της να βοηθήσει κάποια $T(k,n_k)$, τότε η $T(i,n_i)$ δε θα βοηθήσει και την $T(k,n_k)$ αλλά θα σταματήσει να βοηθά την $T(j,n_j)$ και θα ολοκληρωθεί ως μη-επιτυχημένη. Αυτό επιτυγχάνεται με το πέρασμα της τιμής FALSE στο όρισμα isInitiator της PerformTrans που εκτελείται στη γραμμή 48 για την $T(j,n_j)$ από την $T(i,n_i)$.

Όπως περιγράφηκε, ο χρήστης έχει ορίσει τη συνάρτηση f κατά την εκκίνηση της $T(i,n_i)$, η οποία θα εφαρμοστεί στα δεδομένα των t-μεταβλητών του Dataset αφού ο SSTM αποκτήσει τις προσωρινές ιδιοκτησίες των t-μεταβλητών αυτών. Για το λόγο αυτό λέμε ότι ο SSTM χρησιμοποιεί εκ των προτέρων απόκτηση προσωρινών ιδιοκτησιών και όχι εκ των υστέρων που ήταν το αναμενόμενο, δεδομένου ότι οι προσωρινές ιδιοκτησίες αποκτώνται κατά την εκτέλεση της λειτουργίας CommitTransaction. Ακόμη, επειδή ο SSTM αποκτά προσωρινές ιδιοκτησίες σε όλες τις t-μεταβλητές που ορίζει ο χρήστης, χωρίς να γνωρίζει ποιες από αυτές θα ενημερωθούν, λέμε ότι ο SSTM χρησιμοποιεί ορατές αναγνώσεις t-αντικειμένων. Επομένως ο SSTM δεν απαιτεί την υλοποίηση κάποιου μηχανισμού ελέγχου συνέπειας των δεδομένων των t-μεταβλητών που αναγνώσθηκαν μέσω λειτουργίας καθολικής ανάγνωσης διότι χρησιμοποιεί ορατές αναγνώσεις t-μεταβλητών και εκ των προτέρων απόκτηση προσωρινών ιδιοκτησιών.

Στη συνέχεια περιγράφεται η συνάρτηση AcquireOwnerships, ο κώδικας της οποίας παρουσιάζεται στο Σχήμα 5.7. Η απόκτηση των επιθυμητών προσωρινών ιδιοκτησιών γίνεται κατά το πρώτο στάδιο εκτέλεσης της PerformTrans μέσω της εκτέλεσης της συνάρτησης AcquireOwnerships. Διατρέχονται όλες οι επιθυμητές προσωρινές ιδιοκτησίες (51 και 53) και για κάθε μία από αυτές, έστω *own*, αρχικά ανακαλύπτεται η δοσοληψία που πιθανώς την κατέχει (γραμμή 55). Εάν η own έχει αποκτηθεί ήδη από την ίδια την $T(i,n_i)$ (γραμμή 57) τότε συνεχίζουμε με την επόμενη ιδιοκτησία. Αν δεν η own δεν έχει αποκτηθεί από καμία δοσοληψία (γραμμή 58) τότε προσπάθεια απόκτησής της από την $T(i,n_i)$ (γραμμή 60), με την εκτέλεση της κατάλληλη εντολής LL/SC. Αν η own έχει αποκτηθεί από κάποια δοσοληψία διαφορετικής της $T(i,n_i)$ (γραμμή 62) τότε η $T(i,n_i)$ χαρακτηρίζεται ως μη επιτυχημένη (γραμμή 63) και ολοκληρώνεται η εκτέλεση της AcquireOwnerships (γραμμή 64). Επομένως, εάν κάποια από τις προσωρινές ιδιοκτησίες που επιθυμεί να αποκτήσει η $T(i,n_i)$ έχει

αποκτηθεί από κάποια άλλη δοσοληψία, τότε η T(i,n$_i$) θα δεν θα καταφέρει να την αποκτήσει και θα ολοκληρώσει το πρώτο στάδιο. Σημειώνεται ότι στην περίπτωση αυτή, η AcquireOwnerships έχει γράψει την τιμή ABORTED στο Trec$_i$.status, μαζί με τον αριθμό της θέσης του πίνακα Ownerhips που η T(i,n$_i$) δεν κατέφερε να αποκτήσει. Όπως περιγράφηκε, βάσει αυτού του αριθμού η T(i,n$_i$) μπορεί στη συνέχεια να εκτελέσει το μηχανισμό βοήθειας.

```
49  AcquireOwnerships (Transaction Treci, int version)
50  int size = Treci→size;
51  for (j=1; j<=size; j++)
52      while TRUE do
53          location = Treci→dataset[j];
54          if (LL(Treci→status) ≠ ACTIVE) return;
55          owner = LL (ownerships[location]);
56          if (Treci.version ≠ version) return;
57          if (owner == Treci) exit while loop;
58          if (owner == NOBODY)
59              if (SC(Treci.status, (ACTIVE,0)))
60                  if (SC(ownerships[location], Treci))
61                      exit while loop;
62          else
63              if SC (Treci.status, (ABORTED, j))
64                  return;
```

Σχήμα 5.7: Κώδικας της λειτουργίας AcquireOwnerships του SSTM για την p$_i$.

Είναι σημαντικό ότι ο SSTM ορίζει ότι η απόκτηση των προσωρινών ιδιοκτησιών κατά το πρώτο στάδιο εκτέλεσης μιας δοσοληψίας, γίνεται με προκαθορισμένη σειρά βάση της αύξουσας διάταξης των θέσεων των t-μεταβλητών στα οποία αντιστοιχούν. Αυτό γίνεται αντιληπτό από τις γραμμές 51, 53, 55 και 60 του κώδικα. Με τον τρόπο αυτό αποφεύγεται η παρουσίαση κατάστασης καθολικής παρατεταμένης στέρησης που θα είχε ως αποτέλεσμα την καταστρατήγηση της ιδιότητας ελευθερία κλειδωμάτων του SSTM. Εάν δεν υπήρχε ο περιορισμός αυτός, θα μπορούσε να παρουσιαστεί κατάσταση παρατεταμένης στέρησης εάν κάποια δοσοληψία T(i,n$_i$) είχε αποκτήσει την προσωρινή ιδιοκτησία της t-μεταβλητής x και προσπαθούσε να αποκτήσει την προσωρινή ιδιοκτησία της t-μεταβλητής y, και κάποια άλλη δοσοληψία T(j,n$_j$) κατείχε τη y και προσπαθούσε να αποκτήσει τη x. Έτσι, η T(i,n$_i$) θα βοηθούσε την T(j,n$_j$), και η T(j,n$_j$) την T(i,n$_i$) και επειδή η μία κατέχει την

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

προσωρινή ιδιοκτησία μιας t-μεταβλητής που η άλλη επιθυμεί, με βάση τα παραπάνω, θα αποτύχουν και οι δύο. Γίνεται κατανοητό ότι εάν οι δοσοληψίες $T(i,n_i)$ και $T(j,n_j)$ ξανά εκτελούνταν από τον χρήστη και το παραπάνω σενάριο ξανά εμφανιζόταν θα καταστρατηγούνταν η ιδιότητα ελευθερία κλειδωμάτων του αλγορίθμου.

Όπως αναφέρθηκε, κατά το δεύτερο στάδιο εκτέλεσής της η $T(i,n_i)$, ανάλογα με την τιμή του $Trec_i$.status, η PerformTrans θα εκτελέσει είτε τις συναρτήσεις AgreeOldValues, f, UpdateMemory και ReleaseOwnerships στην περίπτωση που έχει τιμή COMMITED, ή μόνο την ReleaseOwnerhips αν η τιμή του είναι ABORTED. Ο κώδικας των συναρτήσεων αυτών παρουσιάζεται στη συνέχεια. Ο κώδικας της συνάρτησης ReleaseOwnerhips παρουσιάζεται στο Σχήμα 5.8. Η συνάρτηση αυτή είναι υπεύθυνη να καταργήσει τις προσωρινές ιδιοκτησίες των t-μεταβλητών που η $T(i,n_i)$ κατάφερε να αποκτήσει. Για το λόγο αυτό εξετάζει όλες τις προσωρινές ιδιοκτησίες (γραμμές 67 και 68) που επιθυμεί η $T(i,n_i)$ και καταργεί (γραμμή 71) όσες έχουν αποκτηθεί από την $T(i,n_i)$ (γραμμή 69).

```
65   ReleaseOwnerships (Transaction *Trec_i, int version)
66   int size = Trec_i→size;
67   for (j=1; j<=size; j++)
68       location = Trec_i→dataset[j];
69       if (LL(ownerships[location])== Trec_i)
70           if (Trec_i→version ≠ version) return;
71           SC (ownerships[location],NOBODY);
```

Σχήμα 5.8: Κώδικας της λειτουργίας ReleaseOwnerships του SSTM για την p_i.

```
72   AgreeOldValues (Transaction *Trec_i, int version)
73   int size = Trec_i→size;
74   for (j=1; j<=size; j++)
75       location = Trec_i→dataset[j];
76       if (LL(Trec_i→oldValues[location]) == (null,0))
77           if (Trec_i→version ≠ version) return;
78           SC (Trec_i→oldValues[location],(memory[location],1));
```

Σχήμα 5.9: Κώδικας της λειτουργίας AgreeOldValues του SSTM για την p_i.

Ο κώδικας της συνάρτησης AgreeOldValues παρουσιάζεται στο Σχήμα 5.9. Η συνάρτηση αυτή είναι υπεύθυνη να αποθηκεύσει τις παλιές τιμές των t-μεταβλητών

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

που η δοσοληψία $T(i,n_i)$ επιθυμεί στον πίνακα oldValues. Ο κώδικας της συνάρτησης UpdateMemory παρουσιάζεται στο Σχήμα 5.10. Η συνάρτηση αυτή ενημερώνει τις t-μεταβλητές που η $T(i,n_i)$ επιθυμεί βάσει των νέων τους τιμών, που υπολογίστηκαν και αποθηκεύτηκαν στον πίνακα newValues, μέσω της εκτέλεσης της συνάρτησης Trec.func (γραμμή 36). Για το λόγο αυτό διατρέχει όλες τις t-μεταβλητές (γραμμές 81 και 82) που ο χρήστης όρισε και τις ενημερώνει βάσει του πίνακα newValues (γραμμή 87).

```
79    UpdateMemory (Transaction *Trec_i, int version, DATA newValues[])
80        int size = Trec_i.size;
81        for (j=1; j<=size; j++)
82            location = Trec_i→dataset[j];
83            oldvalue = LL(memory[location]);
84            if (Trec_i→AllWritten == TRUE) return;
85            if (version ≠ Trec_i→version) return;
86            if (oldvalue ≠ newValues[j])
87                SC (memory[location], newValues[j]);
88        if (LL(Trec_i→AllWritten) == FALSE)
89            if (version ≠ rec→version) return;
90            SC (Trec_i→AllWritten, TRUE);
```

Σχήμα 5.10: Κώδικας της λειτουργίας UpdateMemory του SSTM για την p_i.

Κατά την περιγραφή του αλγορίθμου SSTM που προηγήθηκε, επικεντρωθήκαμε σκοπίμως στις βασικότερες γραμμές του κώδικά του, χάριν απλούστευσης της παρουσίασής του. Στο σημείο αυτό θα γίνει κατανοητή η χρησιμότητα των γραμμών που παραλήφθηκαν.

Ο αλγόριθμος SSTM έχει δύο χαρακτηριστικά που περιπλέκουν αρκετά την απόδειξη της ορθότητάς του. Το πρώτο προκύπτει από το γεγονός ότι εάν κάποια δοσοληψία $T(i,n_i)$ δεν καταφέρει να αποκτήσει τις προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που επιθυμεί κατά το πρώτο στάδιο (κατά την εκτέλεση της αντίστοιχης συνάρτησης PerformTrans) τότε κατά το δεύτερο στάδιο θα προσπαθήσει να βοηθήσει κάποια άλλη δοσοληψία, εκτελώντας την αντίστοιχη συνάρτηση PerformTrans. Αυτό σημαίνει ότι ο κώδικας της συνάρτησης PerformTrans και όλων των συναρτήσεων που αυτή καλεί (ReleaseOwnership, AgreeOldValues, UpdateMemory) μπορεί να εκτελείται ταυτόχρονα από πολλές διεργασίες (από τη

διεργασία που εκκίνησε τη δοσοληψία και από τις «βοηθητικές» της δοσοληψίες). Έτσι, πρέπει ο αλγόριθμος SSTM να εξασφαλίζει ότι δεν δημιουργούνται προβλήματα συγχρονισμού μεταξύ όλων αυτών των διεργασιών. Είναι σημαντικό να παρατηρηθεί ότι οι βοηθητικές δοσοληψίες μιας δοσοληψίας $T(i,n_i)$ μπορούν να εκτελούνται ανεξάρτητα από την $T(i,n_i)$, κάτι που σημαίνει ότι μπορούν να εκτελούνται και μετά τον τερματισμό της $T(i,n_i)$. Η δεύτερη δυσκολία που πρέπει να αντιμετωπίσει ο αλγόριθμος είναι η χρήση της ίδιας δομής $Trec_i$ για την περιγραφή των πληροφοριών όλων των δοσοληψιών που εκκινήθηκαν από τη διεργασία p_i. Πρέπει να διασφαλίζεται ότι οι βοηθητικές δοσοληψίες μιας δοσοληψίας $T(i,n_i)$ βοηθούν πράγματι την $T(i,n_i)$ και όχι κάποια επόμενή της $T(i,k)$, όπου $k>n_i$.

Για να γίνουν κατανοητά τα δύο παραπάνω προβλήματα και να παρουσιαστεί η χρησιμότητα των επιπλέον γραμμών του κώδικα του SSTM που δεν περιγράφηκαν, ας ασχοληθούμε με τον κώδικα της συνάρτησης ReleaseOwnerships. Με βάση τα παραπάνω, δε θα ήταν σωστό μία βοηθητική δοσοληψία της $T(i,n_i)$ να καταργήσει κάποια προσωρινή ιδιοκτησία μετά το τέλος της $T(i,n_i)$, διότι αυτό μπορεί να είχε αποκτηθεί από κάποια άλλη δοσοληψία $T(j,n_j)$, $j\neq i$, ή να είχε αποκτηθεί από κάποια δοσοληψία που έχει εκκινηθεί από την p_i μετά το τέλος της $T(i,n_i)$. Ο έλεγχος που δεν επιτρέπει την πραγματοποίηση των δύο αυτών μη επιθυμητών σεναρίων, όπως θα αποδειχθεί στην Ενότητα 5.2, εμπεριέχεται στην εκτέλεση της εντολής LL/SC των γραμμών 69 και 71, στην οποία ενδιάμεσα παρεμβάλλεται ο έλεγχος του version της δοσοληψίας της γραμμής 70.

Είναι σημαντικό ότι παρόμοιοι έλεγχοι πραγματοποιούνται σε πολλά σημεία του κώδικα και η χρησιμότητα τους είναι η εξής: i) Η ύπαρξη της εντολής LL/SC για την ενημέρωση κάποιας t-μεταβλητής x από μια δοσοληψία $T(i,n_i)$ αποσκοπεί στο να ενημερωθεί η x μία μόνο φορά από την $T(i,n_i)$, είτε από την ίδια ή από κάποια εκ των βοηθητικών της δοσοληψιών, και ii) η ύπαρξη του ελέγχου του version μεταξύ της εντολής LL και SC στον x, αποσκοπεί στην μη ενημέρωση του x μετά την ολοκλήρωση της $T(i,n_i)$, από τις βοηθητικές της δοσοληψίες.

Στην επόμενη Ενότητα, παρουσιάζεται φορμαλιστική απόδειξη της ορθότητας του αλγορίθμου SSTM. Είναι αξιοσημείωτο ότι κατά την φορμαλιστική του απόδειξη,

εμφανίστηκαν κάποια μικρά προβλήματα που ο αλγόριθμος αυτός παρουσίαζε σε κάποια σενάρια εκτελέσεων, τα οποία δεν του επέτρεπαν να είναι ορθός. Τα προβλήματα αυτά επιλύθηκαν και έτσι ο κώδικας που παρουσιάστηκε στην παρούσα Ενότητα, διαφέρει από τον κώδικα που παρουσιάζουν οι συγγραφείς του SSTM στην εργασίας τους [19].

5.2. Απόδειξη Αλγορίθμου SSTM

Υπενθυμίζεται ότι η n_i-οστή δοσοληψία που εκκινείται από μια διεργασία p_i συμβολίζεται με $T(i,n_i)$ και η p_i ονομάζεται *δημιουργός* της $T(i,n_i)$. Μια διεργασία p_j ονομάζεται *βοηθητική διεργασία* μιας δοσοληψίας $T(i,n_i)$ αν η p_j καλεί την PerformTrans με ορίσματα ($Trec(i,n_i)$, n_i, FALSE) (γραμμή 48 του κώδικα). Συμβολίζουμε με $PT_j(i,n_i)$ την κλήση αυτή. Η δοσοληψία $T(j,n_j)$ που εκτελείται από την p_j κατά την κλήση της $PT_j(i,n_i)$ ονομάζεται *βοηθητική δοσοληψία* της $T(i,n_i)$ και η $PT_j(i,n_i)$ ονομάζεται *βοηθητική PerformTrans* της $T(i,n_i)$. Οι *executing διεργασίες* μιας δοσοληψίας $T(i,n_i)$ αποτελούνται από τον δημιουργό της και τις βοηθητικές της διεργασίες. Επίσης οι *executing PerformTrans* της δοσοληψίας αποτελούνται από τις PerformTrans που εκτελούν οι executing διεργασίες της.

Υπενθυμίζεται ότι το *διάστημα εκτέλεσης* μιας δοσοληψίας $T(i,n_i)$ συμβολίζεται ως $E(T(i,n_i))$.Είναι αξιοσημείωτο ότι το $E(T(i,n_i))$ δεν ταυτίζεται με το διάστημα εκτέλεσης των βοηθητικών δοσοληψιών της $T(i,n_i)$, εφόσον οι βοηθητικές δοσοληψίες της μπορεί να τερματίσουν πριν το τέλος του $E(T(i,n_i))$ ή να συνεχίζουν να εκτελούνται μετά το τέλος του.

Υπενθυμίζεται ότι κάθε διεργασία p_i διατηρεί στη διαμοιραζόμενη δομή δεδομένων $Trec_i$ ένα συγκεκριμένο σύνολο δομών δεδομένων και μεταβλητών με τις οποίες περιγράφει τη δοσοληψία που εκτελεί. Μία από τις δομές δεδομένων που διατηρούνται στην $Trec_i$ είναι ο πίνακας $rec_i.add$ ο οποίος αποθηκεύει τις θέσεις του πίνακα ownerships που η τρέχουσα εκκρεμής δοσοληψία της p_i, έστω $T(i,n_i)$, επιθυμεί να αποκτήσει. Επίσης στην $Trec_i$ περιέχεται η μεταβλητή $rec_i.size$ που περιγράφει το πλήθος των στοιχείων του $rec_i.add$. Στην συνέχεια θα χρησιμοποιούμε τον συμβολισμό $l \in Trec_i.add$ αντί του συμβολισμού $l \in \{Trec_i.add[1], Trec_i.add[2], \ldots,$

rec_i.add[rec_i.size]}, για να υποδηλώσουμε κάποιο στοιχείο *l* του $Trec_i$.add. Ακόμη, επειδή η p_i μπορεί να είναι ο δημιουργός πολλών δοσοληψιών, στην $Trec_i$ περιέχεται η μεταβλητή version που δηλώνει την τρέχουσα εκκρεμή δοσοληψία της p_i, π.χ. για την $T(i,n_i)$ ισχύει $Trec_i$.version == n_i. Στη συνέχεια θα χρησιμοποιούμε τον συμβολισμό $Trec_i$ για να αναφερόμαστε στην δομή δεδομένων Trec της p_i και τον συμβολισμό $Trec(i,n_i)$ για να αναφερόμαστε στην δομή δεδομένων Trec της $T(i,n_i)$.

Κάθε φορά που μια executing PerformTrans μιας δοσοληψίας, έστω $T(i,n_i)$, εγγράφει σε κάποια θέση, έστω j, του πίνακα ownerships την $Trec_i$ με $Trec_i$.version==n_i εκτελώντας επιτυχώς την εντολή SC της γραμμής 60 του κώδικα για την j, λέμε ότι η $T(i,n_i)$ *αποκτά* το ownerhips[j]. Επίσης κάθε φορά που μια δοσοληψία εγγράφει σε κάποια θέση, έστω j, του πίνακα ownerships την τιμή nobody εκτελώντας επιτυχώς την εντολή SC της γραμμής 71 του κώδικα για την j, λέμε ότι η δοσοληψία *καταργεί* το ownerships[j].

Για να αποδείξουμε την ορθότητα του αλγορίθμου SSTM πρέπει να δείξουμε ότι ικανοποιεί την ιδιότητα ορθότητας σειριοποιησιμότητα και επίσης ότι ικανοποιεί την ιδιότητα τερματισμού ελευθερία κλειδωμάτων.

5.2.1. Σειριοποιησιμότητα

Έστω α μια οποιαδήποτε εκτέλεση του SSTM και έστω μια δοσοληψία $T(i,n_i)$. Στην συνέχεια της απόδειξης της ιδιότητας της σειριοποιησιμότητας, υποθέτουμε ότι τουλάχιστον μία executing PerformTrans της $T(i,n_i)$ επιστρέφει από την κλήση της AcquireOwnerships και έστω $Caof(i,n_i)$ η καθολική κατάσταση στην οποία ολοκληρώνεται για πρώτη φορά μια κλήση της AcquireOwnerships από κάποια executing PerformTrans της $T(i,n_i)$. Υπενθυμίζεται ότι εάν η $T(i,n_i)$ ολοκληρωθεί, συμβολίζουμε με $Cf(i,n_i)$ την τελευταία καθολική κατάσταση του $E(T(i,n_i))$.

Όπως έχει αναφερθεί η εκτέλεση της $T(i,n_i)$ πραγματοποιείται σε δύο φάσεις. Κατά την πρώτη φάση οι executing PerformTrans της $T(i,n_i)$ προσπαθούν να αποκτήσουν τις προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που η $T(i,n_i)$ επιθυμεί, εκτελώντας τη συνάρτηση AcquireOwnerships. Εάν καταφέρουν να αποκτήσουν όλες

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

τις προσωρινές ιδιοκτησίες, τότε η $T(i,n_i)$ χαρακτηρίζεται ως επιτυχημένη, ενώ εάν δεν καταφέρουν να τις αποκτήσουν χαρακτηρίζεται ως μη-επιτυχημένη. Ανάλογα με το εάν η $T(i,n_i)$ είναι επιτυχημένη ή μη-επιτυχημένη, θα εκτελεστούν οι γραμμές 35 έως 38 ή οι γραμμές 40 έως 48 της συνάρτησης PerformTrans, αντίστοιχα. Επίσης, βάσει του μηχανισμού βοήθειας του SSTM, οι executing PerformTrans της $T(i,n_i)$ μπορεί να είναι παραπάνω από μία, κάτι που σημαίνει ότι όλες θα πρέπει να εκτελέσουν τις ίδιες γραμμές του κώδικα για την $T(i,n_i)$. Για να γίνεται αυτό, πρέπει να αποδειχθεί ότι υπάρχει κάποια καθολική κατάσταση, έστω *Cst(i,n_i)*, η οποία προηγείται της πρώτης εκτέλεσης της γραμμής 19 από τις executing PerformTrans της $T(i,n_i)$ και στην οποία το $Trec_i$.status της $T(i,n_i)$ θα έχει τιμή COMMITED ή ABORTED. Σημειώνεται ότι η τιμή αυτή δεν θα πρέπει να μπορεί να αλλάξει πριν την $Cf(i,n_i)$. Αυτός είναι ο πρώτος στόχος της απόδειξης.

Εάν η $T(i,n_i)$ χαρακτηριστεί ως επιτυχημένη, τότε κατά το δεύτερο στάδιο της εκτέλεσής της οι executing PerformTrans της θα εκτελέσουν τις γραμμές 35 έως 38 του κώδικα ώστε να ενημερώσουν τις t-μεταβλητές. Για να εξασφαλιστεί η ατομικότητα των αλλαγών αυτών είναι σημαντικό οι executing PerformTrans της $T(i,n_i)$ να έχουν αποκλειστική πρόσβαση στις t-μεταβλητές που πρόκειται να ενημερωθούν. Επίσης επειδή οι executing PerformTrans της $T(i,n_i)$ μπορεί να είναι περισσότερες από μία, πρέπει όλες να ενημερώσουν τις t-μεταβλητές με τις ίδιες τιμές. Επομένως πρέπει να αποδειχθεί ότι i) η δοσοληψία $T(i,n_i)$ χαρακτηρίζεται ως επιτυχημένη εάν και μόνο εάν έχει αποκτήσει όλες τις προσωρινές ιδιοκτησίες που επιθυμεί, ii) οι γραμμές 35 έως 38 εκτελούνται από τουλάχιστον μία executing PerformTrans της $T(i,n_i)$, εάν και μόνο εάν η $T(i,n_i)$ είναι επιτυχημένη, iii) καμία executing PerformTrans δοσοληψίας διαφορετικής της $T(i,n_i)$ δεν μπορεί να ενημερώσει κάποια t-μεταβλητή ενόσω η $T(i,n_i)$ κατέχει την αντίστοιχη ιδιοκτησία, και iv) ότι όλες οι executing δοσοληψίες της $T(i,n_i)$ θα ενημερώσουν με τις ίδιες τιμές τις t-μεταβλητές. Τα παραπάνω αποτελούν το δεύτερο στόχο της απόδειξης.

Ανεξάρτητα με το χαρακτηρισμό της δοσοληψίας ως επιτυχημένης ή μη, γίνεται κατάργηση των ιδιοκτησιών που πιθανώς έχει αποκτήσει η $T(i,n_i)$ υποδηλώνοντας ότι η $T(i,n_i)$ έχει ολοκληρώσει την τροποποίηση των αντίστοιχων t-μεταβλητών. Έτσι πρέπει να αποδειχθεί ότι i) καμία από τις executing PerformTrans της $T(i,n_i)$ δεν

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

αποκτά κάποια προσωρινή ιδιοκτησία, μετά την κατάργησή της και ii) καμία από τις executing PerformTrans της $T(i,n_i)$ δεν μπορεί να ενημερώσει κάποια t-μεταβλητή x, μετά την κατάργηση της προσωρινής ιδιοκτησίας της x. Αυτός είναι ο τρίτος στόχος της απόδειξης.

Είναι σημαντικό ότι οι βοηθητικές PerformTrans της $T(i,n_i)$ μπορούν να συνεχίσουν να εκτελούνται μετά το τέλος της $T(i,n_i)$. Επίσης όλες οι δοσοληψίες που εκκινούνται από την ίδια διεργασία p_i χρησιμοποιούν την ίδια δομή $Trec_i$ για την αποθήκευση των απαραίτητων πληροφοριών τους. Έτσι, θα πρέπει να αποδειχθεί ότι οι βοηθητικές PerformTrans της $T(i,n_i)$ δεν μπορούν να ενημερώσουν κανένα διαμοιραζόμενο ατομικό αντικείμενο μετά το τέλος της εκτέλεσης της $T(i,n_i)$, δηλαδή μετά το τέλος του $E(T(i,n_i))$. Αυτός είναι ο τέταρτος στόχος της απόδειξης.

Η απόδειξη των παραπάνω τεσσάρων στόχων θα πραγματοποιηθεί σε δύο βήματα. Στο πρώτο βήμα, θα αποδειχθεί ο πρώτος στόχος, θα αποδειχθούν τα μέρη i), ii), iii) από τον δεύτερο στόχο, το μέρος i) του τρίτου στόχου, και θα αποδειχθεί ότι καμία executing PerformTrans της $T(i,n_i)$ δε μπορεί να ενημερώσει τα στοιχεία του πίνακα ownerships (δηλαδή να αποκτήσει ή να καταργήσει κάποια προσωρινή ιδιοκτησία) ή οποιοδήποτε διαμοιραζόμενο ατομικό αντικείμενο διατηρείται στη δομή $Trec_i$, μετά το τέλος της $T(i,n_i)$, κάτι το οποίο αποτελεί ένα κομμάτι του τέταρτου στόχου. Κατά το δεύτερο βήμα θα αποδειχθεί το μέρος iv) του δεύτερου στόχου, το μέρος ii) του τρίτου στόχου, και θα αποδειχθεί ότι καμία executing PerformTrans της $T(i,n_i)$ δε μπορεί να ενημερώσει τα στοιχεία του πίνακα memory (δηλαδή να ενημερώσει τα δεδομένα των t-μεταβλητών) μετά το τέλος της $T(i,n_i)$, ότι απέμεινε δηλαδή από τον τέταρτο στόχο.

Σημειώνεται ότι κατά την απόδειξη της ιδιότητας της σειριοποιησιμότητας υποθέτουμε ότι εάν η $T(i,n_i)$ είναι επιτυχημένη τότε έχει μία τουλάχιστον ενεργή executing PerformTrans (κατά την απόδειξη του Λήμματος 5.3 θα παρουσιαστούν οι προϋποθέσεις βάσει των οποίων αυτό μπορεί να συμβαίνει), έτσι ώστε να είμαστε σε θέση να αποδείξουμε τα Λήμματα που απαιτούνται, ανεξάρτητα από καταρρεύσεις διεργασιών.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Αρχικά αποδεικνύονται δύο λήμματα τα οποία θα χρειαστούν στη συνέχεια. Από τον κώδικα (γραμμή 41) προκύπτει ότι μόνο ο δημιουργός μιας δοσοληψίας μπορεί να βοηθήσει μία άλλη δοσοληψία και αυτό θα το κάνει το πολύ μία φορά. Έτσι, κάθε δοσοληψία $T(j,n_j)$ μπορεί να είναι βοηθητική μιας μόνο άλλης δοσοληψίας $T(i,n_i)$, με $i \neq j$. Επίσης για μια δοσοληψία $T(i,n_i)$, από τον κώδικα προκύπτει (γραμμές 28-31, 54-59, 54-63, 55-60, 69-71, 76-78, 83-87, 88-90) ότι ανάμεσα σε κάθε εντολή LL και SC σε κάποιο διαμοιραζόμενο ατομικό αντικείμενο ελέγχεται εάν το τρέχον $Trec_i$.version είναι διαφορετικό από το version (n_i) της $T(i,n_i)$. Ονομάζουμε *VerCheck* κάθε τέτοιο έλεγχο.

Λήμμα 5.1: *Έστω $T(i,n_i)$ μια δοσοληψία και C η καθολική κατάσταση στην οποία η $T(i,n_i)$ αυξάνει το $Trec_i$.version στην γραμμή 18 του κώδικα. Τότε μία βοηθητική PerformTrans της $T(i,n_i)$ που εκτελεί κάποιο VerCheck μετά την C, δεν εκτελεί καμία εντολή SC στους διαμοιραζόμενους ατομικούς καταχωρητές LL/SC μετά το VerCheck.*

Απόδειξη: Όπως αναφέρθηκε πριν από την εκτέλεση μιας εντολής SC από κάποια PerformTrans υπάρχει ένας VerCheck έλεγχος. Αυτός ο VerCheck έλεγχος ελέγχει εάν το n_i είναι διαφορετικό από το $Trec_i$.version. Μετά την έναρξη της $T(i,n_i)$ και πριν από την C ισχύει n_i==$Trec_i$.version. Όμως στην C η $T(i,n_i)$ αυξάνει το $Trec_i$.version (εκτελώντας την γραμμή 18 του κώδικα). Σημειώνεται ότι από τον κώδικα προκύπτει ότι το $Trec_i$.version αυξάνεται μόνο από την p_i και δεν μπορεί να μειωθεί από κανέναν, άρα μόνο αυξάνεται. Έτσι οποιοσδήποτε VerCheck έλεγχος εκτελεστεί από κάποια βοηθητική PerformTrans της $T(i,n_i)$ μετά την C θα αποτιμηθεί ως αληθής, θα εκτελεστεί η return λειτουργία του και έτσι θα αποτραπεί η εκτέλεση της αντίστοιχης εντολής SC σε κάποιο διαμοιραζόμενο καταχωρητή LL/SC. Συμπεραίνουμε ότι στην περίπτωση αυτή δε μπορεί να εκτελεστεί καμία εντολή SC από βοηθητικές PerformTrans της $T(i,n_i)$ σε οποιαδήποτε διαμοιραζόμενο ατομικό καταχωρητή LL/SC. ■

Λήμμα 5.2: *Αν το $Trec_i$.status κάποιας δοσοληψίας $T(i,n_i)$ αποκτήσει τιμή διαφορετική από ACTIVE, τότε το $Trec_i$.status μπορεί να αλλάξει μόνο από την p_i κατά την εκτέλεση της Initiate.*

Απόδειξη: Έστω ότι σε κάποια καθολική κατάσταση C, ισχύει $Trec_i$.status == v, όπου v∈{COMMITED, ABORTED}, και έστω δια της μεθόδου της εις άτοπο απαγωγής, ότι το $Trec_i$.status μπορεί να αλλάξει τιμή πριν η p_i επανεκτελέσει την Initiate. Έστω SC_1 η πρώτη εντολή SC που αλλάζει το $Trec_i$.status από v σε v', όπου v'∈{ COMMITED, ABORTED, ACTIVE}. Από τον κώδικα προκύπτει ότι το $Trec_i$.status μπορεί να αλλάξει κατά την εκτέλεση της γραμμής 31 ή της 59 ή της 63. Όμως σε όλες τις περιπτώσεις η LL που αντιστοιχεί στην SC που κάνει την αλλαγή πρέπει να έχει δει στο $Trec_i$.status τιμή ίση με ACTIVE. Άρα, αφού όταν εκτελείται η SC_1 το $Trec_i$.status==v και v≠ACTIVE, η SC_1 αποτυγχάνει. Άτοπο! ■

Πριν την παρουσίαση της απόδειξης των ισχυρισμών του πρώτου βήματος θα δοθούν κάποιοι συμβολισμοί, η απόδειξη της ορθότητας των οποίων αποδεικνύεται στη συνέχεια. Ορίζουμε ότι η $T(i,n_i)$ ονομάζεται επιτυχημένη εάν το $Trec(i,n_i)$.status της έχει τιμή COMMITED στην $Cst(i,n_i)$ και αποτυχημένη εάν έχει τιμή ABORTED. Εάν η $T(i,n_i)$ είναι επιτυχημένη τότε από τις executing PerformTrans της $T(i,n_i)$ γράφεται για μία και μοναδική φορά i) η τιμή COMMITED στο $Trec_i$.status και συμβολίζουμε με $Cc(i,n_i)$ την καθολική αυτή κατάσταση, ii) η τιμή TRUE στην $Trec_i$.AllWritten και συμβολίζουμε με $Caw(i,n_i)$ την καθολική αυτή κατάσταση, iii) η τιμή memory[l], l∈$Trec_i$.add, στο $Trec_i$.OldValues[l] και συμβολίζουμε ως $Cov(i,n_i,l)$ την καθολική κατάσταση στην οποία αυτό συμβαίνει. Συμβολίζουμε με W_{PT} τον βρόχο while της συνάρτησης PerformTrans (γραμμές 14 έως 17) και W_{AO} τον βρόχο while της συνάρτησης AcquireOwnerships (γραμμές 37 έως 49). Εάν η $T(i,n_i)$ είναι αποτυχημένη τότε η τιμή ABORTED γράφεται στο $Trec_i$.status μία και μοναδική φορά κατά την εκτέλεση του W_{AO} για κάποιο l, l∈$Trec_i$.add, από τις executing PerformTrans της $T(i,n_i)$. Αυτή η θέση l ονομάζεται *θέση σφάλματος* και συμβολίζεται ως $FM(i,n_i)$. Συμβολίζουμε ως $Crof(i,n_i)$ την καθολική κατάσταση στην οποία ολοκληρώνεται για πρώτη φορά μια κλήση της ReleaseOwnerships από κάποια executing PerformTrans της $T(i,n_i)$. Ακόμη, συμβολίζουμε ως $Caov(i,n_i)$ την καθολική κατάσταση στην οποία έχει τερματίσει για πρώτη φορά μία κλήση της συνάρτησης AgreeOldValues από τις executing PerformTrans της $T(i,n_i)$.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Στη συνέχεια παρουσιάζεται και αποδεικνύεται το Λήμμα 5.3 το οποίο περιέχει τους απαραίτητους ισχυρισμούς για την απόδειξη του πρώτου βήματος, που περιγράφηκε παραπάνω.

Λήμμα 5.3: *Έστω μια δοσοληψία $T(i,n_i)$, $n_i \geq 1$, για την οποία ορίζεται η $Caof(i,n_i)$ τότε:*

i) Έστω ένα οποιοδήποτε l, $l \in Trec_i.add$. Κάθε executing PerformTrans της $T(i,n_i)$ που εκτελεί τον W_{AO} για κάποιο l πριν την $Caof(i,n_i)$, μπορεί να φύγει από αυτόν είτε επειδή έχει αποκτηθεί το ownerships[l] από τις executing PerformTrans της $T(i,n_i)$ (συμπεριλαμβανομένου και της ίδιας) ή επειδή έχει γραφτεί (ή έγραψε) στο $Trec_i.status$ η τιμή ABORTED από τις executing PerformTrans της $T(i,n_i)$.

ii) Καμία executing PerformTrans της $T(i,n_i)$ δεν μπορεί να κολλήσει επ' άπειρο στην εκτέλεση του βρόχου W_{PT}.

iii) Η $T(i,n_i)$ χαρακτηρίζεται ως επιτυχημένη εάν και μόνο εάν στην $Caof(i,n_i)$ ισχύει $Trec_i.status == ACTIVE$.

iv) Μία τουλάχιστον executing PerformTrans της $T(i,n_i)$ θα εκτελέσει τις γραμμές 35 έως 38 πριν την $Cf(i,n_i)$, εάν και μόνο εάν η δοσοληψία $T(i,n_i)$ είναι επιτυχημένη.

v) Εάν η $T(i,n_i)$ είναι επιτυχημένη, τότε :

a. Η $Caw(i,n_i)$ είναι καλά ορισμένη, έπεται της $Cc(i,n_i)$ και περιέχεται στο $E(T(i,n_i))$.

b. Μεταξύ της $Cc(i,n_i)$ και της $Caw(i,n_i)$ η $T(i,n_i)$ κατέχει όλες τις προσωρινές ιδιοκτησίες που επιθυμεί και καμία άλλη δοσοληψία δε μπορεί να κατέχει τις συγκεκριμένες προσωρινές ιδιοκτησίες.

c. Εάν μία δοσοληψία $T(j,n_j)$, $j \neq i$, επιθυμεί να αποκτήσει μία θέση l, $l \in Trec_i.add$, του πίνακα ownerships που κατέχει η $T(i,n_i)$, τότε η $T(j,n_j)$ δε μπορεί να εκτελέσει κάποια από τις γραμμές 35 έως 38 μεταξύ της $Cc(i,n_i)$ και της $Caw(i,n_i)$.

d. Για κάθε l, $l \in Trec_i.add$, γράφεται πριν την $Caov(i,n_i)$ (που προηγείται της $Caw(i,n_i)$) για μία και μοναδική φορά η τιμή memory[l] στο $Trec_i.OldValues[l]$, στην καθολική κατάσταση $Cov(i,n_i,l)$, από τις executing PerformTrans της $T(i,n_i)$.

vi) a. Δεν υπάρχει καθολική κατάσταση μετά την $Crof(i,n_i)$ στην οποία ο πίνακας ownerships να περιέχει την τιμή $Trec_i$ με $Trec_i.version \leq n_i$ και b. η $Crof(i,n_i)$ προηγείται της $Cf(i,n_i)$.

vii) Καμία βοηθητική PerformTrans της $T(i,n_i')$, $n_i' \leq n_i$, δεν εκτελεί κάποια επιτυχημένη SC στον πίνακα ownerships και σε οποιαδήποτε διαμοιραζόμενο ατομικό αντικείμενο που περιέχεται στην $Trec_i$, μετά την $Cf(i,n_i)$.

Απόδειξη: Κατά το πρώτο βήμα της απόδειξης προσπαθούμε να αποφύγουμε τα προβλήματα που εισάγει το ότι οι βοηθητικές PerformTrans δοσοληψιών που εκκινήθηκαν από την p_i πριν την εκκίνηση της $T(i,n_i)$, δηλαδή δοσοληψίες τις μορφής $T(i,b)$, όπου $b<n_i$, μπορούν να εκτελούνται και μετά το τέλος των $T(i,b)$. Για το λόγο αυτό, η απόδειξη των Ισχυρισμών του Λήμματος 5.3 θα γίνει με επαγωγή στο πλήθος των δοσοληψιών που η διεργασία p_i έχει εκκινήσει.

Σημειώνεται ότι η απόδειξη της βάσης της επαγωγής και η απόδειξη του επαγωγικού βήματος θα γίνει ταυτόχρονα. Έστω ότι όλες οι προτάσεις του λήμματος ισχύουν για τις k-1, όπου k>0, πρώτες δοσοληψίες που εκκινούνται από την δοσοληψία p_i. Θα αποδειχθεί ότι οι Ισχυρισμοί του Λήμματος ισχύουν και για την $T(i,k)$, την k-οστή δοσοληψία που εκκινείται από την διεργασία p_i.

Σημειώνεται ότι πριν από την $Caof(i,k)$ οι διαμοιραζόμενες δομές δεδομένων της $T(i,k)$ που περιέχονται στην $Trec_i$ μπορούν να αλλάξουν μόνο από τις executing PerformTrans της $T(i,k)$ και τις βοηθητικές PerformTrans των προηγούμενων δοσοληψιών που εκκινήθηκαν από την p_i. Εάν k==1, εφόσον η $T(i,1)$ είναι η πρώτη δοσοληψία που εκκινείται από την p_i, δεν υπάρχουν προηγούμενες δοσοληψίες από την p_i, δηλαδή δοσοληψίες της μορφής $T(i,n_i')$ με $n_i'<1$ και άρα ούτε και βοηθητικές δοσοληψίες τέτοιων δοσοληψιών που να μπορούν να τροποποιήσουν τον ownerships ή οποιαδήποτε άλλη διαμοιραζόμενη μεταβλητή. Εάν k>1, τότε από τον Ισχυρισμό vii) της επαγωγικής υπόθεσης προκύπτει ότι βοηθητικές PerformTrans προηγούμενων δοσοληψιών που εκκινήθηκαν από την p_i δεν μπορούν να τροποποιήσουν τον πίνακα ownerships ή οποιαδήποτε ατομικό αντικείμενο περιέχεται στην $Trec_i$, μετά την $TRf(i,k-1)$ άρα και μετά την εκκίνηση της $T(i,k)$. Έτσι προκύπτει άμεσα το παρακάτω πόρισμα, για την $T(i,k)$.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Πόρισμα 5.4: Στο E(T(i,k)) οι διαμοιραζόμενες μεταβλητές της T(i,k) που περιέχονται στο $Trec_i$ μπορούν να αλλάξουν μόνο από τις executing PerformTrans της T(i,k).

Εάν k==1 προκύπτει ότι πριν την εκκίνηση της T(i,1) δεν μπορεί να προϋπάρχει στον πίνακα ownerships η τιμή $Trec_i$ (υποθέτουμε ότι όταν το σύστημα εκκινείται ο ownerships περιέχει την τιμή null σε κάθε θέση του). Εάν k>1 από τον Ισχυρισμό vi) της επαγωγικής υπόθεσης προκύπτει ότι ο πίνακας ownerships δεν μπορεί να περιέχει το $Trec_i$ με $Trec_i$.version < k μετά την TRf(i,k-1). Έτσι προκύπτει ότι δεν μπορεί να προϋπάρχει η τιμή $Trec_i$ στον πίνακα ownerships κατά την εκκίνηση της T(i,k). Επίσης, εάν k>1 με βάση τον Ισχυρισμό vii) της επαγωγικής υπόθεσης δεν μπορεί να γραφεί ή να διαγραφεί η τιμή $Trec_i$ στον πίνακα ownerships από τις executing PerformTrans προηγούμενων δοσοληψιών που εκκινήθηκαν από την p_i, δηλαδή δοσοληψίες της μορφής $T(i,n_i')$ με $n_i'<k$, μετά την εκκίνηση της T(i,k). Έτσι προκύπτει άμεσα το παρακάτω πόρισμα.

Πόρισμα 5.5: i) Κατά την εκκίνηση της T(i,k) ο πίνακας ownerships δεν μπορεί να περιέχει σε καμία θέση του την τιμή $Trec_i$, ii) στο E(T(i,k)) η $Trec_i$ μπορεί να γραφεί στον πίνακα ownerships μόνο από τις executing PerformTrans της T(i,k).

Έστω α μια εκτέλεση της υλοποίησης του SSTM. Για κάθε *l*, $l \in Trec_i$.add, συμβολίζουμε με $C_{AC}(l)$ την καθολική κατάσταση κατά την οποία αποκτάται για πρώτη φορά από κάποια executing PerformTrans της T(i,k) το ownerships[*l*] (εάν αυτό συμβαίνει). Εάν η $C_{AC(l)}$ ορίζεται, συμβολίζουμε με $α_l$ το διάστημα εκτέλεσης της α που ξεκινά από την $C_{AC(l)}$ και καταλήγει στην τελευταία καθολική κατάσταση, έστω C_l, που προηγείται της πρώτης κατάργησης του ownerships[*l*] από κάποια executing PerformTrans της T(i,k) (εάν αυτό συμβαίνει). Αν η C_l δεν ορίζεται τότε το $α_l$ είναι το επίθεμα της α που ξεκινά με την $C_{AC(l)}$. Αποδεικνύουμε αργότερα ότι εάν η $C_{AC(l)}$ ορίζεται τότε θα ορίζεται και η C_l, η οποία θα έπεται της $C_{AC(l)}$.

Λήμμα 5.6: Έστω ένα οποιοδήποτε l, $l \in Trec_i$.add, για το οποίο ορίζεται η $α_l$. Καμία PerformTrans δεν μπορεί i) να αποκτήσει το ownerships[l] στην $α_l$ ή ii) να καταργήσει το ownerships[l] στην $α_l$.

Απόδειξη: Σημειώνεται ότι η απόκτηση του ownerships[l] μπορεί να γίνει με την επιτυχή εκτέλεση της εντολής SC της γραμμής 60 και η κατάργησή του με την επιτυχή εκτέλεση της εντολής SC της γραμμής 71. Ας υποθέσουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι b>0 executing PerformTrans κάποιας δοσοληψίας $T(z,n_z)$ (μπορεί να ισχύει z==i) εκτέλεσαν επιτυχώς την SC της γραμμής 60 ή της 71 στην α_l. Έστω $PT_j(z,n_z)$ η πρώτη από αυτές που εκτελεί την SC της γραμμής 60 ή της 71, έστω SC_1. Διακρίνουμε περιπτώσεις.

i) Η SC_1 είναι SC της γραμμής 60. Η $PT_j(z,n_z)$ εκτέλεσε επιτυχώς την SC_1 στην α_l (μετά την $C_{AC(l)}$), δηλαδή κατάφερε να αποκτήσει το ownerships[l] μετά την $C_{AC(l)}$. Για να έφτασε η $PT_j(z,n_z)$ στη γραμμή 60 πρέπει να εκτέλεσε την εντολή LL της γραμμής 55 στο ownerships[l], βλέποντας ownerships[l] == nobody. Σημειώνεται ότι στη $C_{AC(l)}$ θα ισχύει ownerships[l] == $Trec_i$. Επίσης, από υπόθεση, η SC_1 είναι η πρώτη εντολή των γραμμών 60 και 71 που εκτελείται μετά την $C_{AC(l)}$, άρα το ownerships[l] δεν μπορεί να έχει καταργηθεί από κάποια PerformTrans, μέσω της εκτέλεσης της SC της γραμμής 71, μετά την $C_{AC(l)}$. Σημειώνεται ότι, από το Πόρισμα 5.5 προκύπτει ότι κατά την εκκίνηση της T(i,k) ο ownerships δεν μπορεί να περιέχει σε καμία θέση του το $Trec_i$. Άρα για να μπορεί να δει η $PT_j(z,n_z)$ ownerships[l] == nobody, πρέπει να εκτέλεσε τη γραμμή 55 πριν την $C_{AC(l)}$. Στην συνέχεια η $PT_j(z,n_z)$ εκτελεί την SC_1 μετά την $C_{AC(l)}$, όμως στην $C_{AC(l)}$ κάποια executing PerformTrans της T(i,k) εκτέλεσε επιτυχώς την SC της γραμμής 60 για το ownerships[l]. Έτσι με βάση τον ορισμό της εντολής LL/SC η εκτέλεση της SC_1 στο ownerships[l] μετά την $C_{AC(l)}$ από την $PT_j(z,n_z)$ θα αποτύχει. Άτοπο.

ii) Η SC_1 είναι SC της γραμμής 71. Σημειώνεται ότι το ownerships[l] δεν μπορεί να καταργηθεί από κάποια executing PerformTrans της T(i,k) στην α_l, εξαιτίας του ορισμού της α_l. Επομένως η $PT_j(z,n_z)$ είναι executing PerformTrans μιας δοσοληψίας $T(z,n_z)$ για την οποία ισχύει $z \neq i$. Η $PT_j(z,n_z)$ εκτέλεσε επιτυχώς την SC_1 στην α_l (μετά την $C_{AC(l)}$), δηλαδή κατάφερε να καταργήσει το ownerships[l] μετά την $C_{AC(l)}$. Για να έφτασε η $PT_j(z,n_z)$ στην γραμμή 71 πρέπει να εκτέλεσε την εντολή LL της γραμμής 69 στο ownerships[l], βλέποντας ownerships[l] == $Trec_z$. Σημειώνεται ότι στην $C_{AC(l)}$ θα ισχύει ownerships[l] == $Trec_i$. Επίσης, από υπόθεση, το ownerships[l] δεν μπορεί να έχει αποκτηθεί, μέσω της εκτέλεσης της SC της γραμμής 60, μετά την $C_{AC(l)}$ από οποιαδήποτε PerformTrans, άρα ούτε και από τις executing PerformTrans της $T(z,n_z)$.

Έτσι για να μπορεί η $PT_j(z,n_z)$ να δει ownerships[*l*] == $Trec_z$ πρέπει να εκτέλεσε την γραμμή 69 πριν την $C_{AC(l)}$. Στην συνέχεια η $PT_j(z,n_z)$ εκτελεί την SC_1 μετά την $C_{AC(l)}$, όμως στην $C_{AC(l)}$ κάποια executing PerformTrans της T(i,k) εκτέλεσε επιτυχώς την SC της γραμμής 60 για το ownerships[*l*], αποθηκεύοντας σε αυτό την $Trec_i$. Έτσι με βάση τον ορισμό της εντολής LL/SC η εκτέλεση της SC της γραμμής 71 στο ownerships[*l*] από την $PT_j(z,n_z)$ μετά την $C_{AC(l)}$ θα αποτύχει. Άτοπο.

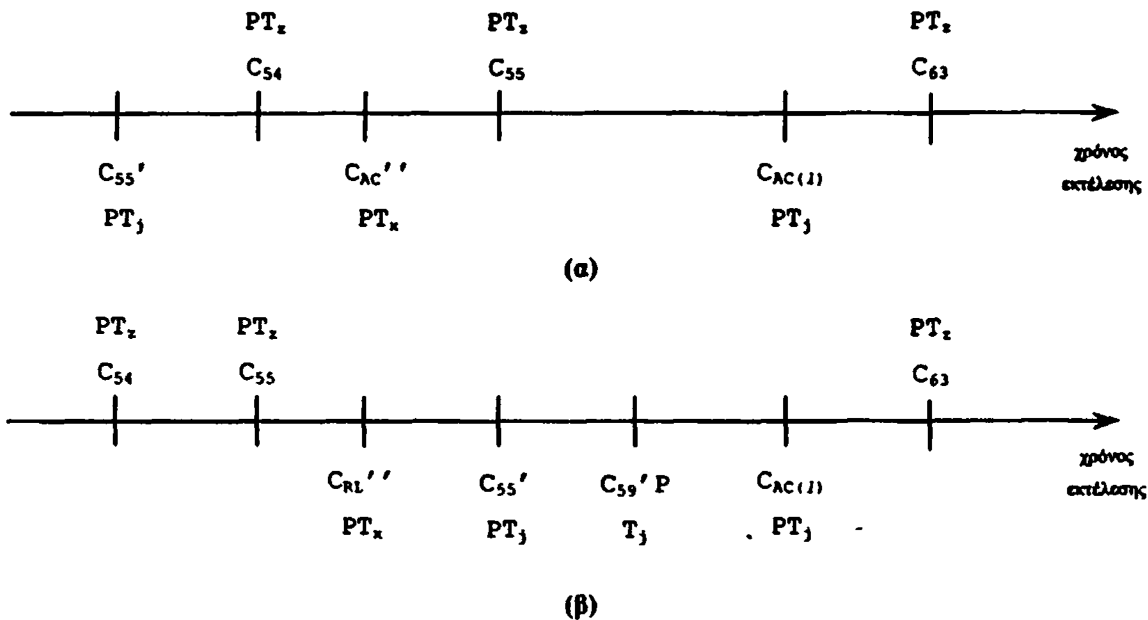
Άρα το Λήμμα ισχύει. ■

Λήμμα 5.7: Έστω ένα οποιοδήποτε l, $l \in Trec_i.add$, για το οποίο ορίζεται η a_l. Τότε το $Trec_i$.status δεν μπορεί να ενημερωθεί μέσω της εκτέλεσης της εντολής SC της γραμμής 63 στην a_l, κατά την εκτέλεση του W_{AO} για το l από τις executing PerformTrans της T(i,k).

Απόδειξη: Εφόσον ορίζεται η a_l σημαίνει ότι ορίζεται η $C_{AC(l)}$, δηλαδή υπάρχει κάποια executing PerformTrans της T(i,k), έστω $PT_j(i,k)$, η οποία εκτελεί επιτυχώς την εντολή SC της γραμμής 60 για το ownerships[*l*], αποθηκεύοντας σε αυτό το Trec(i,k). Μετά την επιτυχή εκτέλεση της SC της γραμμής 60 από κάποια executing PerformTrans της T(i,k), αυτή η PerformTrans αποχωρεί από τον W_{AO} για το *l*. Αυτό σημαίνει ότι δεν μπορεί η $PT_j(i,k)$ να εκτελέσει επιτυχώς την SC της γραμμής 60 για το ownerships[*l*] και στην συνέχεια να εκτελέσει επιτυχώς την SC της γραμμής 63, κατά την εκτέλεση του W_{AO} για το *l*. Ακόμη, από το Πόρισμα 5.4 προκύπτει ότι στο E(T(i,k)) το $Trec_i$.status μπορεί να αλλάξει μόνο από τις executing PerformTrans της T(i,k). Έτσι, υποθέτουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι υπάρχει κάποια executing PerformTrans της T(i,k), έστω $PT_z(i,k)$ και $PT_z(i,k) \neq PT_j(i,k)$, η οποία εκτελεί επιτυχώς την εντολή SC της γραμμής 63 στην a_l, κατά την εκτέλεση του W_{AO} για το *l*, μετά την $C_{AC(l)}$.

Σημειώνεται ότι αρχικά εκτελέστηκε επιτυχώς η SC της γραμμής 60 από την $PT_j(i,k)$ στην $C_{AC(l)}$, και στην συνέχεια εκτελέστηκε επιτυχώς η SC της γραμμής 63 από την $PT_z(i,k)$, έστω στην καθολική κατάσταση C_{63}. Για να έφτασε η $PT_z(i,k)$ στη γραμμή 63 πρέπει εκτελώντας την εντολή LL της γραμμής 55, έστω στην καθολική κατάσταση C_{55}, να είδε ownerships[*l*] == $Trec_m$, $m \neq i$. Από το Λήμμα 5.6 προκύπτει ότι σε όλες τις καθολικές καταστάσεις της a_l θα ισχύει ownerships[*l*] == $Trec_i$. Έτσι η C_{55} πρέπει να προηγείται της $C_{AC(l)}$. Άρα και η εντολή LL της γραμμής 54 πρέπει να

έχει εκτελεστεί από την PT_z πριν την $C_{AC(l)}$, έστω στην καθολική κατάσταση C_{54}. Το Σχήμα 5.11 αναπαριστά τη σειρά εκτέλεσης των εντολών αυτών (καθώς και άλλων που περιγράφονται στη συνέχεια).



Σχήμα 5.11: Εκτελέσεις που περιγράφονται κατά την απόδειξη του Λήμματος 5.7.

Στην $C_{AC(l)}$ έχει εκτελεστεί επιτυχώς η εντολή SC της γραμμής 60 από την $PT_j(i,k)$ και για να μπορεί αυτό να συμβεί πρέπει η $PT_j(i,k)$ μέσω της εντολής LL της γραμμής 55 να είδε ownerships[l] == nobody, έστω στην καθολική κατάσταση C_{55}'. Η C_{55}' προηγείται της $C_{AC(l)}$ και μπορεί είτε να προηγείται ή να έπεται της C_{55}. Εάν η C_{55}' προηγείται της C_{55} τότε στην C_{55}' θα ισχύει ownerships[l] == nobody και στην C_{55} ownerships[l] == $Trec_m$. Αυτό σημαίνει ότι μετά την C_{55}' και πριν την C_{55} πρέπει να αποκτήθηκε το ownerships[l] από κάποια executing PerformTrans της $T(m,n_m)$, έστω $PT_x(m,n_m)$, έστω στην καθολική κατάσταση C_{AC}''. Όμως τότε δεν μπορεί να ορίζεται η κατάσταση $C_{AC(l)}$, διότι η εκτέλεση της εντολής SC της γραμμής 60 από την $PT_j(i,k)$ θα αποτύχει εξαιτίας του ορισμού της εντολής LL/SC. Άτοπο. ΤοΣχήμα 5.11 (α) αναπαριστά την σειρά εκτέλεσης των εντολών αυτών.

Επομένως ας υποθέσουμε τώρα ότι η C_{55}' έπεται της C_{55}. Υπενθυμίζουμε ότι στη C_{55} θα ισχύει ownerships[l] == $Trec_m$ και στη C_{55}' θα ισχύει ownerships[l] == nobody. Αυτό σημαίνει ότι μετά την C_{55} και πριν την C_{55}', πρέπει να καταργήθηκε το ownership[l] από κάποια executing PerformTrans της $T(m,n_m)$, έστω $PT_y(m,n_m)$, έστω

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

στην καθολική κατάσταση C_{RL}''. Σημειώνεται ότι για να μπορεί η $PT_j(i,k)$ να εκτελέσει την SC της γραμμής 60 πρέπει μετά την C_{55}' και πριν την $C_{AC(l)}$ να εκτελέσει επιτυχώς την εντολή SC της γραμμής 59, έστω στην καθολική κατάσταση C_{59}'. Έτσι η εκτέλεση της εντολής SC της γραμμής 63 από την $PT_z(i,k)$ μετά την $C_{AC(l)}$ και άρα μετά την C_{59}' θα αποτύχει λόγω του ορισμού της εντολής LL/SC. Άτοπο. Το Σχήμα 5.11 (β) αναπαριστά την σειρά εκτέλεσης των εντολών αυτών. Άρα το Λήμμα 5.7 ισχύει. ■

Λήμμα 5.8: Εάν υπάρχει καθολική κατάσταση C στην οποία ισχύει Trec(i,k).status ≠ ACTIVE, τότε καμία executing PerformTrans της T(i,k) δεν μπορεί να ενημερώσει το $Trec_i$.status μετά την C.

Απόδειξη: Σημειώνεται, ότι από τον κώδικα προκύπτει ότι το $Trec_i$.status μπορεί να ενημερωθεί με την εκτέλεση των εντολών SC των γραμμών 63, 59 και 31. Η απόδειξη θα δοθεί μόνο για την εντολή SC της γραμμής 63 και η απόδειξη για τις υπόλοιπες γραμμές είναι όμοια. Έτσι, ας υποθέσουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι κάποια executing PerformTrans της T(i,k), έστω $PT_j(i,k)$, εκτελεί επιτυχώς την εντολή SC της γραμμής 63, μετά την C. Για να μπορέσει η $PT_j(i,k)$ να εκτελέσει τη γραμμή 63, πρέπει να εκτελέσει την εντολή LL της γραμμής 54 βλέποντας $Trec_i$.status == ACTIVE. Σημειώνεται ότι στην C ισχύει $Trec_i$.status==ABORTED. Επίσης από το Πόρισμα 5.4 προκύπτει ότι στο E(T(i,k)) το $Trec_i$.status μπορεί να αλλάξει μόνο από τις executing PerformTrans της T(i,k).

Έτσι εάν η γραμμή 54 εκτελέστηκε από την $PT_j(i,k)$ μετά την C τότε για να δει η $PT_j(i,k)$ $Trec_i$.status == null πρέπει, σύμφωνα με το Λήμμα 5.2, να εκτελέστηκε η συνάρτηση Initiate από την p_i, κάτι που σημαίνει ότι η εκτέλεση της T(i,k) έχει ολοκληρωθεί και άρα έχει εκτελεστεί η γραμμή 18 του κώδικα από την T(i,k). Όμως τότε σύμφωνα με το Λήμμα 5.1, η εκτέλεση του VerCheck της γραμμής 56 θα επιτύγχανε και έτσι η γραμμή 63 δεν θα εκτελούνταν από την $PT_j(i,k)$.

Επομένως πρέπει η γραμμή 54 να εκτελέστηκε από την $PT_j(i,k)$ πριν την C. Στην συνέχεια η $PT_j(i,k)$ εκτελεί την εντολή SC της γραμμής 63 μετά την C. Όμως στην C ισχύει $Trec_i$.status == ABORTED, κάτι που σημαίνει ότι κάποια άλλη executing PerformTrans της T(i,k) κατάφερε να εκτελέσει επιτυχώς την εντολή SC της γραμμής

63. Βάσει του ορισμού της εντολής LL/SC, η εκτέλεση της γραμμής 63 από την $PT_j(i,k)$ μετά την C θα αποτύχει. Άτοπο, άρα το Λήμμα 5.8 ισχύει. ■

Λήμμα 5.9: Στην Caof(i,k), το $Trec_i$.status μπορεί να έχει τιμή είτε null ή failure.

Απόδειξη: Από τον κώδικα (γραμμή 6) προκύπτει ότι κατά την εκκίνηση της T(i,k) το $Trec_i$.status έχει τιμή ACTIVE. Από το Πόρισμα 5.4 προκύπτει ότι στο E(T(i,k)) το $Trec_i$.status μπορεί να αλλάξει μόνο από τις executing PerformTrans της T(i,k). Από τον κώδικα της AcquireOwnerships προκύπτει ότι κατά την εκτέλεσή της από οποιαδήποτε δοσοληψία το Trec.status μπορεί να ενημερωθεί είτε σε ACTIVE είτε σε ABORTED. Ακόμη, από τον κώδικα προκύπτει ότι μόνο με την εκτέλεση της εντολή SC της γραμμής 31 του κώδικα μπορεί να γραφεί στο $Trec_i$.status τιμή διαφορετική από ACTIVE ή ABORTED (συγκεκριμένα γράφεται η τιμή COMMITED). Εφόσον η Caof(i,k) είναι η καθολική κατάσταση στην οποία ολοκληρώνεται για πρώτη φορά μια κλήση της AcquireOwnerships από κάποια executing PerformTrans της T(i,k), η SC της γραμμής 31 δεν έχει εκτελεστεί μέχρι την Caof(i,k) από κάποια executing PerformTrans της T(i,k). Επομένως συμπεραίνουμε ότι στην Caof(i,k) το $Trec_i$.status μπορεί να έχει τιμή ACTIVE ή ABORTED.

Λήμμα 5.10: Έστω ένα οποιοδήποτε l, $l \in Trec_i$.add. Κάθε executing PerformTrans της T(i,k) που εκτελεί τον W_{AO} για κάποιο l πριν την Caof(i,k), μπορεί να φύγει από αυτόν είτε επειδή έχει αποκτηθεί το ownerships[l] από τις executing PerformTrans της T(i,k) (συμπεριλαμβανομένου και της ίδιας) ή επειδή έχει γραφτεί (ή έγραψε) στο $Trec_i$.status η τιμή ABORTED από τις executing PerformTrans της T(i,k).

Απόδειξη: Από τον κώδικα της AcquireOwnerships προκύπτει ότι μία executing PerformTrans της T(i,k) που εκτελεί τον W_{AO} για κάποιο *l* μπορεί να φύγει από τον W_{AO} είτε επειδή το $Trec_i$.status έχει τιμή διαφορετική από ACTIVE (γραμμή 54), ή επειδή έχει ολοκληρωθεί η T(i,k) (γραμμή 56), ή επειδή έχει αποκτηθεί το ownerships[*l*] (γραμμή 57), ή επειδή απέκτησε το ownerships[*l*] (γραμμές 60 και 61), ή επειδή έγραψε την τιμή ABORTED στο $Trec_i$.status (γραμμές 63 και 64).

Από το Λήμμα 5.9 προκύπτει ότι στην Caof(i,k) η τιμή του $Trec_i$.status μπορεί να είναι είτε ACTIVE ή ABORTED. Επομένως μία executing PerformTrans της T(i,k)

μπορεί να εξέλθει του W_{AO} λόγω της γραμμής 54, μόνο εάν το $Trec_i$.status έχει πάρει την τιμή ABORTED, άρα ο ισχυρισμός ισχύει σε αυτή την περίπτωση. Επίσης πριν την Caof(i,k) δεν έχει ολοκληρωθεί η εκτέλεση της T(i,k), άρα καμία executing PerformTrans της T(i,k) δεν μπορεί να εξέλθει του W_{AO} λόγω της γραμμής 56.

Ακόμη, από το Πόρισμα 5.5 προκύπτει ότι κατά την εκκίνηση της T(i,k) ο ownerships δεν μπορεί να περιέχει σε καμία θέση του το $Trec_i$ και επίσης στο E(T(i,k)) το $Trec_i$ μπορεί να γραφεί στον πίνακα ownerships μόνο από τις executing PerformTrans της T(i,k). Γίνεται κατανοητό ότι πριν την Caof(i,k) κάθε executing PerformTrans της T(i,k) που εκτελεί τον W_{AO} για το *l* μπορεί να φύγει από αυτόν είτε επειδή έχει αποκτηθεί (ή απέκτησε) το ownerships[*l*] ή επειδή έχει γραφτεί (ή έγραψε) στο $Trec_i$.status η τιμή ABORTED. ■

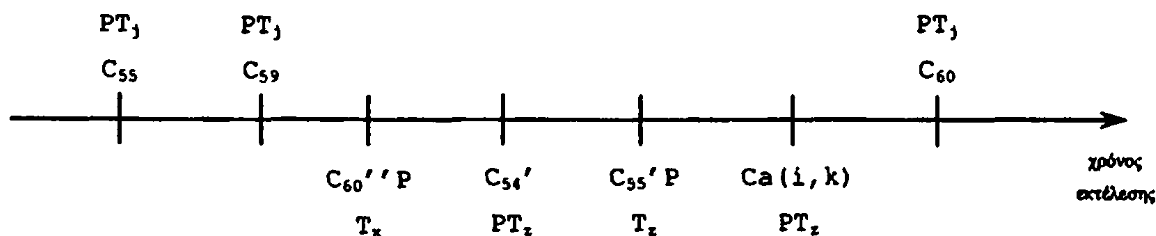
Από τον κώδικα της συνάρτησης AcquireOwnerships (γραμμές 51 και 53) προκύπτει ότι τουλάχιστον μία executing PerformTrans της T(i,k) θα προσπαθήσει να αποκτήσει τις θέσεις του πίνακα ownerships που επιθυμεί η T(i,k) (οι θέσεις που περιέχονται στον πίνακα $Trec_i$.add) με προκαθορισμένη σειρά από την μικρότερη προς την μεγαλύτερη, πριν την Caof(i,k). Αυτό σημαίνει ότι τουλάχιστον μία από τις executing PerformTrans της T(i,k) θα εκτελέσει τον W_{AO} για κάποια *l*, $l \in Trec_i$.add, (ίσως και όλα) πριν την Caof(i,k). Επίσης από το Λήμμα 5.10 προκύπτει ότι πριν την Caof(i,k) οι executing PerformTrans της T(i,k) που θα εκτελέσουν τον W_{AO} για κάποιο *l*, $l \in Trec_i$.add, θα αποχωρήσουν από τον W_{AO} είτε αποκτώντας το ownerships[*l*] είτε γράφοντας την τιμή ABORTED στο $Trec_i$.status. Σημειώνεται ότι από το Λήμμα 5.8 προκύπτει ότι η τιμή ABORTED γράφεται στο $Trec_i$.status το πολύ μία φορά από τις executing PerformTrans της T(i,k). Έτσι, εάν υποθέσουμε ότι για κάποιο *l*, $l \in Trec_i$.add, εκτελείται επιτυχώς η εντολή SC της γραμμής 63 από κάποια executing PerformTrans της T(i,k) πριν την Caof(i,k), συμβολίζουμε με *Ca(i,k)* την καθολική κατάσταση στην οποία γράφεται για μία και μοναδική φορά η τιμή ABORTED στο $Trec_i$.status. Επίσης συμβολίζουμε *FM(i,k)* αυτή την τιμή του *l*, για την οποία εκτελέστηκε ο βρόχος W_{AO} και γράφτηκε η τιμή ABORTED στο $Trec_i$.status. Εάν η Ca(i,k) δεν ορίζεται τότε η FM(i,k) είναι null. Σημειώνεται ότι εάν ορίζεται η Ca(i,k) τότε εξ ορισμού της θα προηγείται της Caof(i,k).

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Λήμμα 5.11: Εάν η Ca(i,k) ορίζεται τότε δεν είναι δυνατό να αποκτηθεί το ownerships[FM(i,k)] από τις executing PerformTrans της T(i,k), μετά την Ca(i,k).

Απόδειξη: Στην Ca(i,k) κάποια executing PerformTrans της T(i,k), έστω $PT_z(i,k)$, εκτέλεσε επιτυχώς την εντολή SC της γραμμής 63. Επίσης, από τον κώδικα προκύπτει ότι μετά την επιτυχή εκτέλεση της SC της γραμμής 63 από την $PT_z(i,k)$ στην Ca(i,k), η $PT_z(i,k)$ αποχωρεί από τη συνάρτηση AcquireOwnerships. Αυτό σημαίνει ότι δεν μπορεί η $PT_z(i,k)$ να εκτελέσει μετά την επιτυχή εκτέλεση της εντολής SC της γραμμής 63, την εντολή SC της γραμμής 60. Έτσι υποθέτουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι υπάρχει κάποια executing PerformTrans της T(i,k), έστω $PT_j(i,k)$ όπου $PT_j(i,k) \neq PT_z(i,k)$, που εκτελεί επιτυχώς την εντολή SC της γραμμής 60 για το ownerships[FM(i,k)] μετά την Ca(i,k).

Για να μπορεί η $PT_j(i,k)$ να εκτελέσει την εντολή SC της γραμμής 60 πρέπει προηγουμένως να εκτελέσει επιτυχώς την εντολή SC της γραμμής 59, έστω στην καθολική κατάσταση C_{59}. Από το Λήμμα 5.8 προκύπτει ότι η C_{59} πρέπει να προηγείται της Ca(i,k). Άρα και η εντολή LL, της γραμμής 55, στο ownerships[FM(i,k)] εκτελέστηκε πριν την Ca(i,k), έστω στην καθολική κατάσταση C_{55}. Για να μπορεί να εκτελεστεί η εντολή SC της γραμμής 60 από την $PT_j(i,k)$ πρέπει η $PT_j(i,k)$ να είδε ownerships[FM(i,k)] = NOBODY στην C_{55}. Το Σχήμα 5.12 αναπαριστά τη σειρά εκτέλεσης των εντολών αυτών (καθώς και άλλων που περιγράφονται στη συνέχεια).



Σχήμα 5.12: Εκτέλεση που περιγράφεται κατά την απόδειξη του Λήμματος 5.11.

Στη συνέχεια η $PT_j(i,k)$ εκτελεί την εντολή SC της γραμμής 60 μετά την Ca(i,k), έστω στην καθολική κατάσταση C_{60}. Όμως στην Ca(i,k) η $PT_z(i,k)$ εκτέλεσε επιτυχώς την εντολή SC της γραμμής 63, γράφοντας έτσι στο $Trec_i$.status την τιμή ABORTED.

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

Υπενθυμίζεται ότι στην C_{59} η $PT_j(i,k)$ εκτελεί επιτυχώς την εντολή SC της γραμμής 59 και εγγράφει η τιμή ACTIVE στο $Trec_i$.status. Έτσι για να μπορεί να εκτελεστεί επιτυχώς η γραμμή 63 από την $PT_z(i,k)$ (καθολική κατάσταση Ca(i,k)) πρέπει η εντολή LL, της γραμμής 54, στο $Trec_i$.status να εκτελέστηκε από την $PT_z(i,k)$ μετά την C_{59}, έστω στην καθολική κατάσταση C_{54}'. Σε αντίθετη περίπτωση, λόγω του ορισμού της εντολής LL/SC, η εκτέλεση της εντολής SC της γραμμής 63 από την $PT_z(i,k)$ μετά την C_{59} θα αποτύγχανε.

Επίσης για να μπορεί να εκτελεστεί η γραμμή 63 από την $PT_z(i,k)$ πρέπει αυτή να διάβασε μέσω της εντολής LL της γραμμής 55 ownerships[FM(i,k)] == $Trec_m$, $m \neq i$, έστω στην καθολική κατάσταση C_{55}' (η οποία έπεται της C_{39}'). Για να μπορεί αυτό να ισχύει πρέπει μετά την C_{40} και πριν την C_{55}' κάποια executing PerformTrans δοσοληψίας διαφορετικής της T(i,k), έστω PT_x, να απέκτησε το ownerhips[FM(i,k)]. Αυτό σημαίνει ότι μετά την C_{55} εκτελέστηκε επιτυχώς από την PT_x η εντολή SC της γραμμής 60 για το ownerships[FM(i,k)], έστω στην καθολική κατάσταση C_{60}''. Επομένως λόγω του ορισμού της εντολής LL/SC η εκτέλεση της εντολής SC της γραμμής 60 στο ownerships[FM(i,k)] από την $PT_j(i,k)$ μετά την Ca(i,k) θα αποτύχει. Άτοπο, άρα το Λήμμα 5.11 ισχύει. ■

Λήμμα 5.12: Εάν η Ca(i,k) ορίζεται, τότε: i) Η FM(i,k) είναι η μικρότερη θέση του πίνακα ownerships που κάποια executing PerformTrans της T(i,k) δεν κατάφερε να αποκτήσει, πριν την Ca(i,k), και ii) Καμία executing PerformTrans της T(i,k) δεν αποκτά κάποια θέση του ownerships που είναι μεγαλύτερη της FM(i,k) μετά την Ca(i,k).

Απόδειξη: Με βάση το Λήμμα 5.11, καμία από τις executing PerformTrans της T(i,k) δεν μπορεί να αποκτήσει το ownerships[FM(i,k)] μετά την Ca(i,k). Έτσι υποθέτουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι η FM(i,k) αποκτήθηκε από τις executing PerformTrans της T(i,k) πριν την Ca(i,k). Από το Λήμμα 5.6 προκύπτει ότι καμία PerformTrans δεν μπορεί να καταργήσει κάποιο ownerships που κατέχει η T(i,k) πριν την κατάργησή του από κάποια executing PerformTrans της T(i,k). Επίσης, από τον κώδικα προκύπτει ότι δεν μπορεί να εκτελεστεί η συνάρτηση ReleaseOwnerships από καμία executing PerformTrans της T(i,k) πριν την Caof(i,k), άρα αυτές οι PerformTrans δεν μπορούν να καταργήσουν κάποιο ownerships πριν την AOf(i,k).

Επομένως, όσα από τα διαστήματα a_l, $l \in Trec_i.add$, ορίζονται, δεν μπορούν να ολοκληρωθούν πριν την Caof(i,k). Τότε από το Λήμμα 5.7 προκύπτει ότι η Ca(i,k) δεν μπορεί να περιέχεται στο διάστημα εκτέλεσης του W_{AO} για το FM(i,k). Άτοπο. Επομένως το ownerships[FM(i,k)] δεν μπορεί να αποκτηθεί από τις executing PerformTrans της T(i,k).

Ας υποθέσουμε, δια της μεθόδους της εις άτοπο απαγωγής, ότι υπάρχει κάποιο l, $l \in Trec_i.add$ και l<FM(i,k), για το οποίο δεν έχει αποκτηθεί το ownerships[l]. Τότε, από το Λήμμα 5.10 προκύπτει ότι κατά την εκτέλεση του W_{AO} για το l γράφετε η τιμή ABORTED στο $Trec_i$.status. Δεδομένου ότι στην Ca(i,k) το $Trec_i$.status ορίζεται σε ABORTED, προκύπτει ότι το $Trec_i$.status θα οριζόταν σε ABORTED δύο φορές, το οποίο είναι άτοπο με βάση το Λήμμα 5.8.

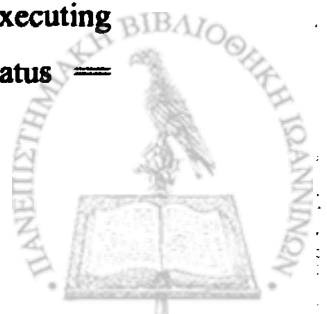
Από τον κώδικα της AcquireOwnerships (γραμμές 51 και 53) προκύπτει ότι οι executing PerformTrans της T(i,k) προσπαθούν να αποκτήσουν τις θέσεις του πίνακα ownerships που επιθυμεί η T(i,k) με προκαθορισμένη σειρά από την μικρότερη προς την μεγαλύτερη. Επίσης όπως αποδείχθηκε, καμία executing PerformTrans της T(i,k) δεν θα αποκτήσει το ownerships[FM(i,k)]. Έτσι, με βάση το Λήμμα 5.10, προκύπτει ότι κανένα ownerships[l'], $l' \in Trec_i.add$ και l' > FM(i,k), δεν μπορεί να αποκτηθεί από τις executing PerformTrans της T(i,k), μετά την Ca(i,k).

Επομένως το Λήμμα 5.12 ισχύει. ■

Λήμμα 5.13: Στην Caof(i,k) ισχύει $Trec_i$.status == ACTIVE εάν και μόνο εάν οι executing PerformTrans της T(i,k) έχουν αποκτήσει, πριν από την Caof(i,k), όλα τα ownerships που η T(i,k) επιθυμεί.

Απόδειξη: Από το Λήμμα 5.9 προκύπτει ότι στην Caof(i,k) το $Trec_i$.status μπορεί να έχει τιμή null ή failure. Εάν στην AOf(i,k) ισχύει $Trec_i$.status == ABORTED τότε από το Λήμμα 5.12 προκύπτει ότι οι executing PerformTrans της T(i,k) δεν έχουν αποκτήσει όλα τα ownerships που η T(i,k) επιθυμεί, κάτι το οποίο έρχεται σε αντίθεση με την υπόθεση του ισχυρισμού (ότι έχουν αποκτηθεί όλα τα ownerships). Έτσι εξετάζεται η περίπτωση που στην Caof(i,k) ισχύει $Trec_i$.status == ACTIVE.

Το Λήμμα 5.2 αναφέρει ότι εάν το Trec(i,k).status λάβει τιμή διαφορετική από ACTIVE τότε το $Trec_i$.status δεν θα μπορεί να αλλάξει ξανά από τις executing PerformTrans της T(i,k). Από υπόθεση, στην Caof(i,k) ισχύει $Trec_i$.status ==



ACTIVE. Άρα, με βάση το Λήμμα 5.2, κατά την εκτέλεση της AcquireOwnerships από τις executing PerformTrans της T(i,k) το $Trec_i$.status δεν θα μπορούσε να έχει λάβει τιμή ABORTED, πριν την Caof(i,k).

Από τον κώδικα της AcquireOwnerships (γραμμές 51 και 53) προκύπτει ότι τουλάχιστον μία executing PerformTrans της T(i,k) θα προσπαθήσει να αποκτήσει τις θέσεις του πίνακα ownerships που επιθυμεί η T(i,k) (οι θέσεις που περιέχονται στον πίνακα $Trec_i$.add) με προκαθορισμένη σειρά από την μικρότερη προς την μεγαλύτερη, πριν την Caof(i,k). Αυτό σημαίνει ότι τουλάχιστον μία από τις executing PerformTrans της T(i,k) θα εκτελέσει τον W_{AO} για κάποια l, $l \in Trec_i$.add, (ίσως και όλα) πριν την AOf(i,k). Επίσης από το Λήμμα 5.10 προκύπτει ότι πριν την Caof(i,k) οι executing PerformTrans της T(i,k) που θα εκτελέσουν τον W_{AO} για κάποιο l θα αποχωρήσουν από αυτόν είτε αποκτώντας το ownerships[l] είτε γράφοντας την τιμή ABORTED στο $Trec_i$.status. Εφόσον ορίζεται η Caof(i,k) σημαίνει ότι ολοκληρώθηκε μία τουλάχιστον φορά η εκτέλεση της AcquireOwnerships από τις executing PerformTrans της T(i,k). Επίσης, όπως έχει αναφερθεί, δεν μπορεί να γραφεί η τιμή failure στο $Trec_i$.status πριν την Caof(i,k), άρα πρέπει να έχουν αποκτηθεί όλα τα ownerships που επιθυμεί η T(i,k) πριν την Caof(i,k). Άρα το Λήμμα 5.13 ισχύει. ■

Από το Λήμμα 5.9 προκύπτει ότι στην Caof(i,k) το $Trec_i$.status μπορεί να έχει τιμή ACTIVE ή ABORTED. Έστω A_{OWN} το σύνολο που περιέχει τις θέσεις του πίνακα ownerships που ζητήθηκαν και αποκτήθηκαν από τις executing PerformTrans της T(i,k). Εάν στην Caof(i,k) ισχύει $Trec_i$.status == ACTIVE τότε από το Λήμμα 5.13 προκύπτει ότι το ownerships κάθε θέσης που ανήκει στο $Trec_i$.add έχει αποκτηθεί από κάποια executing PerformTrans της T(i,k), πριν την Caof(i,k). Αυτό σημαίνει ότι το A_{OWN} περιέχει όλες τις θέσεις του πίνακα ownerships που περιγράφονται από το $Trec_i$.add. Από την άλλη εάν στην Caof(i,k) ισχύει $Trec_i$.status == ABORTED ορίζεται η Ca(i,k) που προηγείται της Caof(i,k). Τότε από το Λήμμα 5.12 προκύπτει ότι οι executing PerformTrans της T(i,k) έχουν αποκτήσει, πριν την Caof(i,k), όλες τις θέσεις του πίνακα ownerships που επιθυμεί η T(i,k) και προηγούνται της FM(i,k). Αυτό σημαίνει ότι το A_{OWN} περιέχει όλες τις θέσεις του πίνακα ownerships που περιγράφονται από το $Trec_i$.add και είναι μικρότερες της FM(i,k). Έτσι, ανεξάρτητα από την τιμή του $Trec_i$.status στην Caof(i,k) ορίζονται όλες οι καθολικές καταστάσεις

$C_{AC(l)}$, $l \in A_{OWN}$, οι οποίες προηγούνται της Caof(i,k). Επίσης, όπως έχει ήδη αναφερθεί, κανένα από τα διαστήματα a_l, $l \in A_{OWN}$, δεν μπορεί να ολοκληρωθεί πριν την Caof(i,k). Έτσι προκύπτει το παρακάτω πόρισμα.

Πόρισμα 5.14: i) Για κάθε $l \in A_{OWN}$ ορίζεται η καθολική κατάσταση $C_{AC}(l)$ και προηγείται της Caof(i,k), καθώς επίσης ορίζεται και το διάστημα εκτέλεσης a_l, και ii) η Caof(i,k) ανήκει στην a_l.

Από τον κώδικα της AcquireOwnerships (γραμμές 51 και 53) προκύπτει ότι τουλάχιστον μία executing PerformTrans της T(i,k) θα προσπαθήσει να αποκτήσει τις θέσεις του πίνακα ownerships που επιθυμεί η T(i,k) (οι θέσεις που περιέχονται στον πίνακα $Trec_i$.add) με προκαθορισμένη σειρά από την μικρότερη προς την μεγαλύτερη, πριν την Caof(i,k). Αυτό σημαίνει ότι τουλάχιστον μία από τις executing PerformTrans της T(i,k) θα εκτελέσει τον W_{AO} για κάποια l, $l \in Trec_i$.add, (ίσως και όλα) πριν την Caof(i,k). Επίσης από το Λήμμα 5.10 προκύπτει ότι πριν την Caof(i,k) οι executing PerformTrans της T(i,k) που θα εκτελέσουν τον W_{AO} για κάποιο l, $l \in Trec_i$.add, θα αποχωρήσουν από τον W_{AO} είτε αποκτώντας το ownerships[l] είτε γράφοντας την τιμή ABORTED στο $Trec_i$.status. Ακόμη από το Λήμμα 5.12 και το Λήμμα 5.6 προκύπτει ότι καμία από τις executing PerformTrans της T(i,k) δεν μπορεί να αποκτήσει κάποιο ownerships μετά την εγγραφή της τιμής ABORTED στο $Trec_i$.status. Έτσι προκύπτει ότι οι θέσεις του πίνακα ownerships που επιθυμεί η T(i,k) να αποκτήσει και περιγράφονται από τον A_{OWN} αποκτώνται από την μικρότερη προς τη μεγαλύτερη.

Επίσης, από το Πόρισμα 5.14 προκύπτει ότι όλες οι καθολικές καταστάσεις $C_{AC(l)}$, $l \in A_{OWN}$, ορίζονται και προηγούνται της Caof(i,k), καθώς επίσης ορίζονται και τα διαστήματα εκτέλεσης a_l. Έτσι η τελευταία θέση του πίνακα ownerships που θα αποκτηθεί είναι η $Trec_i$.add[$|A_{OWN}|$]. Συμβολίζουμε με $C_{AC(last)}$ την πρώτη καθολική κατάσταση κατά την οποία αποκτήθηκε το ownerships[$Trec_i$.add[$|A_{OWN}|$]] από κάποια executing PerformTrans της T(i,k). Από το Πόρισμα 5.14 προκύπτει ότι όλες οι καθολικές καταστάσεις $C_{AC(l)}$, $l \in Trec_i$.add και $l < Trec_i$.add[$|A_{OWN}|$], προηγούνται της $C_{AC(last)}$ και η $C_{AC(last)}$ προηγείται της Caof(i,k). Έστω l_{first}, $l_{first} \in Trec_i$.add, η πρώτη θέση του ownerships που καταργούνται από κάποια executing PerformTrans της

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

T(i,k) (εάν αυτό συμβαίνει). Συμβολίζουμε με a_{ROs} το διάστημα εκτέλεσης της α που ξεκινά από την $C_{AC(last)}$ και καταλήγει στην πρώτη καθολική κατάσταση που προηγείται της πρώτης κατάργησης του ownerships[l_{first}] από κάποια executing PerformTrans της T(i,k) (εάν αυτό συμβαίνει). Εάν αυτό δεν συμβαίνει, τότε η a_{ROs} είναι το επίθεμα της α που ξεκινά από την $C_{AC(last)}$. Σε αυτή την περίπτωση παρατηρούμε ότι καμία θέση του πίνακα ownerships δεν καταργούνται ποτέ από κάποια executing PerformTrans της T(i,k). Από τα παραπάνω και με βάση το Λήμμα 5.6 και το Λήμμα 5.13 προκύπτει το παρακάτω πόρισμα.

Πόρισμα 5.15: *Εάν και μόνο εάν στην Caof(i,k) ισχύει Trec$_i$.status == ACTIVE, τότε στην a_{ROs} η T(i,k) κατέχει όλα τα ownerships που επιθυμεί και καμία άλλη δοσοληψία δε μπορεί να κατέχει τα συγκεκριμένα ownerhips.*

Επίσης με βάση τον ορισμό της a_{ROs}, το Λήμμα 5.7 και το Λήμμα 5.13 προκύπτει το παρακάτω πόρισμα.

Πόρισμα 5.16: *Εάν στην Caof(i,k) ισχύει Trec$_i$.status == ACTIVE, τότε μετά την Caof(i,k) και πριν το τέλος της a_{ROs} το Trec$_i$.status δεν μπορεί να ενημερωθεί μέσω της εκτέλεσης της εντολής SC της γραμμής 63.*

Λήμμα 5.17: *Εάν στην Caof(i,k) ισχύει Trec$_i$.status == ACTIVE, τότε μετά την Caof(i,k) και πριν το τέλος της a_{ROs} το Trec$_i$.status μπορεί να ενημερωθεί το πολύ μία φορά μέσω της εκτέλεσης της εντολής SC της γραμμής 59 από τις executing PerformTrans της T(i,k).*

Απόδειξη: Από το Πόρισμα 5.4 προκύπτει ότι στο E(T(i,k)) το Trec$_i$.status μπορεί να αλλάξει μόνο από τις executing PerformTrans της T(i,k). Αρχικά αποδεικνύεται ότι υπάρχει μία εκτέλεση α του SSTM, η οποία χαρακτηρίζεται ως «κακή», τέτοια ώστε το Trec$_i$.status να ενημερώνεται από τις executing δοσοληψίες της T(i,k), μία φορά μέσω της εκτέλεσης της εντολής SC της γραμμής 59 μετά την Caof(i,k) και πριν το τέλος της a_{ROs}. Στη συνέχεια κατασκευάζουμε μια τέτοια εκτέλεση.

Στο Σχήμα 5.13 παρουσιάζεται σχηματικά η «κακή» εκτέλεση, η οποία περιγράφεται στη συνέχεια. Έστω ότι το σύστημα βρίσκεται σε μία καθολική

κατάσταση C στην οποία καμία δοσοληψία δεν κατέχει κάποια θέση του πίνακα ownerhips και όλες οι executing PerformTrans δοσοληψιών που έχουν εκκινηθεί από όλες τις διεργασίες έχουν ολοκληρωθεί. Δηλαδή το σύστημα μοιάζει σαν να ξεκινά από το μηδέν. Έστω ότι μετά την C στο σύστημα εμφανίζονται δύο μόνο εκκρεμείς δοσοληψίες που εκτελούν τον SSTM, έστω T(i,k) και T(j,k') (ισχύει i≠j). Οι δύο αυτές δοσοληψίες επιθυμούν να αποκτήσουν ακριβώς τις ίδιες θέσεις του πίνακα ownerships, έστω b>1 στο αριθμό. Αρχικά επιτρέπουμε στην T(i,k) να τρέξει μόνη της και να εκτελέσει τη συνάρτηση AcquireOwnerships. Επιτρέπουμε στην T(i,k) να αποκτήσει τα b-1 πρώτα ownerhips που επιθυμεί και την σταματάμε καθώς θα εκτελεί τον βρόχο while της συνάρτησης AcquireOwnerships για το b-οστό (το τελευταίο) ownerships στη γραμμή 60, δηλαδή πριν εκτελέσει την εντολή SC για να αποκτήσει το b-οστό ownerships. Στην συνέχεια επιτρέπουμε στην T(j,k') να εκτελεστεί μόνη της. Καθώς η T(j,k') θα εκτελεί την AcquireOwnerships, στο πρώτο ownerships που επιθυμεί να αποκτήσει θα παρατηρήσει (μέσω της εντολής LL της γραμμής 55) ότι το κατέχει η T(i,k). Έτσι η T(j,k') θα εκτελέσει την γραμμή 63, γράφοντας την τιμή ABORTED στο $Trec_j$.status της, θα επιστρέψει από την κλήση της AcquireOwnerships και θα συνεχίσει εκτελώντας τη γραμμή 48 για να βοηθήσει την T(i,k). Έτσι, από δω και στο εξής η T(j,k') είναι βοηθητική δοσοληψία της T(i,k). Η T(j,k') θα εκτελέσει την συνάρτηση AcquireOwnerships και θα παρατηρήσει (μέσω της εντολής LL της γραμμής 57) για τα (b-1)-οστά ownerships που η T(i,k) επιθυμεί να αποκτήσει, ότι έχουν ήδη αποκτηθεί από την T(i,k). Οπότε η T(j,k') συνεχίζει εκτελώντας και αυτή τον βρόχο while της AcquireOwnerships για το b-οστό ownerships που η T(i,k) επιθυμεί να αποκτήσει. Κατά την εκτέλεση του βρόχου αυτού, σταματάμε την T(j,k') πριν εκτελέσει την εντολή SC της γραμμής 59, δηλαδή πριν γράψει στο $Trec_i$.status την τιμή ACTIVE. Στην συνέχεια η T(i,k) εκτελεί την εντολή SC της γραμμής 60 για το b-οστό ownerships επιτυχώς, διότι δεν υπάρχει καμία άλλη δοσοληψία που να μπορεί να γράψει στο ίδιο ownerships και να προκαλέσει την αποτυχημένη εκτέλεση της γραμμής 60 από την T(i,k). Έτσι η T(i,k) ολοκληρώνει την εκτέλεση της AcquireOwnerships και στο σημείο αυτό ορίζεται η καθολική κατάσταση Caof(i,k). Στη συνέχεια (σημειώνεται μετά την Caof(i,k)) επιτρέπουμε στην T(j,k') να εκτελέσει την SC της γραμμής 59. Η εκτέλεση αυτή θα είναι επιτυχής, διότι από την στιγμή που η T(j,k') διάβασε το $Trec_i$.status (μέσω της εντολής LL της γραμμής 54) και έως τώρα δεν έχει εκτελεστεί κάποιο SC στο

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

$Trec_i$.status από άλλη δοσοληψία. Έτσι, κατασκευάστηκε μια εκτέλεση α στην οποία η εντολή SC της γραμμής 59 εκτελείται επιτυχώς μετά την Caof(i,k) και πριν το τέλος της α_{ROs}.

T(j,k')

```
// T(j,k') wants to obtain b ownerships
// that are the same with T(i,k)'s.

     -- Start of AcquireOwnerships --

53:      location = 1;

// 1 is the first position of Ownerships
// that T(j,k') wants

54:      if (LL(Trecj.status) ≠ ACTIVE) (F)
55:      owner = LL(ownerships[1]);
56:      if (Trecj.version ≠ version) (F)
57:      if (owner==Trecj)   (F)
58:      if (owner==NOBODY) (F)
62:      else
63:       if(SC(Trecj.status,(ABORTED,j))) (T)
```

T(i,k)

```
// T(i,k) wants to obtain b ownerships
// that are the same with T(j,k')'s.

     -- Start of AcquireOwnerships --

// T(i,k) obtains all b-1 ownerships that
// wants and now is executing the while
// loop of AcquireOwnerships for the b-th
// ownerships

...
...
53:      location = 1;

// 1 is the last not obtained position of
// Ownerships that T(i,k) wants

54:      if (LL(Trec.status) ≠ null) (F)
55:      owner = LL(ownerships[1]);
56:      if (Treci.version ≠ version) (F)
57:      if (owner==Treci) (F)
58:      if (owner==nobody) (T)
59:       if(SC(Treci.status,(NULL,0))) (T)

// In next step PTy will execute SC of
// line 60 successfully
```

```
64:        return;
...
...
48:     PerformTrans(Trec_i, 1, FALSE);

// From this point T(j,k') is a helping
// transaction of T(i,k)

...
     -- Start of AcquireOwnerships --

// T(j,k') see that b-1 ownerships are
// already obtained by T(i,k) and now
// executes the while loop for b-th
// ownerships

53:     location = l;

// l is the last (not obtained) position
// of ownerships that T(i,k') requires

54:     if (LL(Trec_i.status) ≠ ACTIVE) (F)
55:     owner = LL(ownerships[l]);
56:     if (Trec_i.version ≠ version) (F)
57:     if (owner==Trec_i) (F)
58:     if (owner==NOBODY) (T)

// In next step PT_x will execute SC of
// line 44 successfully

                                             60:          if(SC(ownerships[l],Trec)) (T)

                                                 -- End of AcquireOwnerships --
                                                        Caof(i,k)

59:     if (SC(Trec_i.status, (NULL,0)))(T)
```

Χρόνος Εκτέλεσης

Σχήμα 5.13: Σχηματική αναπαράσταση του «κακού» σεναρίου που περιγράφεται κατά την απόδειξη του Λήμματος 5.17.

Επομένως, βάσει της «κακής» εκτέλεσης είναι πράγματι πιθανή η ενημέρωση του $Trec_i$.status από κάποια executing PerformTrans της T(i,k), μέσω της εντολής SC της γραμμής 59, μετά την Caof(i,k) και πριν το τέλος της a_{ROs}.

Στη συνέχεια θα δειχθεί ότι μετά την Caof(i,k) δεν μπορεί να ενημερωθεί για δεύτερη φορά το $Trec_i$.status μέσω της εκτέλεσης της εντολής SC της γραμμής 59, με βάση το Πόρισμα 5.4, από κάποια executing PerformTrans της T(i,k). Υποθέτουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι επιτυγχάνουν $b > 1$ λειτουργίες SC μετά από την Caof(i,k) και πριν το τέλος της a_{ROs}, και έστω SC_1 και SC_2 οι δύο πρώτες από αυτές. Επίσης έστω PT_1 και PT_2 οι executing PerformTrans της T(i,k) που εκτελούν επιτυχημένα τις SC_1 και SC_2, αντίστοιχα. Για να μπορέσει μία executing PerformTrans της T(i,k) να εκτελέσει την εντολή SC της γραμμής 59 πρέπει για κάποιο *l*, $l \in Trec_i$.add, να διάβασε, μέσω της εντολής LL της γραμμής 55, ownerships[*l*] == nobody. Το ίδιο ισχύει και για τις PT_1 και PT_2. Όμως, αφού οι SC_1 και SC_2 εκτελούνται μετά την Caof(i,k) και πριν το τέλος της a_{ROs} προκύπτει από το Πόρισμα 5.15 ότι στην Caof(i,k) θα ισχύει ownerships[*l*] == $Trec_i$. Έτσι συμπεραίνουμε ότι οι PT_1 και PT_2 εκτελούν τη γραμμή 55, άρα και την εντολή LL της γραμμής 54 στο $Trec_i$.status, πριν από τη Caof(i,k). Στην συνέχεια, εξ ορισμού των PT_1 και PT_2, η PT_1 εκτελεί επιτυχώς την SC_1 μετά την Caof(i,k). Άρα, βάσει του ορισμού της εντολής LL/SC, η εκτέλεση της SC_2 από την PT_2, η οποία έπεται της PT_1, μετά την Caof(i,k) θα αποτύχει. Άτοπο. Επομένως το Λήμμα 5.17 ισχύει. ∎

Συμβολίζουμε με C_W την πρώτη καθολική κατάσταση (εάν υπάρχει) κατά την οποία έχει ολοκληρωθεί η εκτέλεση του W_{PT} από οποιαδήποτε executing PerformTrans της T(i,k). Σημειώνεται ότι η C_W έπεται της Caof(i,k) και προηγείται του τέλους της a_{ROs}.

Λήμμα 5.18: *i) Καμία executing PerformTrans της T(i,k) δεν μπορεί να κολλήσει επ' άπειρο στον βρόχο W_{PT} και έτσι η C_W είναι καλά ορισμένη, ii) Στην C_W το $Trec_i$.status θα έχει τιμή COMMITED ή ABORTED, iii) Στην C_W το $Trec_i$.status θα έχει λάβει τιμή COMMITED εάν και μόνο εάν στην Caof(i,k) ισχύει $Trec_i$.status == ACTIVE.*

Απόδειξη: Από το Λήμμα 5.9 προκύπτει ότι στην Caof(i,k) το $Trec_i$.status θα έχει τιμή ACTIVE ή ABORTED. Εάν η τιμή του είναι ABORTED, τότε από το Λήμμα 5.2 προκύπτει ότι δεν μπορεί να αλλάξει σε ACTIVE ή COMMITED πριν την Cf(i,k).

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Επομένως όσες από τις executing PerformTrans της T(i,k) εκτελέσουν τη συνθήκη της γραμμής 29 πριν την Cf(i,k), θα την αποτιμήσουν σε FALSE και έτσι δεν θα εκτελέσουν το σώμα του W_{PT}. Σημειώνεται ότι πριν την Cf(i,k) θα έχει εκτελεστεί η γραμμή 18 του κώδικα. Έτσι, όσες από τις executing PerformTrans της T(i,k) εκτελέσουν τη συνθήκη της γραμμής 29 μετά την Cf(i,k) θα αποχωρήσουν από την εκτέλεση της συνάρτησης PerformTrans, με βάση το Λήμμα 5.1. Έτσι όλες οι executing PerformTrans της T(i,k) θα περάσουν τον βρόχο W_{PT}, με την πρώτη από αυτές να ορίζει την C_W.

Από την άλλη εάν στην AOf(i,k) το $Trec_i$.status έχει τιμή ACTIVE τότε από το Πόρισμα 5.16 προκύπτει ότι καμία από τις executing PerformTrans της T(i,k) δεν πρόκειται να γράψει στο $Trec_i$.status την τιμή ABORTED μετά την Caof(i,k) και πριν το τέλος της α_{ROs}. Έτσι τουλάχιστον μία από τις executing PerformTrans της T(i,k) που ολοκληρώσαν την AcquireOwnerships θα εκτελέσει τον W_{PT} μετά την Caof(i,k), διότι θα ισχύει η συνθήκη εισόδου στον W_{PT} (γραμμή 29). Σημειώνεται ότι από το Πόρισμα 5.4 προκύπτει ότι στο E(T(i,k)) το $Trec_i$.status μπορεί να αλλάξει μόνο από τις executing PerformTrans της T(i,k). Επίσης σύμφωνα με το Λήμμα 5.17, είναι πιθανό να εκτελεστεί το πολύ μία φορά επιτυχώς η εντολή SC της γραμμής 59, από κάποια executing PerformTrans της T(i,k), μετά την Caof(i,k) και πριν το τέλος της α_{ROs}. Άρα μία executing PerformTrans της T(i,k) που εκτελεί τον W_{PT} μπορεί να αποτραπεί το πολύ μία φορά από την επιτυχή εκτέλεση της εντολής SC της γραμμής 31, λόγω της επιτυχής εκτέλεσης της γραμμής 59. Με βάση τα παραπάνω εάν στην Caof(i,k) το $Trec_i$.status έχει τιμή ACTIVE, τότε οι executing PerformTrans της T(i,k) θα εκτελέσουν το πολύ δύο φορές το σώμα του W_{PT} έως ότου μία από αυτές καταφέρει να γράψει στο $Trec_i$.status την τιμή COMMITED, οπότε και θα φύγουν από τον W_{PT}. Έτσι εάν στην Caof(i,k) ισχύει $Trec_i$.status == ACTIVE καμία executing PerformTrans της T(i,k) δεν μπορεί να κολλήσει επ' άπειρο στην εκτέλεση του βρόχου W_{PT}, η C_W ορίζεται και σε αυτήν το $Trec_i$.status θα έχει τιμή COMMITED.

Απομένει να αποδειχθεί ότι εάν στην C_W ισχύει $Trec_i$.status == COMMITED τότε στην Caof(i,k) ίσχυε $Trec_i$.status == ACTIVE. Σημειώνεται ότι το $Trec_i$.status μπορεί να πάρει την τιμή COMMITED μόνο εάν εκτελεστεί η εντολή SC που περιέχεται στον W_{PT} (γραμμή 16) και για την είσοδο στον W_{PT} απαιτείται να ισχύει $Trec_i$.status == ACTIVE (έλεγχος γραμμής 14). Σημειώνεται ότι εάν στην Caof(i,k) ίσχυε

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

$Trec_i$.status == ABORTED, τότε με βάση το Λήμμα 5.2 η τιμή αυτή δεν θα μπορούσε να αλλάξει πριν την Cf(i,k). Έτσι καμία από τις executing PerformTrans της T(i,k) δεν θα εκτελούσε το σώμα του W_{PT} (διότι η συνθήκη της γραμμής 29 δεν θα ίσχυε) πριν την C_W και στην C_W το $Trec_i$.status θα είχε τιμή ABORTED. Άρα για να ισχύει $Trec_i$.status == COMMITED στην C_W πρέπει να ισχύει $Trec_i$.status == ACTIVE στην Caof(i,k). ∎

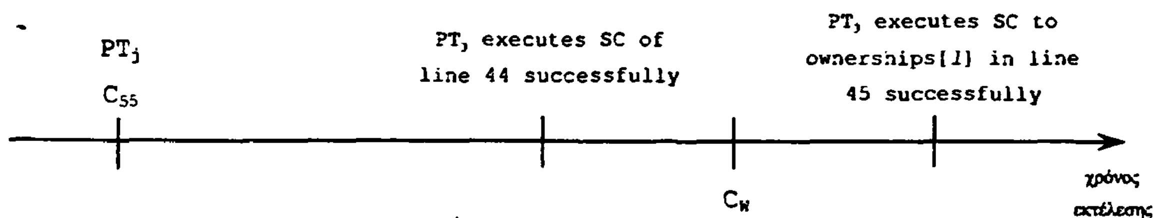
Από το Λήμμα 5.18 προκύπτει ότι στην C_W θα ισχύει Trec(i,k).status ≠ ACTIVE και με βάση το Λήμμα 5.2 η τιμή αυτή δεν πρόκειται να αλλάξει πριν το τέλος της εκτέλεσης της T(i,k). Επίσης η C_W προηγείται της πρώτης εκτέλεσης της γραμμής 34 από κάποια executing PerformTrans της T(i,k), επομένως ταυτίζεται με την καθολική κατάσταση Cst(i,k). Υπενθυμίζεται ότι η T(i,k) ονομάζεται *επιτυχημένη* εάν το Trec(i,k).status της έχει την τιμή COMMITED στην Cst(i,k) και *αποτυχημένη* εάν το Trec(i,k).status της έχει την τιμή ABORTED. Εάν η T(i,k) είναι επιτυχημένη, όπως έχει αναφερθεί, συμβολίζουμε με Cc(i,k) την καθολική κατάσταση στην οποία γράφτηκε για πρώτη φορά η τιμή COMMITED στο $Trec_i$.status από κάποια executing PerformTrans της T(i,k). Επίσης, εάν η T(i,k) είναι αποτυχημένη, όπως έχει αναφερθεί, ορίζεται η Ca(i,k) (στην οποία γράφτηκε για μία και μοναδική φορά η τιμή ABORTED στο $Trec_i$.status από κάποια executing PerformTrans της T(i,k)) και η FM(i,k) (η θέση του πίνακα ownerships που αυτή δεν κατάφερε να αποκτήσει). Σημειώνεται ότι η Cc(i,k) και η Ca(i,k) προηγούνται της C_W.

Από το Λήμμα 5.8 προκύπτει ότι η τιμή COMMITED γράφεται στο $Trec_i$.status το πολύ μία φορά από τις executing PerformTrans της T(i,k). Έτσι, εάν η T(i,k) είναι επιτυχημένη υπάρχει μία μόνο executing PerformTrans της T(i,k) που γράφει COMMITED στο $Trec_i$.status, συγκεκριμένα αμέσως πριν την καθολική κατάσταση Cc(i,k). Επίσης από το Λήμμα 5.18 προκύπτει ότι στην C_W το $Trec_i$.status θα έχει τιμή COMMITED ή ABORTED, έτσι από το Λήμμα 5.8 προκύπτει το παρακάτω πόρισμα.

Πόρισμα 5.19: *Καμία executing PerformTrans της T(i,k) δεν εκτελεί επιτυχώς την SC της γραμμής 59 ή της γραμμής 63 ή της γραμμής 31 μετά την C_W.*

Λήμμα 5.20: Καμία executing PerformTrans της T(i,k) δεν εκτελεί επιτυχώς την SC της γραμμής 60 μετά την C_W.

Απόδειξη: Ας υποθέσουμε, δια της εις άτοπο απαγωγής, ότι κάποια executing PerformTrans της T(i,k), έστω $PT_j(i,k)$, εκτελεί επιτυχώς για κάποιο *l*, $l \in Trec_i.add$, την εντολή SC της γραμμής 60, μετά την C_W. Η $PT_j(i,k)$ πρέπει να εκτέλεσε την εντολή SC της γραμμής 59 επιτυχώς και σύμφωνα με το Πόρισμα 5.19 αυτό σημαίνει ότι η εντολή SC της γραμμής 59 εκτελέστηκε πριν από την C_W. Αυτό συνεπάγεται ότι και οι λειτουργίες των γραμμών 54 και 55 εκτελέστηκαν πριν από την C_W. Επίσης για να μπορεί η $PT_j(i,k)$ να εκτελέσει επιτυχώς την εντολή SC της γραμμής 60 πρέπει να εκτέλεσε την εντολή LL της γραμμής 55 και να είδε ownerships[*l*] == nobody, έστω στην καθολική κατάσταση C_{55}. Το Σχήμα 5.14 αναπαριστά την σειρά εκτέλεσης των εντολών αυτών.



Σχήμα 5.14: Εκτέλεση που περιγράφεται κατά την απόδειξη του Λήμματος 5.20.

Από το Λήμμα 5.18 προκύπτει ότι στην C_W το $Trec_i.status$ μπορεί να έχει τιμή COMMITED ή ABORTED. Εάν στην C_W ισχύει $Trec_i.status$ == COMMITED τότε με βάση το Λήμμα 5.18 και το Πόρισμα 5.15 προκύπτει ότι έχουν αποκτηθεί όλα τα ownerships που επιθυμεί η T(i,k) πριν την C_W, άρα και το ownerhsips[*l*]. Επομένως κάποια executing PerformTrans της T(i,k) κατάφερε να εκτελέσει επιτυχώς την εντολή SC της γραμμής 60 για το ownerships[*l*] μετά την C_{55} και πριν την C_W και άρα βάσει του ορισμού της εντολής LL/SC η εκτέλεση της εντολής SC της γραμμής 60 από την $PT_j(i,k)$ μετά την C_W θα αποτύχει. Από την άλλη, εάν στην C_W ισχύει $Trec_i.status$ == ABORTED ορίζεται η Ca(i,k), η οποία εξ ορισμού της προηγείται της Caof(i,k) και άρα προηγείται και της C_W. Από το Λήμμα 5.12 και το Λήμμα 5.6 προκύπτει ότι δεν είναι δυνατό να εκτελεστεί επιτυχώς η εντολή SC της γραμμής 45 από την $PT_j(i,k)$ μετά την Ca(i,k) και άρα μετά την C_W. Επομένως σε κάθε περίπτωση

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

η εκτέλεση της γραμμής 60 από την $PT_j(i,k)$ μετά την C_W αποτυγχάνει. Άτοπο. Άρα το Λήμμα 5.20 ισχύει. ∎

Έστω α μια εκτέλεση του αλγορίθμου SSTM. Εάν ο πίνακας A_{OWN} είναι μη κενός, ορίζουμε a_h το διάστημα εκτέλεσης της α το οποίο ξεκινά με την $C_{AC(first)}$, όπου first=A_{OWN}[1], και καταλήγει στην καθολική κατάσταση που προηγείται της πρώτης κατάργησης της τελευταίας θέσης του πίνακα ownerships (εάν αυτό συμβαίνει), δηλαδή της μοναδικής που δεν έχει ήδη καταργηθεί για μία τουλάχιστον φορά, από τις executing PerformTrans της T(i,k). Εάν αυτό δεν συμβαίνει τότε το a_h είναι το επίθεμα της α που ξεκινά από την $C_{AC(first)}$. Σημειώνεται ότι σε κάθε καθολική κατάσταση του διαστήματος a_h η T(i,k) κατέχει μία τουλάχιστον θέση του ownerships, δηλαδή μία τουλάχιστον θέση του ownerships περιέχει το Trec(i,k). Επίσης πριν την έναρξη του a_h η T(i,k) δεν κατέχει καμία θέση του ownerships. Επειδή η $C_{AC(first)}$ προηγείται της AOf(i,k), σημαίνει ότι το a_h ξεκινά πριν την Caof(i,k).

Ορίζουμε $F_{Aown}(C)$, $C \in a_h$, μία συνάρτηση που περιγράφει το μη-κενό σύνολο των θέσεων του πίνακα ownerships που κατέχει η T(i,k) σε κάθε καθολική κατάσταση C του a_h. Σημειώνεται ότι το πεδίο τιμών της $F_{Aown}(C)$ είναι τα στοιχεία του δυναμοσυνόλου του συνόλου A_{OWN}. Από τον κώδικα (γραμμές 48, 46, 42, 33, 63 και 55) προκύπτει ότι για να μπορεί μια PerformTrans μιας δοσοληψίας $T(j,n_j)$, $j \neq i$, έστω $PT_z(j,n_j)$, να γίνει βοηθητική της T(i,k) πρέπει να επιθυμεί να αποκτήσει κάποια θέση l, $l \in Trec_j$.add, του ownerships που επιθυμεί και η T(i,k), δηλαδή να ισχύει και $l \in Trec_i$.add, και το ownerships[l] να έχει ήδη αποκτηθεί από την T(i,k). Δηλαδή η $PT_z(j,n_j)$ πρέπει να έχει δει στο ownerships[l] το Trec(i,k). Με βάση τα παραπάνω προκύπτει ότι κάτι τέτοιο είναι πιθανό σε όλες τις καθολικές καταστάσεις C του a_h εάν η $T(j,n_j)$ επιθυμεί να αποκτήσει κάποια θέση του ownerships που περιέχεται στο σύνολο $F_{Aown}(C)$.

Έτσι προκύπτει ότι μία PerformTrans μιας δοσοληψίας $T(j,n_j)$, $j \neq i$, μπορεί να γίνει βοηθητική της T(i,k) εάν σε κάποια καθολική κατάσταση C, $C \in a_h$, προσπαθήσει να αποκτήσει κάποια θέση του ownerships που περιέχεται στο σύνολο $F_{Aown}(C)$. Επίσης, προκύπτει ότι μια δοσοληψία $T(z,n_z)$, η οποία έχει εκκινηθεί από μια διεργασία p_z

που έχει καταρρεύσει, δεν μπορεί να βοηθηθεί από άλλες δοσοληψίες ενεργών διεργασιών εάν η p_z έσφαλε πριν προλάβει να γίνει η έναρξη του διαστήματος a_h, δηλαδή πριν η p_z αποκτήσει το πρώτο ownerships που η $T(z,n_z)$ επιθυμεί.

Με βάση το Πόρισμα 5.15 και τον Ισχυρισμό iii) του Λήμματος 5.18 προκύπτει ότι το a_h ορίζεται εάν η T(i,k) είναι επιτυχημένη. Έτσι, με βάση τα παραπάνω, προκύπτει ότι εάν η T(i,k) είναι επιτυχημένη και υπάρχει μία τουλάχιστον ενεργή διεργασία στο σύστημα που εκτελεί μια δοσοληψία PT, η οποία σε κάποια καθολική κατάσταση C του a_h προσπαθεί να αποκτήσει κάποια θέση του ownerships που περιέχεται στο σύνολο $F_{Aown}(C)$, δηλαδή που κατέχει ήδη η T(i,k), τότε η T(i,k) μπορεί να αποκτήσει μία τουλάχιστον βοηθητική δοσοληψία ανεξάρτητα από τα σφάλματα των διεργασιών.

Στη συνέχεια της απόδειξης θεωρούμε ότι εάν η T(i,k) είναι επιτυχημένη θα υπάρχει τουλάχιστον μία ενεργή PerformTrans, έστω PT_j, που επιθυμεί να αποκτήσει κάποιο ownerships που κατέχει η T(i,k) και έτσι η PT_j θα γίνει βοηθητική της T(i,k). Έτσι προκύπτει το παρακάτω πόρισμα.

***Πόρισμα 5.21**: Εάν η T(i,k) είναι επιτυχημένη, τότε έχει μία τουλάχιστον ενεργή βοηθητική δοσοληψία στο a_h, ανεξάρτητα από τις αποτυχίες των διεργασιών.*

***Λήμμα 5.22**: Υπάρχει μία τουλάχιστον executing PerformTrans της T(i,k) που εκτελεί την συνάρτηση ReleaseOwnerships μετά την Caof(i,k) και πριν την Cf(i,k).*

Απόδειξη: Από το Πόρισμα 5.21 προκύπτει ότι η T(i,k) έχει μία τουλάχιστον ενεργή βοηθητική δοσοληψία στο a_h, έστω $PT_j(i,k)$, ανεξάρτητα από τις αποτυχίες των διεργασιών. Όπως αναφέρθηκε το a_h ξεκινά πριν την Caof(i,k), άρα στην Caof(i,k) η $PT_j(i,k)$ είναι ήδη βοηθητική δοσοληψία της T(i,k). Από το Λήμμα 5.18 προκύπτει ότι η καθολική κατάσταση C_W είναι καλά ορισμένη για την T(i,k), που σημαίνει ότι τουλάχιστον η $PT_j(i,k)$, από τις executing PerformTrans της T(i,k), θα περάσει τον βρόχο W_{PT}, μετά την Caof(i,k). Επίσης, από τον κώδικα (γραμμές 38 και 40) προκύπτει ότι ανεξάρτητα με την τιμή που έχει το $Trec_i$.status στην C_W η

ReleaseOwnerships θα εκτελεστεί τουλάχιστον από την $PT_j(i,k)$, μετά την Caof(i,k) και πριν την Cf(i,k). Άρα το Λήμμα 5.22 ισχύει. ■

Λήμμα 5.23: Έστω κάποιο l, $l \in Trec_i.add$. Εάν ορίζεται η καθολική κατάσταση $C_{AC}(l)$ τότε υπάρχει μία τουλάχιστον executing PerformTrans της T(i,k) που καταργεί το ownerships[l] μετά την Caof(i,k) και πριν την Cf(i,k).

Απόδειξη: Εφόσον ορίζεται η $C_{AC(l)}$, ορίζεται και η α_l. Από το Λήμμα 5.6 προκύπτει ότι σε όλες τις καθολικές καταστάσεις της α_l ισχύει ownerships[*l*] == $Trec_i$. Από το Λήμμα 5.22 προκύπτει ότι τουλάχιστον μία executing PerformTrans της T(i,k) θα εκτελέσει τη συνάρτηση ReleaseOwnerships μετά την Caof(i,k) και πριν την CF(i,k). Επίσης από το Πόρισμα 5.21 προκύπτει ότι η T(i,k) έχει μία τουλάχιστον ενεργή βοηθητική δοσοληψία στο α_h, έστω $PT_z(i,k)$, ανεξάρτητα από τις αποτυχίες των διεργασιών. Σημειώνεται ότι από τον ορισμό του α_h προκύπτει ότι το α_l εμπεριέχεται στο α_h.

Έστω Α το σύνολο των executing PerformTrans της T(i,k) που εκτέλεσαν τον έλεγχο ownerships[*l*] == $Trec_i$ της γραμμής 69 στην α_l. Εξ ορισμού του α_l και από το Λήμμα 5.6 ισχύει ότι ownerships[*l*] == $Trec_i$ όταν εκτελείται ο έλεγχος της γραμμής 54 από τις executing PerformTrans του Α και άρα ο έλεγχος αυτός δεν μπορεί να αποτύχει. Έστω $PT_j(i,k)$ (μπορεί να ισχύει j==z) εκείνη η PerformTrans του Α που κάλεσε πρώτη την εντολή SC της γραμμής 71 για το ownerships[*l*], έστω SC_1 η εντολή αυτή και έστω C η καθολική κατάσταση στην οποία εκτελείται. Σημειώνεται ότι ακόμη δεν έχει ολοκληρωθεί το α_l, επομένως και πάλι με βάση το Λήμμα 5.6 ισχύει ότι ownerships[*l*] == $Trec_i$ όταν εκτελείται η SC_1 από την $PT_j(i,k)$ και έτσι η εκτέλεσή της στην C θα είναι επιτυχής. Επομένως το ownerships[*l*] θα καταργηθεί για πρώτη φορά στην C. Άρα το Λήμμα 5.23 ισχύει. ■

Το Λήμμα 5.23 μας επιτρέπει να συμβολίσουμε με $C_{RL(l)}$, $l \in Trec_i.add$, την καθολική κατάσταση στην οποία καταργείται για πρώτη φορά το ownerships[*l*], εάν αυτό συμβαίνει. Επίσης από το Λήμμα 5.23 προκύπτει ότι εάν ορίζεται η $C_{AC(l)}$ τότε ορίζεται και η $C_{RL(l)}$, η οποία έπεται της $C_{AC(l)}$. Όμως, από το Πόρισμα 5.14 προκύπτει ότι για κάθε *l*, $l \in A_{OWN}$, ορίζεται η καθολική κατάσταση $C_{AC(l)}$ η οποία προηγείται της Caof(i,k), καθώς επίσης ορίζεται και το διάστημα εκτέλεσης α_l. Επομένως οι

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

καθολικές καταστάσεις $C_{RL(l)}$ ορίζονται για κάθε *l*, $l \in A_{OWN}$, και το διάστημα εκτέλεσης α_l ξεκινά από την $C_{AC(l)}$ και καταλήγει στην $C_{RL(l)}$. Επίσης με βάση το Λήμμα 5.22 συμβολίζουμε με C_{RO}, την καθολική κατάσταση που προηγείται της πρώτης $C_{RL(l)}$, δηλαδή την καθολική κατάσταση αμέσως πριν την πρώτη κατάργηση οποιασδήποτε θέσης *l* του πίνακα ownerships. Σημειώνεται ότι η C_{RO}, έπεται της Caof(i,k) και όλες οι καθολικές καταστάσεις $C_{RL(l)}$, $l \in Trec_i$.add, έπονται της C_{RO}.

Λήμμα 5.24: Έστω οποιοδήποτε l, $l \in$ Treci.add. Εάν ορίζεται η καθολική κατάσταση $C_{AC}(l)$ τότε δεν υπάρχει καθολική κατάσταση μετά την $C_{RL}(l)$, στην οποία το ownerships[l] να έχει τιμή Trec(i,k).

Απόδειξη: Από το Λήμμα 5.23 προκύπτει ότι στην $C_{RL(l)}$ καταργήθηκε για πρώτη φορά το ownerships[*l*] από κάποια executing PerformTrans της T(i,k). Σημειώνεται ότι το Trec(i,k) μπορεί να γραφεί στον πίνακα ownerships μόνο από τις executing PerformTrans της T(i,k). Ας υποθέσουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι κάποια executing PerformTrans της T(i,k) εκτελεί επιτυχώς την εντολή SC της γραμμή 60 γράφοντας και πάλι το Trec(i,k) στο ownerships[*l*], μετά την $C_{RL(l)}$. Από το Λήμμα 5.20 προκύπτει ότι καμία από τις executing PerformTrans της T(i,k) δεν γράφει το $Trec_i$ στο ownerships[*l*] μετά την C_W. Από τον κώδικα προκύπτει ότι η C_W προηγείται της $C_{RL(l)}$ και έτσι καταλήγουμε σε άτοπο. Άρα το Λήμμα 5.24 ισχύει. ■

Λήμμα 5.25: Έστω οποιοδήποτε l, $l \in Trec_i$.add. Εάν ορίζεται η καθολική κατάσταση $C_{AC}(l)$ τότε καμία βοηθητική PerformTrans της T(i,k) δεν εκτελεί επιτυχώς την εντολή SC της γραμμής 71 μετά την $C_{RL}(l)$.

Απόδειξη: Ας υποθέσουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι κάποια executing PerformTrans της T(i,k), έστω PT_j(i,k), εκτελεί επιτυχώς την εντολή SC της γραμμής 71 για το ownerships[*l*], μετά την $C_{RL(l)}$. Η PT_j(i,k) πρέπει να εκτέλεσε την εντολή LL της γραμμής 69 για το *l* βλέποντας ownerships[*l*] == Trec(i,k). Από το Λήμμα 5.24 προκύπτει ότι στην $C_{RL(l)}$ και μετά από αυτή θα ισχύει ownerships[*l*] $\neq$ Trec(i,k). Έτσι εάν η γραμμή 69 εκτελέστηκε από την PT_j(i,k) μετά την $C_{RL(l)}$ τότε η PT_j(i,k) θα μπορούσε να δει ownerships[*l*] == $Trec_i$, όμως με $Trec_i$.version > k. Τότε

σύμφωνα με το Λήμμα 5.1, η εκτέλεση του VerCheck της γραμμής 70 θα επιτύγχανε και έτσι η γραμμή 71 δεν θα εκτελούνταν από την $PT_j(i,k)$.

Επομένως η $PT_j(i,k)$ πρέπει να εκτέλεσε την γραμμή 69 πριν την $C_{RL(l)}$ και στην συνέχεια να εκτέλεσε την γραμμή 71 μετά την $C_{RL(l)}$. Όμως στην $C_{RL(l)}$, όπως προαναφέρθηκε, ισχύει ownerships[*l*] ≠ Trec(i,k), που σημαίνει ότι κάποια executing PerformTrans της T(i,k) κατάφερε να εκτελέσει την εντολή SC της γραμμής 71 για το ownerships[*l*] επιτυχώς. Έτσι με βάση τον ορισμό της εντολής LL/SC η εκτέλεση της εντολής SC της γραμμής 71 από την $PT_j(i,k)$ μετά την $C_{RL(l)}$ θα αποτύχει. Άτοπο, έτσι το Λήμμα 5.25 ισχύει. ■

Το Λήμμα 5.25 αποδεικνύει ότι για κάθε *l*, $l \in A_{OWN}$, υπάρχει μία μόνο executing PerformTrans που εκτελεί επιτυχώς την εντολή SC της γραμμής 71 για το ownerships[*l*], συγκεκριμένα αμέσως πριν την καθολική κατάσταση $C_{RL(l)}$. Όπως έχει αναφερθεί, συμβολίζουμε ως *Crof(i,k)* την καθολική κατάσταση στην οποία ολοκληρώνεται για πρώτη φορά μια κλήση της ReleaseOwnerships από κάποια executing PerformTrans της T(i,k). Σημειώνεται ότι η Crof(i,k) έπεται της C_{ROs}, έπεται των $C_{RL(l)}$, $l \in$ $l \in A_{OWN}$, και προηγείται της Cf(i,k).

***Λήμμα 5.26**: Μία τουλάχιστον από τις executing PerformTrans της T(i,k) θα εκτελέσει τις γραμμές 35 έως 38 του κώδικα πριν την Cf(i,k), εάν και μόνο εάν η T(i,k) είναι επιτυχημένη.*

Απόδειξη: Αρχικά αποδεικνύεται ότι εάν η T(i,k) είναι επιτυχημένη τότε μία τουλάχιστον από τις executing PerformTrans της, θα εκτελέσει τις γραμμές 35 έως 38. Σημειώνεται ότι σύμφωνα με το Λήμμα 5.18 το $Trec_i$.status θα λάβει τιμή COMMITED ή ABORTED στην C_W και η τιμή αυτή, σύμφωνα με το Λήμμα 5.2, δεν μπορεί να αλλάξει πριν την Cf(i,k). Επίσης η C_W προηγείται του ελέγχου της γραμμής 34. Έτσι προκύπτει ότι όσες από τις executing PerformTrans της T(i,k) εκτελέσουν τον έλεγχο της γραμμής 34 πριν την Cf(i,k), θα τον περάσουν επιτυχώς. Σημειώνεται ότι μία από αυτές τις PerformTrans θα είναι αυτή που εκτελεί η p_i, δηλαδή ο δημιουργός της $T(i,n_i)$, η οποία είναι και αυτή που ορίζει την καθολική κατάσταση Cf(i,k). Οπότε για να είναι δυνατό να οριστεί η Cf(i,k), πρέπει τουλάχιστον μία από αυτές να ολοκληρώσει την εκτέλεση των γραμμών 35 έως 38.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Στη συνέχεια αποδεικνύεται ότι θα υπάρχει μία τουλάχιστον τέτοια executing PerformTrans της T(i,k) ακόμα και εάν οι διεργασίες μπορούν να αποτύχουν, δηλαδή σταματήσουν ξαφνικά να εκτελούνται. Εφόσον η T(i,k) είναι επιτυχημένη από το Λήμμα 5.13 και το Λήμμα 5.18 προκύπτει ότι έχει αποκτήσει πριν από την Caof(i,k) όλα τα ownerships που επιθυμεί, επομένως και το $C_{AC(first)}$, first=A_{OWN}[1], και έτσι ορίζεται το α_h το οποίο ξεκινά πριν την Caof(i,k). Επίσης επειδή η T(i,k) είναι επιτυχημένη, το $Trec_i$.status της έχει τιμή COMMITED και η τιμή αυτή δεν μπορεί να αλλάξει πριν την Cf(i,k). Έτσι καμία executing PerformTrans της T(i,k) δεν μπορεί να εκτελέσει τη συνάρτηση ReleaseOwnerships μέσω της γραμμής 40 πριν την Cf(i,k). Άρα για κάθε *l*, $l \in A_{OWN}$, η $C_{RL(l)}$ θα οριστεί λόγω της εκτέλεσης της ReleaseOwnerships της γραμμής 38. Επομένως το α_h δεν μπορεί να ολοκληρωθεί πριν την κατάργηση του τελευταίου ownerships που η T(i,k) κατέχει, μέσω της εκτέλεσης της ReleaseOwnerships της γραμμής 38. Έτσι από το Πόρισμα 5.21 προκύπτει ότι πριν την ολοκλήρωση της εκτέλεσης της γραμμής 38 από μία τουλάχιστον PerformTrans, η T(i,k) έχει μία τουλάχιστον ενεργή βοηθητική δοσοληψία ανεξάρτητα από τις αποτυχίες των διεργασιών.

Για την απόδειξη του αντίστροφου αρκεί να αναφερθεί ότι για να μπορεί μια οποιαδήποτε executing PerformTrans της T(i,k) να εκτελέσει τις γραμμές 35 έως 38 πρέπει να έχει περάσει επιτυχώς τον έλεγχο της γραμμής 34, δηλαδή πρέπει η T(i,k) να είναι επιτυχημένη. Άρα το Λήμμα 5.26 ισχύει. ∎

Λήμμα 5.27: Εάν η T(i,k) είναι επιτυχημένη τότε i) η SC της γραμμής 78 θα εκτελεστεί ακριβώς μία φορά για κάθε $Trec_i$.OldValues[l], $l \in Trec_i$.add, από τις executing δοσοληψίες της T(i,k) πριν την Caov(i,k) και ii) η SC της γραμμής 90 θα εκτελεστεί ακριβώς μία φορά, από κάποια executing PerformTrans της T(i,k) πριν την Cf(i,k).

Απόδειξη: Από το Λήμμα 5.26 προκύπτει ότι μία τουλάχιστον από τις executing PerformTrans της T(i,k) θα εκτελέσει τις συναρτήσεις AgreeOldValues (γραμμή 35 του κώδικα) και UpdateMemory (γραμμή 37 του κώδικα) πριν την Cf(i,k).

i) Έστω A το σύνολο των executing PerformTrans της T(i,k) που κάλεσαν την AgreeOldValues και εκτέλεσαν τον έλεγχο $Trec_i$.OldValues == (null,0) της γραμμής 76 για το $Trec_i$.OldValues[*l*] πριν την πρώτη εκτέλεση της εντολής SC της γραμμής 78 για το $Trec_i$.OldValues[*l*] από τις executing PerformTrans της T(i,k).

Συμβολίζουμε ως $SC_{78(l)}$ την εντολή SC της γραμμής 78 για το $Trec_i$.OldValues[l]. Καμία executing PerformTrans της T(i,k) δεν έχει εκτελέσει ακόμα την $SC_{78(l)}$ και κατά την εκκίνηση της T(i,k) ισχύει $Trec_i$.OldValues[l] == (null,0). Επίσης, από το Πόρισμα 5.4 προκύπτει ότι στο E(T(i,k)) μόνο οι executing PerformTrans της T(i,k) μπορούν να τροποποιήσουν τον διαμοιραζόμενο πίνακα $Trec_i$.OldValues. Επομένως η εκτέλεση του ελέγχου της γραμμής 76 από τις PerformTrans του A για το $Trec_i$.OldValues[l] θα είναι επιτυχής. Έστω $PT_j(i,k)$ εκείνη η PerformTrans του A που κάλεσε πρώτη την $SC_{78(l)}$ για το $Trec_i$.OldValues[l], έστω στην καθολική κατάσταση C_{OV}. Εξ ορισμού της C_{OV}, η εκτέλεση της $SC_{78(l)}$ από την $PT_j(i,k)$ στην C_{OV} θα είναι επιτυχής και έτσι θα γραφεί η τιμή (memory[l],1) στο $Trec_i$.OldValues[l]. Σημειώνεται ότι η C_{OV} προηγείται της Caov(i,k).

Στην συνέχεια αποδεικνύεται ότι η ενημέρωση αυτή είναι μοναδική. Ας υποθέσουμε δια της μεθόδου της εις άτοπο απαγωγής ότι υπάρχουν κάποιες executing PerformTrans της T(i,k) οι οποίες εκτελούν επιτυχώς την $SC_{78(l)}$ για το $Trec_i$.OldValues[l] μετά την C_{OV}, και έστω $PT_z(i,k)$ η πρώτη από αυτές. Για να μπορεί η $PT_z(i,k)$ να εκτελέσει την $SC_{78(l)}$ πρέπει να διάβασε, μέσω της εντολής LL της γραμμής 76, $Trec_i$.OldValues[l] == (null,0). Σημειώνεται ότι στην C_{OV} ισχύει $Trec_i$.OldValues[l] == (memory[l],1).

Εάν η γραμμή 76 εκτελέστηκε από την $PT_z(i,k)$ μετά την C_{OV} τότε για να δει η PT_z $Trec_i$.OldValues[l] == (null,0) πρέπει, να εκτελέστηκε η Initiate από την p_i, κάτι που σημαίνει ότι η εκτέλεση της T(i,k) έχει ολοκληρωθεί και άρα έχει εκτελεστεί η γραμμή 18 του κώδικα από την T(i,k). Όμως τότε σύμφωνα με το Λήμμα 5.1, η εκτέλεση του VerCheck της γραμμής 77 θα επιτύγχανε και έτσι η γραμμή 78 δεν θα εκτελούνταν από την $PT_z(i,k)$. Άρα η $PT_z(i,k)$ πρέπει να εκτέλεσε την γραμμή 76 πριν την C_{OV}. Στην συνέχεια η $PT_z(i,k)$ εκτελεί την $SC_{78(l)}$ μετά την C_{OV}, έτσι με βάση τον ορισμό της εντολής LL/SC αυτή η SC θα αποτύχει. Άτοπο.

ii) Έστω A το σύνολο των executing PerformTrans της T(i,k) που κάλεσαν την UpdateMemory και συγκεκριμένα εκτέλεσαν τον έλεγχο $Trec_i$.AllWritten == FALSE της γραμμής 88 πριν την πρώτη εκτέλεση της εντολής SC της γραμμής 90 από τις executing PerformTrans της T(i,k). Συμβολίζουμε ως SC_{90} την εντολή SC της γραμμής 90. Καμία executing PerformTrans της T(i,k) δεν έχει εκτελέσει ακόμα την SC_{90} και κατά την εκκίνηση της T(i,k) ισχύει $Trec_i$.AllWritten == false. Επίσης, από το Πόρισμα 5.4 προκύπτει ότι στο E(T(i,k)) μόνο οι executing PerformTrans της

T(i,k) μπορούν να τροποποιήσουν την διαμοιραζόμενη μεταβλητή $Trec_i$.AllWritten. Επομένως η εκτέλεση του ελέγχου της γραμμής 88 από τις PerformTrans του A θα είναι επιτυχής. Έστω $PT_j(i,k)$ εκείνη η PerformTrans του A που κάλεσε πρώτη την SC_{90}, έστω στην καθολική κατάσταση C_{AW}. Εξ ορισμού της C_{AW}, η εκτέλεση της SC_{90} από την $PT_j(i,k)$ στην C_{AW} θα είναι επιτυχής και έτσι θα γραφεί η τιμή true στο $Trec_i$.AllWritten.

Στην συνέχεια αποδεικνύεται ότι η ενημέρωση αυτή είναι μοναδική. Ας υποθέσουμε δια της μεθόδου της εις άτοπο απαγωγής ότι υπάρχει μία ακόμη executing PerformTrans της T(i,k), έστω $PT_z(i,k)$, η οποία εκτελεί επιτυχώς την SC_{90} μετά την C_{AW}. Για να μπορεί η $PT_z(i,k)$ να εκτελέσει την SC_{90}, πρέπει να διάβασε, μέσω της εντολής LL της γραμμής 88, $Trec_i$.AllWritten == FALSE. Σημειώνεται ότι στην C_{AW} ισχύει $Trec_i$.AllWritten == TRUE.

Εάν η γραμμή 88 εκτελέστηκε από την $PT_z(i,k)$ μετά την C_{AW} τότε για να δει η $PT_z(i,k)$ $Trec_i$.AllWritten == false πρέπει, να εκτελέστηκε η Initiate από την p_i, κάτι που σημαίνει ότι η εκτέλεση της T(i,k) έχει ολοκληρωθεί και άρα έχει εκτελεστεί η γραμμή 6 του κώδικα από την T(i,k). Όμως τότε σύμφωνα με το Λήμμα 5.1, η εκτέλεση του VerCheck της γραμμής 89 θα επιτύγχανε και έτσι η γραμμή 90 δεν θα εκτελούνταν από την $PT_z(i,k)$. Άρα η $PT_z(i,k)$ πρέπει να εκτέλεσε την γραμμή 88 πριν την C_{AW}. Στην συνέχεια η $PT_z(i,k)$ εκτελεί την SC_{90} μετά την C_{AW}, έτσι με βάση τον ορισμό της εντολής LL/SC αυτή η SC θα αποτύχει. Άτοπο.

Επομένως το Λήμμα 5.27 ισχύει. ■

Εάν η T(i,k) είναι επιτυχημένη, τότε με βάση το Λήμμα 5.27, όπως έχει προαναφερθεί, συμβολίζουμε ως *Caw*(i,k) την καθολική κατάσταση στην οποία γράφεται μία και μοναδική φορά η τιμή TRUE στο $Trec_i$.AllWritten από τις executing PerformTrans της T(i,k).

Στην συνέχεια παρουσιάζουμε τον τρόπο με τον οποίο προκύπτουν οι Ισχυρισμοί του Λήμματος με βάση τα παραπάνω. Από τον ορισμό της Caw(i,k) ισχύει άμεσα ο Ισχυρισμός v)a. του Λήμματος. Ακόμη, ισχύουν οι εξής Ισχυρισμοί του λήμματος: ο Ισχυρισμός i) βάσει του Λήμματος 5.10, οι Ισχυρισμοί ii) και iii) βάσει του Λήμματος 5.18, ο Ισχυρισμός vi) βάσει του Λήμματος 5.24, ο Ισχυρισμός iv) λόγω του

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

Λήμματος 5.26 και ο Ισχυρισμός v)d. λόγω του Λήμματος 5.27 (σημειώνεται ότι η Caov(i,k) προηγείται της Caw(i,k)).

Η Caw(i,k) ορίζεται μόνο εάν η T(i,k) είναι επιτυχημένη. Από το Λήμμα 5.26 προκύπτει ότι εάν και μόνο εάν η T(i,k) είναι επιτυχημένη τότε μία τουλάχιστον από τις executing PerformTrans της T(i,k) θα εκτελέσει τις γραμμές 35 έως 38, πριν την Cf(i,k). Επίσης εάν η T(i,k) είναι επιτυχημένη σημαίνει ότι στην C_W το $Trec_i$.status έχει τιμή success και σύμφωνα με το Λήμμα 5.2 η τιμή αυτή δεν μπορεί να αλλάξει πριν την Cf(i,k). Έτσι καμία executing PerformTrans της T(i,k) δεν μπορεί να εκτελέσει την ReleaseOwnerships μέσω της γραμμής 40 πριν την Cf(i,k). Επομένως από τον κώδικα προκύπτει ότι όλες οι Cov(i,k,l), για κάθε $l \in Trec_i$.add, προηγούνται της Caov(i,k), η Caov(i,k) προηγείται της Caw(i,k) και η Caw(i,k) προηγείται της C_{ROs}. Επίσης, όπως έχει προαναφερθεί, η Cc(i,k) έπεται της Caof(i,k). Έτσι με βάση το μέρος iii) του Λήμματος 5.18 και το Πόρισμα 5.15 προκύπτει ότι ισχύει ο Ισχυρισμός v)b. του Λήμματος.

Από το Λήμμα 5.27 προκύπτει ότι όταν η T(i,k) είναι επιτυχημένη η γραμμή 78 δεν μπορεί να εκτελεστεί επιτυχώς μετά την Caov(i,k) και η γραμμή 90 δεν μπορεί να εκτελεστεί επιτυχώς μετά από την Caw(i,k). Σημειώνεται ότι η Caov(i,k) και η Caw(i,k) προηγούνται της Cf(i,k). Έτσι προκύπτει άμεσα ότι καμία βοηθητική PerformTrans της T(i,k) δεν εκτελεί επιτυχώς την SC της γραμμής 78 ή της 90 μετά την Cf(i,k). Σημειώνεται ότι η C_W και όλες οι $C_{RL(l)}$, $l \in A_{OWN}$, προηγούνται της Cf(i,k). Έτσι, με βάση τα παραπάνω και από το Πόρισμα 5.19, το Λήμμα 5.20 και το Λήμμα 5.25 προκύπτει ότι καμία βοηθητική PerformTrans της T(i,k) δεν εκτελεί κάποια επιτυχημένη SC στον πίνακα ownerhsips και σε οποιαδήποτε διαμοιραζόμενη δομή δεδομένων ή μεταβλητή που περιέχεται στην $Trec_i$, μετά την Cf(i,k). Προσθέτοντας το Πόρισμα 5.4 και το Πόρισμα 5.5 προκύπτει ότι ισχύει ο Ισχυρισμός vii) του Λήμματος.

Ακόμη, με βάση το Λήμμα 5.13, το Λήμμα 5.18 και τον Ισχυρισμό v)b. του Λήμματος που αποδείχθηκε παραπάνω, προκύπτει ότι ισχύει ο Ισχυρισμός v)c. του Λήμματος. Άρα ισχύουν όλοι οι Ισχυρισμοί του Λήμματος 5.3. ∎

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Στο σημείο αυτό ολοκληρώθηκε το πρώτο βήμα της απόδειξης και συνεχίζουμε με το δεύτερο βήμα της απόδειξης.

Λήμμα 5.28: Για κάθε process index $j \neq i$, $1 \leq j \leq n$, υπάρχει το πολύ μια PerformTrans που εκτελείται από την p_j, η οποία είναι βοηθητική της $T(i,n_i)$.

Απόδειξη: Έστω, δια της μεθόδου της εις άτοπο απαγωγής, ότι για κάποια διεργασία p_j υπάρχουν δύο PerformTrans που εκκινήθηκαν από την p_j και είναι βοηθητικές της $T(i,n_i)$. Έστω ότι η πρώτη από αυτές εκτελείται από την δοσοληψία $T(j,n_j)$ και η δεύτερη από την $T(j,n_j')$. Αν $n_j = n_j'$ σημαίνει ότι η ίδια δοσοληψία βοήθησε δύο φορές την $T(i,n_i)$, δηλαδή εκτέλεσε δύο φορές τη γραμμή 48 του κώδικα. Όμως αυτό δεν μπορεί να γίνει διότι την πρώτη φορά που η $T(j,n_j)$ θα βοηθήσει την $T(i,n_i)$ θα εκτελέσει τη γραμμή 48 με όρισμα IsInitiator == false, έτσι στη συνέχεια δεν μπορεί να βοηθήσει και πάλι την $T(i,n_i)$, διότι η εκτέλεση του ελέγχου της γραμμής 41 θα είναι μη επιτυχής. Άτοπο.

Έτσι πρέπει να ισχύει $n_j \neq n_j'$. Έστω, χωρίς βλάβη της γενικότητας, ότι $n_j' < n_j$ δηλαδή η $T(j,n_j)$ εκκινήθηκε από την p_j, μετά το τέλος της $T(j,n_j')$. Αφού οι $T(j,n_j')$, $T(j,n_j)$ βοηθούν την $T(i,n_i)$ καλούν στη γραμμή 48 του κώδικα την συνάρτηση PerformTrans με ορίσματα ($Trec_i$, n_i, false). Έστω $PT_k(j,n_j')$ και $PT_z(j,n_j)$ οι κλήσεις αυτές. Παρατηρούμε ότι η $PT_k(j,n_j')$ τερματίζει, αφού στη συνέχεια εκτελείται η $PT_z(j,n_j)$ από την p_j. Έτσι από τον κώδικα προκύπτει ότι η $PT_k(j,n_j')$ πριν την επιστροφή της, καλεί την ReleaseOwnerships για να καταργήσει τα ownerships που κατέχει η $T(i,n_i)$ και έστω C η καθολική κατάσταση στην οποία η κλήση της συνάρτησης αυτής επιστρέφει. Εξ ορισμού της $Crof(i,n_i)$ η C έπεται ή είναι η $Crof(i,n_i)$. Από τον Ισχυρισμό vi) του Λήμματος 5.3 προκύπτει ότι σε οποιαδήποτε καθολική κατάσταση μετά την $Crof(i,n_i)$, ο πίνακας ownerships δεν περιέχει σε καμία εγγραφή του το $Trec_i$ με $Trec_i.version \leq n_i$. Αφού η $T(j,n_j)$ κλήθηκε από την p_j μετά την C, η $T(j,n_j)$ δεν μπορεί να δει σε κάποια εγγραφή του πίνακα ownerships την τιμή $Trec(i,n_i)$. Έτσι η $T(j,n_j)$ δεν μπορεί να καλέσει στη γραμμή 48 του κώδικα την συνάρτηση PerformTrans με ορίσματα ($Trec_i$, n_i, false) και άρα δεν βοηθάει την $T(i,n_i)$. Άτοπο, άρα το Λήμμα 5.28 ισχύει. ■

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Λήμμα 5.29: Έστω $T(i,n_i)$ μια επιτυχημένη δοσοληψία. Οι παρακάτω ιδιότητες ισχύουν:

i) Έστω οποιοδήποτε l, $l \in Trec_i.add$. Μόνο οι executing PerformTrans της $T(i,n_i)$ μπορούν να τροποποιήσουν το memory[l] μεταξύ της $Cc(i,n_i)$ και της $Caw(i,n_i)$.

ii) Μετά την $Caov(i,n_i)$ και πριν την $Cf(i,n_i)$ ο $Trec_i.OldValues$ θα περιέχει τις τιμές που είχαν οι αντίστοιχες θέσεις (όπως περιγράφονται από τον πίνακα $rec_i.add$) του πίνακα memory, στο σημείο $Cc(i,n_i)$.

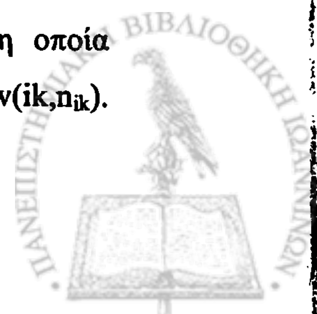
iii) Οι executing PerformTrans της $T(i,n_i)$ που έχουν την δυνατότητα να ενημερώσουν τον πίνακα memory συμφωνούν στις ίδιες τιμές του πίνακα newValues.

iv) Έστω οποιοδήποτε l, $l \in Trec_i.add$. Η SC της γραμμής 87 για το memory[l] θα εκτελεστεί το πολύ μία φορά από τις executing δοσοληψίες της $T(i,n_i)$ και αυτό μπορεί να συμβεί μετά την $Cc(i,n_i)$ και πριν την $Caw(i,n_i)$.

Απόδειξη: Έστω α μια εκτέλεση του αλγορίθμου SSTM και έστω *l* μια οποιαδήποτε θέση του πίνακα ownerships. Εστιάζουμε στις επιτυχημένες δοσοληψίες της α που εκτελούνται από executing PerformTrans δοσοληψιών που επιθυμούν να αποκτήσουν το ownerships[*l*] και να ενημερώσουν το memory[*l*]. Έστω $T(i1,n_{i1})$, $T(i2,n_{i2})$, ... , $T(ik,n_{ik})$ οι δοσοληψίες αυτές που αποκτούν το ownerships[*l*] με βάση την συγκεκριμένη σειρά. Για κάθε j, $i \leq j \leq k$, από τον Ισχυρισμό v)b. του Λήμματος 5.3 προκύπτει ότι, μεταξύ της $Cc(ij,n_{ij})$ και της $Caw(ij,n_{ij})$, η $T(ij,n_{ij})$ κατέχει όλα τα ownerships που επιθυμεί. Επίσης, από τον ίδιο Ισχυρισμό ότι τα διαστήματα εκτέλεσης του συνόλου $\{[Cc(ix,n_{ix}), Caw(ix,n_{ix})] \mid i \leq x \leq k\}$ δεν επικαλύπτονται. Συμβολίζουμε με $T(ik,n_{ik})$ την δοσοληψία εκείνη της οποίας το $Cc(ik,n_{ik})$ είναι το k-οστό στην α.

Η απόδειξη του Λήμματος θα γίνει με βάση τη θέση στην οποία ορίζεται επιτυχημένη (καθολική κατάσταση Cc) μία δοσοληψία. Σημειώνεται ότι η απόδειξη της βάσης της επαγωγής και η απόδειξη του επαγωγικού βήματος θα γίνει ταυτόχρονα. Υποθέτουμε ότι οι Ισχυρισμοί του Λήμματος ισχύουν για όλες τις k-1, όπου k>0, πρώτες επιτυχημένες δοσοληψίες και θα αποδείξουμε ότι ισχύουν για την k-οστή δοσοληψία που ορίζεται ως επιτυχημένη, δηλαδή την $T(ik,n_{ik})$.

i) Υποθέτουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι υπάρχει κάποια executing PerformTrans μιας δοσοληψίας $T(j,n_j)$, j<ik, έστω $PT_z(j,n_j)$, η οποία καταφέρνει να τροποποιήσει το memory[*l*] μεταξύ της $Cc(ik,n_{ik})$ και την $Caw(ik,n_{ik})$.


ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Έστω ότι k==1, επειδή η $T(i1,n_{i1})$ είναι η πρώτη επιτυχημένη δοσοληψία που αποκτά το ownerships[*l*] δεν μπορούν να υπάρχουν προηγούμενες της $T(i1,n_{i1})$ δοσοληψίες που να είναι επιτυχημένες και να έχουν αποκτήσει το ownerships[*l*] πριν την $Cc(i1,n_{i1})$, άτοπο. Έτσι, πρέπει να ισχύει k>1, δηλαδή η $T(j,n_j)$ ανήκει στις k-1 προηγούμενες (της $T(ik,n_{ik})$) επιτυχημένες δοσοληψίες.

Για να τροποποιήσει η $PT_z(j,n_j)$ το memory[*l*] πρέπει να εκτελέσει τη γραμμή 37 του κώδικα. Από τον Ισχυρισμό iv) του Λήμματος 5.3 προκύπτει ότι η $PT_z(j,n_j)$ εκτελεί τη γραμμή 37 μόνο εάν είναι επιτυχημένη, το οποίο με βάση τον Ισχυρισμό v)b. του Λήμματος 5.3 σημαίνει ότι έχει αποκτήσει όλα τα ownerships που επιθυμεί, άρα και το ownerships[*l*].

Βάσει του Ισχυρισμού v)c. του Λήμματος 5.3 προκύπτει ότι εάν η $PT_z(j,n_j)$, προσπαθήσει να αποκτήσει το ownerships[*l*] μετά την $Cc(ik,n_{ik})$ και πριν την $Caw(ik,n_{ik})$, δεν θα μπορέσει να εκτελέσει τη γραμμή 37 του κώδικα (με την οποία μπορεί να ενημερωθεί το memory[*l*]) πριν από την $Caw(ik,n_{ik})$. Έτσι πρέπει η $PT_z(j,n_j)$ να απέκτησε το ownerships[*l*] πριν την $Cc(ik,n_{ik})$. Σημειώνεται ότι η $PT_z(j,n_j)$ αποκτά το ownerships[*l*] πριν την $Cc(j,n_j)$. Όπως αναφέρθηκε, τα διαστήματα $[Cc(j,n_j),Caw(j,n_j)]$ και $[Cc(ik,n_{ik}),Caw(ik,n_{ik})]$ δεν επικαλύπτονται, και επειδή από επαγωγική υπόθεση η $Cc(ik,n_{ik})$ προηγείται της $Cc(j,n_j)$, προκύπτει ότι η $Caw(j,n_j)$ προηγείται της $Cc(ik,n_{ik})$. Βάσει του μέρους iv) της επαγωγικής υπόθεσης η $PT_z(j,n_j)$ δεν μπορεί να ενημερώσει το memory[*l*] μετά την $Caw(j,n_j)$. Έτσι, η $PT_z(j,n_j)$ δεν μπορεί να ενημερώσει το memory[*l*] μεταξύ της $Cc(ik,n_{ik})$ και της $Caw(ik,n_{ik})$. Άτοπο.

ii) Υποθέτουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι υπάρχει κάποιο *l*, *l* $\in$ $Trec_i$.add, για το οποίο το $Trec_i$.OldValues[*l*] δεν έχει την τιμή που είχε το memory[*l*] στην $Cc(ik,n_{ik})$. Σημειώνεται ότι οι τιμές του πίνακα $Trec_i$.OldValues μπορούν να τροποποιηθούν μόνο από τις executing PerformTrans της $T(ik,n_{ik})$ και από executing PerformTrans προηγούμενων δοσοληψιών που εκκινήθηκαν από την p_{ik}. Όμως, από τον Ισχυρισμό vii) του Λήμματος 5.3 προκύπτει ότι οι βοηθητικές PerformTrans προηγούμενων δοσοληψιών που εκκινήθηκαν από την p_{ik} δεν μπορούν να ενημερώσουν τις τιμές του πίνακα $Trec_i$.OldValues μετά την εκκίνηση της $T(ik,n_{ik})$. Άρα οι τιμές του πίνακα $Trec_i$.OldValues μπορούν να ενημερωθούν μόνο από τις executing PerformTrans της $T(ik,n_{ik})$.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Έστω v η τιμή που υπάρχει στο memory[*l*] στην SU(ik,n_{ik}). Από τον κώδικα της AgreeOldValues προκύπτει ότι στο $Trec_i$.OldValues[*l*] γράφεται η τιμή που υπάρχει στο memory[*l*] την στιγμή που εκτελείται η εντολή SC της γραμμής 78 για το $Trec_i$.OldValues[*l*]. Από τον Ισχυρισμό v)d. του Λήμματος 5.3, προκύπτει η τιμή αυτή γράφεται μία και μοναδική φορά, από κάποια executing PerformTrans της T(i,n_i), στην Cov(ik,n_{ik},*l*). Έτσι για να παρατηρηθεί στο $Trec_i$.OldValues[*l*] τιμή διαφορετική από v πρέπει κάποια PerformTrans, έστω PT_j, να άλλαξε την τιμή του memory[*l*] μεταξύ της Cc(ik,n_{ik}) και της Cov(ik,n_{ik},*l*). Σημειώνεται ότι η ενημέρωση της memory[*l*] προϋποθέτει την κλήση της συνάρτησης UpdateMemory. Από τον Ισχυρισμό i) του Λήμματος, προκύπτει ότι μόνο οι executing PerformTrans της T(ik,n_{ik}) μπορούν ενημερώσουν το memory[*l*] μεταξύ της Cc(ik,n_{ik}) και της Caw(ik,n_{ik}) (η οποία έπεται της Cov(ik,n_{ik},*l*)). Όμως, η PT_j δεν μπορεί να ανήκει στις executing PerformTrans της T(ik,n_{ik}), διότι από Ισχυρισμό v)d. του Λήμματος 5.3 και λόγω του ορισμού της Caov(i,n_i) προκύπτει ότι η Cov(ik,n_{ik},*l*) προηγείται της εκτέλεσης της συνάρτησης UpdateMemory από αυτές τις PerformTrans. Επομένως το memory[*l*] δεν μπορεί να ενημερωθεί μεταξύ της Cc(ik,n_{ik}) και της Cov(ik,n_{ik},*l*), το οποίο είναι Άτοπο.

iii) Έστω οποιοδήποτε *l*, $l \in Trec_i$.add. Έστω α_{OV} το διάστημα εκτέλεσης που ξεκινά από την Caov(ik,n_{ik}) και καταλήγει στην Cf(ik,n_{ik}). Από τον κώδικα της PerformTrans (γραμμή 21) προκύπτει ότι οι PerformTrans της T(ik,n_{ik}) υπολογίζουν το newValues[*l*] με βάση το $Trec_i$.oldValues[*l*], μετά την Caov(ik,n_{ik}). Υπενθυμίζουμε ότι πριν την Cf(ik,n_{ik}) εκτελείται η γραμμή 18 του κώδικα από την T(ik,n_{ik}) και αυξάνεται το $Trec_{ik}$.version σε n_{ik}+1. Έτσι με βάση το Λήμμα 5.1 όσες από τις executing PerformTrans της T(ik,n_{ik}) υπολογίσουν το newValues[*l*] (γραμμή 21) μετά την Cf(ik,n_{ik}), δηλαδή μετά το τέλος της α_{OV}, δεν θα μπορέσουν να εκτελέσουν την εντολή SC της γραμμής 87 για να ενημερώσουν το memory[*l*], διότι ο VerCheck έλεγχος της γραμμής 75 θα αποτιμηθεί αληθής.

Επίσης, με βάση τον Ισχυρισμό ii) που αποδείχθηκε παραπάνω, στο α_{OV} το $Trec_i$.oldValues[*l*] θα είναι ίσο με την τιμή του memory[*l*] στην Cc(ik,n_{ik}). Έτσι όσες από τις executing PerformTrans της T(ik,n_{ik}) υπολογίζουν το newValues[*l*] στο α_{OV}, θα υπολογίσουν την ίδια τιμή. Επομένως οι executing PerformTrans της T(ik,n_{ik}) που έχουν την δυνατότητα να εκτελέσουν την γραμμή 87 για το memory[*l*], και άρα μπορούν να το ενημερώσουν, θα έχουν υπολογίσει την ίδια τιμή για το newValues[*l*].

iv) Από τον Ισχυρισμό i) του Λήμματος, προκύπτει ότι μόνο οι executing PerformTrans της $T(ik,n_{ik})$ μπορούν ενημερώσουν το memory[l] μετά την $Cc(ik,n_{ik})$ και πριν την $AW(ik,n_{ik})$. Επίσης από τον Ισχυρισμό iii) του Λήμματος, προκύπτει ότι όλες οι executing δοσοληψίες της $T(ik,n_{ik})$ που έχουν την δυνατότητα να ενημερώσουν τον πίνακα memory υπολογίζουν την ίδια τιμή newvalues[l] για το memory[l]. Έστω V_{new} η νέα τιμή και V_{old} η τιμή που είχε το memory[l] στην $Cc(ik,n_{ik})$. Ακόμη, επειδή η $T(ik,n_{ik})$ είναι επιτυχημένη, από τον Ισχυρισμό iv) του Λήμματος 5.3 προκύπτει ότι μία τουλάχιστον από τις executing PerformTrans της $T(ik,n_{ik})$ θα εκτελέσει τη συνάρτηση UpdateMemory.

Σημειώνεται ότι για να μπορεί μία executing PerformTrans της $T(ik,n_{ik})$, έστω $PT_m(ik,n_{ik})$, να εκτελέσει την εντολή SC της γραμμής 87 για το memory[l] πρέπει να είδε στην γραμμή 84 $Trec_i$.AllWritten == FALSE. Στην $Caw(ik,n_{ik})$ θα ισχύει $Trec_i$.AllWritten == TRUE και από τον κώδικα προκύπτει ότι η τιμή αυτή δεν μπορεί να αλλάξει πριν την $Cf(ik,n_{ik})$. Έτσι εάν η γραμμή 84 εκτελέστηκε από την $PT_m(ik,n_{ik})$ μετά την $Caw(ik,n_{ik})$ τότε για να δει η $PT_m(ik,n_{ik})$ $Trec_i$.AllWritten == FALSE πρέπει να εκτελέστηκε η συνάρτηση Initiate από την p_{ik}, κάτι που σημαίνει ότι η εκτέλεση της $T(ik,n_{ik})$ έχει ολοκληρωθεί και άρα έχει εκτελεστεί η γραμμή 18 του κώδικα από την $T(ik,n_{ik})$. Όμως τότε σύμφωνα με το Λήμμα 5.1, η εκτέλεση του VerCheck της γραμμής 85 θα αποτιμηθεί ως αληθής και έτσι η γραμμή 87 δε θα εκτελούνταν από την $PT_m(ik,n_{ik})$. Επομένως για να μπορεί η $PT_m(ik,n_{ik})$ να εκτελέσει την εντολή SC της γραμμής 87 πρέπει να εκτέλεσε τη γραμμή 84 πριν την $Caw(ik,n_{ik})$.

Αρχικά εξετάζεται η περίπτωση κατά την οποία η νέα τιμή V_{new} που υπολογίστηκε για το memory[l] είναι ίδια με την τιμή που είχε το memory[l] στην $Cc(ik,n_{ik})$, δηλαδή ισχύει V_{new}==V_{old}. Στην συνέχεια αποδεικνύεται ότι στην περίπτωση αυτή δεν θα ενημερωθεί το memory[l] από τις executing PerformTrans της $T(ik,n_{ik})$. Ας υποθέσουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι υπάρχει τουλάχιστον μία executing PerformTrans της $T(ik,n_{ik})$ που εκτελεί επιτυχώς την εντολή SC της γραμμής 87 για το memory[l], μετά την $Cc(ik,n_{ik})$. Έστω $PT_z(ik,n_{ik})$ η πρώτη από αυτές τις executing PerformTrans και SC_1 η εντολή SC της γραμμής 87 που αυτή εκτελεί. Όπως αναφέρθηκε παραπάνω, για να μπορεί η $PT_z(ik,n_{ik})$ να εκτελέσει την SC_1 πρέπει να εκτέλεσε τη γραμμή 84 πριν την $Caw(ik,n_{ik})$. Έτσι η $PT_z(ik,n_{ik})$ εκτέλεσε την εντολή LL της γραμμής 83 πριν από το σημείο $AW(ik,n_{ik})$,

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

έστω στην καθολική κατάσταση C_{LL}. Λόγω του Ισχυρισμού i) που του Λήμματος, δε μπορεί κάποια executing PerformTrans μιας δοσοληψίας διαφορετικής της $T(ik,n_{ik})$ να τροποποιήσει το memory[*l*] μεταξύ της $Cc(ik,n_{ik})$ και της $Caw(ik,n_{ik})$. Επίσης η $PT_z(ik,n_{ik})$ είναι η πρώτη executing PerformTrans της $T(i,n_i)$ που θα εκτελέσει την SC_1 επιτυχώς και η C_{LL} προηγείται της $Caw(i,n_i)$, έτσι στην C_{LL} το memory[*l*] εξακολουθεί να έχει τιμή $V_{old}=V_{new}$. Πριν η $PT_z(ik,n_{ik})$ εκτελέσει την SC_1 πρέπει να περάσει τον έλεγχο της γραμμή 86 για το oldvalue. Όμως η $PT_z(ik,n_{ik})$ δεν περνά τον έλεγχο της γραμμής 86 διότι ισχύει oldvalue = newValues[*l*] = V_{new} και έτσι δεν εκτελείται ποτέ η SC_1. Άτοπο.

Από την άλλη εάν ισχύει $V_{new} \neq V_{old}$ τότε ορίζουμε A το σύνολο των executing PerformTrans της $T(ik,n_{ik})$ που εκτελούν τον έλεγχο της γραμμής 86 για το memory[*l*] πριν την πρώτη εκτέλεση της εντολής SC της γραμμής 87 για το memory[*l*], έστω SC' η εντολή αυτή. Εφόσον καμία executing PerformTrans της $T(ik,n_{ik})$ δεν έχει εκτελέσει την SC', η εκτέλεση του ελέγχου της γραμμής 71 από τις PerformTrans της A θα είναι επιτυχής. Έστω $PT_x(ik,n_{ik})$ εκείνη η PerformTrans του A που εκτελεί πρώτη την SC_1, στην καθολική κατάσταση C. Εξ ορισμού της C, η εκτέλεση της SC' από την $PT_x(ik,n_{ik})$ στην C θα είναι επιτυχής και έτσι θα γραφεί η τιμή V_{new} στο memory[*l*]. Σημειώνεται ότι η C προηγείται της $AW(ik,n_{ik})$.

Στη συνέχεια αποδεικνύεται ότι η ενημέρωση αυτή είναι μοναδική, δηλαδή ότι δεν μπορεί να ενημερωθεί το memory[*l*] μετά την C από κάποια executing PerformTrans της $T(ik,n_{ik})$. Ας υποθέσουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι b>1 executing PerformTrans της $T(ik,n_{ik})$ ενημέρωσαν την θέση memory[*l*], $l \in Trec_i.add$, μετά την C. Έστω $PT_j(ik,n_{ik})$ η πρώτη από αυτές. Η ενημέρωση της θέσης memory[*l*] γίνεται από την $PT_j(ik,n_{ik})$ με την εκτέλεση της εντολής SC της γραμμής 87 του κώδικα για το memory[*l*], έστω SC_1 αυτή η εντολή SC. Σημειώνεται ότι στην C ισχύει memory[*l*] = V_{new}. Εάν η $PT_j(ik,n_{ik})$ εκτέλεσε την εντολή LL της γραμμής 83 μετά την C, τότε στην γραμμή 86 είτε θα ισχύει oldvalue = V_{new} και άρα δεν θα εκτελέσει την SC_1 ή θα ισχύει oldvalue $\neq V_{new}$. Εάν ισχύει oldvalue $\neq V_{new}$, προκύπτει ότι έχει αλλάξει η τιμή memory[*l*] μετά την C από κάποια executing PerformTrans $PT(z,n_z)$ μιας δοσοληψίας $T(z,n_z)$, όπου $z \neq ik$. Βάσει του Ισχυρισμού iv) του Λήμματος 5.3 και του Ισχυρισμού v)b. του Λήμματος 5.3, προκύπτει ότι για να ενημερώσει η $PT(z,n_z)$ το memory[*l*], πρέπει για την $T(z,n_z)$ να έχει οριστεί η $Cc(z,n_z)$. Όπως αναφέρθηκε, τα διαστήματα $[Cc(j,n_j),Caw(j,n_j)]$ και

$[Cc(ik,n_{ik}),Caw(ik,n_{ik})]$ δεν επικαλύπτονται, και από την επαγωγική υπόθεση προκύπτει ότι η $Cc(z,n_z)$ θα έπεται της $Cc(ik,n_{ik})$. Έτσι, προκύπτει ότι η $Caw(ik,n_{ik})$ προηγείται της $Cc(ik,n_{ik})$. Επομένως η $PT_j(ik,n_{ik})$ εκτελεί τη γραμμή 84 μετά την $Caw(ik,n_{ik})$. Όπως αναφέρθηκε παραπάνω, για να μπορεί η $PT_j(ik,n_{ik})$ να εκτελέσει την SC_1 πρέπει να εκτέλεσε τη γραμμή 84 πριν την $Caw(ik,n_{ik})$. Άρα, άτοπο.

Επομένως πρέπει η $PT_j(ik,n_{ik})$ να εκτέλεσε την εντολή LL της γραμμής 83 για το memory[*l*] πριν την C και στην συνέχεια να εκτέλεσε την SC_1 μετά την C. Όμως στην C έχει επιτύχει η εκτέλεση της εντολής SC της γραμμής 87 για το memory[*l*] από την $PT_j(ik,n_{ik})$ και έτσι βάσει του ορισμού της εντολής LL/SC, η εκτέλεση της SC_1 από την $PT_j(ik,n_{ik})$ μετά την C θα αποτύχει. Άτοπο.

Άρα το Λήμμα 5.29 ισχύει. ■

Λήμμα 5.30: Η εκτέλεση μίας αποτυχημένης δοσοληψίας T(i,ni) που εκκινήθηκε από την διεργασία p_i, δεν μπορεί να επηρεάσει τις τιμές του πίνακα memory.

Απόδειξη: Λόγω του Ισχυρισμού iv) του Λήμματος 5.29, οι βοηθητικές PerformTrans προηγούμενων δοσοληψιών που εκκινήθηκαν από την p_i δεν μπορούν να ενημερώσουν τον πίνακα memory μετά το σημείο Caw τους και άρα μετά την εκκίνηση της $T(i,n_i)$. Έτσι αρκεί να εξεταστεί η συμπεριφορά των executing PerformTrans της $T(i,n_i)$. Από τον κώδικα προκύπτει ότι η ενημέρωση του πίνακα memory γίνεται μόνο με την εκτέλεση της συνάρτησης της γραμμής 37 του κώδικα. Επίσης από τον Ισχυρισμό iv) του Λήμματος 5.3 προκύπτει ότι η γραμμή 37 του κώδικα εκτελείται από κάποια executing PerformTrans της $T(i,n_i)$ μόνο εάν η $T(i,n_i)$ είναι επιτυχημένη. Έτσι οι executing PerformTrans της αποτυχημένης δοσοληψίας $T(i,n_i)$ δεν θα εκτελέσουν ποτέ την γραμμή 37 του κώδικα και άρα δεν θα ενημερώσουν τον πίνακα memory. Επομένως το Λήμμα 5.30 ισχύει. ■

Στην συνέχεια αποδίδουμε σημεία σειριοποίησης στις δοσοληψίες. Αρχικά σημειώνεται ότι δεν χρειάζεται να αποδώσουμε σημείο σειριοποίησης σε κάποια αποτυχημένη δοσοληψία $T(i,n_i)$, διότι πρώτον, σύμφωνα με το Λήμμα 5.30 η $T(i,n_i)$ δεν επηρεάζει τον πίνακα memory και δεύτερον, η $T(i,n_i)$ (εάν δεν σφάλει) επιστρέφει μήνυμα αποτυχίας στην διεργασία από την οποία εκκινήθηκε. Έτσι αποδίδουμε σημεία σειριοποίησης μόνο στις επιτυχημένες δοσοληψίες. Συγκεκριμένα ορίζουμε

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ως σημείο σειριοποίησης μιας επιτυχημένης δοσοληψίας T(i,n_i) την καθολική κατάσταση AW(i,n_i) στην οποία γράφεται η τιμή true στο $Trec_i$.AllWritten. H Caw(i,n_i) σύμφωνα με τον Ισχυρισμό v) του Λήμματος 5.3 είναι καλά ορισμένη.

Έστω μια επιτυχημένη δοσοληψία T(i,n_i) και έστω O(Caw(i,n_i)) = {d_1, d_2, ..., $d_{Trec_i.size}$}, το σύνολο των τιμών που περιέχεται στον πίνακα $Trec_i$.OldValues, στην καθολική κατάσταση Caw(i,n_i). Βάσει της ιδιότητας της σειριοποιησιμότητας κάθε τιμή d_l του συνόλου O(Caw(i,n_i)) είναι συνεπής εάν είτε περιέχει την αρχική τιμή της αντίστοιχης θέσης του πίνακα memory (memory[$Trec_i$.add[l]]) είτε γράφεται στην αντίστοιχη θέση του πίνακα memory, από την τελευταία δοσοληψία που ενημερώνει το memory[$Trec_i$.add[l]] και σειριοποιείται πριν το σημείο σειριοποίησης της T(i,n_i), δηλαδή πριν την Caw(i,n_i). Στο παρακάτω Λήμμα αποδεικνύεται ότι αυτό ισχύει για τον αλγόριθμο SSTM.

Λήμμα 5.31: *Τα δεδομένα του συνόλου O(Caw(i,n_i)) είναι συνεπή.*

Απόδειξη: Έστω O(Caw(i,n_i)) = {d_1, d_2, ..., $d_{Trec_i.size}$}. Θα δειχθεί ότι για κάθε l, $l \in Trec_i$.add, εάν το d_l δεν δείχνει στην αρχική τιμή των δεδομένων του memory[l] τότε εγγράφεται από την τελευταία επιτυχημένη δοσοληψία που σειριοποιείται πριν την T(i,n_i) και ενημερώνει το memory[l]. Έστω, δια της μεθόδου της εις άτοπο απαγωγής, ότι υπάρχει κάποια τιμή d_l που ανήκει στο O(Caw(i,n_i)) τέτοια ώστε η τελευταία επιτυχημένη δοσοληψία, έστω T(j,n_j), που ενημερώνει το memory[l] και σειριοποιείται πριν την T(i,n_i) δεν γράφει την τιμή d_b, αλλά την τιμή d_b'. Έστω ότι η τελευταία δοσοληψία που ενημερώνει το memory[l] με την τιμή d_b και σειριοποιείται πριν την T(i,n_i) είναι η T(k,n_k).

Σημειώνεται ότι η T(j,n_j) σειριοποιείται στην καθολική κατάσταση Caw(j,n_j) και η T(k,n_k) στην καθολική κατάσταση Caw(k,n_k). Από τον Ισχυρισμό iv) του Λήμματος 5.29 προκύπτει ότι οι executing PerformTrans της T(j,n_j) ενημερώνουν το memory[l] μία και μοναδική φορά στο διάστημα [Cc(j,n_j), Caw(j,n_j)], έστω στην καθολική κατάσταση C_j. Επίσης από τον ίδιο Ισχυρισμό προκύπτει ότι οι executing PerformTrans της T(k,n_k) ενημερώνουν το memory[l] μία και μοναδική φορά στο διάστημα [Cc(k,n_k), Caw(k,n_k)], έστω στην καθολική κατάσταση C_j.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Από τον Ισχυρισμό v)d. του Λήμματος 5.3, προκύπτει οι executing PerformTrans της $T(i,n_i)$ γράφουν για μία και μοναδική φορά την τιμή του memory[l] στον πίνακα $Trec_i$.oldValues, δηλαδή αναθέτουν τιμή στο d_l, στην καθολική κατάσταση $Cov(i,n_i,l)$. Από τον ίδιο Ισχυρισμό προκύπτει ότι η $Cov(i,n_i,l)$ προηγείται της $Caov(i,n_i)$, δηλαδή προηγείται της $Caw(i,n_i)$. Σημειώνεται ότι η $Cov(i,n_i,l)$ ορίζεται κατά την εκτέλεση της συνάρτησης AgreeOldValues που εκτελείται στη γραμμή 35 του κώδικα. Από τον Ισχυρισμό iv) του Λήμματος 5.3 προκύπτει ότι οι executing PerformTrans της $T(i,n_i)$ εκτελούν τη γραμμή 35 του κώδικα μόνο εάν η $T(i,n_i)$ είναι επιτυχημένη, επομένως εάν έχει ήδη οριστεί η $Cc(i,n_i)$. Έτσι προκύπτει ότι η $Cov(i,n_i,l)$ έπεται της $Cc(i,n_i)$. Επομένως η $Cov(i,n_i,l)$ ορίζεται στο διάστημα $[Cc(i,n_i), Caw(i,n_i)]$.

Από υπόθεση η $Caw(k,n_k)$ προηγείται της $Caw(j,n_j)$ και επίσης και οι δύο προηγούνται της $Caw(i,n_i)$. Για κάθε b, $b \in \{i,j,k\}$, από τον Ισχυρισμό v)b. του Λήμματος 5.3 προκύπτει ότι, μεταξύ της $Cc(b,n_b)$ και της $Caw(b,n_b)$, η $T(b,n_b)$ κατέχει όλα τα ownerships που επιθυμεί. Επίσης, από τον ίδιο Ισχυρισμό ότι τα διαστήματα εκτέλεσης του συνόλου $\{[Cc(b,n_b), Caw(b,n_b)] \mid b \in \{i,j,k\}\}$ δεν επικαλύπτονται. Επομένως τα διαστήματα εμφανίζονται με την εξής σειρά: $[Cc(k,n_k), Caw(k,n_k)]$, $[Cc(j,n_j), Caw(j,n_j)]$, $[Cc(i,n_i), Caw(i,n_i)]$. Άρα και οι παρακάτω καθολικές καταστάσεις εμφανίζονται με την εξής σειρά: C_k, C_j, $Cov(i,n_i,l)$. Έτσι το d_l, θα έπρεπε να έχει την τιμή d_b' και όχι την τιμή d_b, το οποίο είναι άτοπο. ■

Έτσι προκύπτει άμεσα το παρακάτω Θεώρημα.

Θεώρημα 5.32: *Ο αλγόριθμος SSTM είναι σειριοποιήσιμος.*

5.2.2. Ελευθερία Κλειδωμάτων

Στην ενότητα αυτή θα αποδειχθεί ότι ο αλγόριθμος SSTM ικανοποιεί την ιδιότητα τερματισμού ελευθερία κλειδωμάτων. Σημειώνεται ότι κάθε διεργασία εκκινεί μία δοσοληψία κάθε φορά. Έτσι στην συνέχεια, η διεργασία p_j και η μοναδική δοσοληψία που εκτελείται από αυτή κάθε χρονική στιγμή, θα θεωρούνται ίδιες.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Έστω ότι η δοσοληψία $T(i,n_i)$ είναι αποτυχημένη και απέτυχε διότι η δοσοληψία $T(j,n_j)$, $i \neq j$, κατείχε τη θέση $FM(i,n_i)$ του ownerships. Πριν η p_i εκτελέσει τη γραμμή 48 του κώδικα για να βοηθήσει την $T(j,n_j)$ εκτελεί στη γραμμή 40 τη συνάρτηση ReleaseOwnerships για να καταργήσει όλες τις θέσεις του ownerships που κατέχει. Από τον Ισχυρισμό vi) του Λήμματος 5.3 προκύπτει ότι, μετά το τέλος της εκτέλεσης αυτής της ReleaseOwnerships η p_i δεν θα κατέχει κανένα ownerships. Έτσι προκύπτει το παρακάτω πόρισμα.

Πόρισμα 5.33: *Έστω μια αποτυχημένη δοσοληψία $T(i,n_i)$ που απέτυχε διότι η $T(j,n_j)$, $i \neq j$, κατείχε τη θέση $FM(i,n_i)$ του ownerships. Τότε η p_i δεν θα κατέχει κανένα ownerships όταν θα εκτελέσει την γραμμή 48 του κώδικα για να βοηθήσει την $T(j,n_j)$.*

Λήμμα 5.34: *Έστω $T(i,n_i)$ μια δοσοληψία. Εάν η $T(i,n_i)$ είναι η μόνη που επιθυμεί να αποκτήσει τα ownerships που περιγράφονται από το $Trec_i$.add τής $T(i,n_i)$, τότε i) η $Caof(i,n_i)$ ορίζεται, ii) το $Trec_i$.status θα έχει τιμή ACTIVE στην $Caof(i,n_i)$ και iii) η $T(i,n_i)$ θα χαρακτηρισθεί επιτυχημένη.*

Απόδειξη : Για να μπορεί να οριστεί η $Caof(i,n_i)$ πρέπει μία τουλάχιστον από τις executing PerformTrans της $T(i,n_i)$ να ολοκληρώσει την εκτέλεση της συνάρτησης AcquireOwnerships. Από τον κώδικα της AcquireOwnerships (γραμμές 51 και 53) προκύπτει ότι τουλάχιστον μία executing PerformTrans της $T(i,n_i)$ θα προσπαθήσει να αποκτήσει τις θέσεις του πίνακα ownerships που επιθυμεί η $T(i,n_i)$ (οι θέσεις που περιέχονται στον πίνακα $Trec_i$.add) με προκαθορισμένη σειρά από την μικρότερη προς την μεγαλύτερη, πριν την $Caof(i,n_i)$. Αυτό σημαίνει ότι τουλάχιστον μία από τις executing PerformTrans της $T(i,n_i)$ θα εκτελέσει τον W_{AO} για κάποια l, $l \in Trec_i$.add, (ίσως και όλα) πριν την $Caof(i,n_i)$. Επίσης από τον Ισχυρισμό i) του Λήμματος 5.3 προκύπτει ότι οι executing PerformTrans της $T(i,n_i)$ που θα εκτελέσουν τον W_{AO} για κάποιο l, $l \in Trec_i$.add, πριν την $Caof(i,n_i)$, θα αποχωρήσουν από τον W_{AO} είτε αποκτώντας το ownerships[l] είτε γράφοντας την τιμή ABORTED στο $Trec_i$.status. Από τον κώδικα (γραμμές 55, 57, 58, 62) προκύπτει ότι για να γραφτεί η τιμή ABORTED στο $Trec_i$.status πρέπει κάποια άλλη δοσοληψία $T(z,n_z)$, $z \neq i$, να απέκτησε το ownerships[l]. Όμως κάτι τέτοιο δεν μπορεί να γίνει από την υπόθεση του Λήμματος, διότι η $T(i,n_i)$ είναι η μόνη που επιθυμεί να αποκτήσει το ownerships[l].

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Έτσι, οι executing PerformTrans της $T(i,n_i)$ που θα εκτελέσουν τον W_{AO} για κάποιο l πριν την $AOf(i,n_i)$, θα αποχωρήσουν από τον W_{AO} μόνο εάν αποκτήσουν το ownerships[l].

Υποθέτουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι οι executing PerformTrans της $T(i,n_i)$ εκτελούν συνεχώς τον W_{AO} για κάποιο l, $l \in Trec_i.add$, και έτσι καμία από αυτές δεν καταφέρνει να ολοκληρώσει την εκτέλεση της AcquireOwnerships. Έστω l_{max} το μεγαλύτερο τέτοιο l και έστω A το σύνολο που περιέχει αυτές τις PerformTrans. Σημειώνεται ότι εφόσον καμία executing PerformTrans της $T(i,n_i)$ δεν έχει ολοκληρώσει την AcquireOwnerships, δεν έχει οριστεί ακόμα η $Caof(i,n_i)$. Έτσι, με βάση τα παραπάνω, για να είναι δυνατό οι PerformTrans του A να εκτελούν συνεχώς τον W_{AO} για το l_{max} πρέπει καμία από αυτές να μην αποκτά το ownerships[l_{max}]. Όμως κάτι τέτοιο δεν μπορεί να γίνει, διότι δεν υπάρχουν executing PerformTrans δοσοληψία διαφορετικής την $T(i,n_i)$ που να επιθυμούν να αποκτήσουν το ownerships[l_{max}], η $T(i,n_i)$ δεν έχει τερματιστεί και επίσης λόγω του Ισχυρισμό i) του Λήμματος 5.3 δεν μπορεί να γραφεί η τιμή COMMITED στο $Trec_i$.status πριν την $Caof(i,n_i)$. Έτσι από τον κώδικα του W_{AO} προκύπτει ότι η πρώτη από τις PerformTrans του A που θα εκτελέσει την εντολή SC της γραμμής 60 θα επιτύχει και έτσι θα αποκτηθεί το ownerships[l_{max}]. Άτοπο.

Έτσι καμία από τις executing PerformTrans της $T(i,n_i)$ δεν μπορεί να εκτελεί επ' άπειρο τον W_{AO} της AcquireOwnerships και επίσης δεν μπορεί να γραφεί η τιμή ABORTED στο $Trec_i$.status πριν την $Caof(i,n_i)$. Έτσι στην $Caof(i,n_i)$ το $Trec_i$.status θα έχει τιμή ACTIVE. Τότε, από τον Ισχυρισμό iii) του Λήμματος 5.3 προκύπτει ότι η $T(i,n_i)$ θα χαρακτηρισθεί ως επιτυχημένη. Άρα το Λήμμα 5.34 ισχύει. ■

Θεώρημα 5.35: *Ο αλγόριθμος SSTM ικανοποιεί την ιδιότητα ελευθερία κλειδωμάτων.*

Απόδειξη: Υποθέτουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι ο αλγόριθμος SSTM δεν ικανοποιεί την ιδιότητα ελευθερία κλειδωμάτων. Επομένως υπάρχει πεπερασμένος ή άπειρος αριθμός δοσοληψιών, οι οποίες έχουν «κολλήσει» σε κάποιο σημείο της εκτέλεσής του και έτσι καμία από αυτές δεν ολοκληρώνεται επιτυχώς. Σημειώνεται ότι οι μοναδικοί άπειροι βρόχοι του κώδικα του SSTM είναι ο W_{AO} και ο W_{PT}. Οι βρόχοι αυτοί αποτελούν τα μόνα σημεία του κώδικα που μπορεί μία

δοσοληψία να εκτελεί επ' άπειρο. Από τον Ισχυρισμό ii) του Λήμματος 5.3 προκύπτει ότι καμία δοσοληψία δεν μπορεί να εκτελεί επ' άπειρο τον W_{PT}.

Επομένως για να μην τερματίζει καμιά δοσοληψία επιτυχώς πρέπει οι executing PerformTrans των δοσοληψιών να μην μπορούν να ολοκληρώσουν την εκτέλεση του βρόχου W_{AO} για κάποια θέση *l* (η οποία μπορεί να διαφέρει από PerformTrans σε PerformTrans) του πίνακα ownerships. Έστω ότι το *l* είναι μοναδικό για κάθε δοσοληψία, δηλαδή δεν υπάρχουν δύο δοσοληψίες που να θέλουν να αποκτήσουν την ίδια θέση *l* του πίνακα ownerships. Σε αυτή την περίπτωση, σύμφωνα με το Λήμμα 5.34, για κάθε δοσοληψία $T(i,n_i)$ θα υπάρχει μία τουλάχιστον executing PerformTrans που θα ολοκληρώσει την εκτέλεση της AcquireOwnerships και επίσης η $T(i,n_i)$ θα χαρακτηρισθεί ως επιτυχημένη. Επίσης η $T(i,n_i)$ θα τερματίσει (διότι από τον Ισχυρισμό ii) του Λήμματος 5.3 δεν μπορεί να κολλήσει επ' άπειρο στον W_{PT}) ως επιτυχημένη. Άτοπο.

Επομένως για να μην υπάρχει πρόοδος στο σύστημα πρέπει να υπάρχουν άπειρες δοσοληψίες που αποτυγχάνουν συνεχώς επειδή επιθυμούν να αποκτήσουν τις ίδιες θέσεις μνήμης. Άρα, κάθε δοσοληψία αποτυγχάνει διότι κάποια άλλη δοσοληψία κατέχει κάποια θέση μνήμης που αυτή επιθυμεί. Επομένως θα υπάρχει μία θέση μνήμης η οποία θα είναι θέση σφάλματος άπειρες φορές. Θεωρούμε x την μεγαλύτερη τέτοια θέση μνήμης και έστω $T(j,n_j)$ η δοσοληψία που κατέχει αυτή την θέση μνήμης. Με βάση το Πόρισμα 5.33, κάθε δοσοληψία που αποτυγχάνει στο σημείο x καταργεί τις θέσεις του ownerships που κατέχει πριν βοηθήσει κάποια άλλη δοσοληψία, επομένως υπάρχουν δοσοληψίες που βοηθούν την $T(j,n_j)$ και αποτυγχάνουν. Όπως αναφέρθηκε, οι executing δοσοληψίες μιας δοσοληψίας $T(z,n_z)$ προσπαθούν να αποκτήσουν τις θέσεις του ownerships που η T επιθυμεί, από την μικρότερη προς την μεγαλύτερη. Έτσι, προκύπτει ότι οι παραπάνω δοσοληψίες που βοηθούν την $T(j,n_j)$ αποτυγχάνουν σε μια θέση μνήμης μεγαλύτερης της x. Όμως, τότε η x δεν είναι η μεγαλύτερη θέση σφάλματος όπως θεωρήθηκε, άτοπο. Άρα το Θεώρημα 5.35 ισχύει.

5.3. Πολυπλοκότητα αλγορίθμου SSTM

Στην παρούσα Ενότητα συζητείται η πολυπλοκότητα του αλγορίθμου SSTM. Ο SSTM απαιτεί δύο καταχωρητές LL/SC πεπερασμένου μεγέθους για κάθε t-μεταβλητή, έναν για την αποθήκευση των δεδομένων της t-μεταβλητής και έναν για την αποθήκευση της προσωρινής ιδιοκτησίας της t-μεταβλητής. Σημειώνεται ότι ο SSTM χρησιμοποιεί την ίδια δομή για τις δοσοληψίες που εκκινούνται από την ίδια διεργασία. Για κάθε δομή που περιγράφει τις δοσοληψίες μιας συγκεκριμένης διεργασίας, ο SSTM απαιτεί έναν πίνακα σταθερού μεγέθους (M) από καταχωρητές ανάγνωσης εγγραφής πεπερασμένου μεγέθους (για την περιγραφή του Dataset), δύο καταχωρητές ανάγνωσης εγγραφής πεπερασμένου μεγέθους (για το status και την f) και έναν καταχωρητή άπειρου μεγέθους για την αποθήκευση ενός άπειρου μετρητή *version* που περιγράφει τον αύξων αριθμό της τρέχουσας δοσοληψίας που εκτελείται από τη διεργασία. Σημαντικό μειονέκτημα του αλγορίθμου SSTM απαιτεί η ύπαρξη ενός μη-πεπερασμένου μετρητή στη δομή κάθε διεργασίας.

Κάθε επιτυχημένη δοσοληψία στον SSTM που προσπελάσει R t-μεταβλητές για ανάγνωση και W t-μεταβλητές για ενημέρωση ελλείψει ανταγωνισμού, απαιτεί 2*(R+W)+1 εντολές LL/SC για την απόκτηση των προσωρινών ιδιοκτησιών των t-μεταβλητών, R+W+1 εντολές LL/SC για την ενημέρωση των δεδομένων των t-μεταβλητών και R+W εντολές LL/SC για την κατάργηση των προσωρινών ιδιοκτησιών των t-μεταβλητών, συνολικά 4*(R+W)+2 εντολές LL/SC.

ΚΕΦΑΛΑΙΟ 6. ΑΛΓΟΡΙΘΜΟΣ NBSTM

6.1 Περιγραφή αλγορίθμου NBSTM
6.2 Απόδειξη ορθότητας αλγορίθμου NBSTM
6.3 Πολυπλοκότητα

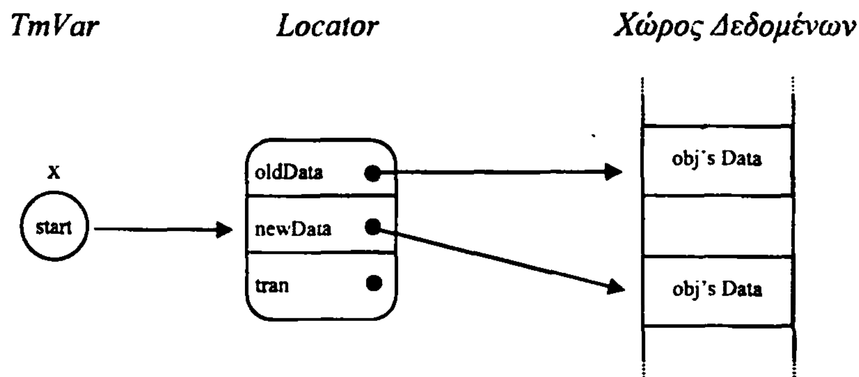
Στο κεφάλαιο αυτό περιγράφεται ο αλγόριθμος Non Blocking STM ή NBSTM, ο οποίο σχεδιάστηκε στα πλαίσια της παρούσας εργασίας. Στην Ενότητα 6.1 περιγράφεται ο κώδικας του αλγορίθμου, στην Ενότητα 6.2 παρουσιάζεται η απόδειξη της ορθότητάς του και στην Ενότητα 6.3 παρουσιάζεται η πολυπλοκότητά του.

6.1. Περιγραφή αλγορίθμου NBSTM

Ο αλγόριθμος NBSTM σχεδιάστηκε στα πλαίσια της παρούσας εργασίας, συνδυάζοντας ιδέες από τους αλγορίθμους SSTM [19] και DSTM [12]. Ο NBSTM υποστηρίζει δυναμικές δοσοληψίες, ικανοποιεί την μη-εμποδιστική ιδιότητα τερματισμού της ελευθερίας κλειδωμάτων και έχει σχεδιαστεί να λειτουργεί σε κλειστό μοντέλο μνήμης. Για να εξασφαλίσει την ιδιότητα της ελευθερίας κλειδωμάτων, ο αλγόριθμος NBSTM υλοποιεί έναν μηχανισμό βοήθειας που εξηγείται στη συνέχεια της παρούσας ενότητας.

Στον NBSTM, όπως και στον DSTM, κάθε t-μεταβλητή x υλοποιείται από έναν καταχωρητή CAS που περιέχεται σε μια δομή η οποία ονομάζεται *TmVar*. Ο καταχωρητής αυτός αποθηκεύει έναν δείκτη *start* που δείχνει σε κάποιο ενδιάμεσο διαμοιραζόμενο αντικείμενο, που ονομάζεται *Locator* και το οποίο διατηρεί i) έναν καταχωρητή ανάγνωσης-εγγραφής *tran*, ο οποίος περιέχει ένα δείκτη σε μια δομή που

περιγράφει τη δοσοληψία που κατέχει την προσωρινή ιδιοκτησία της x, ii) έναν καταχωρητή ανάγνωσης-εγγραφής *oldData* που περιέχει έναν δείκτη προς τα παλιά δεδομένα της x και iii) έναν καταχωρητή ανάγνωσης-εγγραφής *newData* που περιέχει έναν δείκτη προς τα νέα δεδομένα της x. Η μορφή μιας t-μεταβλητής x που αναπαριστά ένα διαμοιραζόμενο αντικείμενο obj παρουσιάζεται στο Σχήμα 2.2. Σημειώνεται ότι ο NBSTM προϋποθέτει ότι μπορεί να οριστεί μία *καθολική διάταξη* μεταξύ t-μεταβλητών, βάσει της τιμής της διεύθυνσης στην οποία είναι αποθηκευμένο η αντίστοιχη δομή TmVar που περιγράφει κάθε t-μεταβλητή. Επιλέγεται η συγκεκριμένη καθολική διάταξη να ορίζεται ως η γνησίως αύξουσα διάταξη των τιμών των παραπάνω διευθύνσεων.



Σχήμα 6.1: Η μορφή ενός διαμοιραζόμενου αντικειμένου obj που περιγράφεται μέσω μιας t-μεταβλητής x, στον NBSTM.

Για κάθε δοσοληψία $T(i,n_i)$ ο αλγόριθμος NBSTM διατηρεί τις ακόλουθες πληροφορίες στη δομή $Trec(i,n_i)$: i) έναν καταχωρητή CAS που ονομάζεται *status* και περιγράφει την κατάσταση της δοσοληψίας, ii) μια λίστα αναγνώσεων *readList* που περιέχει πληροφορίες για τις t-μεταβλητές που η δοσοληψία έχει προσπελάσει με καθολική ανάγνωση, και iii) μια λίστα ενημερώσεων *writeList* που περιέχει πληροφορίες για τις t-μεταβλητές που η δοσοληψία έχει προσπελάσει για ενημέρωση. Συγκεκριμένα κάθε στοιχείο item της readList περιγράφει κάποια t-μεταβλητή x και περιέχει i) έναν καταχωρητή ανάγνωσης εγγραφής *tvar* τύπου TmVar που περιγράφει τη x, ii) έναν καταχωρητή ανάγνωσης-εγγραφής *tvarVersion* που περιέχει ένα δείκτη προς τα δεδομένα της x που αναγνώσθηκαν (ο οποίος αναπαριστά την έκδοση της x τη χρονική στιγμή της καθολικής ανάγνωσης) και iii) έναν καταχωρητή ανάγνωσης-εγγραφής *newLoc* που περιέχει ένα δείκτη σε κάποιον Locator που χρησιμοποιείται

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

από την $T(i,n_i)$ για την απόκτηση της προσωρινής ιδιοκτησίας της x. Κάθε στοιχείο item της writeList περιγράφει κάποια t-μεταβλητή x και περιέχει τους καταχωρητές obj και newLoc, με λειτουργικότητα όμοια με τους αντίστοιχους καταχωρητές της readList. Για μια συγκεκριμένη t-μεταβλητή μπορεί να διατηρείται παράλληλα εγγραφή της στη readList και στη writeList. Αυτό μπορεί να συμβεί μόνο εάν η t-μεταβλητή προσπελαστεί με καθολική ανάγνωση και στη συνέχεια προσπελαστεί για ενημέρωση.

```
Transaction {                    Read_List {
   int status;                      TmVar *tvar;
   Read_List *readList;             DATA *tvarVersion;
   Write_List *writeList;           Locator *newLoc
}                                   Read_List *next;
                                 }
Locator {
   Transaction *Trec;            Write_List {
   DATA *oldData;                   TmVar *tvar;
   DATA *newData;                   Locator *newLoc;
}                                   Write_List *next;
                                 }
TmVar {
   Locator *start;
}
```

Σχήμα 6.2: Οι δομές που χρησιμοποιούνται από τον αλγόριθμο NBSTM.

Στο Σχήμα 6.2 παρουσιάζονται οι δομές που χρησιμοποιούνται από τον αλγόριθμο NBSTM. Σημειώνεται ότι το status μιας δοσοληψίας μπορεί να παίρνει τιμές από το σύνολο {ACTIVE, COMMITTED, ABORTED}, όπου κάθε μία χαρακτηρίζει την $T(i,n_i)$ ως ενεργή, επιτυχώς ολοκληρωμένη και μη επιτυχώς ολοκληρωμένη, αντίστοιχα. Επίσης τα δεδομένα που ο χρήστης μπορεί να αποθηκεύει στις t-μεταβλητές περιγράφονται με τον τύπο *DATA*. Σημειώνεται ότι επειδή ο χρήστης μπορεί να αποθηκεύει τις t-μεταβλητές μόνο δείκτες ο τύπος δεδομένων DATA, για παράδειγμα στη γλώσσα προγραμματισμού της C θα αντιστοιχούσε στον τύπο *void **. Για τη λίστα αναγνώσεων readList υπάρχουν οι συναρτήσεις: i) *Read_List*

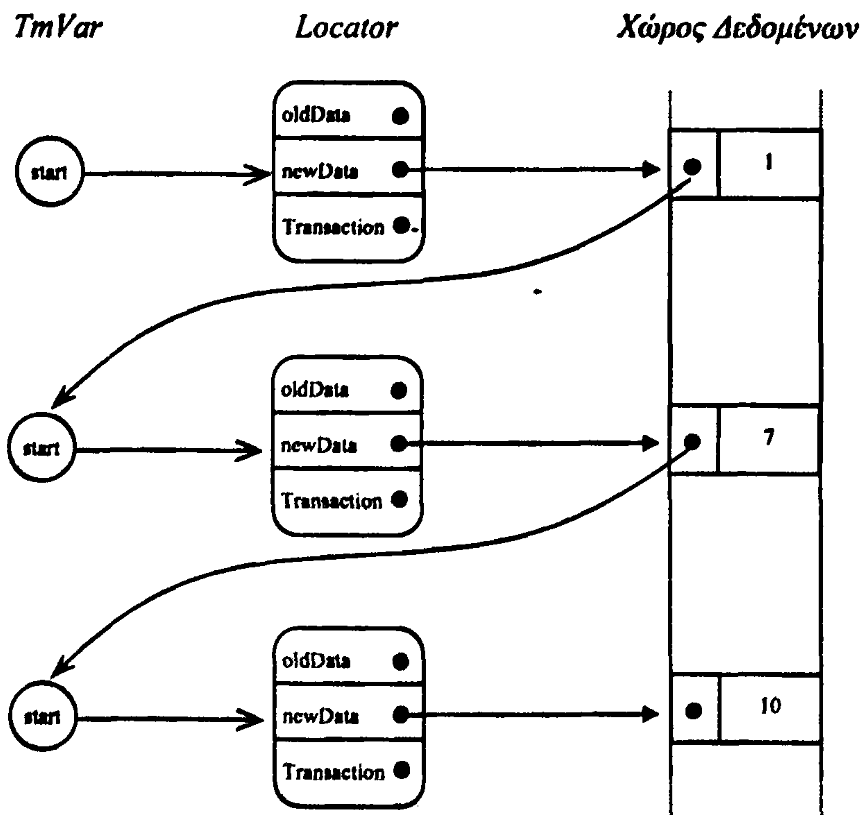
ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

NewReadList() που αρχικοποιεί μία νέα λίστα αναγνώσεων και την επιστρέφει, ii) *Read_List *InsertInReadList(Read_List *readList, TmVar tvar, DATA *tvarVersion, Locator *newLoc)* που εισάγει το στοιχείο tvar με έκδοση tvarVersion και νέο Locator τον newLoc, στη λίστα αναγνώσεων readList, iii) *Read_List *SearchInReadList(Read_List readList, TmVar tvar)* που ερευνά εάν η λίστα readList διατηρεί κάποια καταχώρηση για την tvar και αν αυτό ισχύει επιστρέφει την καταχώρηση αυτή, ενώ σε αντίθετη περίπτωση επιστρέφει null, iv) *Read_List *GetNextItemFromReadList(Read_List *readList)* που σε κάθε κλήση της επιστρέφει με τη σειρά όλα τα στοιχεία της λίστας readList και όταν αυτά τελειώσουν επιστρέφει null, και v) *SortByTMObject(Read_List *readList)* που ταξινομεί τα στοιχεία της λίστας readList με βάση την καθολική διάταξη που ορίζεται μεταξύ των διαμοιραζόμενων αντικειμένων που περιγράφουν. Επίσης για τη λίστα ενημερώσεων writeList υπάρχουν οι συναρτήσεις: i) *Write_List *NewWriteList()*, ii) *Write_List *InsertInWriteList(Write_List *writeList, TmVar tvar, Locator *newLoc)*, iii) *Write_List *SearchInWriteList(Write_List writeList, TmVar tvar)*, και iv) *SortByTMObject(Write_List * writeList)*, με λειτουργικότητα όμοια με αυτή των αντίστοιχων συναρτήσεων της readList. Επιπλέον υπάρχει η συνάρτηση *(Read_List *, Write_List *) GetNextSortedItem (Read_List *readList, Write_List *writeList)* που παίρνει ως όρισμα μία ταξινομημένη λίστα αναγνώσεων readList και μία ταξινομημένη λίστα εγγραφών writeList (οι οποίες έχουν ταξινομηθεί βάσει της καθολικής διάταξης που ορίζεται μεταξύ των διαμοιραζόμενων αντικειμένων που περιγράφουν), και σε κάθε κλήση της επιστρέφει την καταχώρηση του επόμενο στοιχείου, έστω x, στην καθολική διάταξη. Η συνάρτηση αυτή επιστρέφει έναν δείκτη σε στοιχείο Read_List και έναν σε στοιχείο Write_List, διότι για το στοιχείο x μπορεί να υπάρχει καταχώρηση είτε μόνο στην readList οπότε η καταχώρηση αυτή επιστρέφεται μέσω του δείκτη Read_List * και ο δείκτης Write_List * ορίζεται σε null, είτε μόνο στην writeList οπότε η καταχώρηση αυτή επιστρέφεται μέσω του δείκτη Write_List * και ο δείκτης Read_List * ορίζεται σε null, είτε και στις δύο οπότε ορίζονται κατάλληλα και οι δύο δείκτες. Όταν η συνάρτηση GetNextSortedItem επιστρέψει τις καταχωρήσεις όλων των στοιχείων που περιγράφονται από τις λίστες readList και writeList, επιστρέφει την τιμή null και στους δύο δείκτες. Επίσης για την λίστα ενημερώσεων writeList υπάρχει η συνάρτηση *boolean IsEmptyWriteList (Write_List *writeList)* που επιστρέφει TRUE ή

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

FALSE ανάλογα με το εάν η λίστα writeList είναι άδεια ή όχι, αντίστοιχα. Οι παραπάνω λειτουργίες υποστηρίζονται από οποιαδήποτε συμβατή υλοποίηση λίστας και έτσι οι κώδικές τους δεν παρουσιάζονται.

Ο αλγόριθμος NBSTM χρησιμοποιεί τον δείκτη tran του Locator μιας t-μεταβλητής x για την περιγραφή της δοσοληψίας που κατέχει την προσωρινή ιδιοκτησία στη x. Συγκεκριμένα, εάν ο δείκτης tran δείχνει σε κάποια δομή $Trec(i,n_i)$ μιας δοσοληψίας $T(i,n_i)$ της οποίας το status είναι ACTIVE τότε η $T(i,n_i)$ κατέχει την προσωρινή ιδιοκτησία της x. Αν για την $T(i,n_i)$ ισχύει ότι status≠ACTIVE, τότε καμία δοσοληψία δεν κατέχει την προσωρινή ιδιοκτησία της x. Για το λόγο αυτό, λέμε ότι ο NBSTM, όπως και ο DSTM, χρησιμοποιεί ανά t-μεταβλητή ανάθεση προσωρινών ιδιοκτησιών. Παρατηρούμε ότι η πληροφορία κατοχής προσωρινής ιδιοκτησίας σχετίζεται με τη δοσοληψία που κατέχει την προσωρινή ιδιοκτησία, είναι δηλαδή επώνυμη, ένα χαρακτηριστικό που έχουν οι μη-εμποδιστικοί αλγόριθμοι STM.



Σχήμα 6.3: Η μορφή μιας ταξινομημένης κατ' αύξουσα διάταξη συνδεδεμένης λίστας στον NBSTM.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Στο Σχήμα 6.3 παρουσιάζεται η μορφή που θα είχε μια ταξινομημένη κατ' αύξουσα διάταξη συνδεδεμένη λίστα στον NBSTM, που περιέχει τα στοιχεία 1,7,10. Σημειώνεται ότι όπως απαιτεί το αντικείμενο STM, ο NBSTM δεν επιτρέπει να υπάρχουν δείκτες απευθείας μεταξύ των δεδομένων, αλλά ένα δεδομένο d_1 μπορεί να περιέχει δείκτη p σε ένα δεδομένο d_2 μόνο εάν ο p δείχνει στην αντίστοιχη t-μεταβλητή που περιγράφει το d_2. Επίσης, χάριν ευκολίας παρουσίασης του συγκεκριμένου παραδείγματος, υποθέτουμε ότι τα δεδομένα των t-μεταβλητών είναι οι κόμβοι της λίστας και όχι δείκτες προς τους κόμβους αυτούς (όπως συμβαίνει στην πράξη).

Ο αλγόριθμος NBSTM υλοποιείται με βάση το βασικό αντικείμενο STM. Ο NBSTM απαιτεί από τον χρήστη να εκκινήσει μία δοσοληψία εκτελώντας τη λειτουργία BeginTransaction χωρίς κανένα όρισμα. Η λειτουργία BeginTransaction επιστρέφει στο χρήστη ένα δείκτη στη δομή Trec της δοσοληψίας που εκκινήθηκε, τον οποίο ο χρήστης θα πρέπει να περνά ως όρισμα στις υπόλοιπες λειτουργίες που θα εκτελέσει μέσω της συγκεκριμένης δοσοληψίας. Ο κώδικας της λειτουργίας BeginTransaction παρουσιάζεται στο Σχήμα 6.4. Η λειτουργία αυτή δημιουργεί μια νέα δομή Trec (γραμμή 2) για τη δοσοληψία που πρόκειται να εκτελεστεί, δηλώνει ότι το status της είναι ACTIVE (γραμμή 3), αρχικοποιεί την λίστα αναγνώσεών της εκτελώντας τη συνάρτηση NewReadList (γραμμή 4), αρχικοποιεί την λίστα ενημερώσεών της εκτελώντας τη συνάρτηση NewWriteList (γραμμή 5) και επιστρέφει τη νέα αυτή δομή στο χρήστη (γραμμή 6).

```
1   Transaction *BeginTransaction ()
2       Transaction *newTrec = allocMemory(Transaction);
3       newTrec->status = ACTIVE;
4       newTrec->readList = NewReadList();
5       newTrec->writeList = NewWriteList();
6       return (newTrec);
```

Σχήμα 6.4: Κώδικας της λειτουργίας BeginTransaction του DSTM.

Κάθε δοσοληψία $T(i,n_i)$ του αλγορίθμου NBSTM εκτελείται σε δύο στάδια. Κατά το πρώτο στάδιο ο χρήστης που εκτελεί την $T(i,n_i)$, είναι ελεύθερος να διαβάζει και να ενημερώνει τα δεδομένα των t-μεταβλητών που επιθυμεί, εκτελώντας τις λειτουργίες

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ReadTmVar και WriteTmVar, αντίστοιχα, μέσω της $T(i,n_i)$. Το δεύτερο στάδιο εκτέλεσης της δοσοληψίας $T(i,n_i)$ ξεκινά με την κλήση της λειτουργίας CommitTransaction από τον χρήστη. Αρχικά περιγράφεται λεπτομερώς το πρώτο στάδιο εκτέλεσης.

Κατά το πρώτο στάδιο, η $T(i,n_i)$ καταγράφει τις απαραίτητες πληροφορίες για τις ι-μεταβλητές που ο χρήστης επιθυμεί να διαβάσει ή να ενημερώσει, στις αντίστοιχες λίστες που διατηρεί στη δομή της. Σημειώνεται ότι, όπως παρουσιάζεται στη συνέχεια, ο NBSTM χρησιμοποιεί εκ των υστέρων απόκτηση προσωρινών ιδιοκτησιών. Αυτό συνεπάγεται ότι ο NBSTM χρησιμοποιεί τη μέθοδο καθυστερημένης ενημέρωσης των δεδομένων των ι-μεταβλητών. Επομένως, η $T(i,n_i)$ πρέπει να διατηρεί στη λίστα ενημέρωσης, τα νέα δεδομένα κάθε ι-μεταβλητής που ο χρήστης ενημερώνει, ώστε στη συνέχεια να είναι σε θέση η $T(i,n_i)$ να εφαρμόσει τις αλλαγές του χρήστη κατά την εκτέλεση της CommitTransaction.

Σημειώνεται ότι ο NBSTM χρησιμοποιεί μη ορατές αναγνώσεις ι-μεταβλητών και έτσι παρέχει έναν μηχανισμό ελέγχου συνέπειας των δεδομένων των ι-μεταβλητών που αναγνώσθηκαν. Υπενθυμίζεται ότι ο μηχανισμός ελέγχου συνέπειας απαιτεί την ύπαρξη έκδοσης για κάθε δεδομένο. Ο NBSTM, όπως και ο DSTM, εκμεταλλεύεται το κλειστό μοντέλο μνήμης στο οποίο εκτελείται για να ορίσει την έκδοση ενός δεδομένου. Το κλειστό μοντέλο μνήμης επιτρέπει στον αλγόριθμο NBSTM να χρησιμοποιεί ως έκδοση ενός δεδομένου τη διεύθυνση στην οποία ξεκινά η μνήμη στην οποία είναι αποθηκευμένο το δεδομένο αυτό, όπως συζητήθηκε στην Ενότητα 3.5. Συμπεραίνουμε ότι απαιτείται η $T(i,n_i)$ να διατηρεί στη λίστα ανάγνωσης την τρέχουσα έκδοση των δεδομένων κάθε ι-μεταβλητής που ο χρήστης διαβάζει, ώστε να είναι δυνατή η εκτέλεση του μηχανισμού ελέγχου συνέπειας.

Είναι σημαντικό ότι κατά το πρώτο στάδιο της εκτέλεσης μια δοσοληψίας $T(i,n_i)$ δεν εφαρμόζεται καμία αλλαγή στα δεδομένα των ι-μεταβλητών που η $T(i,n_i)$ προσπελάζει για ενημέρωση και επίσης η $T(i,n_i)$ δε μπορεί να εντοπιστεί από οποιαδήποτε άλλη δοσοληψία. Αυτό έρχεται σε αντίθεση με τον τρόπο λειτουργίας του DSTM. Στη συνέχεια περιγράφονται οι υλοποιήσεις των λειτουργιών

ReadTmVar και WriteTmVar από τον NBSTM, που καλούνται κατά το πρώτο στάδιο από τον χρήστη, καθώς και οι συναρτήσεις που αυτές χρησιμοποιούν.

Ο κώδικας της λειτουργίας ReadTmVar παρουσιάζεται στο Σχήμα 6.5. Μέτα την εκκίνηση της δοσοληψίας $T(i,n_i)$ ο χρήστης είναι υπεύθυνος να προσπελάζει μέσω της λειτουργίας ReadTmVar τα δεδομένα κάθε t-μεταβλητής x που επιθυμεί να αναγνώσει, περνώντας ως παράμετρο στην ReadTmVar την αντίστοιχη TmVar *tvar* της x. Στην περίπτωση που η προσπέλαση των δεδομένων γίνει επιτυχώς, η λειτουργία ReadTmVar επιστρέφει την τιμή TRUE και τα δεδομένα της x. Σε αντίθετη περίπτωση, επιστρέφεται μόνο η τιμή FALSE και ο χρήστης είναι υποχρεωμένος να ολοκληρώσει τη δοσοληψία που εκτελεί ως μη-επιτυχημένη, καλώντας μία εκ των λειτουργιών CommitTransaction και AbortTransaction. Σημειώνεται ότι στην περίπτωση αυτή, ακόμα και αν κληθεί η λειτουργία CommitTransaction, ο NBSTM εξασφαλίζει ότι η δοσοληψία θα ολοκληρωθεί ως μη επιτυχημένη.

```
7   (Boolean, DATA) ReadTmVar (Transaction *Trec, TmVar *tvar)
8     DATA *returnData, *curData;
9     Loocator *loc, *newLoc;
10    Read_List *readList = Trec→readList;
11    Read_List *readListItem = SearchInReadList(readList,tvar);
12    Write_List *writeList = Trec→writeList;
13    Write_List *writeListItem = SearchInWriteList(WriteList,tvar);
14    (loc, curData) = GetCurrentData (Trec, tvar);
15    if (readListItem!=null && curData != readListItem→tvarVersion)
16        CAS (Trec→status, ACTIVE, ABORTED);
17        return (FALSE, null);
18    if (writeListItem!=null)
19        returnData = writeListItem→newLoc→newData;
20    else
21        returnData = curData;
22        if (readListItem == null)
23            newLoc = InitializeNewLocator (Trec, curData, null);
24            InsertInReadList (readList, tvar, curData, newLoc);
25    if (Validate(Trec) == TRUE)
26        return (TRUE, *returnData);
27    else
28        CAS (Trec→status, ACTIVE, ABORTED);
29        return (FALSE, null);
```

Σχήμα 6.5: Κώδικας της λειτουργίας ReadTmVar του NBSTM.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Αρχικά, η λειτουργία ReadTmVar προσπαθεί να ανακτήσει τα *τρέχοντα δεδομένα* της tvar, χρησιμοποιώντας τη συνάρτηση GetCurrentData (γραμμή 14). Όπως αναφέρθηκε, ο Locator που περιέχει κάθε t-μεταβλητή x, περιέχει δείκτες προς τα παλιά και τα καινούργια δεδομένα της x. H GetCurrentData επιλέγει τα τρέχοντα δεδομένα της x βάσει του status της δοσοληψία που απέκτησε τελευταία την προσωρινή ιδιοκτησία της x. Εάν το status της δοσοληψίας αυτής είναι COMMITTED, η GetCurrentData επιλέγει τα νέα δεδομένα, ενώ εάν το status της είναι ABORTED, επιλέγει τα παλιά δεδομένα. (Περισσότερες λεπτομέρειες για τη συνάρτηση GetCurrentData παρουσιάζονται στη συνέχεια της παρούσας Ενότητας.)

Μετά την ανάκτηση των τρεχόντων δεδομένων της tvar, η λειτουργία ReadTmVar δηλώνει ότι τα δεδομένα που πρόκειται να επιστραφούν στο χρήστη, εάν αυτή ολοκληρωθεί επιτυχώς, είναι τα τρέχοντα δεδομένα της tvar (γραμμή 21), που έχουν ήδη ανακτηθεί. Σημειώνεται ότι παραβλέπουμε προσωρινά τις γραμμές 15 έως 19, η χρησιμότητα των οποίων παρουσιάζεται στη συνέχεια της παρούσας Ενότητας. Εάν είναι η πρώτη φορά που προσπελάσετε μέσω της ReadTmVar η συγκεκριμένη t-μεταβλητή, πρέπει να καταχωρηθούν οι κατάλληλες πληροφορίες για αυτή στη λίστα αναγνώσεων readList της $T(i,n_i)$. Σημειώνεται ότι η ReadTmVar έχει ήδη ψάξει στην readList εάν υπάρχει κάποια καταχώρηση *readListItem* για την tvar (γραμμές 10 και 11), με τη βοήθεια της συνάρτησης SearchInReadList, η οποία αν δεν υπάρχει τέτοια καταχώρηση επιστρέφει την τιμή null. Έτσι, εάν δεν υπάρχει το readListItem (γραμμή 22), τότε η $T(i,n_i)$ καλεί τη συνάρτηση InitializeNewLocator (γραμμή 23) με ορίσματα $Trec(i,n_i)$, τα τρέχοντα δεδομένα της tvar και null, ώστε να δημιουργηθεί και να αρχικοποιηθεί κατάλληλα ένας νέος Locator. Σημειώνεται ότι αυτός ο νέος Locator θα χρησιμοποιηθεί για να αποκτηθεί η προσωρινή ιδιοκτησία της tvar, όπως θα παρουσιαστεί στη συνέχεια της παρούσας Ενότητας. Ο κώδικας της συνάρτησης InitializeNewLocator παρουσιάζεται στο Σχήμα 6.6 και παρατηρούμε ότι δημιουργείται ένας νέος Locator newLoc (γραμμή 31), ιδιοκτήτης του συγκεκριμένου Locator ορίζεται η $T(i,n_i)$ (το newLoc→tran αρχικοποιείται σε $Trec(i,n_i)$ στη γραμμή 32) και ο δείκτης παλιών δεδομένων του newLoc αρχικοποιείται ώστε να δείχνει στα τρέχοντα δεδομένα της tvar (γραμμή 33). Στη συνέχεια, επειδή ισχύει newdata == null, αρχικοποιείται ο δείκτης των νέων δεδομένων του newLoc ώστε να δείχνει

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

επίσης στα τρέχοντα δεδομένα της tvar (γραμμή 38). Η συνάρτηση InitializeNewLocator επιστρέφει στην ReadTmVar αυτό τον δείκτη newLoc, ο οποίος μαζί με την tvar και τα τρέχοντα δεδομένα της καταχωρείται στη λίστα αναγνώσεων της $T(i,n_i)$ (γραμμή 24).

Στο σημείο αυτό εξηγείται η χρησιμότητα των γραμμών 15 έως 19. Στη διάρκεια της εκτέλεσης μιας δοσοληψίας $T(i,n_i)$ σε μια εκτέλεση α του αλγορίθμου NBSTM, είναι πιθανό ο χρήστης να καλέσει τη λειτουργία ReadTmVar για κάποια t-μεταβλητή x. Διακρίνω περιπτώσεις ανάλογα με το εάν η λειτουργία αυτή είναι καθολική προσπέλαση για ανάγνωση (δηλαδή δεν προηγείται εκτέλεση λειτουργίας ενημέρωσης της x) ή όχι. Εάν είναι καθολική ανάγνωση, θα έχει ως αποτέλεσμα να δημιουργηθεί μια καταχώρηση $listItem_x$ για το x στην $Trec(i,n_i)$→readList. Στη συνέχεια ο χρήστης υπάρχει περίπτωση να καλέσει τη λειτουργία ReadTmVar και πάλι για την x. Στην περίπτωση αυτή, πριν την επιστροφή των δεδομένων της x στο χρήστη πρέπει να ελεγχθεί ότι τα δεδομένα που περιέχονται στο $listItem_x$→tvarVersion είναι συνεπή, δηλαδή δεν έχουν τροποποιηθεί, ώστε η εκτέλεση της $T(i,n_i)$ στην α να είναι σειριοποιήσιμη σε κάθε περίπτωση. Για το λόγο αυτό, ελέγχεται αν τα αποθηκευμένα δεδομένα $listItem_x$→tvarVersion της x στη $Trec(i,n_i)$.readList είναι διαφορετικά από τα τρέχοντα δεδομένα της (γραμμή 15) και εάν αυτό ισχύει, η $T(i,n_i)$ χαρακτηρίζεται ως μη-επιτυχημένη (γραμμή 16) και επιστρέφεται η τιμή FALSE στο χρήστη (γραμμή 17).

```
30 Locator *InitializeNewLocator (Transaction *Trec, DATA *curData, DATA newData)
31    Locator *newLoc = allocMemory (Locator);
32    newLoc→tran = Trec;
33    newLoc→oldData = curData;
34    if (newData != null)
35        newLoc→newData = allocMemory (DATA);
36        *(newLoc→newData) = newData;
37    else
38        newLoc→newData = curData;
39    return (newLoc);
```

Σχήμα 6.6: Κώδικας της συνάρτησης InitializeNewLocator του NBSTM.

Εάν η λειτουργία ReadTmVar δεν είναι καθολική ανάγνωση, σημαίνει ότι η $T(i,n_i)$ έχει ήδη εκτελέσει κάποια λειτουργία ενημέρωσης, δηλαδή κάποια WriteTmVar, της x. Όπως θα περιγραφεί κατά την παρουσίαση της WriteTmVar, στην περίπτωση αυτή θα έχει δημιουργηθεί μια καταχώρηση $listItem_x$ για την x στη $Trec(i,n_i)$→writeList. Η ReadTmVar πρέπει να επιστρέψει τα δεδομένα που έχουν γραφεί από προηγούμενη ενημέρωση της $T(i,n_i)$ στη x και περιέχονται στην $listItem_x$, συγκεκριμένα στο σημείο $listItem_x$→newLoc→newData. Σημειώνεται ότι η ReadTmVar έχει ήδη ψάξει στην writeList της $T(i,n_i)$ εάν υπάρχει κάποια καταχώρηση *writeListItem* για την x (γραμμές 12 και 13), με τη βοήθεια της συνάρτησης SearchInWriteList, η οποία αν δεν υπάρχει τέτοια καταχώρηση επιστρέφει την τιμή null. Έτσι, εάν δεν υπάρχει η writeListItem (γραμμή 18), τότε η $T(i,n_i)$ δηλώνει ότι τα δεδομένα που πρόκειται να επιστραφούν στο χρήστη, εάν αυτή ολοκληρωθεί επιτυχώς, είναι τα writeListItem→newLoc→newData (γραμμή 19).

Στο τέλος της εκτέλεσης της ReadTmVar, εξετάζεται εάν τα δεδομένα που η δοσοληψία $T(i,n_i)$ έχει προσπελάσει για ανάγνωση και περιέχονται στην readList της $T(i,n_i)$ είναι συνεπή, εκτελώντας τη συνάρτηση Validate (γραμμή 25). Η συνάρτηση αυτή επιστρέφει TRUE εάν τα δεδομένα που διάβασε η $T(i,n_i)$ είναι ακόμα συνεπή και FALSE σε αντίθετη περίπτωση. Εάν η Validate επιστρέψει TRUE, τότε η ReadTmVar επιστρέφει την τιμή επιτυχίας TRUE και τα δεδομένα που έχουν ήδη οριστεί ότι πρόκειται να επιστραφούν στον χρήστη (γραμμή 26), ενώ εάν η Validate επιστρέψει FALSE, η ReadTmVar επιστρέφει στο χρήστη την τιμή αποτυχίας FALSE (γραμμή 29). Σημειώνεται ότι η συνάρτηση Validate υλοποιεί τον μηχανισμό ελέγχου συνέπειας των δεδομένων που προσπελάστηκαν για ανάγνωση από την $T(i,n_i)$. Υπενθυμίζεται ότι ο έλεγχος συνέπειας αρκεί να εκτελεστεί για τις t-μεταβλητές που έχουν προσπελαστεί την πρώτη φορά με καθολική ανάγνωση, όπως συζητήθηκε στην Ενότητα 3.1.3. Σημειώνεται ότι από τον κώδικα (γραμμές 12, 13, 18, 19, 20 και 24) προκύπτει ότι η ιδιότητα αυτή εξασφαλίζεται διότι και πάλι θα υπάρχει κάποια καταχώρηση της x στη writeList της $T(i,n_i)$, εξαιτίας της υλοποίησης της λειτουργίας WriteTmVar που περιγράφεται στη συνέχεια της παρούσας Ενότητας. Σημειώνεται επίσης, ότι ο μηχανισμός ελέγχου συνέπειας εκτελείται από τον ίδιο τον αλγόριθμο NBSTM και κάθε φορά που ο χρήστης προσπελάζει για ανάγνωση μια t-μεταβλητή.

Επομένως ο αλγόριθμος NBSTM χρησιμοποιεί αυτόματο και αυξητικό έλεγχο συνέπειας.

Στη συνέχεια περιγράφεται η συνάρτηση GetCurrentData, ο κώδικας της οποίας παρουσιάζεται στο Σχήμα 6.7. Η συνάρτηση αυτή χρησιμοποιείται για την ανάκτηση των τρεχόντων δεδομένων μιας t-μεταβλητής. Όπως αναφέρθηκε, ο Locator που περιγράφει κάθε t-μεταβλητή x, περιέχει δείκτες προς τα παλιά και τα καινούργια δεδομένα της x. Τα τρέχοντα δεδομένα της x καθορίζονται με βάση το status της δοσοληψίας που απέκτησε τελευταία την προσωρινή ιδιοκτησία της x. Έστω $T(j,n_j)$ η δοσοληψία που απέκτησε τελευταία την προσωρινή ιδιοκτησία της x και έστω $T(i,n_i)$ μία δοσοληψία που θέλει να εξακριβώσει τα τελευταία δεδομένα του x. Αν η κατάσταση της $T(j,n_j)$ είναι COMMITTED, τότε τα τρέχοντα δεδομένα της x είναι τα νέα δεδομένα (γραμμή 53), ενώ εάν είναι ABORTED είναι τα παλιά δεδομένα (γραμμή 54). Σημειώνεται ότι και στις δύο αυτές περιπτώσεις σημαίνει ότι η $T(j,n_j)$ έχει καταργήσει την προσωρινή ιδιοκτησία της x.

```
40 (Locator, DATA *) GetCurrentData (Transaction *Trec, TmVar *tvar)
41   Locator *loc;
42   int status;
43   loc = tvar→start;
44   if (loc→tran == Trec)
45       return (loc, loc→newData);
46   status = loc→tran→status;
47   if (status==COMMITTED) return (loc, loc→newData);
48   else if (status==ABORTED) return (loc, loc→oldData);
49   else if (loc→oldData == loc→newData)
50       return (loc, loc→oldData);
51   else
52       TryHelpTransaction (loc→tran);
53       if (loc→tran→status == COMMITTED)
54           return (loc, loc→newData);
55       else return (loc, loc→oldData);
```

Σχήμα 6.7: Κώδικας της συνάρτησης GetCurrentData του NBSTM.

Όπως θα αναφερθεί στη συνέχεια της παρούσας Ενότητας, ο αλγόριθμος NBSTM αποκτά τις ιδιοκτησίες όλων των t-μεταβλητών που προσπέλασε είτε για ανάγνωση (δηλαδή χρησιμοποιεί ορατές αναγνώσεις t-μεταβλητών) είτε μόνο για ενημέρωση.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Αυτό σημαίνει ότι ο αλγόριθμος NBSTM, αποκτά προσωρινές ιδιοκτησίες ενημέρωσης και προσωρινές ιδιοκτησίες ανάγνωσης. Σημειώνεται ότι, ο NBSTM επιτρέπει στις δοσοληψίες να διακρίνουν τις δύο αυτές προσωρινές ιδιοκτησίες, όπως περιγράφεται στη συνέχεια. Σημειώνεται ότι σε έναν Locator loc μιας t-μεταβλητής x, ο οποίος περιγράφει την προσωρινή ιδιοκτησία κάποιας δοσοληψίας, ο δείκτης loc.oldData θα δείχνει στα παλιά δεδομένα της x και ο δείκτης loc.newData θα δείχνει στα νέα δεδομένα της x. Επειδή τα παλιά και τα νέα δεδομένα της x αποθηκεύονται σε διαφορετικές θέσεις της μνήμης, προκύπτει ότι οι δείκτες loc.oldData και loc.newData θα έχουν διαφορετικές τιμές, εάν ο συγκεκριμένος Locator loc περιγράφει κάποια ενημέρωση της x ή αντίστοιχα κάποια προσωρινή ιδιοκτησία ενημέρωσης. Εκμεταλλευόμαστε αυτή την ιδιότητα και ορίζουμε ότι ένας Locator loc που περιγράφει κάποια προσωρινή ιδιοκτησία ανάγνωσης θα έχει τους δείκτες loc.oldData και loc.newData να δείχνουν και οι δύο στα ίδια δεδομένα, συγκριμένα στα τρέχοντα δεδομένα της x και άρα θα έχουν την ίδια τιμή. Σημειώνεται ότι η ιδιότητα αυτή, εξασφαλίζεται από τον κώδικα της ReadTmVar (γραμμές 24, 33, 37 και 38) για τις t-μεταβλητές που αναγνώσθηκαν.

Με βάση τα παραπάνω, κατά την εκτέλεση της GetCurrentData εάν η κατάστασή της $T(j,n_j)$ είναι ACTIVE και η $T(j,n_j)$ κατέχει προσωρινή ιδιοκτησία ανάγνωσης στο x (γραμμή 49), τότε τα τρέχοντα δεδομένα της x είναι τα παλιά δεδομένα (τα οποία είναι ίδια με τα νέα δεδομένα) που δεικτοδοτούνται μέσω του Locator της x (γραμμή 50). Εάν η κατάστασή της $T(j,n_j)$ είναι ACTIVE και η $T(j,n_j)$ κατέχει προσωρινή ιδιοκτησία ενημέρωσης στη x (γραμμή 51) τότε δεν μπορεί να γίνει αποτίμηση των τρεχόντων δεδομένων της x από την $T(i,n_i)$ πριν την ολοκλήρωση της $T(j,n_j)$ ως επιτυχημένης ή μη-επιτυχημένης. Ο NBSTM διαθέτει έναν *μηχανισμό βοήθειας* με τον οποίο μία δοσοληψία μπορεί να βοηθήσει κάποια άλλη να ολοκληρωθεί. Ο μηχανισμός βοήθειας εγγυάται ότι εάν μια δοσοληψία T_1 βοηθήσει κάποια άλλη δοσοληψία T_2 να ολοκληρωθεί, τότε όταν η T_1 θα ολοκληρώσει την βοήθεια της προς την T_2, η T_2 θα έχει ολοκληρωθεί είτε ως επιτυχημένη είτε ως μη-επιτυχημένη, όπως αποδεικνύεται στην Ενότητα 6.2. Ο μηχανισμός βοήθειας του NBSTM περιγράφεται λεπτομερώς στη συνέχεια της παρούσας Ενότητας. Έτσι, η $T(i,n_i)$ προσπαθεί να βοηθήσει την $T(j,n_j)$ να ολοκληρωθεί εκτελώντας τη συνάρτηση TryHelpTransaction (γραμμή 52) του μηχανισμού βοήθειας. Ο κώδικας της λειτουργίας

TryHelpTransaction παρουσιάζεται στο Σχήμα 6.8 και παρατηρείται ότι καλείται η συνάρτηση PerformTrans (η οποία περιγράφεται παρακάτω στην παρούσα Ενότητα), μόνο εάν η κατάσταση της $T(j,n_j)$ είναι ενεργή (γραμμή 57), με όρισμα $Trec(j,n_j)$, ώστε να βοηθηθεί η $T(j,n_j)$. Όπως αναφέρθηκε, ο αλγόριθμος NBSTM εγγυάται ότι μετά την ολοκλήρωση της TryHelpTransaction η $T(j,n_j)$ θα έχει ολοκληρωθεί είτε επιτυχώς ή μη-επιτυχώς και έτσι πραγματοποιούνται ενέργειες αντίστοιχες με πριν (γραμμές 53 έως 55).

```
56    TryHelpTransaction (Transaction *Trec)
57       if (Trec→status==ACTIVE)
58          PerformTrans(Trec);
```

Σχήμα 6.8: Κώδικας της συνάρτησης TryHelpTransaction του NBSTM.

Σημειώνεται ότι εάν η δοσοληψία $T(i,n_i)$ προσπαθεί να μάθει τα τρέχοντα δεδομένα μιας t-μεταβλητής x της οποίας κατέχει την προσωρινή ιδιοκτησία (γραμμή 50), τότε τα τρέχοντα δεδομένα της x είναι τα νέα δεδομένα και επιστρέφονται (γραμμή 51). Σημειώνεται ότι με βάση τα παραπάνω προκύπτει ότι σε κάθε κλήση της η συνάρτηση GetCurrentData επιστρέφει τα τρέχοντα δεδομένα μιας t-μεταβλητής x που δεικτοδοτούνται από κάποιον Locator, ο οποίος μετά την επιστροφή της GetCurrentData είτε περιγράφει κάποια ολοκληρωμένη δοσοληψία, είτε περιγράφει κάποια ενεργή δοσοληψία που κατέχει προσωρινή ιδιοκτησία ανάγνωσης στη x.

Στη συνέχεια περιγράφεται η συνάρτηση Validate, ο κώδικας της οποίας παρουσιάζεται στο Σχήμα 6.9. Η συνάρτηση Validate χρησιμοποιείται από τον NBSTM για να ελέγξει τη συνέπεια των δεδομένων των t-μεταβλητών που μια δοσοληψία $T(i,n_i)$ έχει προσπελάσει για ανάγνωση, υλοποιεί δηλαδή το μηχανισμό ελέγχου συνέπειας. Αρχικά ελέγχεται η τιμή του $Trec(i,n_i)$.status (γραμμή 65) και εάν η τιμή του είναι ABORTED, η Validate επιστρέφει την τιμή αποτυχίας FALSE (γραμμή 66). Σε αντίθετη περίπτωση, διατρέχονται όλες οι t-μεταβλητές που περιέχονται στη λίστα $Trec(i,n_i)$.readList (γραμμή 67), με τη βοήθεια της συνάρτησης GetNextItemFromReadList. Για κάθε στοιχείο item της λίστας αυτής; i) ανακτώνται τα τρέχοντα δεδομένα (curData) της t-μεταβλητής item.tvar (γραμμή 68), με τη βοήθεια της συνάρτησης GetCurrentData, και ii) ελέγχεται εάν τα τρέχοντα δεδομένα

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

της είναι τα ίδια με αυτά που αναγνώσθηκαν και περιγράφονται από το στοιχείο item της λίστας, μέσω του δείκτη item→ tvarVersion (γραμμή 69). Η συνάρτηση Validate επιστρέφει TRUE εάν όλα τα στοιχεία της readList είναι συνεπή, δηλαδή εάν τα δεδομένα των t-μεταβλητών που η $T(i,n_i)$ προσπέλασε με καθολική ανάγνωση είναι συνεπή, και FALSE σε αντίθετη περίπτωση.

```
59 Boolean Validate (Transaction *Trec)
60    Locator *loc;
61    DATA *curData;
62    Read_List *item, *readList = Trec→readList;
63    if (Trec→status == ABORTED)
64       return (FALSE);
65    while ((item = GetNextItemFromReadList(readList)) != null)
66       (loc,curData) = GetCurrentData (Trec,item→tvar);
67       if (curData != item→tvarVersion)
68          return (FALSE);
69    return (TRUE);
```

Σχήμα 6.9: Κώδικας της συνάρτησης Validate του NBSTM.

Στη συνέχεια περιγράφεται η λειτουργία WriteTmVar, ο κώδικας της οποίας παρουσιάζεται στο Σχήμα 6.10. Μέτα την εκκίνηση της δοσοληψίας $T(i,n_i)$ ο χρήστης είναι υπεύθυνος να ενημερώσει μέσω της λειτουργίας WriteTmVar τα δεδομένα κάθε t-μεταβλητής x που επιθυμεί, περνώντας ως παράμετρο στην ReadTmVar την αντίστοιχη TmVar *tvar* της x και τα δεδομένα *data* με τα οποία επιθυμεί να ενημερώσει τη x. Αρχικά, η WriteTmVar εκτελεί τη συνάρτηση InitializeNewLocator, ώστε να δημιουργήσει και να αρχικοποιήσει κατάλληλα έναν νέο Locator. Σημειώνεται ότι αυτός ο νέος Locator θα χρησιμοποιηθεί για να αποκτηθεί η προσωρινή ιδιοκτησία της tvar, όπως θα παρουσιαστεί στη συνέχεια της παρούσας Ενότητας. Ο κώδικας της συνάρτησης InitializeNewLocator παρουσιάζεται στο Σχήμα 6.6 και παρατηρούμε ότι δημιουργείται ένας νέος Locator newLoc (γραμμή 31), ιδιοκτήτης του συγκεκριμένου Locator ορίζεται η $T(i,n_i)$ (το newLoc.tran αρχικοποιείται σε $Trec(i,n_i)$ στη γραμμή 32) και ο δείκτης παλιών δεδομένων του newLoc αρχικοποιείται ώστε να δείχνει στα τρέχοντα δεδομένα της tvar (γραμμή 33), τα οποία είναι null, διότι η WriteTmVar δεν τα γνωρίζει. Στη συνέχεια, επειδή ισχύει newdata≠null, αρχικοποιείται ο δείκτης των νέων δεδομένων

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

του newLoc (αφού δεσμευθεί η απαραίτητη μνήμη στη γραμμή 35) ώστε να δείχνει στη διεύθυνση που είναι αποθηκευμένα τα δεδομένα data (γραμμή 36) με τα οποία ο χρήστης θέλει να ενημερώσει την tvar. Η συνάρτηση InitializeNewLocator επιστρέφει στην WriteTmVar αυτό τον δείκτη newLoc, ο οποίος μαζί με την tvar καταχωρείται στη λίστα ενημερώσεων της $T(i,n_i)$ (γραμμή 78). Σημειώνεται ότι ο νέος αυτός Locator θα χρησιμοποιηθεί για την απόκτηση προσωρινής ιδιοκτησίας ενημέρωσης, διότι οι δείκτες newLoc.oldData και newLoc.newData είναι διαφορετικοί. Σημειώνεται ότι η διεύθυνση στην οποία είναι αποθηκευμένα τα νέα δεδομένα data της x, διαφέρει από τη διεύθυνση στην οποία είναι αποθηκευμένα τα παλιά δεδομένα της x (που η WriteTmVar δε γνωρίζει). Αυτό είναι σημαντικό, διότι συμπεραίνουμε ότι οι δείκτες newLoc.newData και newLoc.oldData, όταν αυτός (ο newLoc.oldData) αρχικοποιηθεί (συγκεκριμένα κατά την εκτέλεση της CommitTransaction, όπως θα περιγραφεί), θα είναι διαφορετικοί και άρα πράγματι ο newLoc θα περιγράφει μια προσωρινή ιδιοκτησία ενημέρωσης, όπως απαιτείται.

```
70  (Boolean, DATA *) WriteTmVar (Transaction *Trec, TmVar *tvar, DATA data)
71      Loocator *newLoc;
72      Write_List *writeList = Trec→writeList;
73      Write_List *writeListItem = SearchInWriteList(writeList, tvar);
74      if (writeListItem!=null)
75          *(writeListItem→newLoc→newData) = data;
76      else
77          newLoc = InitializeNewLocator (Trec, null, data);
78          InsertInWriteList (writeList, tvar, newLoc);
79      return (TRUE);
```

Σχήμα 6.10: Κώδικας της λειτουργίας WriteTmVar του NBSTM.

Σημειώνεται ότι η λειτουργία WriteTmVar επιτρέπεται να εκτελεστεί πολλαπλές φορές για τη tvar από την $T(i,n_i)$. Στην περίπτωση αυτή, αρκεί να δημιουργηθεί την πρώτη φορά μία καταχώρηση για την tvar στης writeList της $T(i,n_i)$ και στη συνέχεια αρκεί να ενημερώνεται η συγκεκριμένη καταχώρηση. Για το λόγο αυτό, αρχικά η WriteTmVar αναζητεί κάποια καταχώρηση της tvar στην writeList της $T(i,n_i)$ με τη βοήθεια της συνάρτησης SearchInWriteList (γραμμή 73), και εάν η καταχώρηση αυτή υπάρχει (γραμμή 74) ενημερώνεται ο δείκτης newLoc→newData ώστε να δείχνει στα νέα δεδομένα data (γραμμή 75). Σημειώνεται ότι σε κάθε περίπτωση η λειτουργία

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

WriteTmVar του NBSTM επιστρέφει την τιμή TRUE. Επειδή η WriteTmVar δεν αποκτά προσωρινή ιδιοκτησία στη tvar, ούτε ενημερώνει άμεσα τα πραγματικά δεδομένα της tvar, προκύπτει ότι ο NBSTM πρέπει να χρησιμοποιεί εκ των υστέρων απόκτηση ιδιοκτησιών και ετεροχρονισμένη ενημέρωση των t-μεταβλητών, όπως παρουσιάζεται στη συνέχεια.

Στη συνέχεια περιγράφεται η λειτουργία CreateNewTmVar του DSTM, ο κώδικας της οποίας παρουσιάζεται στο Σχήμα 6.11. Υπενθυμίζεται ότι ο χρήστης εκτελεί τη συγκεκριμένη λειτουργία κάθε φορά που επιθυμεί να δημιουργήσει μια νέα t-μεταβλητή και να την αρχικοποιήσει με τα δεδομένα *data*, κατά τη διάρκεια εκτέλεσης μιας δοσοληψίας $T(i,n_i)$. Η CreateNewTmVar δημιουργεί μια καινούργια t-μεταβλητή *newTvar* (γραμμή 73) και έναν καινούργιο Locator *newLoc* (74). Στη συνέχεια αρχικοποιεί τον newLoc έτσι ώστε ο δείκτης των παλιών δεδομένων του να είναι null (γραμμή 75), ο δείκτης των νέων δεδομένων του (αφού δεσμευθεί η απαραίτητη μνήμη στη γραμμή 76) να δείχνει στα δεδομένα data (γραμμή 77) και ο κάτοχος της ιδιοκτησίας της newtvar να είναι η $T(i,n_i)$ (γραμμή 78). Έπειτα, αρχικοποιείται ο δείκτης start της newTvar ώστε να δείχνει στον newLoc (γραμμή 79) και επιστρέφεται (γραμμή 80).

```
80    TmVar *CreateNewTmVar (Transaction *Trec, DATA data)
81       TmVar *newTvar = allocMemory(TmVar);
82       Locator *newLoc = allocMemory (Locator);
83       newLoc→oldData = null;
84       newLoc→newData = allocMemory(DATA);
85       *(newLoc→newData) = data;
86       newLoc→tran = Trec;
87       newTvar→start = newLoc;
88       return (newTvar);
```

Σχήμα 6.11: Κώδικας της λειτουργίας CreateNewTmVar του NBSTM.

Μόλις ο χρήστης επιθυμεί να ολοκληρώσει τη δοσοληψία $T(i,n_i)$, πρέπει να καλέσει τη λειτουργία CommitTransaction ή τη λειτουργία AbortTransaction για να γίνει προσπάθεια ολοκλήρωσης της $T(i,n_i)$ ως επιτυχημένης ή μη-επιτυχημένης, αντίστοιχα. Στο σημείο αυτό, ορίζεται το τέλος του 1[ου] σταδίου και η έναρξη του 2[ου] σταδίου εκτέλεσης της $T(i,n_i)$, με την $T(i,n_i)$ να γνωρίζει τις t-μεταβλητές που ο

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

χρήστης έχει προσπελάσει για ανάγνωση και ενημέρωση, μαζί με τις ενημερώσεις που επιθυμεί να εφαρμόσει σε αυτές. Εάν κληθεί η λειτουργία AbortTransaction, τότε το δεύτερο στάδιο είναι πολύ μικρό και περιλαμβάνει απλά τον ορισμό του status της $T(i,n_i)$ σε ABORTED. Ο κώδικας της λειτουργίας AbortTransaction παρουσιάζεται στο Σχήμα 6.12. Παρατηρούμε ότι η τιμή ABORTED ορίζεται στο status της $T(i,n_i)$ με την εκτέλεση της λειτουργίας CAS της γραμμής 90. Σημειώνεται ότι η συγκεκριμένη λειτουργία είναι αρκετή ώστε να εγγυηθεί τον ορισμό του status της δοσοληψίας σε ABORTED, διότι όπως αποδεικνύεται στην Ενότητα 6.2, η συγκεκριμένη λειτουργία CAS εκτελείται σε κάθε περίπτωση επιτυχώς (επειδή καμία άλλη δοσοληψία δε μπορεί να γνωρίζει την ύπαρξη της δοσοληψίας $T(i,n_i)$).

```
89    AbortTransaction (Transaction *Trec)
90        CAS (Trec→status, ACTIVE, ABORTED);
```

Σχήμα 6.12: Κώδικας της λειτουργίας AbortTransaction του NBSTM.

Εάν ο χρήστης καλέσει την λειτουργία CommitTransaction, σημαίνει ότι επιθυμεί να εφαρμόσει τις αλλαγές στις t-μεταβλητές που ενημέρωσε και στη συνέχεια περιγράφεται συνοπτικά η διαδικασία που ακολουθείται στην περίπτωση αυτή. Για να γίνει με σωστό τρόπο η ενημέρωση, κατά το 2° στάδιο της εκτέλεσής της η δοσοληψία $T(i,n_i)$ προσπαθεί αρχικά να αποκτήσει τις προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που ο χρήστης προσπέλασε είτε για ανάγνωση είτε για ενημέρωση, οι οποίες περιέχονται στις λίστες ανάγνωσης και ενημέρωσης που η $T(i,n_i)$ διατηρεί, και στη συνέχεια να ενημερώσει τις t-μεταβλητές. Επίσης για κάθε t-μεταβλητή x για την οποία διατηρεί εγγραφή readListItem στην λίστα ανάγνωσής της, πρέπει να εξασφαλίσει ότι τα δεδομένα που διατηρούνται στο readListItem για τη x είναι συνεπή, πριν την απόκτηση της ιδιοκτησίας της x. Εάν η δοσοληψία καταφέρει να αποκτήσει προσωρινές ιδιοκτησίες σε όλες τις t-μεταβλητές που επιθυμεί τότε χαρακτηρίζεται ως επιτυχημένη, εφαρμόζει τις ενημερώσεις που επιθυμεί και καταργεί αυτές τις ιδιοκτησίες. Είναι σημαντικό ότι η διαδικασία ενημέρωσης και κατάργησης γίνεται με την εκτέλεση μίας μόνο λειτουργίας CAS. Εάν η δοσοληψία δεν καταφέρει να αποκτήσει τις προσωρινές ιδιοκτησίες όλως των t-μεταβλητών που επιθυμεί, τότε χαρακτηρίζεται ως αποτυχημένη και καταργεί τις ιδιοκτησίες των t-

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

μεταβλητών που έχει αποκτήσει (και πάλι με την εκτέλεση μίας μόνο λειτουργίας CAS). Στη συνέχεια περιγράφεται λεπτομερώς το δεύτερο στάδιο εκτέλεσης της δοσοληψίας, εάν ο χρήστης καλέσει τη λειτουργία CommitTransaction.

Ο κώδικας της λειτουργίας CommitTransaction παρουσιάζεται στο Σχήμα 6.13. Η λειτουργία αυτή, ελέγχει αρχικά εάν τα δεδομένα των t-μεταβλητών που προσπέλασε η δοσοληψία $T(i,n_i)$ για ανάγνωση είναι συνεπή, εκτελώντας τη συνάρτηση Validate (γραμμή 92). Εάν τα δεδομένα αυτά δεν είναι συνεπή, η $T(i,n_i)$ χαρακτηρίζεται ως αποτυχημένη (γραμμή 100) και επιστρέφεται η τιμή FALSE (γραμμή 101). Εάν τα δεδομένα αυτά είναι συνεπή τότε ταξινομεί τις λίστες ανάγνωσης και ενημέρωσης της $T(i,n_i)$ (γραμμές 96 και 97), χρησιμοποιώντας τις συναρτήσεις SortReadListByTMObj και SortWriteListByTMObj, αντίστοιχα, καλεί τη συνάρτηση PerformTrans και επιστρέφει ότι επιστρέψει η συνάρτηση αυτή (γραμμή 98). Η συνάρτηση PerformTrans επιστρέφει TRUE εάν η $T(i,n_i)$ ολοκληρώθηκε επιτυχώς και FALSE σε αντίθετη περίπτωση. Σημειώνεται ότι στην περίπτωση που η εκτέλεση της συνάρτησης Validate της γραμμής 92 επιστρέψει TRUE και η λίστα ενημερώσεων της $T(i,n_i)$ είναι κενή (γραμμή 93), η $T(i,n_i)$ χαρακτηρίζεται ως επιτυχημένη (γραμμή 94) και γίνεται επιστροφή της τιμής TRUE (γραμμή 95), ενημερώνοντας με αυτό τον τρόπο τον χρήστη ότι τα δεδομένα που διάβασε είναι συνεπή.

```
91  Boolean CommitTransaction (Transaction *Trec)
92     if (Validate(Trec)==TRUE)
93         if (IsEmptyWriteList (Trec→writeList) == TRUE)
94             CAS (Trec→status,ACTIVE,COMMITTED);
95             return (TRUE);
96         SortReadListByTMObj (Trec→readList);
97         SortWriteListByTMObj (Trec→writeList);
98         return (PerformTrans(Trec));
99     else
100        CAS (Trec→status, ACTIVE, ABORTED);
101        return (FALSE);
```

Σχήμα 6.13: Κώδικας της λειτουργίας CommitTransaction του NBSTM.

Στη συνέχεια περιγράφεται η συνάρτηση PerformTrans ο κώδικας της οποίας παρουσιάζεται στο Σχήμα 6.14. Αρχικά εκτελείται η συνάρτηση AcquireOwnerships

(γραμμή 104), η οποία προσπαθεί να αποκτήσει τις προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που η $T(i,n_i)$ έχει προσπελάσει. Όταν η συνάρτηση AcquireOwnerships επιστρέψει είτε θα έχουν αποκτηθεί όλες οι επιθυμητές προσωρινές ιδιοκτησίες ή όχι και η AcquireOwnerships εγγυάται ότι το status της $T(i,n_i)$ θα έχει τιμή ACTIVE ή ABORTED, αντίστοιχα. Έτσι εάν το status της $T(i,n_i)$ έχει τιμή ACTIVE (γραμμή 106) μετά την επιστροφή της AcquireOwnerships, η $T(i,n_i)$ ορίζεται ως επιτυχημένη (γραμμή 107) και επιστρέφεται η τιμή TRUE στο χρήστη (γραμμή 108). Ο χαρακτηρισμός της $T(i,n_i)$ ως επιτυχημένης έχεις ως αποτέλεσμα να ενημερωθούν τα δεδομένα των διαμοιραζόμενων αντικειμένων που ο χρήστης επιθυμεί και να καταργηθούν οι προσωρινές ιδιοκτησίες που η $T(i,n_i)$ κατέχει. Σημειώνεται ότι ο χαρακτηρισμός της $T(i,n_i)$ ως επιτυχημένης πραγματοποιείται με την εκτέλεση της λειτουργίας CAS της γραμμής 107, με την οποία ορίζεται η τιμή COMMITTED στο status της $T(i,n_i)$. Όπως αποδεικνύεται στην Ενότητα 6.2, η συγκεκριμένη λειτουργία CAS δε μπορεί να αποτύχει και επομένως είναι αρκετή για τον χαρακτηρισμό της $T(i,n_i)$ ως επιτυχημένης. Εάν μετά την επιστροφή της AcquireOwnerships το status της $T(i,n_i)$ έχει τιμή ABORTED (γραμμή 109) επιστρέφεται η τιμή FALSE στον χρήστη (γραμμή 110). Σημειώνεται ότι στην καθολική κατάσταση που το status έλαβε την τιμή ABORTED, έγινε κατάργηση των προσωρινών ιδιοκτησιών που κατείχε η $T(i,n_i)$. Επειδή ο NBSTM αποκτά τις προσωρινές ιδιοκτησίες και ενημερώνει δεδομένα των t-μεταβλητών κατά την εκτέλεση της λειτουργίας CommitTransaction, λέμε ότι ο NBSTM χρησιμοποιεί εκ των υστέρων απόκτηση προσωρινών ιδιοκτησιών και ετεροχρονισμένη ενημέρωση των t-μεταβλητών.

```
102 Boolean PerformTrans (*Trec)
103   int status;
104   AcquireOwnerships (Trec);
105   status = Trec→status;
106   if (status == ACTIVE)
107       CAS (Trec→status, ACTIVE, COMMITED);
108       return (TRUE);
109   else
110       return (FALSE);
```

Σχήμα 6.14: Κώδικας της συνάρτησης PerformTrans του NBSTM.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Ο κώδικας της συνάρτησης AcquireOwnerships παρουσιάζεται στο Σχήμα 6.15. Με την συνάρτηση αυτή γίνεται προσπάθεια απόκτησης των επιθυμητών προσωρινών ιδιοκτησιών. Ο αλγόριθμος NBSTM ορίζει ότι πριν την εφαρμογή των ενημερώσεων, πρέπει να αποκτηθούν όλες οι προσωρινές ιδιοκτησίες t-μεταβλητών που ο χρήστης προσπέλασε είτε για ανάγνωση είτε για ενημέρωση. Αυτές οι t-μεταβλητές είναι καταχωρημένες στις λίστες ανάγνωσης Trec(i,n_i).readList και ενημέρωσης Trec(i,n_i).writeList που διατηρεί η T(i,n_i). Για το λόγο αυτό χρησιμοποιείται η συνάρτηση GetNextSortedItem (γραμμή 115), η οποία παίρνει ως όρισμα τις λίστες αυτές και επιστρέφει τα στοιχεία τους. Σημειώνεται ότι εάν μια t-μεταβλητή είναι καταχωρημένη και στις δύο λίστες, τότε επιστρέφονται και οι δύο καταχωρήσεις της. Για κάθε t-μεταβλητή x που επιστρέφεται από την συνάρτηση GetNextSortedItem, αρχικά ανακτώνται τα τρέχοντα δεδομένα της, με τη βοήθεια της συνάρτησης GetCurrentData (ανάλογα με τη λίστα στην οποία είναι καταχωρημένο το x καλείται μια εκ των γραμμών 118 και 119). Στη συνέχεια, εάν υπάρχει καταχώρηση για τη x στη readList της T(i,n_i), ελέγχεται εάν τα αποθηκευμένα δεδομένα για το x είναι ακόμα συνεπή (γραμμή 123). Εάν αυτό ισχύει γίνεται προσπάθεια απόκτησης της προσωρινής ιδιοκτησίας της x, ανάλογα με την λίστα στην οποία ανήκει, όπως περιγράφεται στη συνέχεια. Εάν τα δεδομένα της x δεν είναι συνεπή τότε η T(i,n_i) χαρακτηρίζεται ως μη επιτυχημένη (γραμμή 124) και η συνάρτηση AcquireOwnerships επιστρέφει.

Παραβλέπουμε προσωρινά τη χρησιμότητα των γραμμών 126 και 127. Η απόκτηση της προσωρινή ιδιοκτησίας μιας t-μεταβλητής x που περιγράφεται από την TmVar $tvar_x$, από την T(i,n_i), γίνεται ενημερώνοντας ατομικά (με τη βοήθεια μιας λειτουργίας CAS) το δείκτη $tvar_x$.start ώστε να δείχνει σε κάποιον νέο Locator, έστω loc, ο οποίος θα περιγράφει ως προσωρινό ιδιοκτήτη της x τη δοσοληψία αυτή, δηλαδή θα ισχύει loc.Tran == Trec(i,n_i) και Trec(i,n_i).status == ACTIVE. Σημειώνεται ότι κάθε καταχώρηση της readList και της writeList περιέχει έναν νέο Locator, ο οποίος περιγράφει μια προσωρινή ιδιοκτησία ανάγνωσης και μια προσωρινή ιδιοκτησία ενημέρωσης, αντίστοιχα, για κάποια t-μεταβλητή. Έτσι, εάν υπάρχει καταχώρηση για το x στη writeList της T(i,n_i), κατά την απόκτηση της προσωρινής ιδιοκτησίας της x χρησιμοποιείται ο νέος Locator της εγγραφής αυτής (γραμμή 130), ώστε να αποκτηθεί προσωρινή ιδιοκτησία εγγραφής στο x και να είναι

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

δυνατή η εφαρμογή των ενημερώσεων του χρήστη στο x. Στην αντίθετη περίπτωση, χρησιμοποιείται ο νέος Locator της εγγραφής του x στην readList και αποκτάται προσωρινή ιδιοκτησία ανάγνωσης (γραμμή 132). Σημειώνεται ότι εάν πρόκειται να αποκτηθεί κάποια προσωρινή ιδιοκτησία ενημέρωσης μιας t-μεταβλητής x, τότε η εγγραφή της x στη writeList της $T(i,n_i)$, έστω writeListItem, πρέπει να ενημερωθεί ώστε ο δείκτης writeListItem→newLoc→oldData, να δείχνει στα παλιά δεδομένα της x, τα οποία είναι τα τρέχοντα δεδομένα της (γραμμή 129). Σημειώνεται ότι ο δείκτης αυτός έχει οριστεί σε null κατά την εκτέλεση της λειτουργίας WriteTmVar (γραμμές 77 και 33).

```
111 AcquireOwnerships (Transaction *Trec)
112     Read_List *readListItem, *readList = Trec→readList;
113     Write_List *writeListItem, *writeList = Trec→writeList;
114 Locator *loc;
115 while ( ((readListItem, writeListItem) = GetNextSortedItem(readList,writeList))
                !- (null, null)
            )
116     while (TRUE)
117         if (readListItem != null)
118            (loc,curData) = GetCurrentData(Trec, readListItem→tvar);
119         else (loc,curData) = GetCurrentData(Trec, writeListItem→tvar);
120         if (Trec->status != ACTIVE)
121            return;
122         if (loc→tran == Trec) break;
123         if (readListItem!=null && curData!=readListItem→tvarVersion)
124            CAS(Trec→status, ACTIVE, ABORTED);
125            return;
126         if (loc→oldData == loc→newData)
127            TryHelpTransaction (loc→tran);
128         if (writeListItem != null)
129            writeListItem→newLoc→oldData = curData;
130         if ( writeListItem != null &&
                   CAS(writeListItem→tvar→start, loc, writeListItem→newLoc)==TRUE
                )
131            break;
132         else if (CAS(readListItem→tvar→start, loc, readListItem→newLoc)==TRUE)
133            break;
```

Σχήμα 6.15: Κώδικας της συνάρτησης PerformTrans του NBSTM.

Η δοσοληψία $T(i,n_i)$ γίνεται *εμφανής* στις υπόλοιπες δοσοληψίες μετά την απόκτηση της πρώτης προσωρινής ιδιοκτησίας κάποιας t-μεταβλητής x, διότι τότε μέσω του

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Locator της x οι υπόλοιπες δοσοληψίες μπορούν να μάθουν τη δομή Trec(i,n_i). Επίσης για να έχει αποκτήσει η T(i,n_i) κάποια προσωρινή ιδιοκτησία σημαίνει ότι έχει κληθεί η λειτουργία CommitTransaction από τον χρήστη και άρα έχει ολοκληρωθεί ήδη το πρώτο στάδιο εκτέλεσης της T(i,n_i). Αυτό σημαίνει ότι όλες οι t-μεταβλητές που η T(i,n_i) προσπέλασε για ανάγνωση ή ενημέρωση, περιέχονται στις αντίστοιχες λίστες Trec(i,n_i).readList και Trec(i,n_i).writeList και οι λίστες αυτές δεν πρόκειται να τροποποιηθούν. Εάν κάποια δοσοληψία T(j,n_j), όπου $i \neq j$, βρει κάποια t-μεταβλητή x η ιδιοκτησία της οποίας έχει αποκτηθεί από την T(i,n_i), μπορεί να βοηθήσει την T(i,n_i) να ολοκληρωθεί (εκτελώντας την αντίστοιχη λειτουργία PerformTrans), διότι μέσω του Locator της x μαθαίνει το Trec(j,n_j) και αποκτά πρόσβαση σε όλες τις απαραίτητες πληροφορίες για την T(j,n_j). Αυτός είναι ο μηχανισμός βοήθειας που χρησιμοποιεί ο αλγόριθμος NBSTM.

Από τα παραπάνω προκύπτει ότι το δεύτερο στάδιο εκτέλεσης μιας δοσοληψίας και συγκεκριμένα οι ενέργειες της συνάρτησης PerformTrans, μπορούν να εκτελούνται ταυτόχρονα από πολλές δοσοληψίες. Έτσι, πρέπει να εξασφαλιστεί ότι δε δημιουργούνται προβλήματα στην περίπτωση αυτή. Η ορθότητα του αλγορίθμου NBSTM στην περίπτωση αυτή αποδεικνύεται στην Ενότητα 6.2. Σημαντικό ρόλο στην ορθότητά του έχει η γραμμή 122, η οποία εξασφαλίζει ότι εάν κάποια δοσοληψία που βοηθά την T(i,n_i) προσπαθήσει να αποκτήσει την προσωρινή ιδιοκτησία κάποιας t-μεταβλητής x η οποία έχει ήδη αποκτηθεί από την T(i,n_i), συνεχίζει με την επόμενη t-μεταβλητή χωρίς να προσπαθήσει να αποκτήσει την προσωρινή ιδιοκτησία της x.

Σημειώνεται ότι η συνάρτηση GetCurrentData μπορεί να επιστρέψει ως τρέχοντα δεδομένα μιας t-μεταβλητής x, δεδομένα που δεικτοδοτούνται από κάποιον Locator, ο οποίος περιγράφει προσωρινή ιδιοκτησία ανάγνωσης της x από κάποια εκκρεμή δοσοληψία T(j,n_j). Στην περίπτωση αυτή (γραμμή 126) η T(i,n_i) δεν μπορεί να αποκτήσει την προσωρινή ιδιοκτησία της x, πριν την ολοκλήρωση της T(j,n_j). Για το λόγο αυτό, η T(i,n_i) προσπαθεί να βοηθήσει την T(j,n_j) να ολοκληρωθεί εκτελώντας τη συνάρτηση TryHelpTransaction (γραμμή 127). Ο αλγόριθμος NBSTM εγγυάται ότι μετά την ολοκλήρωση της TryHelpTransaction, η T(j,n_j) θα έχει ολοκληρωθεί είτε επιτυχώς ή μη-επιτυχώς.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Επίσης, η σειρά με την οποία η δοσοληψία θα προσπαθήσει να αποκτήσει προσωρινές ιδιοκτησίες στις t-μεταβλητές είναι καθολική. Αυτό σημαίνει ότι είναι αδύνατο μια διεργασία T_1 να τις προσωρινές ιδιοκτησίες των t-μεταβλητών x_i, x_j με την συγκεκριμένη σειρά και μια άλλη δοσοληψία T_2 να τις αποκτήσει με την αντίστροφη σειρά, δηλαδή x_j, x_i. Με βάση τον μηχανισμό βοήθειας που ο NBSTM χρησιμοποιεί και το χαρακτηριστικό της καθολικής σειράς απόκτησης των προσωρινών ιδιοκτησιών των επιθυμητών t-μεταβλητών, ο αλγόριθμος NBSTM εγγυάται την ιδιότητα τερματισμού της ελευθερίας κλειδωμάτων.

Αν οι προσωρινές ιδιοκτησίες δεν αποκτιούνταν βάσει της καθολικής διάταξης, θα μπορούσε να παρουσιαστεί κατάσταση καθολικής παρατεταμένης στέρησης, όπως περιγράφεται στη συνέχεια. Έστω ότι η δοσοληψία $T(i,n_i)$ είχε αποκτήσει την προσωρινή ιδιοκτησία της t-μεταβλητής x και προσπαθεί να αποκτήσει την προσωρινή ιδιοκτησία της t-μεταβλητής y. Έστω ότι κάποια άλλη δοσοληψία $T(j,n_j)$ κατέχει την προσωρινή ιδιοκτησία της y και προσπαθεί να αποκτήσει την προσωρινή ιδιοκτησία της x. Έτσι, η $T(i,n_i)$ θα βοηθήσει την $T(j,n_j)$ και η $T(j,n_j)$ την $T(i,n_i)$, και αυτό θα επαναλαμβάνεται συνεχώς οδηγώντας τις δοσοληψίες σε κατάσταση παρατεταμένης στέρησης. Εάν οι δύο αυτές δοσοληψίες ήταν οι μόνες που εκτελούνταν, τότε στην περίπτωση αυτή θα καταστρατηγείτο η ιδιότητα της ελευθερίας κλειδωμάτων του αλγορίθμου και το σύστημα θα βρίσκονταν σε κατάσταση καθολικής παρατεταμένης στέρησης.

6.2. Απόδειξη ορθότητας αλγορίθμου NBSTM

Υπενθυμίζεται ότι η n_i-οστή δοσοληψία που εκκινείται από μια διεργασία p_i συμβολίζεται με *T(i,ni)* και η διεργασία p_i είναι ο *δημιουργός* της $T(i,n_i)$. Ακόμη, υπενθυμίζεται ότι κάθε δοσοληψία $T(i,n_i)$ διατηρεί μια διαμοιραζόμενη δομή δεδομένων *Trec(i,n_i)* στην οποία περιέχεται ένας καταχωρητής CAS που ονομάζεται *status*, μια λίστα αναγνώσεων *readList* και μια λίστα ενημερώσεων *writeList*. Μια διεργασία p_j ονομάζεται *βοηθητική διεργασία* μιας δοσοληψίας $T(i,n_i)$ αν η p_j καλεί τη λειτουργία PerformTrans της γραμμής 58 με όρισμα $Trec(i,n_i)$. Συμβολίζουμε με $PT_j(i,n_i)$ την κλήση αυτή. Η δοσοληψία $T(j,n_j)$ που εκτελείται από την p_j κατά την

κλήση της $PT_j(i,n_i)$ ονομάζεται *βοηθητική δοσοληψία* της $T(i,n_i)$ και η $PT_j(i,n_i)$ ονομάζεται *βοηθητική PerformTrans* της $T(i,n_i)$. Οι *executing διεργασίες* μιας δοσοληψίας $T(i,n_i)$ αποτελούνται από τον initiator της και τις βοηθητικές της διεργασίες. Επίσης οι *executing PerformTrans* της δοσοληψίας αποτελούνται από τις PerformTrans που εκτελούν οι executing διεργασίες της. Υπενθυμίζεται ότι το *διάστημα εκτέλεσης* μιας δοσοληψίας $T(i,n_i)$ συμβολίζεται ως *E(T(i,n_i))*. Είναι αξιοσημείωτο ότι το $E(T(i,n_i))$ δεν ταυτίζεται με το διάστημα εκτέλεσης των βοηθητικών δοσοληψιών της $T(i,n_i)$, εφόσον οι βοηθητικές δοσοληψίες της μπορεί να τερματίσουν πριν το τέλος του $E(T(i,n_i))$ ή να συνεχίζουν να εκτελούνται μετά το τέλος του.

Έστω ότι μια δοσοληψία $T(i,n_i)$ έχει εκτελέσει τις γραμμές 77 και 78 του κώδικα, που σημαίνει ότι έχει δημιουργήσει έναν καινούργιο Locator newLoc για κάποια t-μεταβλητή x και ο newLoc έχει καταχωρηθεί μαζί με την x σε μια εγγραφή της writeList της $T(i,n_i)$. Επίσης για τον newLoc ισχύει newLoc→tran == $Trec(i,n_i)$ και μπορεί να χρησιμοποιηθεί για την απόκτηση προσωρινής ιδιοκτησίας ενημέρωσης από την $T(i,n_i)$. Κάθε φορά που μια executing PerformTrans της $T(i,n_i)$ εκτελεί επιτυχώς τη λειτουργία CAS της γραμμής 130 του κώδικα με ορίσματα newLoc και την x, λέμε ότι η $T(i,n_i)$ *γίνεται προσωρινός ιδιοκτήτης* της *x* και *αποκτά προσωρινή ιδιοκτησία ενημέρωσης* σε αυτή. Έστω ότι η $T(i,n_i)$ έχει εκτελέσει τις γραμμές 23 και 24 του κώδικα, που σημαίνει ότι έχει δημιουργήσει έναν καινούργιο Locator newLoc για κάποια t-μεταβλητή x και ο newLoc έχει καταχωρηθεί μαζί με την x σε μια εγγραφή της readList της $T(i,n_i)$. Επίσης για τον newLoc ισχύει newLoc→tran == $Trec(i,n_i)$ και μπορεί να χρησιμοποιηθεί για την απόκτηση προσωρινής ιδιοκτησίας ανάγνωσης από την $T(i,n_i)$. Κάθε φορά που μια executing PerformTrans της $T(i,n_i)$ εκτελεί επιτυχώς τη λειτουργία CAS της γραμμής 132 του κώδικα με ορίσματα newLoc και την x, λέμε ότι η $T(i,n_i)$ *γίνεται προσωρινός ιδιοκτήτης* της *x* και *αποκτά προσωρινή ιδιοκτησία ανάγνωσης* σε αυτή. Λέμε ότι μια δοσοληψία $T(i,n_i)$ αποκτά την προσωρινή ιδιοκτησία μιας t-μεταβλητής x *βάσει ενός Locator loc*, εάν η $T(i,n_i)$ εκτελέσει επιτυχώς την εντολή CAS μιας εκ των γραμμών 130 και 132 με πρώτο όρισμα την TmVar της x και τρίτο όρισμα τον loc.

Κάθε φορά που μια δοσοληψία $T(i,n_i)$ εκτελεί επιτυχώς τη λειτουργία CAS μιας εκ των γραμμών 16, 90, 94 και 100 στο status της, λέμε ότι η $T(i,n_i)$ *καταργεί* τις προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που αυτή κατείχε. Κάθε φορά που μια executing PerformTrans της $T(i,n_i)$ εκτελεί επιτυχώς τη λειτουργία CAS της γραμμής 124 στο status της $T(i,n_i)$, λέμε ότι η $T(i,n_i)$ *καταργεί* τις προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που αυτή κατείχε. Κάθε φορά που μια executing PerformTrans της $T(i,n_i)$ εκτελεί επιτυχώς τη λειτουργία CAS της γραμμής 107 στο status της $T(i,n_i)$, λέμε ότι η $T(i,n_i)$ *εφαρμόζει* τις *ενημερώσεις* της στις t-μεταβλητές που κατείχε προσωρινή ιδιοκτησία ενημέρωσης και *καταργεί* τις προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που αυτή κατείχε.

Για να αποδείξουμε την ορθότητα του αλγορίθμου NBSTM πρέπει να δείξουμε ότι ικανοποιεί την ιδιότητα της σειριοποιησιμότητας και την ιδιότητα τερματισμού ελευθερία κλειδωμάτων.

6.2.1. Σειριοποιησιμότητα

Έστω α μια οποιαδήποτε εκτέλεση του NBSTM και έστω μια δοσοληψία $T(i,n_i)$. Υπενθυμίζεται ότι εάν η $T(i,n_i)$ τερματίσει, συμβολίζουμε με $Cf(i,n_i)$ την τελευταία καθολική κατάσταση του $E(T(i,n_i))$.

Λήμμα 6.1:*Εάν το Trec(i,n_i).status μιας δοσοληψίας $T(i,n_i)$ έχει τιμή διαφορετική από ACTIVE σε κάποια καθολική κατάσταση C της α, τότε καμία επιτυχημένη λειτουργία CAS στο Trec(i,n_i).status δεν πραγματοποιείται μετά την C.*

Απόδειξη: Στη C το $Trec(i,n_i)$.status έχει τιμή v, όπου $v \in$ {COMMITTED, ABORTED}. Ας υποθέσουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι πραγματοποιείται μια επιτυχημένη CAS στο $Trec(i,n_i)$.status μετά την C. Έστω CS_1 η *πρώτη* λειτουργία CAS που αλλάζει μετά την C το $Trec(i,n_i)$.status από v σε v', όπου $v' \in$ {ACTIVE, COMMITTED, ABORTED}. Σημειώνεται ότι $Trec(i,n_i) \neq Trec(i,n_j)$ αν $n_i \neq n_j$, δηλαδή η δομή που περιγράφει μια δοσοληψία είναι μοναδική, ακόμη και για δοσοληψίες που εκκινήθηκαν από την ίδια διεργασία. Από τον κώδικα προκύπτει ότι το $Trec(i,n_i)$.status μπορεί να αλλάξει κατά την εκτέλεση μιας εκ των γραμμών 16, 28,

90, 94, 100, 107 ή 124. Σε όλες τις περιπτώσεις, η λειτουργία CAS εκτελείται έχοντας ως δεύτερο όρισμα την τιμή ACTIVE που σημαίνει ότι πρέπει να ισχύει $Trec(i,n_i)$.status==ACTIVE για να επιτύχει. Ωστόσο, όταν εκτελείται η CS_1 το $Trec(i,n_i)$.status≠ACTIVE. Άρα, η CS_1 αποτυγχάνει, το οποίο είναι άτοπο! ■

Από τον κώδικα (γραμμή 3) προκύπτει ότι κατά την εκκίνηση της $T(i,n_i)$ ισχύει $Trec(i,n_i)$.status==ACTIVE. Από το Λήμμα 6.1 προκύπτει ότι το $Trec(i,n_i)$.status της $T(i,n_i)$ μπορεί να ενημερωθεί σε COMMITED ή ABORTED μία μόνο φορά. Από τον κώδικα προκύπτει ότι το $Trec(i,n_i)$.status μπορεί να λάβει την τιμή COMMITED μέσω της εκτέλεσης της εντολής CAS της γραμμής 94 από την $T(i,n_i)$ ή της γραμμής 107 από κάποια executing PerformTrans της $T(i,n_i)$. Σε αυτή την περίπτωση συμβολίζουμε με *Cc(i,n_i)* την καθολική κατάσταση στην οποία αυτό συμβαίνει για μία και μοναδική φορά και χαρακτηρίζουμε την $T(i,n_i)$ ως *επιτυχημένη*. Επίσης το $Trec(i,n_i)$.status μπορεί να λάβει την τιμή ABORTED μέσω της εκτέλεσης της κατάλληλης εντολής CAS είτε από την $T(i,n_i)$ (γραμμές 16, 28, 90 και 100) είτε από κάποια executing PerformTrans της $T(i,n_i)$ (γραμμή 124). Σε κάθε περίπτωση συμβολίζουμε με *Ca(i,n_i)* την καθολική κατάσταση στην οποία αυτό συμβαίνει για μία και μοναδική φορά και χαρακτηρίζουμε την $T(i,n_i)$ ως *μη-επιτυχημένη*. Σημειώνεται ότι από το Λήμμα 6.1 προκύπτει ότι θα ορίζεται το πολύ μία από τις $Cc(i,n_i)$ και $Ca(i,n_i)$.

Κατά την απόδειξη της ιδιότητας της σειριοποιησιμότητας θα θεωρήσουμε ότι μία δοσοληψία $T(i,n_i)$ εκτελείται σε δύο στάδια. Στο *πρώτο στάδιο* η $T(i,n_i)$ εκτελεί τις λειτουργίες ReadTmVar, WriteTmVar και CreateNewTmVar του αντικειμένου STM. Όταν η $T(i,n_i)$ επιθυμεί να ολοκληρώσει το πρώτο στάδιο εκτέλεσής της καλεί μια εκ των λειτουργιών AbortTransaction και CommitTransaction. Εάν η $T(i,n_i)$ εκτελέσει τη λειτουργία CommitTransaction τότε λέμε ότι ξεκινά το *δεύτερο στάδιο* εκτέλεσής της, το οποίο ολοκληρώνεται μόλις η λειτουργία αυτή ολοκληρωθεί. Σημειώνεται ότι μια άλλη δοσοληψία $T(j,n_j)$ μπορεί να βοηθήσει την $T(i,n_i)$ εκτελώντας τη συνάρτηση TryHelpTransaction μιας εκ των γραμμών 52 και 127, με όρισμα την $Trec(i,n_i)$. Από τον κώδικα προκύπτει, ότι η $T(j,n_j)$ διαβάζει τη δομή $Trec(i,n_i)$ μέσω του Locator κάποιας t-μεταβλητής. Αυτό σημαίνει ότι για να μπορεί η $T(j,n_j)$ να βοηθήσει την $T(i,n_i)$, πρέπει η $T(i,n_i)$ να έχει αποκτήσει την προσωρινή ιδιοκτησία τουλάχιστον

μίας t-μεταβλητής. Η $T(i,n_i)$ αποκτά προσωρινές ιδιοκτησίες κατά το δεύτερο στάδιο εκτέλεσής της. Επομένως, μια δοσοληψία $T(i,n_i)$ δε μπορεί να έχει βοηθητικές δοσοληψίες ενόσω βρίσκεται στο πρώτο στάδιο εκτέλεσής της. Επίσης, κατά το δεύτερο στάδιο της εκτέλεσής της, η $T(i,n_i)$ αποκτά προσωρινές ιδιοκτησίες με την εκτέλεση της συνάρτησης AcquireOwnerships της γραμμής 104, η οποία έπεται της κλήσης της συνάρτησης PerformTrans της γραμμής 98. Επομένως, η $T(i,n_i)$ μπορεί να έχει βοηθητικές δοσοληψίες μόνο μετά την εκτέλεση της συνάρτησης PerformTrans της γραμμής 98. Έτσι προκύπτει άμεσα το παρακάτω Πόρισμα.

Πόρισμα 6.2: *Έστω μια δοσοληψία $T(i,n_i)$. Η εκτέλεση της εντολής CAS μιας εκ των γραμμών 16, 28, 90, 94 και 100 στο $Trec(i,n_i)$.status από την $T(i,n_i)$ γίνεται πάντα επιτυχώς.*

Από τον κώδικα (γραμμή 3) προκύπτει ότι κατά την εκκίνηση της $T(i,n_i)$ ισχύει $Trec(i,n_i)$.status==ACTIVE. Για να μπορεί η $T(i,n_i)$ να εκτελέσει τη συνάρτηση PerformTrans της γραμμής 98 πρέπει η συνάρτηση Validate που εκτελέστηκε στη γραμμή 92 να επέστρεψε την τιμή TRUE. Από τον κώδικα της Validate (γραμμή 65) προκύπτει ότι η τιμή TRUE επιστρέφεται εάν και μόνο εάν ισχύει $Trec(i,n_i)$.status != ABORTED, δηλαδή εάν το status της $T(i,n_i)$ είναι ACTIVE ή COMMITED. Σημειώνεται ότι πριν την κλήση της PerformTrans της γραμμής 98 από την $T(i,n_i)$ η τιμή COMMITED μπορεί να γραφεί μόνο με την επιτυχή εκτέλεση της εντολής CAS της γραμμής 94. Όμως, η γραμμή 94 δεν εκτελείται διότι εάν αυτό συνέβαινε δε θα ήταν δυνατό να εκτελεστεί η γραμμή 98. Επομένως προκύπτει ότι πριν την εκτέλεση της γραμμής 95, το $Trec(i,n_i)$.status μπορεί να έχει ενημερωθεί μόνο με την τιμή ABORTED. Με βάση τα παραπάνω, προκύπτει ότι κατά την εκτέλεση της συνάρτησης PerformTrans της γραμμής 98 από την $T(i,n_i)$ ισχύει $Trec(j,n_j)$.status == ACTIVE.

Πόρισμα 6.3: *Έστω C η καθολική κατάσταση στην οποία εκτελείται η συνάρτηση PerformTrans της γραμμής 98 από κάποια δοσοληψία $T(i,n_i)$. Στη C ισχύει $Trec(i,n_i)$.status == ACTIVE.*

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Έστω *Caof(i,n$_i$)* η καθολική κατάσταση στην οποία ολοκληρώνεται για πρώτη φορά μια κλήση της συνάρτησης AcquireOwnerships από κάποια executing PerformTrans της T(i,n$_i$). Από τον κώδικα προκύπτει ότι μετά την κλήση της PerformTrans της γραμμής 98 από την T(i,n$_i$), το Trec(i,n$_i$).status μπορεί να λάβει την τιμή ABORTED μέσω της εκτέλεσης της εντολής CAS της γραμμής 124 ή την τιμή COMMITED μέσω της εκτέλεσης της εντολής CAS της γραμμής 107, έστω CS_{107} η εντολή αυτή, από κάποια executing PerformTrans της T(i,n$_i$). Πριν την Caof(i,n$_i$) καμία executing PerformTrans της T(i,n$_i$) δε μπορεί να εκτελέσει την CS_{107}.

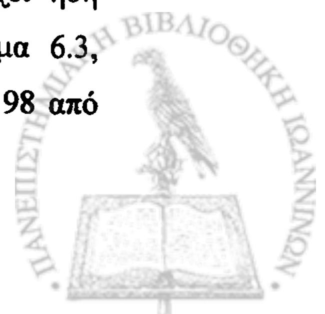
Πόρισμα 6.4: *Στην Caof(i,n$_i$) το status της δοσοληψίας T(i,n$_i$) μπορεί να είναι ABORTED ή ACTIVE.*

Από τον κώδικα προκύπτει ότι το status της T(i,n$_i$) μπορεί να ενημερωθεί με την εκτέλεση της εντολής CAS μιας εκ των γραμμών 16, 28, 90, 94 και 100 από την T(i,n$_i$) ή μιας εκ των γραμμών 107 και 124 από τις executing PerformTrans της T(i,n$_i$). Έστω CS μια οποιαδήποτε τέτοια CAS που εκτελείται στην καθολική κατάσταση C. Εάν η CS αποτύχει, τότε το Trec(i,n$_i$).status είχε τιμή διαφορετική από ACTIVE στη C. Επομένως είτε η Cc(j,n$_j$) είτε η Ca(j,n$_j$) είναι καλά ορισμένη και προηγείται της C. Έτσι, προκύπτει το παρακάτω πόρισμα.

Πόρισμα 6.5: *Έστω C η καθολική κατάσταση στην οποία εκτελείται μια εντολή CAS στο status μιας δοσοληψίας T(i,n$_i$) και αποτυγχάνει. Τότε μία εκ των Cc(i,n$_i$) και Ca(i,n$_i$) είναι καλά ορισμένη και προηγείται της C.*

Λήμμα 6.6: *Έστω κάποια δοσοληψία T(i,n$_i$) η οποία καλεί τη συνάρτηση PerformTrans (μιας εκ των γραμμών 58 και 98) με όρισμα Trec(j,n$_j$) (για την γραμμή 98 ισχύει i==j), δηλαδή βοηθά την T(j,n$_j$), και έστω Cptf(i,n$_i$) η πρώτη καθολική κατάσταση στην οποία έχει ολοκληρωθεί η εκτέλεση της συνάρτηση αυτής. Πριν την Cptf ορίζεται μία εκ των Cc(j,n$_j$) και Ca(j,n$_j$).*

Απόδειξη: Εφόσον η T(i,n$_i$) βοηθά την T(j,n$_j$) σημαίνει ότι η T(j,n$_j$) έχει ήδη εκτελέσει τη συνάρτηση PerformTrans της γραμμής 98. Από το Πόρισμα 6.3, προκύπτει ότι κατά την κλήση της συνάρτησης PerformTrans της γραμμής 98 από


ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

την $T(j,n_j)$ ισχύει $Trec(j,n_j).status == ACTIVE$. Επίσης από το Πόρισμα 6.4 προκύπτει ότι το $Trec(j,n_j).status$ στην $Caof(j,n_j)$ μπορεί να έχει τιμή ABORTED ή ACTIVE. Σημειώνεται ότι η $Caof(j,n_j)$ προηγείται της $Cptf(i,n_i)$.

Εάν στην $Caof(j,n_j)$ το $Trec(j,n_j).status$ έχει τιμή ABORTED, σημαίνει ότι η $Ca(j,n_j)$ είναι καλά ορισμένη και προηγείται της $Cptf(j,n_j)$. Εάν στην $Caof(j,n_j)$ το $Trec(j,n_j).status$ έχει τιμή ACTIVE, τότε από τον κώδικα (γραμμή 106) προκύπτει ότι θα εκτελεστεί η εντολή CAS της γραμμής 107, έστω CS_{107} η εντολή αυτή. Έστω A το σύνολο των executing PerformTrans της $T(j,n_j)$ που βλέπουν ACTIVE το $Trec(j,n_j).status$ μετά την $Caof(j,n_j)$ και εκτελούν την CS_{107}. Έστω C η καθολική κατάσταση στην οποία εκτελείται για πρώτη φορά η CS_{107} από κάποια executing PerformTrans που ανήκει στο σύνολο A. Εάν στη C η εκτέλεση της CS_{107} γίνει επιτυχώς, ορίζεται η $Cc(j,n_j)$ και επειδή η C προηγείται της $Cptf(j,n_j)$, η $Cc(j,n_j)$ θα προηγείται της $Cptf(j,n_j)$. Εάν στη C αποτύχει η εκτέλεση της CS_{107}, τότε με βάση το Πόρισμα 6.5 προκύπτει ότι η $Ca(j,n_j)$ θα είναι καλά ορισμένη και θα προηγείται της C, η οποία προηγείται της $Cptf(j,n_j)$. Έτσι σε κάθε περίπτωση το Λήμμα ισχύει. ■

Έστω ότι κάποια δοσοληψία $T(i,n_i)$ εκτελεί τη συνάρτηση TryHelpTran με όρισμα $Trec(j,n_j)$, όπου $i \neq j$, για να βοηθήσει τη δοσοληψία $T(j,n_j)$. Έστω C η καθολική κατάσταση στην οποία η συνάρτηση αυτή επιστρέφει. Από τον κώδικα της συνάρτησης TryHelpTran προκύπτει ότι η συνάρτηση αυτή καλεί τη συνάρτηση PerformTrans μόνο εάν ισχύει $Trec(j,n_j).status == ACTIVE$ κατά την εκτέλεση της γραμμής 57 από την $T(i,n_i)$. Στην περίπτωση αυτή, από το Λήμμα 6.6 προκύπτει ότι πριν την ολοκλήρωση της PerformTrans, και άρα πριν την C, θα έχει οριστεί είτε η $Cc(j,n_j)$ είτε η $Ca(j,n_j)$. Εάν κατά την εκτέλεση της γραμμής 57 από την $T(i,n_i)$ ισχύει $Trec(j,n_j).status \neq ACTIVE$, σημαίνει ότι πριν την εκτέλεση της γραμμής αυτής, και άρα πριν την C, θα έχει οριστεί είτε η $Cc(j,n_j)$ είτε η $Ca(j,n_j)$. Έτσι προκύπτει το παρακάτω Πόρισμα.

Πόρισμα 6.7: *Έστω ότι κάποια δοσοληψία $T(i,n_i)$ εκτελεί τη συνάρτηση TryHelpTran με όρισμα $Trec(j,n_j)$, όπου $i \neq j$, για να βοηθήσει τη δοσοληψία $T(j,n_j)$. Έστω C η καθολική κατάσταση στην οποία η συνάρτηση αυτή επιστρέφει. Μία εκ των $Cc(j,n_j)$ και $Ca(j,n_j)$ είναι καλά ορισμένη και προηγείται της C.*

Εάν η $Cf(i,n_i)$ ορίζεται, τότε πριν ολοκληρωθεί η $T(i,n_i)$ έχει κληθεί μια εκ των λειτουργιών CommitTransaction και AbortTransaction από την $T(i,n_i)$. Εάν έχει κληθεί η AbortTransaction, τότε από το Πόρισμα 6.2 προκύπτει ότι πριν την $Cf(i,n_i)$ έχει οριστεί η $Ca(i,n_i)$. Εάν έχει κληθεί η CommitTransaction, τότε από τον κώδικα προκύπτει ότι θα κληθεί είτε η εντολή CAS της γραμμής 94, είτε η συνάρτηση PerformTrans στη γραμμή 98, είτε η εντολή CAS της γραμμής 100. Έστω ότι εκτελείται μίας από τις CAS εντολές, έστω CS η εντολή αυτή. Εάν η CS εκτελεστεί επιτυχώς τότε είτε η $Cc(i,n_i)$ είτε η $Ca(i,n_i)$, αντίστοιχα, θα προηγείται της $Cf(i,n_i)$. Εάν η CS εκτελεστεί μη-επιτυχώς, τότε από το Πόρισμα 6.5 προκύπτει ότι μία εκ των $Cc(i,n_i)$ και $Ca(i,n_i)$ είναι καλά ορισμένη και προηγείται την $Cf(i,n_i)$. Εάν κληθεί η η συνάρτηση PerformTrans στη γραμμή 98, τότε από το Λήμμα 6.6 προκύπτει ότι πριν την επιστροφή της, άρα πριν την $Cf(i,n_i)$, θα έχει οριστεί μια εκ των $Cc(i,n_i)$ και $Ca(i,n_i)$. Έτσι προκύπτει το παρακάτω Πόρισμα.

Πόρισμα 6.8: Έστω μία δοσοληψία $T(i,n_i)$. Εάν η $Cf(i,n_i)$ ορίζεται, τότε μια εκ των $Cc(i,n_i)$ και $Ca(i,n_i)$ είναι καλά ορισμένη και προηγείται της $Cf(i,n_i)$.

Έστω ότι οι executing PerformTrans της $T(i,n_i)$ καταφέρνουν να αποκτήσουν την προσωρινή ιδιοκτησία μιας t-μεταβλητής x και έτσι η $T(i,n_i)$ γίνεται ιδιοκτήτης της x. Συμβολίζουμε με $C_x(i,n_i)$ την πρώτη καθολική κατάσταση στην οποία αυτό συμβαίνει. Εάν το $C_x(i,n_i)$ ορίζεται και αν προηγείται της $Cc(i,n_i)$ ή της $Ca(i,n_i)$ (ανάλογα με το ποια ορίζεται), συμβολίζουμε με $\alpha_x(i,n_i)$ το διάστημα εκτέλεσης της α που ξεκινά από την $C_x(i,n_i)$ και καταλήγει στην τελευταία καθολική κατάσταση που προηγείται είτε της $Cc(i,n_i)$ ή της $Ca(i,n_i)$. Επομένως κατά τη διάρκεια του $\alpha_x(i,n_i)$ ισχύει $Trec(i,n_i).status == ACTIVE$. Σημειώνεται ότι το $\alpha_x(i,n_i)$ δεν ορίζεται, εάν το $C_x(i,n_i)$ είτε δεν ορίζεται, είτε το $C_x(i,n_i)$ δεν προηγείται της $Cc(i,n_i)$ ή της $Ca(i,n_i)$. Σημειώνεται ότι χρησιμοποιούμε το συμβολισμό C_x αντί του $C_x(i,n_i)$ και τον α_x αντί του $\alpha_x(i,n_i)$, όταν η δοσοληψία στην οποία αναφέρεται ο συμβολισμός εξάγεται εύκολα από τα συμφραζόμενα.

Λήμμα 6.9: Έστω μια οποιαδήποτε δοσοληψία $T(i,n_i)$ και έστω μια t-μεταβλητή x για την οποία ορίζεται η $\alpha_x(i,n_i)$. Καμία δοσοληψία δε μπορεί να αποκτήσει την προσωρινή ιδιοκτησία ενημέρωσης ή ανάγνωσης της x, στην $\alpha_x(i,n_i)$, δηλαδή δε μπορεί να εκτελέσει

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ ΒΙΒΛΙΟΘΗΚΗ

επιτυχώς την εντολή CAS μιας εκ των γραμμών 130 και 132, αντίστοιχα, για τη x στην $\alpha_x(i,n_i)$.

Απόδειξη: Έστω, δια της μεθόδου της εις άτοπο απαγωγής, ότι στην $\alpha_x(i,n_i)$ υπάρχει τουλάχιστον μία executing PerformTrans που καταφέρνει να αποκτήσει την προσωρινή ιδιοκτησία της x. Έστω, PT(j,n_j) η πρώτη από αυτές, η οποία είναι executing PerformTrans μιας δοσοληψίας T(j,n_j) και έστω C η καθολική κατάσταση στην οποία αυτή αποκτά την προσωρινή ιδιοκτησία της x. Εφόσον η PT(j,n_j) αποκτά την προσωρινή ιδιοκτησία της x, προκύπτει ότι εκτελεί επιτυχώς την εντολή CAS, έστω CS_x, μιας εκ των γραμμών 130 και 132 με όρισμα την TmVar που περιγράφει τη x, έστω $tvar_x$.

Πριν η PT(j,n_j) εκτελέσει τη CS_x, εκτελεί είτε στη γραμμή 118 είτε στη γραμμή 119 τη συνάρτηση GetCurrentData για την $tvar_x$, η οποία επιστρέφει τα δεδομένα της x και τον αντίστοιχο Locator, έστω loc_x. Κατά την εκτέλεση της συνάρτησης GetCurrentData από την PT(j,n_j), γίνεται ανάγνωση της τιμής του loc_x στη γραμμή 43. Εάν η γραμμή 43 εκτελεστεί μετά την $C_x(i,n_i)$, τότε το loc_x→tran θα έχει τιμή Trec(i,n_i) και το loc_x→tran→status θα έχει τιμή ACTIVE (εξαιτίας του ορισμού της CS_x και του ότι η CS_x προηγείται του τέλους της $\alpha_x(i,n_i)$). Εάν ισχύει i==j, τότε η συνάρτηση GetCurrentData θα επιστρέψει στη γραμμή 51, εξαιτίας του ελέγχου της γραμμής 50, και η PT(j,n_j) δεν θα εκτελούσε την CS_x λόγω της γραμμής 122, το οποίο είναι Άτοπο. Εάν ισχύει i≠j, από τον κώδικα (γραμμές 46 έως 55) προκύπτει ότι για να ολοκληρωθεί η εκτέλεση της συνάρτησης GetCurrentData πρέπει είτε να αποτιμηθεί ως αληθής ο έλεγχος της γραμμής 49, είτε να εκτελεστεί η συνάρτηση TryHelpTransaction της γραμμής 52 με όρισμα Trec(i,n_i). Στην περίπτωση που ο έλεγχος της γραμμής 49 αποτιμηθεί ως αληθής, κατά την εκτέλεση της PT(j,n_j) θα αποτιμηθεί ως αληθής και ο έλεγχος της γραμμής 126, το οποίο σημαίνει ότι θα εκτελεστεί και πάλι η συνάρτηση TryHelpTransaction με όρισμα Trec(i,n_i) στη γραμμή 127. Επομένως σε κάθε περίπτωση εκτελείται η συνάρτηση TryHelpTransaction με όρισμα Trec(i,n_i), πριν την CS_x. Από το Πόρισμα 6.7 προκύπτει ότι πριν την ολοκλήρωση της TryHelpTransaction θα έχει οριστεί μία εκ των Cc(i,n_i) και Ca(i,n_i). Έτσι, πριν τη CS_x θα έχει οριστεί είτε η CM(i,n_i) ή η AB(i,n_i), το οποίο σημαίνει ότι η CS_x εκτελείται μετά το τέλος της $\alpha_x(i,n_i)$, το οποίο είναι Άτοπο.

Συμπεραίνουμε ότι η γραμμή 43 εκτελείται πριν την $C_x(i,n_i)$. Στη $C_x(i,n_i)$, η $T(i,n_i)$ θα αποκτήσει την προσωρινή ιδιοκτησία της x, το οποίο σημαίνει ότι θα ορίσει έναν νέο Locator για τη x, διαφορετικό του loc_x. Έτσι, η εκτέλεση της CS_x μετά την $C_x(i,n_i)$ που καλείται είτε ως CAS($tvar_x$, loc_x, Trec→readList.newLoc) είτε ως CAS($tvar_x$, loc_x, Trec→writeList.newLoc), θα αποτύχει διότι ο Locator της $tvar_x$ δεν θα είναι ο loc_x [6.6]. Άτοπο. ■

Έστω W_{AO} ο βρόχος των γραμμών 116 έως 133. Σημειώνεται ότι οι executing PerformTrans της $T(i,n_i)$ προσπαθούν να αποκτήσουν τις προσωρινές ιδιοκτησίες των t-μεταβλητών που η $T(i,n_i)$ επιθυμεί, κατά την εκτέλεση της συνάρτησης AcquireOwnerships. Σημειώνεται ότι η προσπάθεια απόκτησης της προσωρινής ιδιοκτησίας μιας t-μεταβλητής x, πραγματοποιείται με την εκτέλεση του W_{AO} για την x. Από τον κώδικα της AcquireOwnerships προκύπτει ότι μία executing PerformTrans της $T(i,n_i)$, έστω PT, που εκτελεί τον W_{AO} για κάποια t-μεταβλητή x μπορεί να φύγει από τον W_{AO} είτε επειδή το $Trec(i,n_i)$.status έχει τιμή διαφορετική από ACTIVE (γραμμές 120 και 121), είτε επειδή έχει ήδη αποκτηθεί η προσωρινή ιδιοκτησία της x από κάποια άλλη executing PerformTrans της $T(i,n_i)$ (γραμμή 122), είτε επειδή η PT έγραψε την τιμή ABORTED στο $Trec(i,n_i)$.status (γραμμές 124 και 125), είτε επειδή η PT απέκτησε την προσωρινή ιδιοκτησία της x (γραμμές 130 έως 133). Σημειώνεται ότι από το Πόρισμα 6.4 προκύπτει ότι στην $Caof(i,n_i)$ η τιμή του $Trec(i,n_i)$.status μπορεί να είναι είτε ACTIVE είτε ABORTED. Επομένως η PT μπορεί να εξέλθει του W_{AO} λόγω της γραμμής 121 πριν την $Caof(i,n_i)$, μόνο εάν το $Trec(i,n_i)$.status έχει πάρει την τιμή ABORTED. Έτσι προκύπτει το παρακάτω Πόρισμα.

Πόρισμα 6.10: *Έστω μια δοσοληψία $T(i,n_i)$ και μια t-μεταβλητή x, $x \in Trec(i,n_i)$.readList ή $x \in Trec(i,n_i)$.writeList. Κάθε executing PerformTrans της $T(i,n_i)$ που εκτελεί τον W_{AO} για τη x πριν την $Caof(i,n_i)$, μπορεί να φύγει από αυτόν είτε*

[6.6] Σημειώνεται ότι δε θα μπορούσε να προκύψει το πρόβλημα ABA διότι, όπως έχει αναφερθεί, ο αλγόριθμος NBSTM εκτελείται σε κλειστό μοντέλο μνήμης και η μεταβλητή loc_x είναι μια από τις ενεργές τοπικές μεταβλητές της p_j.

επειδή έχει αποκτηθεί η προσωρινή ιδιοκτησία της x από τις executing PerformTrans της $T(i,n_i)$ (συμπεριλαμβανομένου και της ίδιας) ή επειδή έχει γραφτεί (ή έγραψε) στο $Trec(i,n_i).status$ η τιμή ABORTED από τις executing PerformTrans της $T(i,n_i)$.

Στη συνέχεια δίνουμε κάποιους ορισμούς για τις t-μεταβλητές που περιέχονται στη readList της $T(i,n_i)$. Έστω $Crl_x(i,n_i)$ η καθολική κατάσταση στην οποία εισάγεται μια t-μεταβλητή x στη readList της $T(i,n_i)$. Από τον κώδικα προκύπτει, ότι η εισαγωγή της x στη readList της $T(i,n_i)$ πραγματοποιείται με την εκτέλεση της συνάρτησης InsertInReadList της γραμμής 24. Εάν υποθέσουμε χάριν απλούστευσης ότι η εκτέλεση της συνάρτησης InsertInReadList πραγματοποιείται με μία μόνο εντολή, τότε η $Crl_x(i,n_i)$ είναι η πρώτη καθολική κατάσταση που έπεται της εκτέλεσης της συνάρτησης αυτής για την x. Η συνάρτηση InsertInReadList αποθηκεύει στη readList της $T(i,n_i)$ το TmVar της x και τα δεδομένα της $drl_x(i,n_i)$, όπως αυτά επιστράφηκαν από την εκτέλεση της συνάρτησης GetCurrentData της γραμμής 14, έστω $GCDrl_x(i,n_i)$ η κλήση της συνάρτησης αυτής. Από τον κώδικα της $GCDrl_x(i,n_i)$ προκύπτει ότι τα δεδομένα $drl_x(i,n_i)$ (που επιστράφηκαν) διαβάστηκαν μέσω ενός Locator $locrl_x(i,n_i)$, ο οποίος διαβάστηκε από την $GCDrl_x(i,n_i)$ στη γραμμή 43, έστω στην καθολική κατάσταση $Crloc_x(i,n_i)$.

Σημειώνεται ότι πριν την $Crloc_x(i,n_i)$ έχει αποκτηθεί μνήμη για την αποθήκευση των δεδομένων $drl_x(i,n_i)$ από κάποια δοσοληψία. Στην $Crloc_x(i,n_i)$ οι θέσεις μνήμης στις οποίες είναι αποθηκευμένα τα δεδομένα $drl_x(i,n_i)$ είναι ενεργές διότι υπάρχει κάποιος δείκτης προς αυτές στον $locrl_x(i,n_i)$. Το κλειστό μοντέλο μνήμης για το οποίο σχεδιάστηκε ο NBSTM εξασφαλίζει ότι καμία δοσοληψία δε μπορεί να ενημερώσει οποιαδήποτε t-μεταβλητή με δεδομένα που είναι αποθηκευμένα στις θέσεις αυτές μετά την $Crloc_x(i,n_i)$ και πριν την ολοκλήρωση της $T(i,n_i)$. Έτσι προκύπτει το παρακάτω Πόρισμα.

Λήμμα 6.11: Έστω x οποιαδήποτε t-μεταβλητή που περιέχεται στην readList μιας δοσοληψίας $T(i,n_i)$. Καμία δοσοληψία δε μπορεί να αποκτήσει την προσωρινή ιδιοκτησία ενημέρωσης οποιαδήποτε t-μεταβλητής (συμπεριλαμβανομένου της x) μεταξύ της $Crloc_x(i,n_i)$ και του τέλους του $E(T(i,n_i))$ βάσει κάποιου Locator loc, για τον οποίο ισχύει $loc.newData == drl_x(i,n_i)$.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Λήμμα 6.12: Έστω μια οποιαδήποτε επιτυχημένη δοσοληψία $T(i,n_i)$. Έστω ότι η $T(i,n_i)$ ενημερώνει κάποια t-μεταβλητή x, βάσει του Locator loc για τον οποίο ο δείκτης loc.newData έχει τιμή d. Έστω $T(j,n_j)$ μια άλλη δοσοληψία, $i \neq j$, που εκτελεί τη συνάρτηση GetCurrentData για τη x και έστω GCD μια οποιαδήποτε τέτοια κλήση της GetCurrentData για τη x από την $T(j,n_j)$. Έστω C_R η καθολική κατάσταση στην οποία η GCD εκτελεί τη γραμμή 43. Εάν η C_R βρίσκεται εντός του διαστήματος $a_x(i,n_i)$, τότε η GCD επιστρέφει (loc, d).

Απόδειξη: Έστω loc_x ο Locator του x που η GCD διαβάζει στη C_R. Εφόσον η C_R εκτελείται εντός του διαστήματος a_x από το Λήμμα 6.9 προκύπτει ότι στη C_R θα ισχύει loc_x == loc. Επομένως, από τον κώδικα της GCD προκύπτει ότι αυτή θα επιστρέψει τον loc ως Locator του x. Έστω, δια της μεθόδου της εις άτοπο απαγωγής ότι η GCD επιστρέφει (loc, d'), όπου $d' \neq d$. Επειδή ο loc περιγράφει κάποια προσωρινή ιδιοκτησία ενημέρωσης, από τον κώδικα (γραμμές 77, 35 και 129) προκύπτει ότι η τιμή του δείκτη loc.oldData είναι διαφορετική από την τιμή του δείκτη loc.newData. Εφόσον η GCD διαβάζει loc στην C_R, για να μπορεί να ισχύει $d' \neq d$, η GCD πρέπει να επιστρέψει την τιμή του δείκτη loc.oldData. Από τον κώδικα προκύπτει ότι η GCD επιστρέφει την τιμή του δείκτη loc.oldData, εάν επιστρέψει σε μία από τις γραμμές 54 και 55 (σημειώνεται ότι δε μπορεί να επιστρέψει στην 50 γιατί ο loc περιγράφει μια προσωρινή ιδιοκτησία ενημέρωσης). Έστω ότι η GCD επιστρέφει στην καθολική κατάσταση C. Εάν επιστρέψει στη γραμμή 54, λόγω του ελέγχου που προηγείται, στην ίδια γραμμή, προκύπτει ότι πριν την C θα έχει οριστεί η $Ca(i,n_i)$. Εάν επιστρέψει στη γραμμή 55, σημαίνει ότι έχει εκτελεστεί και έχει ολοκληρωθεί η συνάρτηση TryHelpTransaction της γραμμής 52. Από το Πόρισμα 6.7 προκύπτει ότι πριν την ολοκλήρωση της συνάρτησης αυτής έχει οριστεί μία εκ των $Cc(i,n_i)$ και $Ca(i,n_i)$. Εφόσον η GCD επιστρέφει στη γραμμή 55 σημαίνει ότι ορίζεται η $Ca(i,n_i)$. Έτσι, σε κάθε περίπτωση πρέπει να έχει οριστεί η $Ca(i,n_i)$ και από το Λήμμα 6.1 προκύπτει ότι στη συνέχεια δε μπορεί να οριστεί η $Cc(i,n_i)$, το οποίο είναι άτοπο. ■

Σημειώνεται ότι οι t-μεταβλητές που η $T(i,n_i)$ διαβάζει και ενημερώνει καταχωρούνται στις λίστες ανάγνωσης και ενημέρωσης, αντίστοιχα, με την εκτέλεση

των λειτουργιών ReadTmVar (γραμμή 24) και WriteTmVar (γραμμή 78), αντίστοιχα, κατά το πρώτο στάδιο της εκτέλεσής της. Μετά την εκκίνηση του δευτέρου σταδίου της $T(i,n_i)$, δηλαδή την κλήση της λειτουργίας CommitTransaction, οι λίστες αυτές δεν μεταβάλλονται. Επίσης πριν την εκτέλεση της συνάρτησης PerformTrans της γραμμής 98, τα στοιχεία των λιστών ανάγνωσης και ενημέρωσης της $T(i,n_i)$ έχουν ταξινομηθεί κατά αύξουσα σειρά στις γραμμές 96 και 97, αντίστοιχα.

Έστω ότι κάποια executing PerformTrans της $T(i,n_i)$ εκτελεί επιτυχώς την εντολή CAS της γραμμής 124, ορίζοντας την $Ca(i,n_i)$, κατά την εκτέλεση του W_{AO} για κάποια t-μεταβλητή x, $x \in Trec(i,n_i)$.readList ή $x \in Trec(i,n_i)$.writeList, πριν την $Caof(i,n_i)$. Συμβολίζουμε με $Csa(i,n_i)$ την καθολική κατάσταση στην οποία αυτό συμβαίνει (για να την ξεχωρίζουμε από την $Ca(i,n_i)$ που μπορεί να οριστεί από οποιαδήποτε εντολή CAS μιας εκ των γραμμών 16, 28, 90, 100 και 124). Συμβολίζουμε με *FM(i,n)* την t-μεταβλητή x για την οποία εκτελέστηκε ο βρόχος W_{AO} και γράφτηκε η τιμή ABORTED στο $Trec(i,n_i)$.status. Εάν η $Csa(i,n_i)$ δεν ορίζεται τότε η $FM(i,n_i)$ είναι null. Σημειώνεται ότι εάν η $Csa(i,n_i)$ ορίζεται τότε εξ ορισμού της θα προηγείται της $Caof(i,n_i)$.

Λήμμα 6.13: Εάν η $Csa(i,n_i)$ ορίζεται τότε δεν είναι δυνατό να αποκτηθεί η t-μεταβλητή $FM(i,n_i)$ από τις executing PerformTrans της $T(i,n_i)$, μετά την $Csa(i,n_i)$.

Απόδειξη: Σημειώνεται ότι στην παρούσα απόδειξη χάριν απλότητας θα χρησιμοποιούμε το συμβολισμό FM αντί του $FM(i,n_i)$. Στην $Csa(i,n_i)$ κάποια executing PerformTrans της $T(i,n_i)$, έστω $PT_z(i,n_i)$, εκτέλεσε επιτυχώς την εντολή CAS της γραμμής 124, έστω CS_{124} η εντολή αυτή. Σημειώνεται ότι επειδή η $PT_z(i,n_i)$ ορίζει την $Csa(i,n_i)$ κατά την εκτέλεση του W_{AO} για την FM, από τον κώδικα (γραμμή 123) προκύπτει ότι η FM ανήκει στη readList της $T(i,n_i)$. Η FM εισάγεται στη readList της $T(i,n_i)$ στην καθολική κατάσταση $Crl_{FM}(i,n_i)$, μαζί με τα δεδομένα της $drl_{FM}(i,n_i)$, που επιστράφηκαν από την εκτέλεση της $GCDrl_{FM}(i,n_i)$ και διαβάστηκαν μέσω του $locrl_{FM}(i,n_i)$, ο οποίος αναγνώσθηκε από την $GCDrl_{FM}(i,n_i)$ στην $Crloc_{FM}(i,n_i)$.

Πριν την εκτέλεση της CS_{124}, η $PT_z(i,n_i)$ εκτελεί τη συνάρτηση GetCurrentData της γραμμής 118 για την FM (όχι της γραμμής 119 επειδή η FM ανήκει στη readList

της $T(i,n_i)$), έστω GCD' η κλήση της συνάρτησης αυτής, η οποία επιστρέφει τα τρέχοντα δεδομένα της FM, έστω $d_{FM}(z)$. Τα δεδομένα αυτά διαβάζονται από έναν Locator $loc_{FM}(z)$, ο οποίος αναγνώσθηκε από την GCD' στη γραμμή 43, έστω στην καθολική κατάσταση $Cloc_{FM}(z)$. Για να μπορεί η $PT_z(i,n_i)$ να εκτελέσει την CS_{124} πρέπει να αποτιμηθεί ως αληθής ο έλεγχος της γραμμής 123, το οποίο σημαίνει ότι πρέπει να ισχύει $d_{FM}(z) \neq drl_{FM}(i,n_i)$. Αυτό σημαίνει ότι τα δεδομένα της FM πρέπει να έχουν ενημερωθεί μετά την εισαγωγή τους στη readList, συγκεκριμένα μετά την $Crloc_{FM}(i,n_i)$, και πριν την ανάγνωσή τους από την $PT_z(i,n_i)$, δηλαδή πριν την $Cloc_{FM}(z)$, από τουλάχιστον μία δοσοληψία. Έστω $T(k,n_k)$, $i \neq k$, η πρώτη από αυτές τις δοσοληψίες, η οποία αποκτά την προσωρινή ιδιοκτησία ενημέρωσης της FM βάσει ενός Locator $loc_{FM}(k,n_k)$, έστω στην καθολική κατάσταση $C_{FM}(k,n_k)$, μετά την $Crloc_{FM}(i,n_i)$ και πριν την $Cloc_{FM}(z)$, και ενημερώνει την FM εκτελώντας επιτυχώς την εντολή CAS της γραμμής 130 πριν την $Cloc_{FM}(z)$, στην καθολική κατάσταση $C_C(k,n_k)$. Σημειώνεται ότι ισχύει $loc_{FM}(k,n_k)$.newData $\neq drl_{FM}(i,n_i)$.

Από τον κώδικα προκύπτει ότι μετά την επιτυχή εκτέλεση της CS_{122} από την $PT_z(i,n_i)$ στην $Csa(i,n_i)$, η $PT_z(i,n_i)$ αποχωρεί από τη συνάρτηση AcquireOwnerships. Αυτό σημαίνει ότι δε μπορεί η $PT_z(i,n_i)$ να αποκτήσει την προσωρινή ιδιοκτησία της FM, μετά την $Csa(i,n_i)$. Έτσι υποθέτουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι υπάρχει κάποια executing PerformTrans της $T(i,n_i)$, έστω $PT_j(i,n_i)$ όπου $PT_j(i,n_i) \neq PT_z(i,n_i)$, που εκτελεί επιτυχώς την εντολή CAS μιας εκ των γραμμών 130 και 132 για την FM μετά την $Csa(i,n_i)$, έστω στην καθολική κατάσταση C. Έστω CS η εντολή CAS μιας εκ των γραμμών 130 και 132. Για να μπορεί η $PT_j(i,n_i)$ να εκτελέσει την CS, πρέπει να έχει δει στη γραμμή 120 να ισχύει $Trec(i,n_i)$.status == ACTIVE, το οποίο σημαίνει ότι η γραμμή 120 πρέπει να εκτελεστεί από την $PT_j(i,n_i)$ πριν την $Csa(i,n_i)$. Αυτό σημαίνει ότι η εκτέλεση της συνάρτησης GetCurrentData στη γραμμή 118 για την FM (και όχι στη γραμμή 119 επειδή η FM ανήκει στη readList της $T(i,n_i)$) από την $PT_j(i,n_i)$, έστω GCD'' η συνάρτηση αυτή, ολοκληρώνεται πριν την $Csa(i,n_i)$. Σημειώνεται ότι η GCD'' επιστρέφει στην $PT_j(i,n_i)$ τον Locator $loc_{FM}(j)$ και τα δεδομένα της FM $d_{FM}(j)$ που διαβάστηκαν μέσω του $loc_{FM}(j)$, ο οποίος διαβάστηκε κατά την εκτέλεση της GCD'' στη γραμμή 43, έστω στη καθολική κατάσταση $Cloc_{FM}(j)$. Επειδή η GCD'' ολοκληρώνεται πριν την $Csa(i,n_i)$, προκύπτει ότι η $Cloc_{FM}(j)$ προηγείται της $Csa(i,n_i)$.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Σημειώνεται ότι η $Cloc_{FM}(j)$ έπεται της $Crloc_{FM}(i,n_i)$. Διακρίνω περιπτώσεις ανάλογα με το εάν η $Cloc_{FM}(j)$ προηγείται ή έπεται της $C_{FM}(k,n_k)$. Έστω ότι η $Cloc_{FM}(j)$ έπεται της $C_{FM}(k,n_k)$. Διακρίνω περιπτώσεις ανάλογα με το εάν η $Cloc_{FM}(j)$ προηγείται ή έπεται της $Cc(k,n_k)$. Εάν η $Cloc_{FM}(j)$ προηγείται της $Cc(k,n_k)$, τότε εφόσον η $Cloc_{FM}(j)$ έπεται της $C_{FM}(k,n_k)$ και προηγείται της $Cc(k,n_k)$ σημαίνει ότι εκτελείται εντός του διαστήματος $\alpha_{FM}(k,n_k)$. Επειδή η $T(k,n_k)$ ολοκληρώνεται επιτυχώς, από το Λήμμα 6.12 προκύπτει ότι η GCD'', θα επιστρέψει την τιμή του δείκτη $loc_{FM}(k,n_k)$.newData όπως αυτός αρχικοποιήθηκε από την $T(k,n_k)$, επομένως θα ισχύει $d_{FM}(j) \neq drl_{FM}(i,n_i)$. Εάν η $Cloc_{FM}(j)$ έπεται της $Cc(k,n_k)$, τότε η GCD'' θα επιστρέψει και πάλι τον δείκτη $loc_{FM}(k,n_k)$.newData, εάν δεν υπάρχει άλλη επιτυχημένη δοσοληψία μετά την $Cc(k,n_k)$ και πριν την $Cloc_{FM}(j)$ που να ενημερώνει την FM, επομένως θα ισχύει $d_{FM}(j) \neq drl_{FM}(i,n_i)$. Εάν υπάρχει κάποια δοσοληψία T που απέκτησε προσωρινή ιδιοκτησία ενημέρωσης στην FM στο διάστημα $[Cc(k,n_k), Cloc_{FM}(j)]$ και ενημέρωσε την FM (δηλαδή ολοκληρώθηκε επιτυχώς) πριν την $Cloc_{FM}(j)$, τότε τα δεδομένα $d_{FM}(j)$ θα διαβάστηκαν από την GCD'' μέσω ενός Locator loc που χρησιμοποιήθηκε για την απόκτηση προσωρινής ιδιοκτησίας ενημέρωσης στην FM από την T, για τον οποίο με βάση το Λήμμα 6.11 θα ισχύει loc.newData $\neq drl_{FM}(i,n_i)$. Επομένως σε κάθε περίπτωση θα ισχύει $d_{FM}(j) \neq drl_{FM}(i,n_i)$. Έτσι μετά την ολοκλήρωση της GCD'' θα αποτιμηθεί ως αληθής ο έλεγχος της 123 και η $PT_j(i,n_i)$ δε θα εκτελέσει τη CS, το οποίο είναι άτοπο.

Επομένως πρέπει η $Cloc_{FM}(j)$ να προηγείται της $C_{FM}(k,n_k)$. Στην περίπτωση αυτή, στην $C_{FM}(k,n_k)$ η $T(k,n_k)$ θα αποκτήσει την προσωρινή ιδιοκτησία της FM, το οποίο σημαίνει ότι θα ορίσει έναν νέο Locator για την FM, διαφορετικό του $loc_{FM}(j)$. Έτσι, η εκτέλεση της CS μετά την $C_{FM}(k,n_k)$, θα αποτύχει διότι ο Locator της FM δεν θα είναι ο $loc_{FM}(j)$[6.7]. Άτοπο. ■

[6.7] Σημειώνεται ότι δε θα μπορούσε να προκύψει το πρόβλημα ABA διότι, όπως έχει αναφερθεί, ο αλγόριθμος NBSTM εκτελείται σε κλειστό μοντέλο μνήμης και η μεταβλητή loc_{fm} είναι μια από τις ενεργές τοπικές μεταβλητές της p_i.

Λήμμα 6.14: Έστω x οποιαδήποτε t-μεταβλητή που περιέχεται στη readList μιας δοσοληψίας $T(i,n_i)$, για την οποία ορίζεται η $C_x(i,n_i)$. Τότε καμία επιτυχημένη δοσοληψία $T(j,n_j)$, $i \neq j$, δε μπορεί να αποκτήσει προσωρινή ιδιοκτησία ενημέρωσης της x μετά την $Crloc_x(i,n_i)$ και πριν το τέλος της $a_x(i,n_i)$.

Απόδειξη: Στη $C_x(i,n_i)$ κάποια executing PerformTrans της $T(i,n_i)$, έστω $PT_z(i,n_i)$, εκτελεί επιτυχώς την εντολή CAS μιας εκ των γραμμών 130 και 132 για τη x, έστω CS_x η εντολή αυτή. Η x εισάγεται στη readList της $T(i,n_i)$ στην καθολική κατάσταση $Crl_x(i,n_i)$, μαζί με τα δεδομένα της $drl_x(i,n_i)$, που επιστράφηκαν από την εκτέλεση της $GCDrl_x(i,n_i)$ και διαβάστηκαν μέσω του $locrl_x(i,n_i)$, ο οποίος αναγνώσθηκε από την $GCDrl_x(i,n_i)$ στην $Crloc_x(i,n_i)$. Υποθέτουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι υπάρχει κάποια δοσοληψία $T(j,n_j)$, $i \neq j$, η οποία εκτελεί επιτυχώς την εντολή CAS της γραμμής 130 για τη x, αποκτώντας προσωρινή ιδιοκτησία ενημέρωσης στη x, μετά την $Crloc_x(i,n_i)$ και πριν το τέλος της $a_x(i,n_i)$, έστω στην καθολική κατάσταση $C_x(j,n_j)$. Έστω ότι η $PT_j(i,n_i)$ αποκτά την προσωρινή ιδιοκτησία ενημέρωσης της x στη $C_x(j,n_j)$ με βάση τον Locator loc. Σημειώνεται ότι ισχύει loc.newData $\neq drl_x(i,n_i)$.

Από το Λήμμα 6.9 προκύπτει ότι η $C_x(j,n_j)$ δε μπορεί να ορίζεται εντός του διαστήματος $a_x(i,n_i)$, επομένως η $C_x(j,n_j)$ προηγείται της $C_x(i,n_i)$ (και έπεται της $Crloc_x(i,n_i)$). Πριν την εκτέλεση της CS_x, η $PT_z(i,n_i)$ εκτελεί τη συνάρτηση GetCurrentData της γραμμής 118 για τη x (όχι της γραμμής 119 επειδή η x ανήκει στη readList της $T(i,n_i)$), έστω GCD η κλήση της συνάρτησης αυτής, η οποία επιστρέφει τα τρέχοντα δεδομένα της x, έστω $d_x(z)$. Τα δεδομένα αυτά διαβάζονται από έναν Locator $loc_x(z)$, ο οποίος αναγνώσθηκε από την GCD στη γραμμή 43, έστω στην καθολική κατάσταση $Cloc_x(z)$. Διακρίνω περιπτώσεις ανάλογα με το εάν η $Cloc_x(z)$ προηγείται ή έπεται της $C_x(j,n_j)$. Έστω ότι η $Cloc_x(z)$ έπεται της $C_x(j,n_j)$. Σημειώνεται ότι από υπόθεση η $T(j,n_j)$ ολοκληρώνεται επιτυχώς στην καθολική κατάσταση $Cc(j,n_j)$. Διακρίνω περιπτώσεις ανάλογα με το εάν η $Cloc_x(z)$ προηγείται ή έπεται της $Cc(j,n_j)$.

Εάν η $Cloc_x(z)$ προηγείται της $Cc(k,n_k)$, τότε εφόσον η $Cloc_x(z)$ έπεται της $C_x(j,n_j)$ και προηγείται της $Cc(j,n_j)$ σημαίνει ότι εκτελείται εντός του διαστήματος $a_x(j,n_j)$. Επειδή η $T(j,n_j)$ ολοκληρώνεται επιτυχώς, από το Λήμμα 6.12 προκύπτει ότι η GCD, θα επιστρέψει την τιμή του δείκτη loc.newData όπως αυτός αρχικοποιήθηκε από την $T(k,n_k)$, επομένως θα ισχύει $d_x(z) \neq drl_x(i,n_i)$. Εάν η $Cloc_x(z)$ έπεται της $Cc(j,n_j)$, τότε

η GCD θα επιστρέψει και πάλι τον δείκτη loc.newData, εάν δεν υπάρχει άλλη επιτυχημένη δοσοληψία μετά την $Cc(j,n_j)$ και πριν την $Cloc_x(z)$ που να ενημερώνει τη x, επομένως θα ισχύει και πάλι $d_x(z) \neq drl_x(i,n_i)$. Εάν υπάρχει κάποια δοσοληψία T που απέκτησε προσωρινή ιδιοκτησία ενημέρωσης στη x στο διάστημα $[Cc(j,n_j), Cloc_x(z)]$ και ενημέρωσε τη x (δηλαδή ολοκληρώθηκε επιτυχώς) πριν την $Cloc_x(z)$, τότε τα δεδομένα $d_x(z)$ θα διαβάστηκαν από την GCD μέσω ενός Locator loc' που χρησιμοποιήθηκε για την απόκτηση προσωρινής ιδιοκτησίας ενημέρωσης στη x από την T, για τον οποίο με βάση το Λήμμα 6.11 θα ισχύει loc'.newData $\neq drl_{FM}(i,n_i)$. Επομένως σε κάθε περίπτωση θα ισχύει $d_x(z) \neq drl_x(i,n_i)$. Έτσι μετά την ολοκλήρωση της GCD θα αποτιμηθεί ως αληθής ο έλεγχος της 123 και η $PT_z(i,n_i)$ δε θα εκτελέσει τη CS_x, το οποίο είναι άτοπο.

Επομένως πρέπει η $Cloc_x(z)$ να προηγείται της $C_x(j,n_j)$. Στην περίπτωση αυτή, στην $C_x(j,n_j)$ η $T(j,n_j)$ θα αποκτήσει την προσωρινή ιδιοκτησία της x, το οποίο σημαίνει ότι θα ορίσει έναν νέο Locator για την x, διαφορετικό του $loc_x(z)$. Έτσι, η εκτέλεση της CS_x μετά την $C_x(j,n_j)$, θα αποτύχει διότι ο Locator της x δεν θα είναι ο $loc_x(z)$[6.8]. Άτοπο. ∎

Λήμμα 6.15: Έστω x οποιαδήποτε t-μεταβλητή που περιέχεται στη readList ή τη writeList μιας δοσοληψίας $T(i,n_i)$, για την οποία ορίζεται η $C_x(i,n_i)$. Τότε το $Trec(i,n_i)$.status δε μπορεί να ενημερωθεί μέσω της εκτέλεσης της εντολής CAS της γραμμής 124, δηλαδή δε μπορεί να οριστεί η $Csa(i,n_i)$, μετά τη $C_x(i,n_i)$ και κατά την εκτέλεση του W_{AO} για τη x από τις executing PerformTrans της $T(i,n_i)$.

Απόδειξη: Στη $C_x(i,n_i)$ κάποια executing PerformTrans της $T(i,n_i)$, έστω $PT_z(i,n_i)$, εκτελεί επιτυχώς την εντολή CAS μιας εκ των γραμμών 130 και 132 για τη x, έστω CS_x η εντολή αυτή, με βάση τον Locator loc για τον οποίο ισχύει loc.tran == $Trec(i,n_i)$. Μετά την επιτυχή εκτέλεση της CS_x από την $PT_z(i,n_i)$, η $PT_z(i,n_i)$ αποχωρεί

[6.8] Σημειώνεται ότι δε θα μπορούσε να προκύψει το πρόβλημα ABA διότι, όπως έχει αναφερθεί, ο αλγόριθμος NBSTM εκτελείται σε κλειστό μοντέλο μνήμης και η μεταβλητή loc_{fm} είναι μια από τις ενεργές τοπικές μεταβλητές της p_i.

από τον W_{AO} για τη x. Αυτό σημαίνει ότι δεν μπορεί η $PT_j(i,n_i)$ να εκτελέσει επιτυχώς την CS_x και στην συνέχεια να εκτελέσει επιτυχώς την εντολή CAS της γραμμής 124, έστω CS_{124} αυτή η εντολή, κατά την εκτέλεση του W_{AO} για τη x. Έτσι, υποθέτουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι υπάρχει κάποια executing PerformTrans της $T(i,n_i)$, έστω $PT_j(i,n_i)$ και $PT_j(i,n_i) \neq PT_z(i,n_i)$, η οποία εκτελεί επιτυχώς τη CS_{124} και ορίζει τη $Csa(i,n_i)$, μετά τη $C_x(i,n_i)$ και κατά την εκτέλεση του W_{AO} για τη x.

Σημειώνεται ότι επειδή η $PT_j(i,n_i)$ ορίζει την $Ca(i,n_i)$ κατά την εκτέλεση του W_{AO} για τη x, από τον κώδικα (γραμμή 123) προκύπτει ότι η x ανήκει στη readList της $T(i,n_i)$. Πριν την εκτέλεση της CS_{124}, η $PT_j(i,n_i)$ εκτελεί τη συνάρτηση GetCurrentData της γραμμής 118 για την x (όχι της γραμμής 119 επειδή η x ανήκει στη readList της $T(i,n_i)$), έστω GCD η κλήση της συνάρτησης αυτής, η οποία επιστρέφει τα τρέχοντα δεδομένα της x, έστω $d_x(j)$. Τα δεδομένα αυτά διαβάζονται από έναν Locator $loc_x(j)$, ο οποίος αναγνώσθηκε από την GCD στη γραμμή 43, έστω στην καθολική κατάσταση $Cloc_x(j)$.

Διακρίνω περιπτώσεις ανάλογα με το εάν η $Cloc_x(j)$ προηγείται ή έπεται της $C_x(i,n_i)$. Έστω ότι η $Cloc_x(j)$ έπεται της $C_x(i,n_i)$. Με βάση το Λήμμα 6.1 για να μπορεί η $PT_j(i,n_i)$ να ορίσει την $Csa(i,n_i)$ μετά την $C_x(i,n_i)$ δεν πρέπει να έχει ήδη οριστεί είτε η $Ca(i,n_i)$ είτε η $Cc(i,n_i)$. Αυτό σημαίνει ότι το διάστημα $\alpha_x(i,n_i)$ ολοκληρώνεται με την επιτυχή εκτέλεση της CS_{124} από την $PT_j(i,n_i)$. Επομένως, στην περίπτωση αυτή η $Cloc_x(j)$ εκτελείται εντός του διαστήματος $\alpha_x(i,n_i)$ διαβάζοντας τον loc βάσει του οποίου αποκτήθηκε η προσωρινή ιδιοκτησία της x από την $PT_z(i,n_i)$ στη $C_x(i,n_i)$. Στην περίπτωση αυτή η GCD επιστρέφει στη γραμμή 51 (λόγω του ελέγχου της γραμμής 50) και ο έλεγχος της γραμμής 122 θα αποτιμηθεί ως μη αληθής από την $PT_j(i,n_i)$, διότι θα ισχύει $loc_x(j)$ == loc και loc.tran == $Trec(i,n_i)$, και δε θα είναι δυνατή η εκτέλεση της CS_{124} από την $PT_j(i,n_i)$, το οποίο είναι άτοπο. Επομένως πρέπει η $Cloc_x(j)$ να προηγείται της $C_x(i,n_i)$.

Για να μπορεί η $PT_j(i,n_i)$ να εκτελέσει την CS_{124} πρέπει να αποτιμηθεί ως αληθής ο έλεγχος της γραμμής 123, το οποίο σημαίνει ότι πρέπει να ισχύει $d_x(j) \neq drl_x(i,n_i)$. Αυτό σημαίνει ότι τα δεδομένα της x πρέπει να έχουν ενημερωθεί μετά την εισαγωγή τους στη readList, συγκεκριμένα μετά την $Crloc_x(i,n_i)$, και πριν την ανάγνωσή τους από την $PT_j(i,n_i)$, δηλαδή πριν την $Cloc_x(j)$, από τουλάχιστον μία δοσοληψία T. Δεδομένου ότι η $Cloc_x(j)$ προηγείται της $C_x(i,n_i)$, από το Λήμμα 6.14 προκύπτει ότι δε μπορεί να υπάρχει τέτοια δοσοληψία T, το οποίο είναι άτοπο. ■

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Από τον κώδικα της συνάρτησης AcquireOwnerships (γραμμή 115) προκύπτει ότι τουλάχιστον μία executing PerformTrans της $T(i,n_i)$ θα προσπαθήσει να αποκτήσει τις προσωρινές ιδιοκτησίες των t-μεταβλητών που η $T(i,n_i)$ επιθυμεί, με προκαθορισμένη σειρά, πριν την $Caof(i,n_i)$. Αυτό σημαίνει ότι τουλάχιστον μία από τις executing PerformTrans της $T(i,n_i)$ θα εκτελέσει τον W_{AO} για κάποιες t-μεταβλητές x (ίσως και όλες), $x \in Trec(i,n_i)$.readList ή $x \in Trec(i,n_i)$.writeList, πριν την $Caof(i,n_i)$. Επίσης από το Λήμμα προκύπτει ότι πριν την $Caof(i,n_i)$ οι executing PerformTrans της $Caof(i,n_i)$ που θα εκτελέσουν τον W_{AO} για κάποια t-μεταβλητή x, $x \in Trec(i,n_i)$.readList ή $x \in Trec(i,n_i)$.writeList, θα αποχωρήσουν από τον W_{AO} είτε αποκτώντας την προσωρινή ιδιοκτησία της x είτε γράφοντας την τιμή ABORTED στο $Trec(i,n_i)$.status, δηλαδή ορίζοντας την $Csa(i,n_i)$.

Με βάση το Λήμμα 6.13 καμία από τις executing PerformTrans της $T(i,n_i)$ δε μπορεί να αποκτήσει την t-μεταβλητή $FM(i,n_i)$ μετά την $Csa(i,n_i)$. Επίσης, η $FM(i,n_i)$ δε μπορεί να αποκτήθηκε από τις executing PerformTrans της $T(i,k)$ πριν την $Csa(i,n_i)$, διότι εάν αυτό συνέβαινε με βάση το Λήμμα προκύπτει ότι δε θα ήταν δυνατό να οριστεί η $Csa(i,n_i)$ κατά την εκτέλεση του W_{AO} για την $FM(i,n_i)$. Εφόσον οι προσωρινές ιδιοκτησίες των t-μεταβλητών που η $T(i,n_i)$ επιθυμεί, αποκτώνται με προκαθορισμένη σειρά, από το Λήμμα 6.13 προκύπτει ότι δε μπορούν να έχουν αποκτηθεί οι t-μεταβλητές της $T(i,n_i)$ που έπονται στην καθορισμένη διάταξη της $FM(i,n_i)$ (εφόσον δε μπορεί να αποκτηθεί η $FM(i,n_i)$). Σημειώνεται ότι η $Csa(i,n_i)$ προηγείται της $Caof(i,n_i)$. Έτσι, προκύπτει ότι έχουν αποκτηθεί οι προσωρινές ιδιοκτησίες των t-μεταβλητών της $T(i,n_i)$ που προηγούνται της $FM(i,n_i)$. Εάν αυτό δεν ίσχυε για κάποια από αυτές, έστω x, τότε κατά την εκτέλεση του W_{AO} για τη x, από το Λήμμα προκύπτει ότι θα γραφεί η τιμή ABORTED στο $Trec(i,n_i)$.status, το οποίο με βάση το Λήμμα 6.1 σημαίνει ότι δε μπορεί να οριστεί η $Csa(i,n_i)$ κατά την εκτέλεση του W_{AO} για την $FM(i,n_i)$ (που έπεται στη διάταξη της x).

Πόρισμα 6.16: *Εάν η $Csa(i,n_i)$ ορίζεται, τότε: i) Η $FM(i,n_i)$ είναι η μικρότερη στη διάταξη t-μεταβλητή που ανήκει στη readList ή στη writeList της $T(i,n_i)$ που κάποια executing PerformTrans της $T(i,n_i)$ δεν κατάφερε να αποκτήσει, πριν την $Csa(i,n_i)$, και*

ii) Καμία executing PerformTrans της $T(i,n_i)$ δεν αποκτά κάποια t-μεταβλητή που είναι μεγαλύτερη της $FM(i,n_i)$ μετά την $Csa(i,n_i)$.

Σημειώνεται ότι από το Πόρισμα 6.4 προκύπτει ότι στην $Caof(i,n_i)$ το status της δοσοληψίας $T(i,n_i)$ μπορεί να είναι ABORTED ή ACTIVE. Εάν έχει τιμή ABORTED σημαίνει ότι η $Csa(i,n_i)$ προηγείται της $Caof(i,n_i)$. Εάν έχει τιμή ACTIVE, με βάση το ότι οι προσωρινές ιδιοκτησίες των t-μεταβλητών που η $T(i,n_i)$ επιθυμεί αποκτώνται με προκαθορισμένη σειρά και το Λήμμα προκύπτει ότι έχουν αποκτηθεί οι προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που η $T(i,n_i)$ επιθυμεί. Έτσι, στην $Caof(i,n_i)$ ισχύει $Trec(i,n_i)$.status == ACTIVE εάν και μόνο εάν οι executing PerformTrans της $T(i,n_i)$ έχουν αποκτήσει, πριν από την $Caof(i,n_i)$, τις προσωρινές ιδιοκτησίες όλως των t-μεταβλητών που η $T(i,n_i)$ επιθυμεί.

Επίσης εάν στην $Caof(i,n_i)$ ισχύει $Trec(i,n_i)$.status == ACTIVE τότε με βάση το Λήμμα προκύπτει ότι δε μπορεί να οριστεί η $Ca(i,n_i)$ λόγω της επιτυχούς εκτέλεσης της εντολής CAS της γραμμής 124. Σημειώνεται ότι ο μόνος τρόπος ορισμού της $Ca(i,n_i)$ κατά το δεύτερο στάδιο της εκτέλεσης της $T(i,n_i)$ και μετά την κλήση της συνάρτησης PerformTrans στη γραμμή 98, είναι με την επιτυχή εκτέλεση της εντολής CAS της γραμμής 124. Ορίζουμε Α το σύνολο που περιέχει τις executing PerformTrans της $T(i,n_i)$ που παρατηρούν $Trec(i,n_i)$.status == ACTIVE στην $Caof(i,n_i)$ και με βάση τον κώδικα (γραμμή 106) εκτελούν την εντολή CAS της γραμμής 107, έστω CS η εντολή αυτή. Με βάση τα παραπάνω, η πρώτη executing PerformTrans του συνόλου Α που θα εκτελέσει πρώτη την CS, θα την εκτελέσει επιτυχώς, ορίζοντας με αυτό τον τρόπο την $Cc(i,n_i)$ και χαρακτηρίζοντας την $T(i,n_i)$ ως επιτυχημένη. Έτσι, προκύπτει το παρακάτω Πόρισμα.

Πόρισμα 6.17: Μια δοσοληψία $T(i,n_i)$ χαρακτηρίζεται ως επιτυχημένη, δηλαδή ορίζεται η $Cc(i,n_i)$, εάν και μόνο εάν οι executing PerformTrans της έχουν αποκτήσει τις προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που αυτή επιθυμεί.

Στο σημείο αυτό αποδίδουμε σημεία σειριοποίησης στις δοσοληψίες. Σημειώνεται ότι για μια δοσοληψία $T(i,n_i)$ από το Λήμμα 6.1 προκύπτει ότι μόνο μία εκ των $Cc(i,n_i)$ και $Ca(i,n_i)$ μπορεί να οριστεί. Επομένως δε χρειάζεται να αποδοθεί σημείο

σειριοποίησης για μια μη επιτυχημένη δοσοληψία $T(i,n_i)$, για την οποία δηλαδή έχει οριστεί η $Ca(i,n_i)$, διότι αυτή δεν μπορεί να εκτελέσει επιτυχώς την εντολή CAS της γραμμής 107 και να ορίσει την $Cc(i,n_i)$, ενημερώνοντας έτσι οποιαδήποτε t-μεταβλητή. Έτσι αποδίδεται σημείο σειριοποίησης σε κάποια δοσοληψία $T(i,n_i)$ μόνο εάν αυτή είναι επιτυχημένη, δηλαδή έχει εκτελέσει τη λειτουργία CommitTransaction και κάποια από τις executing PerformTrans της $T(i,n_i)$ έχει εκτελέσει επιτυχώς την εντολή CAS της γραμμής 107, ορίζοντας τη $Cc(i,n_i)$. Τοποθετούμε το σημείο σειριοποίησης της $T(i,n_i)$ στη $Cc(i,n_i)$.

Λήμμα 6.18: *Έστω GCD μια οποιαδήποτε κλήση της λειτουργίας GetCurrentData από την $T(i,n_i)$ για κάποια t-μεταβλητή x. Έστω ότι η GCD ολοκληρώνεται στην καθολική κατάσταση C και επιστρέφει έναν δείκτη σε δεδομένα, τον οποίο έχει διαβάσει από έναν Locator loc. Έστω $T(j,n_j)$, $i \neq j$, η δοσοληψία που δημιούργησε το loc ώστε να αποκτήσει βάσει αυτού την προσωρινή ιδιοκτησία της x. Πριν την C είτε θα έχει οριστεί μια εκ των $Cc(j,n_j)$ και $Ca(j,n_j)$, είτε θα ισχύει loc.oldData == loc.newData.*

Απόδειξη: Η GCD διαβάζει στην γραμμή 43 την τιμή του loc, έστω στην καθολική κατάσταση C_{43}. Σημειώνεται ότι ο έλεγχος της γραμμής 50 δε μπορεί να αποτιμηθεί σε TRUE διότι από υπόθεση $i \neq j$, επομένως η GCD δε μπορεί να επιστρέψει στη γραμμή 51. Στη συνέχεια η GCD διαβάζει στη γραμμή 46 την τιμή του $Trec(j,n_j)$.status, έστω στην καθολική κατάσταση C_{46}. Εάν ο έλεγχος μιας εκ των γραμμών 53 και 54 αποτιμηθεί ως αληθής, η GCD θα επιστρέψει στις γραμμές αυτές, και θα έχει οριστεί πριν την C_{46}, άρα και πριν την C, είτε η $Cc(j,n_j)$ είτε η $Ca(j,n_j)$, αντίστοιχα. Σε αντίθετη περίπτωση, στην C_{46}, θα ισχύει $Trec(j,n_j)$.status == ACTIVE. Εάν ο έλεγχος της γραμμής 49 αποτιμηθεί ως αληθής τότε η GCD θα επιστρέψει στη γραμμή 50 και θα ισχύει loc.oldData == loc.newData. Σε αντίθετη περίπτωση η GCD θα εκτελέσει τη συνάρτηση TryHelpTransaction της γραμμής 52. Από το Πόρισμα 6.7 προκύπτει ότι πριν την ολοκλήρωση της συνάρτησης αυτής και άρα πριν την C, θα έχει οριστεί μία εκ των $Cc(j,n_j)$ και $Ca(j,n_j)$. ∎

Έστω μια εκτέλεση α και έστω C μια οποιαδήποτε καθολική κατάσταση της α στην οποία κάποια δοσοληψία $T(i,n_i)$ είναι εκκρεμής. Συμβολίζουμε με $O_C = \{x_1, x_2, \ldots, x_k\}$ $O_C = \{x_1, x_2, \ldots, x_k\}$ το σύνολο των t-μεταβλητών που περιέχονται στη readList της

$T(i,n_i)$ στην C. Σημειώνεται ότι το πλήθος k των στοιχείων του συνόλου O_C εξαρτάται από την καθολική κατάσταση C. Συμβολίζουμε με $D(O_C) = \{d_1, d_2, \ldots, d_k\}$ το σύνολο των δεικτών στα δεδομένα των t-μεταβλητών του O_C που περιέχονται στη readList της $T(i,n_i)$ στην C (υπενθυμίζεται ότι κάθε δείκτης d_b, $1 \leq b \leq k$, αναπαριστά την έκδοση της x_b τη χρονική στιγμή της καθολικής ανάγνωσής της). Υπενθυμίζουμε ότι κάθε επιτυχημένη δοσοληψία σειριοποιείται βάσει μιας καθολικής κατάστασης της α και πιο συγκεκριμένα της τελευταίας που προηγείται της τελευταίας Validate που εκτελεί η δοσοληψία. Έτσι, δεδομένης μιας καθολικής κατάστασης C και των σημείων σειριοποίησης των επιτυχημένων δοσοληψιών στην α, μπορούμε να καθορίσουμε την επακολουθία των επιτυχημένων δοσοληψιών της α που σειριοποιούνται σε καθολικές καταστάσεις που προηγούνται της C. Λέμε ότι το σύνολο $D(O_C)$ είναι συνεπές αν κάθε στοιχείο d_b', $1 \leq b \leq k$, του συνόλου αυτού δείχνει στα δεδομένα που εγγράφονται από την τελευταία επιτυχημένη δοσοληψία που σειριοποιείται πριν τη C και ενημερώνει την αντίστοιχη x_b.

Λήμμα 6.19: Έστω μια επιτυχημένη δοσοληψία $T(i,n_i)$. Το σύνολο $D(O_{Cc(i,ni)})$ είναι συνεπές στη $Cc(i,n_i)$.

Απόδειξη: Έστω $D(O_{Cc(i,ni)}) = \{d_1, d_2, \ldots, d_k\}$. Θα δειχθεί ότι για κάθε b, $1 \leq b \leq k$, το d_b εγγράφεται από την τελευταία επιτυχημένη δοσοληψία που σειριοποιείται πριν την $Cc(i,n_i)$ και ενημερώνει το x_b. Χάριν απλότητας στη συνέχεια θα αναφερόμαστε στα x_b και d_b ως x και d, αντίστοιχα. Σημειώνεται ότι το d διαβάζεται από την $T(i,n_i)$ μέσω ενός Locator, ο οποίος διαβάζεται από την $GCDrl_x(i,n_i)$ στην $Crloc_x(i,n_i)$ και εγγράφεται στη readList της $T(i,n_i)$ μαζί με τη x στην $Crl_x(i,n_i)$. Έστω, δια της μεθόδου της εις άτοπο απαγωγής, ότι υπάρχει κάποια t-μεταβλητή $x \in O_C$ τέτοια ώστε η τελευταία επιτυχημένη δοσοληψία, έστω $T(j,n_j)$, που ενημερώνει τη x και σειριοποιείται πριν την $Cc(i,n_i)$ γράφει την τιμή $d' \neq d$ στη x, με βάση κάποιο Locator $loc_x(j,n_j)$ (για τον οποίο ισχύει $loc_x(j,n_j)$.newData == d'). Σημειώνεται ότι η $T(j,n_j)$ σειριοποιείται στην $Cc(j,n_j)$, η οποία από υπόθεση προηγείται της $Cc(i,n_i)$.

Από το Πόρισμα 6.17, προκύπτει ότι η $T(i,n_i)$ έχει αποκτήσει την ιδιοκτησία του x στην καθολική κατάσταση $C_x(i,n_i)$ η οποία προηγείται της $Cc(i,n_i)$. Επίσης με βάση το ίδιο Πόρισμα προκύπτει ότι η $T(j,n_j)$ έχει αποκτήσει την ιδιοκτησία του x στην καθολική κατάσταση $C_x(j,n_j)$ η οποία προηγείται της $Cc(j,n_j)$. Εφόσον η $Cc(j,n_j)$

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

προηγείται της $Cc(i,n_i)$ προκύπτει ότι η $C_x(j,n_j)$ προηγείται της $Cc(i,n_i)$. Από το Λήμμα 6.9 προκύπτει ότι η $C_x(j,n_j)$ δε μπορεί να ορίζεται εντός του διαστήματος $\alpha_x(i,n_i)$, επομένως η $C_x(j,n_j)$ προηγείται της $C_x(i,n_i)$. Από το Λήμμα 6.14 προκύπτει ότι η $C_x(j,n_j)$ δε μπορεί να ορίζεται εντός του διαστήματος $[Crloc_x(i,n_i), Cc(i,n_i)]$. Επομένως η $C_x(j,n_j)$ πρέπει να προηγείται της $Crloc_x(i,n_i)$.

Διακρίνω περιπτώσεις ανάλογα με το εάν η $Cc(j,n_j)$ προηγείται ή έπεται της $Crloc_x(i,n_i)$. Έστω ότι η $Cc(j,n_j)$ έπεται της $Crloc_x(i,n_i)$. Αυτό σημαίνει ότι η $Crloc_x(i,n_i)$ έπεται της $C_x(j,n_j)$ και προηγείται της $Cc(j,n_j)$, δηλαδή ορίζεται εντός του διαστήματος $\alpha_x(j,n_j)$. Επειδή η $T(j,n_j)$ ολοκληρώνεται επιτυχώς, από το Λήμμα 6.12 προκύπτει ότι η $GCDrl_x(i,n_i)$, θα επιστρέψει την τιμή $d' \neq drl_{FM}(i,n_i)$ του δείκτη $loc_x(j,n_j)$.newData όπως αυτός αρχικοποιήθηκε από την $T(j,n_j)$, το οποίο είναι άτοπο. Επομένως πρέπει η $Cc(j,n_j)$ να προηγείται της $Crloc_x(i,n_i)$. Στην περίπτωση αυτή, για να μπορεί η $T(i,n_i)$ να διαβάσει στην $Crloc_x(i,n_i)$ την τιμή d για τη x πρέπει να υπάρχει κάποια επιτυχημένη δοσοληψία $T(k,n_k)$, $k \neq i \neq j$, που αποκτά ιδιοκτησία ενημέρωσης στη x, έστω στην καθολική κατάσταση $C_x(k,n_k)$, μετά τη $Cc(j,n_j)$ και πριν την $Crloc_x(i,n_i)$, βάσει κάποιου Locator loc' για τον οποίο ισχύει loc'.newData == d.

Η $T(k,n_k)$ σειριοποιείται στην καθολική κατάσταση $Cc(k,n_k)$. Η $Cc(k,n_k)$ μπορεί να έπεται ή να προηγείται της $Crloc_x(i,n_i)$. Έστω ότι η $Cc(k,n_k)$ έπεται της $Crloc_x(i,n_i)$. Στην περίπτωση αυτή η $Crloc_x(i,n_i)$ εκτελείται εντός του διαστήματος $\alpha_x(k,n_k)$ και επομένως η $GCDrl_x(i,n_i)$ επιστρέφει τον Locator loc'. Επειδή η $T(k,n_k)$ ολοκληρώνεται επιτυχώς και ο loc' περιγράφει κάποια προσωρινή ιδιοκτησία ενημέρωσης (το οποίο σημαίνει ότι ισχύει loc'.oldData ≠ loc'.newData), από το Λήμμα 6.18 προκύπτει ότι πριν την ολοκλήρωση της $GCDrl_x(i,n_i)$ θα έχει οριστεί η $Cc(k,n_k)$. Επειδή η ολοκλήρωση της $GCDrl_x(i,n_i)$ προηγείται της $Crloc_x(i,n_i)$, προκύπτει ότι η $Cc(k,n_k)$ προηγείται της $Crloc_x(i,n_i)$. Επομένως σε κάθε περίπτωση η $Cc(k,n_k)$ προηγείται της $Crloc_x(i,n_i)$ και επειδή η $Crloc_x(i,n_i)$ προηγείται της $Cc(i,n_i)$, προκύπτει ότι η $Cc(k,n_k)$ προηγείται της $Cc(i,n_i)$. Επίσης, η $Cc(k,n_k)$ έπεται της $Cc(j,n_j)$. Επομένως η $T(j,n_j)$ δεν είναι η τελευταία δοσοληψία επιτυχημένη δοσοληψία που ενημερώνει τη x και σειριοποιείται πριν από τη $Cc(i,n_i)$, το οποίο είναι άτοπο. ■

Από το Πόρισμα 6.8 προκύπτει ότι η $Cc(i,n_i)$ είναι καλά ορισμένη. Επίσης από το Λήμμα 6.19 προκύπτει ότι τα δεδομένα του $D(O_{Cc(i,ni)})$ είναι συνεπή στη $Cc(i,n_i)$. Έτσι προκύπτει το παρακάτω Θεώρημα.

Θεώρημα 6.20: Ο αλγόριθμος NBSTM είναι σειριοποιήσιμος.

6.2.2. Ελευθερία Κλειδωμάτων

Στην παρούσα Ενότητα αποδεικνύεται ότι ο αλγόριθμος NBSTM ικανοποιεί την ιδιότητα τερματισμού της ελευθερίας κλειδωμάτων.

***Λήμμα 6.21**: Ο κώδικας του χρήστη δε μπορεί να εκτελεί επ' άπειρο κάποιο βρόχο λόγω ασυνέπειας των δεδομένων που χρησιμοποιεί.*

Απόδειξη: Η απόδειξη είναι αντίστοιχη με την απόδειξη του ίδιου Λήμματος στον αλγόριθμο DSTM και παραλείπεται.

Από τον κώδικα (γραμμές 43, 51, 52, 126 και 127) προκύπτει ότι μια δοσοληψία $T(i,n_i)$ βοηθά μία άλλη δοσοληψία $T(j,n_j)$ να ολοκληρωθεί, μόνο εάν η $T(j,n_j)$ κατέχει κάποια από τις προσωρινές ιδιοκτησίες που η $T(i,n_i)$ επιθυμεί. Επίσης από τον κώδικα (γραμμές 96, 97, 115, 130 και 132) προκύπτει ότι οι δοσοληψίες στον NBSTM αποκτούν τις ιδιοκτησίες που επιθυμούν με συγκεκριμένη καθολική σειρά. Έτσι προκύπτει το παρακάτω πόρισμα.

***Πόρισμα 6.22**: Καμία δοσοληψία δε μπορεί να εκτελεί επ' άπειρο κάποιο βρόχο εξαιτίας της υλοποίησης του μηχανισμού βοήθειας του NBSTM.*

***Θεώρημα 6.23**: Ο αλγόριθμος NBSTM ικανοποιεί την ιδιότητα τερματισμού της ελευθερίας κλειδωμάτων.*

Απόδειξη: Έστω μια οποιαδήποτε άπειρη εκτέλεση α του αλγορίθμου NBSTM. Έστω ότι στην α εκτελούνται k δοσοληψίες $T(1,n_1)$, $T(2,n_2)$, ... $T(k,n_k)$ από τις διεργασίες p_1, p_2, ..., p_k, αντίστοιχα. Υποθέτουμε, δια της μεθόδου της εις άτοπο απαγωγής, ότι ο αλγόριθμος NBSTM δεν ικανοποιεί την ιδιότητα τερματισμού της ελευθερίας κλειδωμάτων. Επομένως οι δοσοληψίες $T(b,n_b)$, $1 \le b \le k$, πρέπει να εκτελούν κάποιον άπειρο βρόχο, έτσι ώστε καμία από αυτές να μην ολοκληρώνεται.

Από το Λήμμα 6.21 προκύπτει ότι ο κώδικας του χρήστη δε μπορεί ευθύνεται για τον μη τερματισμό των δοσοληψιών αυτών. Επομένως πρέπει οι δοσοληψίες να εκτελούν κάποιον άπειρο βρόχο του κώδικα του NBSTM. Από το Πόρισμα 6.23 προκύπτει ότι η υλοποίηση του μηχανισμού βοήθειας δε μπορεί ευθύνεται για τον μη τερματισμό των δοσοληψιών $T(b,n_b)$, $1 \le b \le k$. Έτσι, από τον κώδικα προκύπτει, ότι οι δοσοληψίες αυτές μπορεί να μην ολοκληρώνονται εξαιτίας της επ' άπειρο εκτέλεσης του μοναδικού βρόχου W_{AO} του NBSTM. Έστω ότι όλες οι δοσοληψίες $T(b,n_b)$, $1 \le b \le k$, και οι executing PerformTrans τους εκτελούν επ' άπειρο τον W_{AO} για κάποια t-μεταβλητή x_b (η x_b μπορεί να διαφορετική ή όχι για κάθε δοσοληψία $T(b,n_b)$). Στη συνέχεια μελετούμε τη συμπρεριφορά μιας οποιαδήποτε από τις δοσοληψίες $T(b,n_b)$, $1 \le b \le k$, έστω $T(i,n_i)$ που εκτελεί τον W_{AO} για τη x_i (δηλαδή b=i).

Από τον κώδικα (γραμμές 120 και 121) προκύπτει ότι η $T(i,n_i)$ δε μπορεί να κολλήσει στον W_{AO} που εκτελείται για τη x_i εάν έχει οριστεί μια εκ των $Cc(i,n_i)$ και $Ca(i,n_i)$. Έτσι καμία από τις executing PerformTrans της $T(i,n_i)$ δε μπορεί να εκτελέσει επιτυχώς την εντολή CAS της γραμμής 124. Από το Πόρισμα 6.4 προκύπτει ότι στη $Caof(i,n_i)$ το $Trec(i,n_i)$.status μπορεί να έχει τιμή ACTIVE ή ABORTED. Εάν έχει τιμή ABORTED σημαίνει ότι έχει οριστεί η $Ca(i,n_i)$, ενώ εάν έχει τιμή ACTIVE από το Πόρισμα 6.17 προκύπτει ότι στη συνέχεια θα οριστεί η $Cc(i,n_i)$. Έτσι, με βάση τα παραπάνω για να μπορεί να η $T(i,n_i)$ να εκτελεί επ' άπειρο τον W_{AO} για τη x_i, πρέπει να μην έχει οριστεί η $Caof(i,n_i)$. Από το Πόρισμα 6.10 προκύπτει ότι πριν την $Caof(i,n_i)$ οι executing PerformTrans της $T(b,n_b)$ που θα εκτελέσουν τον W_{AO} για τη x_i θα αποχωρήσουν από τον W_{AO} είτε αποκτώντας την προσωρινή ιδιοκτησία της x_i είτε γράφοντας την τιμή ABORTED στο $Trec(i,n_i)$.status, δηλαδή ορίζοντας την $Ca(i,n_i)$. Επίσης από τον κώδικα (γραμμή 122) προκύπτει ότι η $T(i,n_i)$ δε μπορεί να κολλήσει στον W_{AO} που εκτελείται για τη x_i εάν έχει αποκτηθεί η προσωρινή ιδιοκτησία της x_i. Έτσι για να μπορεί η $T(i,n_i)$ να εκτελεί επ' άπειρο τον W_{AO} για τη x_i πρέπει καμία από τις executing PerformTrans της να μην αποκτήσει την προσωρινή ιδιοκτησία της x_i.

Η $T(i,n_i)$ προσπαθεί να αποκτήσει την ιδιοκτησία της x_i, με την εκτέλεση της εντολής CAS μιας εκ των γραμμών 130 και 132, έστω CS_i η εντολή αυτή. Από τον κώδικα προκύπτει ότι σε κάποια καθολική κατάσταση πριν την εκτέλεση της CS_i η $T(i,n_i)$ έχει βεβαιωθεί ότι η ιδιοκτησία της x_i είναι ελεύθερη. Έτσι, για να αποτύχει η εκτέλεση της CS_i πρέπει κάποια άλλη δοσοληψία $T(j,n_j)$ να απέκτησε την ιδιοκτησία

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

της x_i. Στη συνέχεια, η T(j,n_j) δεν καταργεί τη ιδιοκτησία της x_i, διότι εάν αυτό συνέβαινε θα σήμαινε ότι είχε οριστεί είτε η Cc(j,n_j) είτε η Ca(j,n_j) και επομένως στη συνέχεια η T(j,n_j) θα ολοκληρωνόταν, κάτι το οποίο αντιτίθεται στην υπόθεση ότι καμία δοσοληψία δεν τερματίζει. Επομένως μόλις η T(i,n_i) επανεκτελέσει τον W_{AO} για τη x_i, θα βοηθήσει την T(j,n_j) να ολοκληρωθεί, εκτελώντας τη συνάρτηση TryHelpTransaction μιας εκ των γραμμών 52 ή 127 για την T(j,n_j). Από το Πόρισμα 6.22, προκύπτει ότι η T(i,n_i) δε μπορεί να κολλήσει κατά την εκτέλεση της συνάρτησης αυτής και επομένως η συνάρτηση αυτή επιστρέφει. Έτσι, από το Πόρισα 6.7 προκύπτει ότι έχει οριστεί μία εκ των Cc(j,n_j) και Ca(j,n_j), και άρα η T(j,n_j) έχει ολοκληρωθεί, κάτι το οποίο είναι Άτοπο. Επομένως το Θεώρημα ισχύει. ■

6.3. Πολυπλοκότητα αλγορίθμου NBSTM

Στην παρούσα Ενότητα συζητείται η πολυπλοκότητα του αλγορίθμου NBSTM. Ο NBSTM απαιτεί για την αναπαράσταση κάθε t-μεταβλητής έναν καταχωρητή CAS πεπερασμένου μεγέθους και 3 καταχωρητές ανάγνωσης εγγραφής πεπερασμένου μεγέθους. Για την περιγραφή της δομής κάθε δοσοληψίας, απαιτεί έναν καταχωρητή CAS πεπερασμένου μεγέθους, 3 καταχωρητές ανάγνωσης εγγραφής για την περιγραφή κάθε στοιχείου της readList και 2 καταχωρητές ανάγνωσης εγγραφής για την περιγραφή κάθε στοιχείου της writeList.

Κάθε επιτυχημένη δοσοληψία στον NBSTM που προσπελάσει R t-μεταβλητές για ανάγνωση και W t-μεταβλητές για ενημέρωση ελλείψει ανταγωνισμού, απαιτεί τουλάχιστον R+W εντολές CAS για την απόκτηση των προσωρινών ιδιοκτησιών των t-μεταβλητών, $O(R^2)$ κόστος για την εκτέλεση του μηχανισμού ελέγχου συνέπειας, O(W*logW) κόστος για την ταξινόμηση των στοιχείων της writeList, O(R*logR) κόστος για την ταξινόμηση των στοιχείων της readList και μία εντολή CAS για την ενημέρωση των δεδομένων των t-μεταβλητών και την κατάργηση των προσωρινών ιδιοκτησιών των t-μεταβλητών.

ΚΕΦΑΛΑΙΟ 7. ΕΠΙΣΚΟΠΗΣΗ ΥΠΑΡΧΟΝΤΩΝ ΑΛΓΟΡΙΘΜΩΝ STM

7.1 Μη-εμποδιστικοί Αλγόριθμοι

7.2 Εμποδιστικοί Αλγόριθμοι

7.3 Συνοπτικοί Παρουσίαση Αλγορίθμων

Στο παρών κεφάλαιο θα γίνει μία επισκόπηση των υπαρχόντων αλγορίθμων STM. Η ερευνητική δραστηριότητα στους συγκεκριμένους αλγορίθμους ξεκίνησε το 1995, όταν και εισήχθη η έννοια της Software Transactional Memory από τους N. Shavit και D. Touitou στην ομώνυμη εργασία τους [19]. Στην συγκεκριμένη εργασία παρουσιάστηκε επίσης ένας μη-εμποδιστικός αλγόριθμος για στατικές δοσοληψίες. Έπειτα, μετά από αρκετά μεγάλο χρονικό διάστημα, το 2003 δημοσιεύθηκε ο δεύτερος αλγόριθμος STM [12], που ονομάζεται DSTM, ο οποίος ήταν επίσης μη-εμποδιστικός αλγόριθμος όμως υποστήριζε δυναμικές δοσοληψίες. Έκτοτε, η ερευνητική δραστηριότητα γύρω από τους αλγορίθμους STM είναι έντονη και έχουν δημοσιευθεί αρκετές εργασίες που προτείνουν νέους τέτοιους αλγορίθμους.

Στο παρόν κεφάλαιο θα περιγραφούν οι κυριότεροι αλγόριθμοι STM. Η παρουσίαση κάθε αλγορίθμου στην παρούσα εργασία πραγματοποιείται βάσει του αντικειμένου STM που παρουσιάστηκε στην Ενότητα 0, κάτι το οποίο αποδεικνύει ότι το προτεινόμενο μοντέλο STM είναι αρκετά γενικό ώστε να μπορεί να περιγράψει τους περισσότερους υπάρχοντες αλγορίθμους STM. Έτσι, η παρουσίαση των αλγορίθμων στο παρών Κεφάλαιο διαφέρει από τον τρόπο παρουσίασής τους από τους ίδιους τους συγγραφείς τους, στις αντίστοιχες εργασίες τους. Η παρουσίασή των αλγορίθμων θα γίνει χωρίζοντάς τους σε δύο κατηγορίες ανάλογα με το εάν εξασφαλίζουν κάποια

εμποδιστική ή μη-εμποδιστική ιδιότητα τερματισμού. Επίσης κατά την παρουσίασή τους θα εστιάσουμε σε συγκεκριμένες σχεδιαστικές επιλογές που ο καθένας ακολουθεί. Στις Ενότητες 7.1 και 7.2 θα παρουσιαστούν οι κυριότεροι αλγόριθμοι STM. Στην Ενότητα 7.3 θα παρουσιαστεί ένας συνοπτικός πίνακας με τις ιδιότητες και τα χαρακτηριστικά κάθε αλγορίθμου.

7.1. Μη-εμποδιστικοί Αλγόριθμοι

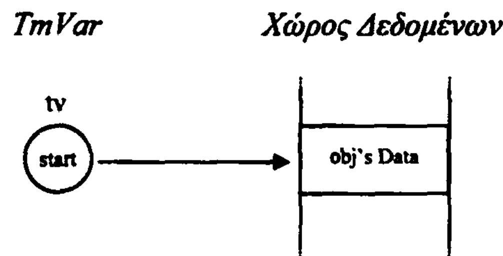
Στην παρούσα ενότητα θα παρουσιαστούν οι σημαντικότεροι υπάρχοντες αλγόριθμοι STM που εγγυώνται κάποια μη-εμποδιστική ιδιότητα τερματισμού. Σημειώνεται ότι έχουν ήδη παρουσιαστεί δύο τέτοιοι αλγόριθμοι STM στα Κεφάλαια 4 και 5, οι αλγόριθμοι SSTM και DSTM.

7.1.1. FSTM-OSTM

Το 2004 ο K.Fraser στην διδακτορική του διατριβή [6] παρουσίασε έναν νέο αλγόριθμο STM, ο οποίος υλοποιεί δυναμική μνήμη STM και ονομάζεται *FSTM*, από το όνομα του συγγραφέα του. Ο αλγόριθμος FSTM υποστηρίζει δυναμικές δοσοληψίες και βελτιώνει τον αλγόριθμο DSTM ικανοποιώντας την ιδιότητα τερματισμού της ελευθερίας κλειδωμάτων, η οποία είναι ισχυρότερη από την αντίστοιχη ελευθερία ανταγωνισμού του DSTM. Για την επίτευξη της ιδιότητας αυτής, ο FSTM χρησιμοποιεί ένα *μηχανισμό επαναλαμβανόμενης βοήθειας* που θα περιγραφεί στη συνέχεια. Ακόμη ο FSTM έχει σχεδιαστεί να λειτουργεί σε κλειστό μοντέλο μνήμης, όπως και ο DSTM.

Ο αλγόριθμος FSTM υλοποιεί δυναμική μνήμη STM, επομένως ο χρήστης είναι υπεύθυνος να χρησιμοποιεί τη λειτουργία CreateNewTmVar του αντικειμένου STM, κάθε φορά που θέλει να δημιουργήσει μια νέα t-μεταβλητή. Για την επίτευξη της ιδιότητας της σειριοποιησιμότητας, ο FSTM χρησιμοποιεί τη μέθοδο απόκτησης προσωρινών ιδιοκτησιών επί των t-μεταβλητών. Στον FSTM κάθε t-μεταβλητή είναι ένας καταχωρητής CAS και περιέχεται σε μια δομή *TmVar*. Η t-μεταβλητή tv είναι ένας δείκτης *start* που δείχνει στα δεδομένα της tv. Η μορφή μιας t-μεταβλητή tv που

περιγράφει τα δεδομένα ενός αντικειμένου obj στον FSTM παρουσιάζεται στο Σχήμα 7.1.



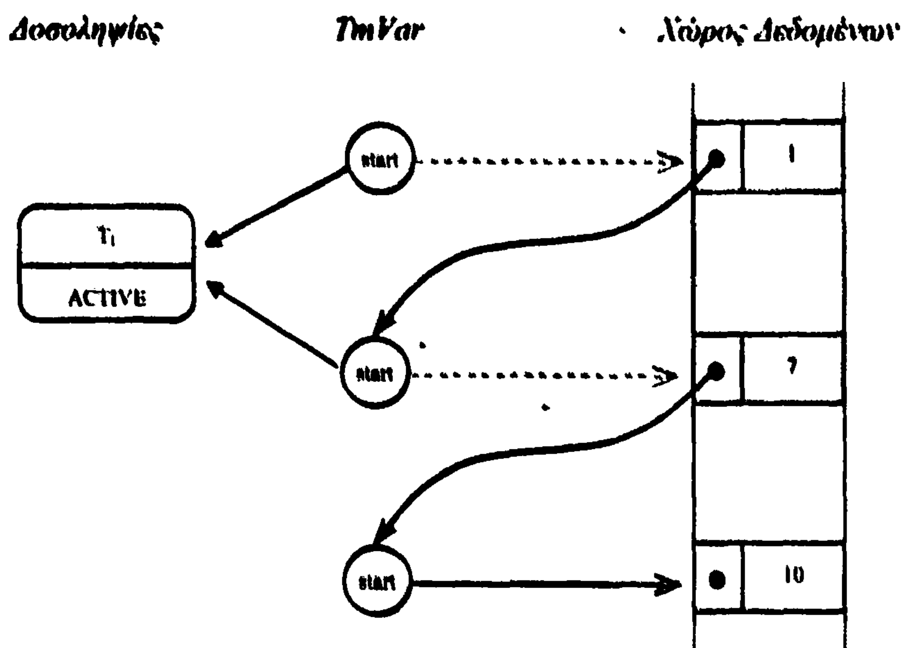
Σχήμα 7.1: Η μορφή ενός διαμοιραζόμενου αντικειμένου obj που περιγράφεται μέσω μιας t-μεταβλητής tv, στον FSTM.

Ο FSTM χρησιμοποιεί τον δείκτη start της TmVar μιας t-μεταβλητής tv για την περιγραφή της δοσοληψίας που κατέχει την προσωρινή ιδιοκτησία της tv. Αυτό σημαίνει ότι ο FSTM χρησιμοποιεί τον ίδιο δείκτη start της tv για την περιγραφή των δεδομένων της tv και της δοσοληψίας που κατέχει την προσωρινή ιδιοκτησία της tv. Για να το επιτύχει αυτό, χρησιμοποιεί ένα bit του δείκτη αυτού (το χαμηλότερο) που δηλώνει εάν ο δείκτης δείχνει σε δεδομένα ή σε κάποια δοσοληψία. Επομένως, ο FSTM χρησιμοποιεί ανα t-μεταβλητή ανάθεση προσωρινών ιδιοκτησιών. Για την απόκτηση της προσωρινής ιδιοκτησίας μιας t-μεταβλητής tv από μια δοσοληψία $T(i,n_i)$, αρκεί η δοσοληψία να καταφέρει να γράψει στον start της tv την τιμή ενός δείκτη προς την δομή της $Trec(i,n_i)$, μέσω της κατάλληλης εντολής CAS και ενόσω καμία δοσοληψία δεν κατέχει την αντίστοιχη προσωρινή ιδιοκτησία. Επομένως ο FSTM χρησιμοποιεί επώνυμη πληροφορία προσωρινής ιδιοκτησίας Για την κατάργηση της προσωρινής ιδιοκτησίας της tv αρκεί η $T(i,n_i)$ να γράψει στον δείκτη start της tv έναν δείκτη προς τα δεδομένα της tv.

Η μορφή του συστήματος FSTM παρουσιάζεται σχηματικά στο Σχήμα 7.2, όπου παρουσιάζεται μια ταξινομημένη κατά αύξουσα σειρά συνδεδεμένη λίστα που περιέχει τα στοιχεία 1,7,10. Όπως απαιτεί το αντικείμενο STM, ο FSTM δεν επιτρέπει να υπάρχουν δείκτες απευθείας μεταξύ των δεδομένων, αλλά ένα δεδομένο d_1 μπορεί να περιέχει δείκτη p σε ένα δεδομένο d_2 μόνο εάν ο p δείχνει στην αντίστοιχη t-μεταβλητή που περιγράφει το d_2. Επίσης, χάριν ευκολίας παρουσίασης του συγκεκριμένου παραδείγματος, υποθέτουμε ότι ο χρήστης διατηρεί ως δεδομένα των

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

t-μεταβλητών τους κόμβους της λίστας και όχι δείκτες προς τους κόμβους αυτούς (ενώ όπως έχει αναφερθεί διατηρεί τους δείκτες). Ακόμη, ο FSTM ορίζει ότι κάθε δοσοληψία διατηρεί στη δομή της έναν καταχωρητή CAS *status*, ο οποίος έχει τιμή ACTIVE όσο η δοσοληψία είναι εκκρεμής και λαμβάνει τιμή COMMITED ή ABORTED όταν η δοσοληψία ολοκληρωθεί επιτυχώς ή μη-επιτυχώς, αντίστοιχα. Στο συγκεκριμένο παράδειγμα οι προσωρινές ιδιοκτησίες των t-μεταβλητών που περιγράφουν τα στοιχεία 1 και 7 της λίστας κατέχονται από την εκκρεμή δοσοληψία T_1 και η προσωρινή ιδιοκτησία της t-μεταβλητής που περιγράφει το στοιχείο 10 δεν έχει αποκτηθεί από καμία δοσοληψία αυτή τη χρονική στιγμή. Σημειώνεται ότι με τα διακεκομμένα και γκρίζα βελάκια παρουσιάζονται τα δεδομένα τα οποία περιγράφουν οι δείκτες start όταν καμία δοσοληψία δεν κατέχει τις προσωρινές ιδιοκτησίες των αντίστοιχων t-μεταβλητών.



Σχήμα 7.2: Η μορφή μιας ταξινομημένης κατ' αύξουσα διάταξη συνδεδεμένης λίστας στον FSTM.

Ο αλγόριθμος FSTM χρησιμοποιεί τη μέθοδο της εκ των υστέρων απόκτησης προσωρινών ιδιοκτησιών, όπως θα περιγραφεί στη συνέχεια, και έτσι χρησιμοποιεί την ετεροχρονισμένη μέθοδο ενημέρωσης των t-μεταβλητών, το οποίο σημαίνει ότι διατηρεί τα νέα δεδομένα των t-μεταβλητών σε μια δομή επανεκτέλεσης. Σημειώνεται ότι για κάθε t-μεταβλητή tv που ενημερώνεται από κάποια δοσοληψία

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

$T(i,n_i)$ διατηρούνται μόνο τα τελευταία δεδομένα της tv στη δομή επανεκτέλεσης της $T(i,n_i)$ και όχι πιθανώς κάποια ενδιάμεσα δεδομένα, εάν η $T(i,n_i)$ ενημερώνει πολλαπλές φορές τη tv. Για την διατήρηση της δομής επανεκτέλεσης ο FSTM ορίζει ότι κάθε δοσοληψία θα διατηρεί στη δομή της μια λίστα ενημερώσεων writeList, όπου κάθε στοιχείο της λίστας θα περιέχει δείκτη προς την t-μεταβλητή που περιγράφει και δείκτη προς τα νέα της δεδομένα.

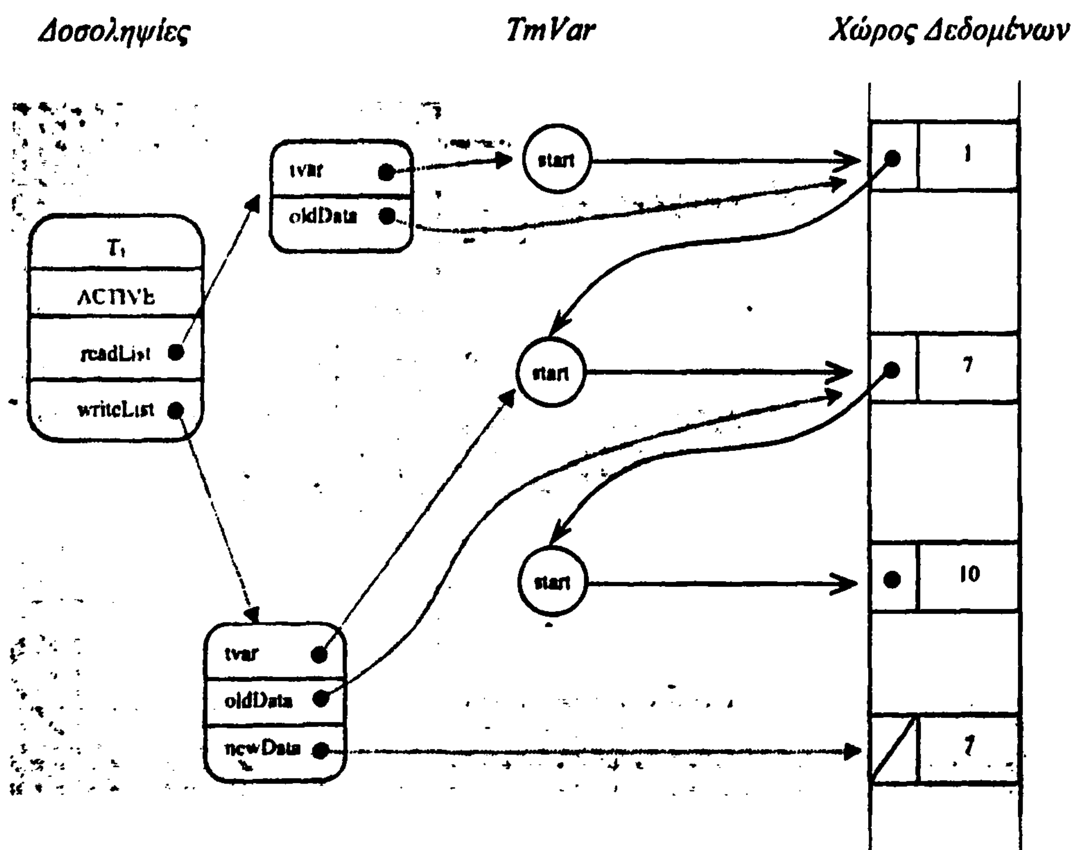
Ο αλγόριθμος FSTM, όπως και ο DSTM, χρησιμοποιεί μη ορατές αναγνώσεις t-μεταβλητών. Επομένως, για να μπορεί να ανιχνεύει συγκρούσεις R-W πρέπει να υλοποιεί έναν μηχανισμό ελέγχου συνέπειας των δεδομένων των t-μεταβλητών που αναγνώσθηκαν με λειτουργία καθολικής ανάγνωσης. Ο μηχανισμός αυτός υλοποιείται από τον FSTM με αντίστοιχο τρόπο όπως και στον DSTM, εκμεταλλευόμενος το κλειστό μοντέλο μνήμης για το οποίο έχει σχεδιαστεί, ώστε να ορίσει με εύκολο τρόπο τις εκδόσεις των δεδομένων.

Ο αλγόριθμος FSTM μπορεί να περιγραφεί με βάση τις λειτουργίες που παρέχει το επεκτεταμένο αντικείμενο STM. Ο FSTM απαιτεί από τον χρήστη να εκκινήσει μία δοσοληψία εκτελώντας τη λειτουργία BeginTransaction, χωρίς να περάσει κανένα όρισμα. Στη συνέχεια ο χρήστης είναι υπεύθυνος να προσπελάσει μέσω των λειτουργιών ReadTmVar και AccessForUpdateTmVar τα δεδομένα των t-μεταβλητών που επιθυμεί να αναγνώσει και να ενημερώσει, αντίστοιχα. Η λειτουργία ReadTmVar εντοπίζει τα τρέχοντα δεδομένα μιας t-μεταβλητής tv, καταγράφει την έκδοσή τους στη readList της δοσοληψίας (ώστε να είναι δυνατή η εκτέλεση του μηχανισμού βοήθειας για την tv) και επιστρέφει τα δεδομένα της tv στο χρήστη. Η λειτουργία AccessForUpdateTmVar εντοπίζει τα τρέχοντα δεδομένα μιας t-μεταβλητής tv, δημιουργεί ένα αντίγραφο των δεδομένων αυτών και επιστρέφει στο χρήστη έναν δείκτη *pd* προς το αντίγραφο αυτό, μέσω του οποίου ο χρήστης μπορεί να διαβάσει και να ενημερώσει τα δεδομένα της tv.

Όπως αναφέρθηκε, μια δοσοληψία $T(i,n_i)$ που εκτελεί την AccessForUpdateTmVar για τη tv πρέπει να καταγράψει τα νέα δεδομένα της tv στην λίστα ενημερώσεών της (writeList). Για να το επιτύχει αυτό αρκεί να καταγράψει τον δείκτη pd στην λίστα ενημερώσεών της, διότι μέσω αυτού περιγράφεται το αντίγραφο των δεδομένων της

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

tv στο οποίο ο χρήστης εφαρμόζει τις ενημερώσεις του. Επίσης ο χρήστης μέσω του pd μπορεί να διαβάσει τα δεδομένα της tv. Έτσι, για να μπορούν να εντοπιστούν συγκρούσεις R-W που αφορούν τα δεδομένα αυτά πρέπει πριν την επιστροφή τους στο χρήστη να καταγραφεί η έκδοσή τους, ώστε να είναι δυνατή η εκτέλεση του μηχανισμού ελέγχου συνέπειας για τα δεδομένα αυτά. Επομένως σε κάθε εγγραφή της λίστας ενημέρωσης writeList μιας δοσοληψίας, εκτός από την t-μεταβλητή και τα νέα της δεδομένα, διατηρείται και η έκδοση των παλιών της δεδομένων.



Σχήμα 7.3: Η μορφή μιας ταξινομημένης κατ' αύξουσα διάταξη συνδεδεμένης λίστας στον FSTM.

Για να γίνει περισσότερο κατανοητή η διαδικασία ανάγνωσης και ενημέρωσης μιας t-μεταβλητής παρουσιάζεται το παράδειγμα του Σχήματος 7.3. Το παράδειγμα αυτό είναι όμοιο με το αντίστοιχο παράδειγμα του Σχήματος 7.2, με τη διαφορά ότι η δοσοληψία T_1 έχει προσπελάσει το στοιχείο 1 της λίστας για ανάγνωση και το στοιχείο 7 της λίστας για ενημέρωση, έτσι ώστε να διαγράψει το στοιχείο 10 από τη λίστα. Παρατηρούμε ότι η T_1 διατηρεί μια καταχώρηση στη λίστα αναγνώσεών της

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

για την t-μεταβλητή και τα δεδομένα του στοιχείου 1 που διάβασε. Επίσης η T_1 διατηρεί μια καταχώρηση στη λίστα ενημερώσεων της με την t-μεταβλητή, τα παλιά και τα νέα δεδομένα του στοιχείου 7 που επιθυμεί να ενημερώσει. Αξίζει να σημειωθεί ότι το παράδειγμα που περιγράφεται στο Σχήμα 7.3 περιγράφει την κατάσταση του συστήματος FSTM πριν την απόκτηση των προσωρινών ιδιοκτησιών που επιθυμεί η T_1 (η οποία πραγματοποιείται κατά την εκτέλεση της λειτουργίας CommitTransaction, όπως περιγράφεται στη συνέχεια), για το λόγο αυτό ο δείκτης start της t-μεταβλητής του στοιχείου 1 δείχνει στα δεδομένα του στοιχείου 1 και όχι στη δομή της δοσοληψίας T_1.

Μόλις ο χρήστης επιθυμεί να ολοκληρώσει τη δοσοληψία, πρέπει να καλέσει τη λειτουργία CommitTransaction ή την AbortTransaction για να γίνει προσπάθεια ολοκλήρωσης της δοσοληψίας ως επιτυχημένης ή μη-επιτυχημένης, αντίστοιχα. Εάν κληθεί η λειτουργία AbortTransaction από τον χρήστη, τότε απλά αποδεσμεύεται ο χώρος που δεσμεύθηκε για την αποθήκευση των δεδομένων της $Trec(i,n_i)$. Έστω ότι εκτελείται η λειτουργία CommitTransaction από τον χρήστη. Κατά την εκτέλεση της λειτουργίας CommitTransaction ο FSTM προσπαθεί να ολοκληρώσει την $T(i,n_i)$ ως επιτυχημένη. Η ολοκλήρωση της $T(i,n_i)$ γίνεται σε δύο στάδια. Κατά το πρώτο στάδιο ο SSTM προσπαθεί να αποκτήσει τις προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που η $T(i,n_i)$ επιθυμεί να ενημερώσει (και περιέχονται στην writeList). Εάν κάποια από τις ιδιοκτησίες αυτές είναι δεσμευμένη από κάποια άλλη δοσοληψία $T(j,n_j)$, σημαίνει ότι η $T(j,n_j)$ θα ενημερώσει τα δεδομένα της αντίστοιχης t-μεταβλητής, εάν ολοκληρωθεί επιτυχώς. Ο FSTM ορίζει ότι η $T(i,n_i)$ θα βοηθήσει την $T(j,n_j)$ να ολοκληρωθεί, όπως περιγράφεται στη συνέχεια.

Σημειώνεται ότι η $T(i,n_i)$ είναι σε θέση να βοηθήσει την $T(j,n_j)$, διότι μέσω της τιμής της προσωρινής ιδιοκτησίας της tv μπορεί να μάθει το αντίστοιχο $Trec(j,n_j)$ της $T(j,n_j)$, το οποίο περιέχει όλες τις απαιτούμενες πληροφορίες για την $T(j,n_j)$. Αυτός είναι και ο *μηχανισμός βοήθειας* που προσδίδει την ιδιότητα τερματισμού ελεύθερη κλειδωμάτων στον αλγόριθμο FSTM. Σημειώνεται ότι η $T(i,n_i)$ βοηθά επαναληπτικά όλες τις δοσοληψίες που πιθανόν χρειάζονται βοήθεια ώστε να ολοκληρωθεί η $T(j,n_j)$. Αυτό σημαίνει ότι εάν η $T(i,n_i)$ βοηθήσει την $T(j,n_j)$ να ολοκληρωθεί και η $T(j,n_j)$ αποφασίζει να βοηθήσει κάποια $T(k,n_k)$ επειδή η $T(k,n_k)$ κατείχε την προσωρινή

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ιδιοκτησία κάποιας t-μεταβλητής που η $T(j,n_j)$ επιθυμούσε, τότε και η $T(i,n_i)$ θα βοηθήσει την $T(k,n_k)$. Για το λόγο αυτό αποδίδεται ο χαρακτηρισμός *επαναληπτικός* στον μηχανισμό βοήθειας του FSTM.

Μετά την ολοκλήρωση του πρώτου σταδίου η $T(i,n_i)$ έχει καταφέρει να αποκτήσει τις προσωρινές ιδιοκτησίες όλων των t-μεταβλητών που ο χρήστης επιθυμεί να ενημερώσει, το οποίο σημαίνει ότι η $T(i,n_i)$ έχει αποκτήσει αποκλειστική πρόσβαση σε αυτές. Στη συνέχεια, κατά το δεύτερο στάδιο της εκτέλεσής εκτελείται ο μηχανισμός ελέγχου συνέπειας των δεδομένων που η $T(i,n_i)$ προσπέλασε για ανάγνωση και ενημέρωση. Εάν τα δεδομένα αυτά είναι συνεπή, η $T(i,n_i)$ χαρακτηρίζεται ως επιτυχημένη (ορίζοντας το status της στην τιμή COMMITED μέσω της κατάλληλης εντολής CAS) και επαναφέρει τους δείκτες start των t-μεταβλητών που κατέχει την προσωρινή ιδιοκτησία ώστε να δείχνουν στα νέα δεδομένα που όρισε ο χρήστης, τα οποία διατηρούνται στην writeList της $T(i,n_i)$. Εάν τα δεδομένα δεν είναι συνεπή, τότε η $T(i,n_i)$ χαρακτηρίζεται ως μη επιτυχημένη (ορίζοντας το status της στην τιμή ABORTED μέσω της κατάλληλης εντολής CAS) και επαναφέρει τους δείκτες start των t-μεταβλητών που κατέχει την προσωρινή ιδιοκτησία ώστε να δείχνουν στα παλιά τους δεδομένα, τα οποία διατηρούνται στην writeList της $T(i,n_i)$. Σημειώνεται ότι ο έλεγχος συνέπειας των δεδομένων των t-μεταβλητών που η $T(i,n_i)$ προσπέλασε για ενημέρωση, πραγματοποιείται κατά το πρώτο στάδιο εκτέλεσης της CommitTransaction, πριν την απόκτηση της αντίστοιχης προσωρινής ιδιοκτησίας.

Επειδή ο FSTM αποκτά την προσωρινή ιδιοκτησία μόνο των t-μεταβλητών που προσπελάσονται για ενημέρωση κατά την εκτέλεση της λειτουργίας CommitTransaction, λέμε ότι χρησιμοποιεί μη ορατές αναγνώσεις t-μεταβλητών και εκ των υστέρων απόκτηση προσωρινών ιδιοκτησιών. Επίσης ο αλγόριθμος FSTM επιτρέπει στον χρήστης να εκτελεί τον μηχανισμό ελέγχου συνέπειας όποτε αυτός κρίνει απαραίτητο. Επομένως ο FSTM χρησιμοποιεί μη αυτόματη μέθοδο εκτέλεσης του μηχανισμού ελέγχου συνέπειας, το οποίο βάσει της συζήτησης που προηγήθηκε στην Ενότητα 3.3 μπορεί να οδηγήσει στην καταστρατήγηση της ιδιότητας τερματισμού της ελευθερίας κλειδωμάτων του FSTM.

Παρατηρούμε ότι για την επιστροφή των δεδομένων μιας t-μεταβλητής tv η λειτουργία ReadTmVar απαιτείται να εντοπίσει τα τρέχοντα δεδομένα της tv. Για την εξακρίβωση των τρεχόντων δεδομένων της tv η ReadTmVar πρέπει να προσπελάσει την αναπαράσταση της tv στο σύστημα FSTM και εάν η προσωρινή ιδιοκτησία της tv δεν κατέχεται από κάποια δοσοληψία θα ακολουθήσει έναν μόνο δείκτη για να εντοπίσει τα τρέχοντα δεδομένα της tv. Επομένως το πλήθος ενδιάμεσων επιπέδων για τον αλγόριθμο FSTM είναι 1, το οποίο είναι μικρότερο από την αντίστοιχη τιμή του DSTM. Υπενθυμίζεται ότι το πλήθος ενδιάμεσων επιπέδων του αλγορίθμου DSTM είναι 3.

Όπως αναφέρθηκε όταν μια δοσοληψία $T(i,n_i)$ προσπαθεί να αποκτήσει την προσωρινή ιδιοκτησία μιας t-μεταβλητής tv, πρέπει να εξασφαλίσει ότι τα δεδομένα της tv που η ίδια έχει διαβάσει (η έκδοση των οποίων περιέχεται στη writeList της $T(i,n_i)$) είναι συνεπή. Για να το επιτύχει αυτό η $T(i,n_i)$ πρέπει να εξακριβώσει τα τρέχοντα δεδομένα της tv, το οποίο ίσως απαιτεί την εκτέλεση του μηχανισμού βοήθειας για κάποια άλλη δοσοληψία $T(j,n_j)$, εάν η $T(j,n_j)$ κατέχει την προσωρινή ιδιοκτησία της tv. Είναι σημαντικό ότι ο FSTM ορίζει ότι η απόκτηση των προσωρινών ιδιοκτησιών κατά το δεύτερο στάδιο εκτέλεσης μιας δοσοληψίας, γίνεται με προκαθορισμένη σειρά βάση της αύξουσας διάταξης των θέσεων των t-μεταβλητών στα οποία αντιστοιχούν. Με τον τρόπο αυτό αποφεύγεται η παρουσίαση κατάστασης καθολικής παρατεταμένης στέρησης που θα είχε ως αποτέλεσμα την καταστρατήγηση της ιδιότητας ελευθερία κλειδωμάτων του FSTM. Εάν δεν υπήρχε ο περιορισμός αυτός, θα μπορούσε να παρουσιαστεί κατάσταση καθολικής παρατεταμένης στέρησης εάν η $T(i,n_i)$ είχε αποκτήσει την προσωρινή ιδιοκτησία της t-μεταβλητής tv_1 και προσπαθούσε να αποκτήσει την προσωρινή ιδιοκτησία της t-μεταβλητής tv_2, και η $T(j,n_j)$ κατείχε την tv_2 και προσπαθούσε να αποκτήσει την tv_1. Έτσι, η $T(i,n_i)$ θα βοηθούσε την $T(j,n_j)$ και η $T(j,n_j)$ την $T(i,n_i)$ και επειδή η μία κατέχει την προσωρινή ιδιοκτησία μιας t-μεταβλητής που η άλλη επιθυμεί, με βάση τα παραπάνω, η μία θα βοηθάει την άλλη επαναλαμβανόμενα, χωρίς καμία να τερματίζει. Επομένως είναι απαραίτητη η απόκτηση των t-μεταβλητών με προκαθορισμένη σειρά και για να είναι αυτό εφικτό απαιτείται η ταξινόμηση των στοιχείων της writeList μιας δοσοληψίας πριν την απόκτηση των αντίστοιχων προσωρινών ιδιοκτησιών κατά το δεύτερο στάδιο εκτέλεσης της δοσοληψίας.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Όμως η ταξινόμηση των στοιχείων της writeList δεν είναι αρκετή για να εμποδίσει την εμφάνιση όλων των πιθανών καταστάσεων καθολικής παρατεταμένης στέρησης, όπως περιγράφεται στη συνέχεια. Είναι πιθανό να προκύψει τέτοια κατάσταση κατά την εκτέλεση του μηχανισμού ελέγχου συνέπειας των δεδομένων που μια δοσοληψία προσπέλασε με καθολική ανάγνωση, ο οποίος εκτελείται μετά την απόκτηση των προσωρινών ιδιοκτησιών που επιθυμεί η δοσοληψία. Έστω ότι κάποια δοσοληψία $T(i,n_i)$ έχει προσπελάσει με καθολική ανάγνωση μια t-μεταβλητή tv_1. Η $T(i,n_i)$ πρέπει να ελέγξει τη συνέπεια των δεδομένων της tv_1 που διάβασε, πριν τον τερματισμό της. Για να το επιτύχει αυτό η $T(i,n_i)$ πρέπει να εξακριβώσει τα τρέχοντα δεδομένα της tv_1, το οποίο ίσως απαιτεί την εκτέλεση του μηχανισμού βοήθειας για κάποια άλλη δοσοληψία $T(j,n_j)$, εάν η $T(j,n_j)$ κατέχει την προσωρινή ιδιοκτησία της tv_1. Έτσι μπορεί να προκύψει το παρακάτω σενάριο. Έστω ότι η $T(j,n_j)$ έχει αποκτήσει την προσωρινή ιδιοκτησία μιας t-μεταβλητής tv_1, έχει διαβάσει με καθολική ανάγνωση τα δεδομένα της tv_2 και προσπαθεί να εξακριβώσει τη συνέπεια των δεδομένων της tv_2. Έστω ότι η $T(i,n_i)$ έχει αποκτήσει την προσωρινή ιδιοκτησία της tv_2 και προσπαθεί να εξακριβώσει τη συνέπεια των δεδομένων της tv_1. Στην περίπτωση αυτή η $T(i,n_i)$ και η $T(j,n_j)$ θα βοηθήσουν η μία την άλλη και μάλιστα επαναληπτικά χωρίς καμία από αυτές να τερματίζει, δημιουργώντας κατάσταση καθολικής παρατεταμένης στέρησης στο σύστημα. Για την αποφυγή της συγκεκριμένης κατάστασης, ο FSTM απαιτεί να ορίζεται μια καθολική διάταξη μεταξύ των δοσοληψιών (π.χ. με βάση την τιμή της διεύθυνσης στην οποία είναι αποθηκευμένη η δομή κάθε δοσοληψίας). Με βάση αυτή την καθολική διάταξη, στο παραπάνω παράδειγμα εάν η δοσοληψία $T(i,n_i)$ προηγούνταν της $T(j,n_j)$, τότε η $T(i,n_i)$ δε θα βοηθούσε την $T(j,n_j)$ αλλά θα την ολοκλήρωνε βίαια ως αποτυχημένη, διότι i) και οι δύο δοσοληψίες εκτελούν τον μηχανισμό ελέγχου συνέπειας των δεδομένων τους (για να μπορεί να το γνωρίζει αυτό η μία δοσοληψία για την άλλη, ο FSTM ορίζει την ότι στη συγκεκριμένη περίπτωση το status της δοσοληψίας θα έχει την τιμή READ_PHASE), ii) η $T(j,n_j)$ κατέχει μια t-μεταβλητή που η $T(i,n_i)$ προσπαθεί να διαβάσει και iii) η $T(i,n_i)$ προηγείται της $T(j,n_j)$ στη διάταξη των δοσοληψιών.

Στο σημείο αυτό συζητείται η πολυπλοκότητα του αλγορίθμου FSTM. Ο FSTM απαιτεί για την αναπαράσταση κάθε t-μεταβλητής έναν καταχωρητή CAS

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

πεπερασμένου μεγέθους. Για την περιγραφή της δομής κάθε δοσοληψίας, απαιτεί έναν καταχωρητή CAS πεπερασμένου μεγέθους, 2 καταχωρητές ανάγνωσης εγγραφής για την περιγραφή κάθε στοιχείου της readList και 3 καταχωρητές ανάγνωσης εγγραφής για κάθε στοιχείο της writeList. Κάθε επιτυχημένη δοσοληψία στον FSTM που προσπελάζει R t-μεταβλητές για ανάγνωση και W t-μεταβλητές για ενημέρωση ελλείψει ανταγωνισμού, απαιτεί τουλάχιστον W εντολές CAS για την απόκτηση των προσωρινών ιδιοκτησιών των t-μεταβλητών και το κόστος για τη δημιουργία των W αντιγράφων, W εντολές CAS για την ενημέρωση των δεδομένων των t-μεταβλητών και την κατάργηση των προσωρινών ιδιοκτησιών των t-μεταβλητών, O((R+W)*R) κόστος για την εκτέλεση του μηχανισμού ελέγχου συνέπειας (στην περίπτωση που εκτελούνταν με την αυξητική μέθοδο) και O(W*logW) κόστος για την ταξινόμηση των στοιχείων της writeList.

7.1.2. ASTM

Το 2005 οι V.Marathe, W.Scherer και M.Scott στη εργασία τους [15] παρουσίασαν έναν νέο αλγόριθμο STM, ο οποίος υλοποιεί δυναμική μνήμη STM και ονομάζεται *Adaptive STM* (*ASTM*). Ο αλγόριθμος ASTM αποτελεί αποτέλεσμα της μελέτης και της σύγκρισης των αλγορίθμων DSTM και FSTM. Συγκεκριμένα οι συγγραφείς εντόπισαν πειραματικά κάποια σενάρια στα οποία άλλοτε ο αλγόριθμος DSTM ήταν καλύτερος από τον FSTM και άλλοτε το αντίθετο, και επιχείρησαν να εξηγήσουν τις διαφορές αυτές βάσει των διαφορετικών σχεδιαστικών επιλογών του κάθε αλγορίθμου. Έτσι οι συγγραφείς προτείνουν τον νέο αλγόριθμο ASTM ο οποίος αποτελεί επέκταση του DSTM και υιοθετεί τις «καλές» σχεδιαστικές επιλογές των αλγορίθμων DSTM και FSTM. Επίσης ο ASTM μπορεί να *προσαρμόζεται* δυναμικά στο είδος των εκτελούμενων λειτουργιών από τις δοσοληψίες (από το χαρακτηριστικό αυτό πήρε το όνομά του), εκμεταλλευόμενος με αυτό τον τρόπο σχεδιαστικές επιλογές των αλγορίθμων DSTM και FSTM που όπως απέδειξαν δουλεύουν καλύτερα σε διαφορετικό είδος εκτελούμενων λειτουργιών. Έτσι ο ASTM καταφέρνει, σύμφωνα με τους συγγραφείς του, να επιτύχει αυξημένη απόδοση σε σχέση με τους DSTM και OSTM. Ο αλγόριθμος ASTM υποστηρίζει δυναμικές δοσοληψίες, ικανοποιεί την ιδιότητα τερματισμού ελευθερία ανταγωνισμού και έχει σχεδιαστεί για κλειστό μοντέλο μνήμης.

Οι συγγραφείς της εργασίας ASTM μελέτησαν τις σχεδιαστικές επιλογές των αλγορίθμων DSTM και FSTM που αφορούν τον χρόνο απόκτησης των κλειδωμάτων, τον τρόπο αναπαράστασης των t-μεταβλητών, την ιδιότητα τερματισμού που ικανοποιούν και το πλήθος ενδιάμεσων επιπέδων κάθε αλγορίθμου. Σημειώνεται ότι για την εύρεση των «καλών» επιλογών για τις σχεδιαστικές αυτές επιλογές των αλγορίθμων DSTM και FSTM οι συγγραφείς, βασίστηκαν επίσης σε μία προηγούμενη εργασία τους [14]. Και στις δύο εργασίες τους, μελετούν τον τρόπο με τον οποίο συμπεριφέρονται οι αλγόριθμοι DSTM και FSTM όταν το μεγαλύτερο μέρος των λειτουργιών που εκτελούν οι δοσοληψίες είναι λειτουργίες ανάγνωσης και όταν είναι λειτουργίες ενημέρωσης.

Στην εργασία τους [14], συμπέραναν ότι ο αλγόριθμος DSTM έχει καλύτερη απόδοση όταν το μεγαλύτερο μέρος των λειτουργιών που εκτελούνται είναι λειτουργίες ενημέρωσης, διότι ο DSTM χρησιμοποιεί εκ των προτέρων απόκτηση κλειδωμάτων και επιτυγχάνει εκ των προτέρων αποτροπή και ανίχνευση συγκρούσεων και έχει μικρότερο κόστος διατήρησης της δομής επαναφοράς σε σχέση με το κόστος διατήρησης της δομής επανεκτέλεσης του FSTM, δεδομένου του ότι απαιτείται και η ταξινόμησή της πριν την ολοκλήρωση μιας δοσοληψίας. Επίσης, στην ίδια εργασία, συμπέραναν ότι ο FSTM έχει καλύτερη απόδοση όταν το μεγαλύτερο μέρος των λειτουργιών είναι λειτουργίες ανάγνωσης, διότι στην περίπτωση αυτή το κόστος ταξινόμησης της δομής επανεκτέλεσης στον FSTM εξαλείφεται, το κόστος διαχείρισης των λιστών ανάγνωσης είναι το ίδιο και ο DSTM πληρώνει υψηλότερο κόστος προσπέλασης των δεδομένων των t-μεταβλητών διότι το πλήθος ενδιάμεσων επιπέδων του έχει τιμή τρία, έναντι του ενός του FSTM.

Οι συγγραφείς του ASTM ισχυρίζονται ότι στις πειραματικές μελέτες που έκαναν δεν κατάφεραν να εντοπίσουν κάποιο πειραματικό σενάριο στο οποίο ο FSTM να ήταν πολύ καλύτερος από τον DSTM και για το λόγο αυτό επέλεξαν να σχεδιάσουν τον αλγόριθμο ASTM έτσι ώστε να υποστηρίζει την ιδιότητα της ελευθερίας ανταγωνισμού, που ικανοποιεί και ο DSTM. Με τον τρόπο αυτό αποφεύγουν το κόστος εκτέλεσης της ταξινόμησης της δομής επανεκτέλεσης που χρησιμοποιεί ο FSTM. Επίσης, οι συγγραφείς του ASTM επέλεξαν να χρησιμοποιήσουν τον τρόπο

αναπαράστασης των t-μεταβλητών που χρησιμοποιεί ο DSTM. Η επιλογή τους αυτή στηρίχθηκε στο ότι η κατάργηση των προσωρινών ιδιοκτησιών των t-μεταβλητών που κατέχει μια δοσοληψία και η ενημέρωση τους (ανάλογα και με το εάν τερματίζει ως επιτυχημένη ή ως αποτυχημένη) μπορεί να γίνει πολύ πιο αποτελεσματικά με τον τρόπο αναπαράστασης των t-μεταβλητών που χρησιμοποιεί ο DSTM. Υπενθυμίζεται ότι η συγκεκριμένη λειτουργικότητα επιτυγχάνεται με μία μόνο εντολή CAS στον DSTM ενώ απαιτεί W εντολές CAS στον FSTM, εάν W είναι το πλήθος των t-μεταβλητών που μια δοσοληψία προσπέλασε για ενημέρωση. Επίσης, ο ASTM ορίζει ότι θα χρησιμοποιούνται μη-ορατές αναγνώσεις t-μεταβλητών, όπως και ο DSTM.

Όμως ο τρόπος αναπαράστασης των t-μεταβλητών του DSTM έχει το μειονέκτημα του υψηλού πλήθους ενδιάμεσων επιπέδων το οποίο επιφέρει μεγάλο κόστος στις λειτουργίες ανάγνωσης, όπως απέδειξαν πειραματικά οι συγγραφείς στην εργασία τους [14]. Για τη μείωση του κόστους αυτού, οι συγγραφείς του ASTM ορίζουν ότι ο δείκτης start της TmVar μιας t-μεταβλητής tv δε θα μπορεί να δείχνει μόνο σε κάποιον Locator της tv, αλλά θα μπορεί να δείχνει απευθείας και στα δεδομένα της tv, όπως ακριβώς και ο αντίστοιχος δείκτης του FSTM μπορεί να δείχνει είτε στα δεδομένα της tv είτε στη δομή της δοσοληψίας που κατέχει την προσωρινή ιδιοκτησία της tv. Για να το επιτύχει αυτό, χρησιμοποιεί ένα bit του δείκτη start (το χαμηλότερο) που δηλώνει εάν ο δείκτης δείχνει σε δεδομένα ή σε κάποιον Locator. Με τον τρόπο αυτό αρκετές από τις λειτουργίες ανάγνωσης επιβαρύνονται με το κόστος ενός μόνο ενδιάμεσου επιπέδου.

Ο ASTM ορίζει ότι ο δείκτης start μιας t-μεταβλητής tv θα δείχνει αρχικά στα αντίστοιχα δεδομένα. Εάν κάποια δοσοληψία επιθυμεί να αποκτήσει την προσωρινή ιδιοκτησία της tv, τότε πρέπει να αλλάξει το δείκτη start της tv ώστε να δείχνει σε κάποιον Locator, με αντίστοιχο τρόπο όπως και στον DSTM, με την εκτέλεση της κατάλληλης εντολής CAS. Λέμε μια t-μεταβλητή είναι σε *κατάσταση κατοχής* εάν ο δείκτης της start δείχνει σε κάποιον locator και σε *κατάσταση μη-κατοχής* σε αντίθετη περίπτωση. Είναι σημαντικό ότι για να επιτευχθεί η βελτίωση του πλήθους ενδιάμεσων επιπέδων από τρία σε ένα, πρέπει όταν μια δοσοληψία ολοκληρώσει την ενημέρωση της tv (ολοκληρωθεί επιτυχώς ή μη επιτυχώς) να επαναφέρει τον δείκτη start ώστε να δείχνει απευθείας στα τρέχοντα δεδομένα (τα οποία μπορεί να είναι τα

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

παλιά ή τα νέα, αντίστοιχα), με την εκτέλεση της κατάλληλης εντολής CAS. Όμως, η επαναφορά των δεικτών αυτών από τη δοσοληψία που ενημέρωσε τη tv μπορεί να προκαλέσει μείωση στην απόδοση του συστήματος, εάν στη συνέχεια έρθει μια άλλη δοσοληψία που θέλει επίσης να ενημερώσει τη tv και άρα αποκτήσει την προσωρινή ιδιοκτησία της. Γίνεται αντιληπτό ότι στην περίπτωση αυτή η επαναφορά του δείκτη της tv στα τρέχοντα δεδομένα και η πληρωμή του κόστους μιας εντολής CAS, έγινε άσκοπα. Η κατάσταση αυτή είναι πολύ πιθανό να προκύψει όταν το μεγαλύτερο μέρος των λειτουργιών που εκτελούν οι δοσοληψίες είναι λειτουργίες ενημέρωσης. Για την αποφυγή του προβλήματος αυτού, αλλά και τον διαμοιρασμό του κόστους επαναφοράς των δεικτών start των t-μεταβλητών που μια δοσοληψία $T(i,n_i)$ ενημέρωσε, ο ASTM ορίζει ότι την επαναφορά αυτή θα την εκτελέσει η πρώτη δοσοληψία που θα αναγνώσει τη tv μετά την $T(i,n_i)$. Με τον τρόπο αυτό οι t-μεταβλητές τείνουν να παραμένουν σε κατάσταση μη-κατοχής εάν το μεγαλύτερο μέρος των λειτουργιών είναι λειτουργίες ανάγνωσης και σε κατάσταση κατοχής εάν το μεγαλύτερο μέρος των λειτουργιών είναι λειτουργίες ενημέρωσης.

Η τελευταία σχεδιαστική επιλογή του ASTM αφορά τον χρόνο απόκτησης των κλειδωμάτων. Υπενθυμίζεται ότι τα δύο είδη χρονικής απόκτησης κλειδωμάτων είναι η εκ των προτέρων και η εκ των υστέρων. Ανάλογα με το είδος χρονικής απόκτησης κλειδωμάτων χαρακτηρίζεται και η μέθοδος αποτροπής και ανίχνευσης των συγκρούσεων που χρησιμοποιεί ο αλγόριθμος ως εκ των προτέρων και εκ των υστέρων αντίστοιχα. Όπως συζητήθηκε στην 3.3 παρουσιάζεται ένας συμβιβασμός (tradeoff) μεταξύ της εκ των προτέρων και της εκ των υστέρων αποτροπής ή ανίχνευσης συγκρούσεων, με την εκ' των προτέρων να επιτρέπει τη γρήγορη ανίχνευση των καταδικασμένων δοσοληψιών και την εκ των υστέρων να μειώνει την πιθανότητα εμφάνισης καταστάσεων άσκοπης πληρωμής και να αυξάνει την πιθανότητα οι εν δυνάμει καταδικασμένες δοσοληψίες να ολοκληρωθούν ως επιτυχημένες. Εξαιτίας αυτού του συμβιβασμού οι συγγραφείς του ASTM αποφάσισαν να σχεδιάσουν δύο εκδόσεις του ASTM, η μία θα χρησιμοποιεί εκ των προτέρων απόκτηση κλειδωμάτων, όπως και ο DSTM, και η άλλη εκ των υστέρων απόκτηση κλειδωμάτων. Ο αλγόριθμος ASTM αρχικά λειτουργεί με την εκ των προτέρων έκδοσή του και επιλέγει να χρησιμοποιήσει τη μία ή την άλλη έκδοση ανάλογα με το είδος των λειτουργιών που εξυπηρετεί. Ο ASTM ορίζει ένα

συγκεκριμένο όριο *w* για το ποσοστό των λειτουργιών ενημέρωσης. Όσο το ποσοστό των λειτουργιών ενημέρωσης ξεπερνά το *w* χρησιμοποιείται η έκδοση με την εκ των προτέρων απόκτηση κλειδωμάτων, ενώ εάν το ποσοστό αυτό πέσει κάτω από το *w* χρησιμοποιείται η έκδοση με την εκ των υστέρων απόκτηση κλειδωμάτων. Σημειώνεται ότι για να το πετύχει αυτό ο ASTM, διατηρεί κάποιου είδους ιστορικό. Με τον τρόπο αυτό ο ASTM μπορεί να προσαρμοστεί δυναμικά στο είδος των λειτουργιών που εκτελούν οι δοσοληψίες. Σημειώνεται ότι ο ASTM χρησιμοποιεί το επεκτεταμένο αντικείμενο STM, όπως και ο DSTM, το οποίο σημαίνει ότι κάθε λειτουργία ενημέρωσης μιας t-μεταβλητής tv περιέχει και μια λειτουργία ανάγνωσης των δεδομένων της tv.

Η έκδοση του ASTM που χρησιμοποιεί εκ των υστέρων απόκτηση πρέπει απαραίτητα να χρησιμοποιεί ετεροχρονισμένη ενημέρωση των t-μεταβλητών, επομένως πρέπει να διατηρεί δομή επανεκτέλεσης. Κάθε δοσοληψία διατηρεί τη δομή επανεκτέλεσης σε μια λίστα ενημέρωσης, όπως και στον FSTM, με την διαφορά ότι όταν πρόκειται να αποκτηθούν κατά την εκτέλεση της λειτουργίας CommitTransaction οι προσωρινές ιδιοκτησίες των t-μεταβλητών, αυτές αποκτώνται χωρίς προηγούμενη ταξινόμηση της δομής επανεκτέλεσης (όπως στον FSTM). Σημειώνεται ότι ο ASTM, και στις δύο εκδόσεις του, βασίζεται στον διαχειριστή ανταγωνισμού που χρησιμοποιεί, με λειτουργικότητα ανάλογη του DSTM, ώστε να επιλύονται οι συγκρούσεις που πιθανώς προκύπτουν (σχετικά τις ενέργειες που πρέπει να ακολουθήσει μια δοσοληψία που συγκρούεται με κάποια άλλη).

Η πολυπλοκότητα χώρου του αλγορίθμου ASTM είναι ίδια με την αντίστοιχη πολυπλοκότητα του αλγορίθμου DSTM, με τη διαφορά ότι η έκδοση του ASTM που χρησιμοποιεί την εκ των υστέρων απόκτηση κλειδωμάτων, απαιτεί επιπλέον 2 καταχωρητές ανάγνωσης-εγγραφής για την αποθήκευση κάθε στοιχείου της λίστας ενημερώσεων (δομή επανεκτέλεσης). Η πολυπλοκότητα χρόνου του αλγορίθμου ASTM σε συνθήκες έλλειψης ανταγωνισμού είναι ίδια με την αντίστοιχη πολυπλοκότητα του αλγορίθμου DSTM.

7.2. Εμποδιστικοί Αλγόριθμοι

Στην παρούσα ενότητα θα παρουσιαστούν οι σημαντικότεροι υπάρχοντες εμποδιστικοί αλγόριθμοι STM. Συγκεκριμένα, παρουσιάζονται συνοπτικά δύο εργασίες που προτείνουν τους αλγορίθμους TL [4] και TLII [3]. Όμως στη βιβλιογραφία υπάρχουν αρκετές ακόμα εργασίες που προτείνουν τέτοιους αλγορίθμους, όπως η εργασία [20] που προτείνει τον αλγόριθμο RingSTM, η εργασία [17] και η εργασία [10] που προτείνει μια τεχνική ενίσχυσης (boosting) υπαρχόντων αλγορίθμων. Είναι αξιοσημείωτο ότι οι εργασίες που προτείνουν εμποδιστικούς αλγορίθμους STM είναι αρκετά περισσότερες από αυτές που προτείνουν κάποιο μη-εμποδιστικό αλγόριθμο STM.

7.2.1. TL

Το 2006 οι D.Dice και N.Shavit στην εργασία τους [4] παρουσιάζουν έναν εμποδιστικό αλγόριθμο STM, ο οποίος υλοποιεί δυναμική μνήμη STM και ονομάζεται *Transactional Locking* (*TL*). Ο αλγόριθμος TL υποστηρίζει δυναμικές δοσοληψίες και έχει σχεδιαστεί για κλειστό μοντέλο μνήμης.

Ο αλγόριθμος TL υλοποιεί δυναμική μνήμη STM, επομένως ο χρήστης είναι υπεύθυνος να χρησιμοποιεί τη λειτουργία CreateNewTmVar του αντικειμένου STM, κάθε φορά που θέλει να δημιουργήσει μια νέα t-μεταβλητή. Για την επίτευξη της ιδιότητας της σειριοποιησιμότητας δοσοληψιών, ο TL χρησιμοποιεί τη μέθοδο απόκτησης προσωρινών ιδιοκτησιών επί των t-μεταβλητών. Στον TL κάθε t-μεταβλητή είναι ένας καταχωρητής ανάγνωσης εγγραφής που περιέχει τα δεδομένα που ο χρήστης διατηρεί σε αυτή την t-μεταβλητή (σημειώνεται ότι στην παρούσα εργασία θεωρούμε ότι μια t-μεταβλητή μπορεί να περιέχει ως δεδομένα της μόνο δείκτες) και περιέχεται σε μια δομή που ονομάζεται TmVar. Ο TL μπορεί να υλοποιηθεί χρησιμοποιώντας ανα t-μεταβλητή ανάθεση κλειδωμάτων ή ανα σύνολο ανάθεση κλειδωμάτων.

Κάθε κλείδωμα στον TL περιγράφεται από έναν καταχωρητή CAS *lock* ο οποίος περιγράφει εάν η t-μεταβλητή tv στην οποία έχει αντιστοιχιστεί είναι κλειδωμένη ή όχι και παίρνει δύο τιμές LOCKED και UNLOCKED, αντίστοιχα. Για να αποκτήσει

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

μια δοσοληψία το κλείδωμα της tv, αρκεί να εκτελέσει την κατάλληλη εντολή CAS και να γράψει την τιμή LOCKED στο lock της tv, ενόσω καμία άλλη δοσοληψία δεν κατέχει το συγκεκριμένο κλείδωμα. Για να καταργήσει μια δοσοληψία το κλείδωμα της tv, αρκεί να εκτελέσει την κατάλληλη εντολή CAS και να γράψει την τιμή UNLOCKED στο lock της tv. Παρατηρούμε ότι η πληροφορία του κλειδώματος δεν περιέχει καμία πληροφορία για τη δοσοληψία που κατέχει το κλείδωμα, είναι δηλαδή ανώνυμη, κάτι το οποίο αποτελεί χαρακτηριστικό των εμποδιστικών αλγορίθμων STM. Σημειώνεται ότι στην περίπτωση της ανα t-μεταβλητής ανάθεσης κλειδωμάτων ο lock θα αντιστοιχιζόταν σε μία μόνο t-μεταβλητή, ενώ στην ανα σύνολο θα αντιστοιχιζόταν σε ένα σύνολο t-μεταβλητών. Το σύστημα STM είναι υπεύθυνο για τη διατήρηση αυτής της αντιστοιχίας μεταξύ t-μεταβλητών και κλειδωμάτων, για το λόγο αυτό ο lock δεν περιέχει πληροφορία για τις t-μεταβλητές στις οποίες αντιστοιχεί.

Ο αλγόριθμος TL χρησιμοποιεί μη ορατές αναγνώσεις t-μεταβλητών. Έτσι, για να μπορεί να ανιχνεύει συγκρούσεις R-W πρέπει να υλοποιεί έναν μηχανισμό ελέγχου συνέπειας των δεδομένων των t-μεταβλητών που αναγνώσθηκαν με λειτουργία καθολικής ανάγνωσης. Υπενθυμίζεται ότι ο μηχανισμός ελέγχου συνέπειας απαιτεί την ύπαρξη έκδοσης για κάθε δεδομένο. Επειδή ο lock χρειάζεται μόνο ένα bit για την υλοποίηση της λειτουργικότητάς του, ο TL χρησιμοποιεί τα υπόλοιπα bits του καταχωρητή CAS για την αποθήκευση ενός μετρητή που δηλώνει την *έκδοση* των δεδομένων της t-μεταβλητής (ή του συνόλου των t-μεταβλητών) που ο lock περιγράφει. Κάθε δοσοληψία που ενημερώνει μια t-μεταβλητή tv και ολοκληρώνεται επιτυχώς, απαιτείται να αυξήσει τον μετρητή της tv πριν τον τερματισμό της, δηλώνοντας με τον τρόπο αυτό ότι η έκδοση της tv έχει ενημερωθεί.

Έτσι, απαιτείται κάθε δοσοληψία να διατηρεί την τρέχουσα έκδοση των δεδομένων κάθε t-μεταβλητής που ο χρήστης διαβάζει καθολικά. Για να είναι αυτό δυνατό ο αλγόριθμος TL ορίζει ότι κάθε δοσοληψία θα διατηρεί στη δομή Tree της μια λίστα αναγνώσεων *readList* που περιέχει τις απαραίτητες πληροφορίες για τις t-μεταβλητές που η δοσοληψία έχει προσπελάσει με καθολική ανάγνωση, συγκεκριμένα την έκδοσή τους και τον lock από τον οποίο διαβάστηκε η κάθε έκδοση, ώστε να είναι δυνατός ο έλεγχος συνέπειας των αντίστοιχων δεδομένων. Επίσης, όπως θα

περιγραφεί στη συνέχεια ο TL έχει δύο εκδόσεις, οι οποίες χρησιμοποιούν ετεροχρονισμένη και άμεση ενημέρωση των δεδομένων των t-μεταβλητών που μια δοσοληψία προσπελάζει για ενημέρωση, αντίστοιχα. Για το λόγο αυτό απαιτείται κάθε δοσοληψία να διατηρεί τα νέα δεδομένα των t-μεταβλητών σε μια *δομή επανεκτέλεσης* ή τα παλιά δεδομένα σε μια *δομή επαναφοράς*, αντίστοιχα, το οποίο στον TL διατηρείται τοπικά στην εκάστοτε δοσοληψία σε μια λίστα ενημέρωσης *writeList* που περιέχεται στη δομή της. Σημειώνεται ότι η δομή μιας δοσοληψίας και οι πληροφορίες που περιέχει δεν είναι προσπελάσιμες από άλλες δοσοληψίες, δηλαδή δε βρίσκονται στη διαμοιραζόμενη μνήμη, κάτι το οποίο έρχεται σε αντίθεση με τη δομή των δοσοληψιών όλων των αλγορίθμων που περιγράφηκαν μέχρι τώρα.

Σημειώνεται ότι ο TL έχει σχεδιαστεί από τους συγγραφείς του ώστε να δουλεύει και με τις δύο μεθόδους χρονικής απόκτησης κλειδωμάτων, δηλαδή υπάρχουν δύο εκδόσεις του. Στη συνέχεια περιγράφεται ο τρόπος υλοποίησης του TL, βάσει των λειτουργιών που παρέχει το βασικό αντικείμενο STM. Ο TL απαιτεί από τον χρήστη να εκκινήσει μία δοσοληψία εκτελώντας τη λειτουργία BeginTransaction, χωρίς να περάσει κανένα όρισμα. Στη συνέχεια ο χρήστης είναι υπεύθυνος να προσπελάσει μέσω των λειτουργιών ReadTmVar και WriteTmVar τα δεδομένα των t-μεταβλητών που επιθυμεί να αναγνώσει και να ενημερώσει, αντίστοιχα. Έστω ότι η δοσοληψία $T(i,n_i)$ εκτελεί τη λειτουργία ReadTmVar για μια t-μεταβλητή tv, η οποία αποτελεί καθολική ανάγνωση της tv. Η ReadTmVar προσπελάζει τον lock που αντιστοιχεί στη tv και ελέγχει εάν είναι κλειδωμένος. Εάν ο lock της tv είναι κλειδωμένος τότε η ReadTmVar είτε περιμένει τη tv να ελευθερωθεί, είτε η $T(i,n_i)$ ολοκληρώνεται ως μη επιτυχημένη. Εάν ο lock δεν είναι κλειδωμένος, τότε η ReadTmVar καταγράφει στην λίστα αναγνώσεών της $T(i,n_i)$ την έκδοση των δεδομένων της tv, που περιέχεται στον lock, ώστε να είναι δυνατή η εκτέλεση του μηχανισμού ελέγχου συνέπειας για την tv, και στη συνέχεια διαβάζει τα δεδομένα της tv, τα οποία και επιστρέφει στον χρήστη.

Έστω ότι η δοσοληψία $T(i,n_i)$ εκτελεί τη λειτουργία WriteTmVar για μια t-μεταβλητή tv. Η WriteTmVar εκτελείται ανάλογα με τη μέθοδο απόκτησης κλειδωμάτων που χρησιμοποιείται. Εάν χρησιμοποιείται η εκ των υστέρων απόκτηση κλειδωμάτων, τότε η WriteTmVar απλά καταγράφει τη νέα τιμή της tv, στη λίστα ενημερώσεων της $T(i,n_i)$. Εάν χρησιμοποιείται η εκ των προτέρων απόκτηση κλειδωμάτων, τότε η

WriteTmVar αρχικά προσπελάσει τον lock που αντιστοιχεί στη tv και ελέγχει εάν είναι κλειδωμένος. Εάν ο lock της tv είναι κλειδωμένος τότε η WriteTmVar είτε περιμένει τη tv να ελευθερωθεί, είτε η $T(i,n_i)$ ολοκληρώνεται ως μη επιτυχημένη. Εάν ο lock δεν είναι κλειδωμένος, τότε η WriteTmVar προσπαθεί να αποκτήσει το κλείδωμα της tv εκτελώντας την κατάλληλη εντολή CAS στον lock της tv. Μόλις η WriteTmVar αποκτήσει το κλείδωμα της tv, διατηρεί τα παλιά δεδομένα της tv στη λίστα ενημερώσεών της και ενημερώνει τα δεδομένα της tv, με βάση τα νέα δεδομένα που όρισε ο χρήστης. Επίσης στην περίπτωση της εκ των προτέρων απόκτησης κλειδωμάτων, κατά την απόκτηση του κλειδώματος lock της tv, ο TL αντικαθιστά την έκδοση των δεδομένων που περιγράφει ο lock (σημειώνεται ότι την τρέχουσα έκδοση τη διατηρεί στη λίστα ενημερώσεων της $T(i,n_i)$) με έναν δείκτη προς την καταχώρηση της writeList στην οποία έχει αποθηκεύσει την παλιά τιμή των δεδομένων της tv. Με τον τρόπο αυτό, εάν η $T(i,n_i)$ ολοκληρωθεί μη επιτυχώς τότε ο TL θα μπορεί να αντικαταστήσει τα δεδομένα της tv με τα παλιά της δεδομένα, άμεσα μέσω του lock της tv, χωρίς να διατρέξει όλα τα στοιχεία της writeList της $T(i,n_i)$. Επομένως στην περίπτωση που χρησιμοποιείται η εκ των υστέρων απόκτηση κλειδωμάτων ο TL χρησιμοποιεί τη μέθοδο ετεροχρονισμένης ενημέρωσης των t-μεταβλητών, ενώ όταν χρησιμοποιείται εκ των προτέρων απόκτηση κλειδωμάτων χρησιμοποιεί την άμεση μέθοδο.

Σημειώνεται ότι ο TL ορίζει ότι πριν την ολοκλήρωση μιας λειτουργίας ReadTmVar πρέπει να εκτελεστεί ο μηχανισμός ελέγχου συνέπειας των δεδομένων των t-μεταβλητών που προσπελάστηκαν με καθολική ανάγνωση. Έτσι προκύπτει ότι ο TL χρησιμοποιεί την αυτόματη και αυξητική μέθοδο εκτέλεσης του μηχανισμού ελέγχου συνέπειας. Επίσης, παρατηρούμε ότι για την επιστροφή των δεδομένων μιας t-μεταβλητής tv η λειτουργία ReadTmVar απαιτεί απλά να εντοπίσει την αντιστοιχία μεταξύ t-μεταβλητής και κλειδώματος. Αυτό σημαίνει ότι θα ακολουθηθεί ένας μόνο δείκτης για τον εντοπισμό των δεδομένων της tv. Επομένως το πλήθος ενδιάμεσων επιπέδων για τον αλγόριθμο TL είναι 1.

Σημειώνεται ότι το αντικείμενο STM επιτρέπει στο χρήστη να προσπελάσει την ίδια t-μεταβλητή tv πολλαπλές φορές για ενημέρωση ή ανάγνωση. Για το λόγο αυτό όταν εκτελείται μια λειτουργία ReadTmVar στη tv, ο TL αναζητεί αρχικά μήπως υπάρχει

ήδη καταχώρηση της tv στη writeList της T(i,n_i) και εάν υπάρχει τέτοια καταχώρηση τότε η ReadTmVar επιστρέφει κατευθείαν τα δεδομένα που περιέχονται είτε στη writeList, εάν χρησιμοποιείται η εκ των υστέρων απόκτηση κλειδωμάτων, είτε στην ίδια την tv, εάν χρησιμοποιείται η εκ των προτέρων απόκτηση κλειδωμάτων. Είναι σημαντικό ότι η συγκεκριμένη αναζήτηση στην writeList της T(i,n_i) πραγματοποιείται με τη βοήθεια ενός κατάλληλου bloom filter [2], κάτι το οποίο επιτρέπει στη ReadTmVar εάν η tv περιέχεται στη writeList της T(i,n_i) και μόνο εάν αυτό συμβαίνει να ψάξει να βρει την καταχώρησή της (και μόνο εάν χρησιμοποιείται η εκ των υστέρων απόκτηση κλειδωμάτων). Επίσης το ίδιο bloom filter χρησιμοποιείται και όταν εκτελείται κάποια λειτουργία WriteTmVar στη tv, ώστε να γνωρίζει και πάλι εάν έχει ήδη αποκτηθεί το κλείδωμα της tv (στην εκ των προτέρων απόκτηση κλειδωμάτων) ή εάν απλά υπάρχει ήδη καταχώρηση της tv στη writeList τη T(i,n_i) (στην εκ των υστέρων απόκτηση κλειδωμάτων) και επομένως πρέπει να ενημερωθεί η συγκεκριμένη καταχώρηση με τα νέα δεδομένα που εισάγει ο χρήστης. Γίνεται κατανοητό ότι το κόστος αναζήτησης στη writeList μειώνεται αισθητά με τη χρήση των bloom filters.

Μόλις ο χρήστης θελήσει να ολοκληρώσει τη δοσοληψία, πρέπει να καλέσει τη λειτουργία CommitTransaction ή την AbortTransaction για να γίνει προσπάθεια ολοκλήρωσης της δοσοληψίας ως επιτυχημένης ή μη-επιτυχημένης, αντίστοιχα. Έστω ότι εκτελείται η λειτουργία AbortTransaction για τη δοσοληψία T(i,n_i). Στην περίπτωση που χρησιμοποιείται η εκ των υστέρων απόκτηση κλειδωμάτων, απλά ολοκληρώνεται ως αποτυχημένη η T(i,n_i). Εάν χρησιμοποιείται η εκ των προτέρων απόκτηση κλειδωμάτων, τότε πρέπει η CommitTransaction να ελευθερώσει το κλείδωμα κάθε t-μεταβλητής που η T(i,n_i) έχει αποκτήσει και ταυτόχρονα να επαναφέρει τις παλιές τιμές των δεδομένων τους, κάτι το οποίο μπορεί να το πετύχει με απλές εγγραφές (όχι απαραίτητα με τη χρήση εντολών CAS).

Έστω ότι εκτελείται η λειτουργία CommitTransaction για τη δοσοληψία T(i,n_i). Η CommitTransaction πρέπει αρχικά να αποκτήσει τα κλειδώματα όλων των ιδιοκτησιών που επιθυμεί η T(i,n_i), μόνο εάν χρησιμοποιείται η εκ των υστέρων απόκτηση κλειδωμάτων. Σημειώνεται ότι τα συγκεκριμένα κλειδώματα μπορούν να αποκτηθούν με οποιαδήποτε σειρά, δηλαδή δε χρησιμοποιείται κάποιου είδους

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

καθολική διάταξη που θα απαιτούσε την ταξινόμησή τους, και η παρουσίαση συγκρούσεων επιλύεται με την μέθοδο αναμονής και τελικά ολοκλήρωσης ως αποτυχημένης μιας δοσοληψίας που δε μπορεί να αποκτήσει ένα συγκεκριμένο κλείδωμα για μεγάλο χρονικό διάστημα. Εάν η CommitTransction καταφέρει να αποκτήσει όλα τα κλειδώματα (όταν χρησιμοποιείται η εκ των υστέρων απόκτηση κλειδωμάτων), τότε στη συνέχεια εκτελείται ο έλεγχος συνέπειας των δεδομένων που η $T(i,n_i)$ προσπέλασε με καθολική ανάγνωση. Εάν ο έλεγχος αυτό ολοκληρωθεί επιτυχώς η CommitTransaction, ενημερώνει τα δεδομένα των επιθυμητών t-μεταβλητών (μόνο εάν χρησιμοποιείται η εκ των υστέρων απόκτηση κλειδωμάτων), ενημερώνει την έκδοση τους που περιγράφεται από τα αντίστοιχα κλειδώματα, ελευθερώνει τα κλειδώματα και ολοκληρώνεται επιτυχώς. Σε αντίθετη περίπτωση, η CommitTransaction, επαναφέρει τα παλιά δεδομένα των t-μεταβλητών (μόνο εάν χρησιμοποιείται η εκ των προτέρων απόκτηση κλειδωμάτων), ελευθερώνει τα κλειδώματα και ολοκληρώνεται μη επιτυχώς. Σημειώνεται ότι η εκτέλεση όλων των λειτουργιών που περιγράφηκαν, μετά την εκτέλεση του μηχανισμού ελέγχου μπορούν να υλοποιηθούν με απλές εγγραφές (όχι απαραίτητα με τη χρήση εντολών CAS).

Στο σημείο αυτό συζητείται η πολυπλοκότητα του αλγορίθμου TL. Ο TL απαιτεί για την αναπαράσταση κάθε t-μεταβλητής έναν καταχωρητή ανάγνωσης εγγραφής και έναν καταχωρητή CAS μη-πεπερασμένης χωρητικότητας. Για την περιγραφή της δομής κάθε δοσοληψίας, απαιτεί 2 καταχωρητές ανάγνωσης εγγραφής για την περιγραφή κάθε στοιχείου της readList, ο ένας από τους οποίους είναι μη πεπερασμένης χωρητικότητας και 2 καταχωρητές ανάγνωσης εγγραφής για κάθε στοιχείο της writeList. Κάθε επιτυχημένη δοσοληψία στον TL που προσπελάζει R t-μεταβλητές για ανάγνωση και W t-μεταβλητές για ενημέρωση ελλείψει ανταγωνισμού, απαιτεί W εντολές CAS για την απόκτηση των προσωρινών ιδιοκτησιών των t-μεταβλητών, 2*W εντολές εγγραφής καταχωρητών ανάγνωσης εγγραφής, για την ενημέρωση των δεδομένων των t-μεταβλητών και την κατάργηση των ιδιοκτησιών των t-μεταβλητών, $O(R^2)$ κόστος για την εκτέλεση του μηχανισμού ελέγχου συνέπειας.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

7.2.2. TL II

Το 2006 οι D.Dice, O.Shalev και N.Shavit στην εργασία τους [3] παρουσιάζουν έναν εμποδιστικό αλγόριθμο STM, ο οποίος υλοποιεί δυναμική μνήμη STM και ονομάζεται *Transactional Locking II* (*TLII*). Ο αλγόριθμος TLII αποτελεί επέκταση του αλγορίθμου TL που παρουσιάστηκε στην εργασία [4]. Συγκεκριμένα οι συγγραφείς εισάγουν έναν καινούργιο μηχανισμό που επιτρέπει την εκτέλεση του μηχανισμού ελέγχου συνέπειας των δεδομένων με αρκετά γρήγορο τρόπο.

Ο TLII υποστηρίζει δυναμικές δοσοληψίες και έχει σχεδιαστεί για κλειστό μοντέλο μνήμης. Ο TLII χρησιμοποιεί την ίδια αναπαράσταση t-μεταβλητών με τον TL και χρησιμοποιεί τη μέθοδο απόκτησης προσωρινών ιδιοκτησιών επί των t-μεταβλητών για την επίτευξη της ιδιότητας της σειριοποιησιμότητας δοσοληψιών. Ο TLII μπορεί να χρησιμοποιεί ανά t-μεταβλητή ανάθεση κλειδωμάτων και ανά σύνολο ανάθεση κλειδωμάτων, όπως ακριβώς και ο TL. Επίσης η μορφή κάθε κλειδώματος lock στον TLII είναι ίδια με την αντίστοιχη μορφή του αλγορίθμου TL, με την πληροφορία κλειδώματος να είναι ανώνυμη. Ο TLII, σε αντίθεση με τον TL, δουλεύει μόνο με τη μέθοδο της εκ των υστέρων απόκτησης κλειδωμάτων, αποκτώντας κλειδώματα μόνο στις t-μεταβλητές που ενημερώνονται. Έτσι ο TLII, όπως και ο TL, χρησιμοποιεί μη-ορατές αναγνώσεις t-μεταβλητών και για να μπορεί να ανιχνεύει συγκρούσεις R-W υλοποιεί έναν μηχανισμό ελέγχου συνέπειας των δεδομένων των t-μεταβλητών που αναγνώσθηκαν με λειτουργία καθολικής ανάγνωσης, όμοιο με τον αντίστοιχο μηχανισμό του αλγορίθμου TL.

Υπενθυμίζεται ότι επειδή ένα κλείδωμα lock χρειάζεται μόνο ένα bit για την υλοποίηση της λειτουργικότητάς του, ο TL χρησιμοποιεί τα υπόλοιπα bits του καταχωρητή CAS (που αποθηκεύεται ο lock) για την περιγραφή της *έκδοσης* των δεδομένων της t-μεταβλητής (ή του συνόλου των t-μεταβλητών) που περιγράφει, η οποία είναι απαραίτητη για την υλοποίηση του μηχανισμού ελέγχου συνέπειας. Ο lock χρησιμοποιείται με τον ίδιο ακριβώς τρόπο και από τον TLII. Στον TL, κάθε δοσοληψία που ενημερώνει μια t-μεταβλητή tv και ολοκληρώνεται επιτυχώς, απαιτείται να αυξήσει την έκδοση της tv πριν τον τερματισμό της. Σημειώνεται ότι στον TL, κάθε επιτυχημένη δοσοληψία που ενημερώνει k t-μεταβλητές, πρέπει να ενημερώσει k εκδόσεις με διαφορετικές τιμές. Στον TLII η διατήρηση και ενημέρωση

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

των εκδόσεων γίνεται με λίγο διαφορετικό τρόπο. Ο TLII ορίζει ότι θα υπάρχει ένας *καθολικός μετρητής*, ο οποίος διατηρείται σε έναν καταχωρητή CAS και αυξάνεται από κάθε δοσοληψία που ενημερώνει τουλάχιστον μία t-μεταβλητή. Στον TLII κάθε επιτυχημένη δοσοληψία που ενημερώνει k t-μεταβλητές, ενημερώνει τις εκδόσεις αυτών των t-μεταβλητών με την νέα τιμή του καθολικού μετρητή, δηλαδή με την ίδια τιμή. Η παραλλαγή αυτή επιτρέπει στον TLII, να εκτελεί τον μηχανισμό ελέγχου συνέπειας αρκετά γρήγορα, όπως περιγράφεται στη συνέχεια.

Ο αλγόριθμος TLII περιγράφεται βάσει των λειτουργιών που παρέχει το βασικό μοντέλο STM. Στη συνέχεια, κατά τη συνοπτική περιγραφή του τρόπου υλοποίησης των λειτουργιών αυτών, παρουσιάζονται μόνο τα σημεία στα οποία διαφέρει ο TLII από τον TL. Κατά την εκκίνηση μιας δοσοληψίας $T(i,n_i)$ (δηλαδή κατά την εκτέλεση της λειτουργίας BeginTransaction) γίνεται ανάγνωση της τρέχουσας τιμής *rv* του καθολικού μετρητή. Στη συνέχεια κατά την εκτέλεση κάποια λειτουργίας ανάγνωσης (ReadTmVar) σε μια t-μεταβλητή tv από την $T(i,n_i)$, γίνεται ανάγνωση της έκδοσης της tv, έστω v_{tv}, αποθήκευση της στη λίστα αναγνώσεων της $T(i,n_i)$ και σύγκρισή της με την rv. Εάν ισχύει $v_{tv} > rv$, σημαίνει ότι τα δεδομένα της tv ενημερώθηκαν μετά την εκκίνηση της $T(i,n_i)$ και έτσι για την αποφυγή R-W συγκρούσεων, η συγκεκριμένη ReadTmVar επιστρέφει την τιμή FALSE που θα έχει ως αποτέλεσμα η $T(i,n_i)$ να ολοκληρωθεί ως μη επιτυχημένη. Επομένως η $T(i,n_i)$ συνεχίζει την εκτέλεσή της μόνο εάν ισχύει $v_{tv} \leq rv$.

Κατά την εκτέλεση της λειτουργίας CommitTransaction, από τον χρήστη, γίνεται απόκτηση των κλειδωμάτων των t-μεταβλητών που ενημερώθηκαν και προσαυξάνεται κατά ένα η τιμή του καθολικού μετρητή, με την εκτέλεση της κατάλληλης λειτουργίας CAS. Έστω wv, η νέα τιμή του καθολικού μετρητή. Στη συνέχεια, η CommitTransaction εκτελεί τον μηχανισμό ελέγχου συνέπειας των δεδομένων των t-μεταβλητών που προσπελάστηκαν με καθολική ανάγνωση (οι οποίες περιέχονται στη λίστα αναγνώσεων της $T(i,n_i)$). Κατά την εκτέλεση του μηχανισμού αυτού, ελέγχεται για κάθε t-μεταβλητή tv που ανήκει στη λίστα αναγνώσεων της $T(i,n_i)$ εάν η τρέχουσα έκδοσή της v_{tv} είναι μικρότερη ή ίση από την rv. Ο έλεγχος συνέπειας ολοκληρώνεται επιτυχώς μόνο εάν η παραπάνω συνθήκη ισχύει για όλες τις t-μεταβλητές. Εάν ο έλεγχος αυτό ολοκληρωθεί επιτυχώς η

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

CommitTransaction, ενημερώνει τα δεδομένα των επιθυμητών t-μεταβλητών, ενημερώνει την έκδοση τους με την τιμή wv, ελευθερώνει τα κλειδώματα και ολοκληρώνεται επιτυχώς. Σε αντίθετη περίπτωση, η CommitTransaction ελευθερώνει τα κλειδώματα και ολοκληρώνεται μη επιτυχώς. Σημειώνεται ότι η εκτέλεση όλων των λειτουργιών που περιγράφηκαν, μετά την εκτέλεση του μηχανισμού ελέγχου μπορούν να υλοποιηθούν με απλές εγγραφές (όχι απαραίτητα με τη χρήση εντολών CAS).

Η ύπαρξη του καθολικού μετρητή, επιτρέπει στον TLII να εγγυηθεί τη συνέπεια των δεδομένων των t-μεταβλητών που προσπελάστηκαν με καθολική ανάγνωση, με την εκτέλεση του ελέγχου συνέπειας μόνο στα δεδομένα μιας t-μεταβλητής που προσπελάζεται για πρώτη φορά μέσω της ReadTmVar. Έτσι μειώνεται αρκετά το κόστος εκτέλεσης του μηχανισμού ελέγχου συνέπειας, σε σύγκριση με άλλους αλγορίθμους που εκτελούν τον μηχανισμό αυτό με την αυξητική μέθοδο, δηλαδή κάθε ReadTmVar ελέγχει τη συνέπεια των δεδομένων όλων των t-μεταβλητών που κάποια δοσοληψία έχει ήδη προσπελάσει. Παρατηρούμε ότι ο αλγόριθμος TL εκτελεί αυτό τον πλήρη μηχανισμό ελέγχου, μόνο κατά την εκτέλεση της λειτουργίας CommitTransaction. Έτσι το κόστος εκτέλεσης του μηχανισμού ελέγχου στον αλγόριθμο TLII, μειώνεται αισθητά.

Στο σημείο αυτό συζητείται η πολυπλοκότητα του αλγορίθμου TLII. Ο TLII έχει την ίδια χωρική πολυπλοκότητα με τον TL. Κάθε επιτυχημένη δοσοληψία στον TLII που προσπελάζει R t-μεταβλητές για ανάγνωση και W t-μεταβλητές για ενημέρωση ελλείψει ανταγωνισμού, απαιτεί W εντολές CAS για την απόκτηση των προσωρινών ιδιοκτησιών των t-μεταβλητών, 2*W εντολές εγγραφής καταχωρητών ανάγνωσης εγγραφής, για την ενημέρωση των δεδομένων των t-μεταβλητών και την κατάργηση των ιδιοκτησιών των t-μεταβλητών, O(R) κόστος για την εκτέλεση του μηχανισμού ελέγχου συνέπειας.

7.3. Συνοπτικοί Παρουσίαση Αλγορίθμων

Στον Πίνακα 3.1 παρουσιάζονται συνοπτικά οι σχεδιαστικές επιλογές και ιδιότητες κάθε αλγορίθμου που παρουσιάστηκε στο παρών Κεφάλαιο. Επίσης στον πίνακα

αυτό συμπεριλαμβάνονται και τα αντίστοιχα στοιχεία του νέου αλγορίθμου NBSTM, που προτάθηκε στην παρούσα εργασία. Στο σημείο αυτό αξίζει να σημειωθεί ότι, μέχρι σήμερα ο αποδοτικότερος αλγόριθμος STM θεωρείται ο αλγόριθμος TLII, όπως ισχυρίζονται οι συγγραφείς του. Στην εργασία τους [3], μέτρησαν την απόδοσή του αλγορίθμου ως προς την ικανότητα διεκπεραίωσης δοσοληψιών και την συνέκριναν με την αντίστοιχη απόδοση των υπολοίπων αλγορίθμων και συμπέραναν ότι στις περισσότερες περιπτώσεις η απόδοσή του είναι υψηλότερη. Οι συγγραφείς της εργασίας αυτής, αλλά και μεγάλο μέρος της ερευνητικής κοινότητας, πιστεύουν ότι η υπεροχή του αλγορίθμου TLII έναντι των υπολοίπων, οφείλεται στο ότι είναι εμποδιστικός, κάτι το οποίο του επιτρέπει να αποφύγει το επιπλέον κόστος που προέρχεται από την υλοποίηση του μηχανισμού βοήθειας που χρησιμοποιούν οι μη-εμποδιστικοί αλγόριθμοι.

Αυτή η αίσθηση, οδηγεί μεγάλο μέρος της ερευνητικής κοινότητας να αναζητεί αποδοτικές υλοποιήσεις μνήμης STM με εμποδιστικούς αλγορίθμους. Όμως, είναι ένα ανοιχτό ερευνητικό θέμα εάν η υψηλή απόδοση που επιτυγχάνει ο αλγόριθμος TLII οφείλεται πράγματι στην εμποδιστική του ιδιότητα, ή στη χρήση ενός καθολικού μετρητή που του επιτρέπει να εκτελεί τον μηχανισμό ελέγχου συνέπειας, πολύ πιο αποδοτικά από τους υπολοίπους αλγορίθμους. Είναι ενδιαφέρουσα η σχεδίαση και υλοποίηση ενός νέου μη-εμποδιστικού αλγορίθμου που θα χρησιμοποιεί τη συγκεκριμένη ιδέα του TLII, ή ιδέες από άλλους εμποδιστικούς αλγορίθμους, και η σύγκρισή της απόδοσής του με την απόδοση των εμποδιστικών αλγορίθμων.

Ακόμη, ένα επιπλέον ενδιαφέρον ερευνητικό ζήτημα είναι το είδος των πειραμάτων που εκτελούνται για τη μέτρηση της απόδοσης ενός αλγορίθμου, και ακόμη περισσότερο ο τρόπος ορισμού της απόδοσης ενός αλγορίθμου. Επομένως, γίνεται κατανοητό ότι υπάρχουν αρκετά ερευνητικά θέματα που πρέπει να επιλυθούν πριν μπορούμε με βεβαιότητα να αιτιολογήσουμε την απόδοση ενός αλγορίθμου. Μια σχετική συζήτηση ακολουθεί στο Κεφάλαιο 8.

Πίνακας 7.1 Σχεδιαστικές επιλογές και ιδιότητες αλγορίθμων STM

	SSTM	DSTM	FSTM	ASTM	TL	TLII	NBSTM
Μοντέλο STM	βασικό / επεκτεταμένο	επεκτεταμένο	επεκτεταμένο	επεκτεταμένο	βασικό	βασικό	βασικό
Ιδιότητα Τερματισμού	ελευθερία κλειδωμάτων	ελευθερία ανταγωνισμού	ελευθερία κλειδωμάτων	ελευθερία ανταγωνισμού	εμποδιστικός	εμποδιστικός	ελευθερία κλειδωμάτων
Στατική ή Δυναμική Μνήμη STM	στατική	δυναμική	δυναμική	δυναμική	δυναμική	δυναμική	δυναμική
Στατικές ή Δυναμικές Δοσοληψίες	στατικές	δυναμικές	δυναμικές	δυναμικές	δυναμικές	δυναμικές	δυναμικές
Τρόπος ανάθεσης κλειδωμάτων	ανά t-μεταβλητή	ανά t-μεταβλητή	ανά t-μεταβλητή	ανά t-μεταβλητή	ανά t-μεταβλητή / ανά σύνολο	ανά t-μεταβλητή / ανά σύνολο	ανα t-μεταβλητή
Χρόνος απόκτησης κλειδωμάτων	εκ των προτέρων	εκ των προτέρων	εκ των υστέρων	εκ των προτέρων / υστέρων	εκ των προτέρων / υστέρων	εκ των υστέρων	εκ των υστέρων
Μέθοδος ενημέρωσης t-μεταβλητών	άμεση	άμεση	ετεροχρονισμένη	άμεση / ετεροχρονισμένη	άμεση / ετεροχρονισμένη	ετεροχρονισμένη	ετεροχρονισμένη
Αναγνώσεις t-μεταβλητών	ορατές	μη-ορατές	μη-ορατές	μη-ορατές	μη-ορατές	μη-ορατές	μη-ορατές
Διάκριση κλειδωμάτων ανάγνωσης - ενημέρωσης	όχι	όχι	όχι	όχι	όχι	όχι	ναι
Υλοποίηση μηχανισμού ελέγχου συνέπειας	-	ναι	ναι	ναι	ναι	ναι	ναι
Τρόπος εκτέλεσης μηχανισμού ελέγχου συνέπειας	-	αυτόματα & αυξητικά	μη -αυτόματα & αυξητικά / ανά κάποιες λειτουργίες / όποτε επιθυμεί ο χρήστης	αυτόματα & αυξητικά	αυτόματα & αυξητικά	αυτόματα & αυξητικά	αυτόματα & αυξητικά
Επώνυμη ή Ανώνυμη πληροφορίας κλειδώματος	επώνυμη	επώνυμη	επώνυμη	επώνυμη	ανώνυμη	ανώνυμη	επώνυμη
Μοντέλο μνήμης	ανοιχτό	κλειστό	κλειστό	κλειστό	κλειστό / ανοιχτό	κλειστό / ανοιχτό	κλειστό
Πλήθος ενδιάμεσων επιπέδων	1	3	1	1	1	1	3

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ΚΕΦΑΛΑΙΟ 8. ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΔΟΥΛΕΙΑ

Όπως αναφέρθηκε και στην εισαγωγή, η μελέτη θεμελιωδών θεωρητικών θεμάτων που αφορούν τους αλγορίθμους μνήμης STM βρίσκεται ακόμα σε πρώιμο στάδιο. Ενώ υπάρχουν αρκετοί αλγόριθμοι που προτείνουν αλγορίθμους για την υλοποίηση μνήμης STM, δεν υπάρχουν οι κατάλληλοι αυστηροί ορισμοί για τη σημασιολογία και τις εγγυήσεις που ένα σύστημα STM πρέπει να παρέχει. Χωρίς αυτούς τους αυστηρούς ορισμούς είναι αδύνατο να ελεγχθεί η ορθότητα των τρεχόντων αλγορίθμων STM, να γίνουν κατανοητές όλες οι λεπτομέρειες της απόδοσής τους και να ανακαλυφθούν οι περιορισμοί τους. Επιπρόσθετα, οι περισσότερες από τις ήδη υπάρχουσες εργασίες περιγράφουν τους προτεινόμενους αλγορίθμους χωρίς καν την παρουσίαση του ψευδοκώδικά τους και σε κάθε περίπτωση χωρίς καμία απόδειξη της ορθότητάς τους. Αυτό έχει ως αποτέλεσμα, η συμπεριφορά των αλγορίθμων αυτών να είναι απρόβλεπτη.

Στην παρούσα εργασία ορίσαμε ένα μοντέλο STM, με τη βοήθεια του οποίου περιγράψαμε κάποιους από τους ήδη υπάρχοντες αλγορίθμους. Επίσης, προσπαθήσαμε να ανακαλύψουμε τις διαφορετικές σχεδιαστικές επιλογές του καθενός και να τις συγκρίνουμε, όσο αυτό ήταν εφικτό, δεδομένου του ελλιπούς θεωρητικού υποβάθρου. Μελλοντικά, η κατηγοριοποίηση που πραγματοποιήθηκε στους αλγορίθμους θα φανεί αρκετά χρήσιμη. Επίσης, ορίσαμε την ιδιότητα της σειριοποιησιμότητας, την οποία κάθε αλγόριθμος πρέπει να ικανοποιεί. Με βάση το προτεινόμενο μοντέλο STM περιγράψαμε λεπτομερώς τις λειτουργίες που παρέχουν δύο από τους ήδη υπάρχοντες αλγορίθμους και με βάση την ιδιότητα της

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

σειριοποιησιμότητας, αποδείξαμε για πρώτη φορά με αυστηρό τρόπο την ορθότητά τους. Αξίζει να σημειωθεί ότι κατά την απόδειξη της ορθότητας ενός από τους αλγόριθμους αυτούς (συγκεκριμένα του SSTM) ανακαλύφθηκαν προβλήματα του κώδικα που παρουσίαζαν οι συγγραφείς του, τα οποία δεν επέτρεπαν στον αλγόριθμο να είναι ορθός. Επίσης, προτείναμε έναν νέο αλγόριθμο μνήμης STM, τον NBSTM, τον οποίο περιγράψαμε λεπτομερώς και αποδείξαμε φορμαλιστικά την ορθότητά του. Γίνεται κατανοητό ότι στην παρούσα εργασία πραγματοποιήθηκαν σημαντικά βήματα για την ενίσχυση της θεωρητικής μελέτης των αλγορίθμων STM.

Στο σημείο αυτό αξίζει να σημειωθεί ότι ο NBSTM παρουσιάζει μερικά καλά χαρακτηριστικά σε σύγκριση με τους υπάρχοντες αλγορίθμους. Εγγυάται την ισχυρότερη ιδιότητα τερματισμού ελευθερία κλειδωμάτων, σε σύγκριση με τους υπάρχοντες αλγορίθμους, η οποία σε συνδυασμό με την υποστήριξη δυναμικής μνήμης και δυναμικών δοσοληψιών εξασφαλίζεται μόνο από τον FSTM. Επίσης, ο NBSTM είναι ο πρώτος αλγόριθμος που εγγυάται αυτή την ιδιότητα τερματισμού και επιτρέπει την ολοκλήρωση μιας δοσοληψίας ως επιτυχημένης ή μη επιτυχημένης με την εκτέλεση μίας μόνο εντολής CAS.

Στη συνέχεια συζητούνται μελλοντικές επεκτάσεις της παρούσας εργασίας. Υπενθυμίζεται ότι η παρουσίαση ενός κοινού μοντέλου μνήμης STM που μπορεί να περιγράψει όλους τους υπάρχοντες αλγορίθμους STM και το οποίο θα χρησιμοποιούν όλοι οι μεταγενέστεροι αλγόριθμοι μνήμης STM, είναι απαραίτητη ώστε να μπορούν να συγκριθούν πιο εύκολα. Το μοντέλο STM που παρουσιάζεται στην παρούσα εργασία είναι αρκετά γενικό. Για λόγους απλότητας, υποστηρίζει την αποθήκευση μόνο δεικτών ως δεδομένα των t-μεταβλητών. Μελλοντικός στόχος αποτελεί η μελέτη των αναγκαίων τροποποιήσεων που πρέπει να γίνουν, ώστε το μοντέλο αυτό να επιτρέπει την αποθήκευση των ίδιων των δεδομένων στις t-μεταβλητές και όχι των δεικτών τους. Φαίνεται ότι για να είναι κάτι τέτοιο εφικτό, ο χρήστης θα πρέπει να παρέχει πληροφορία στο σύστημα STM για την μορφή των δεδομένων που αποθηκεύει σε κάθε t-μεταβλητή, έτσι ώστε το σύστημα να μπορεί να τα διαχειριστεί ανάλογα με τις ανάγκες του. Το πρόβλημα γίνεται κατανοητό εάν σκεφτούμε ότι ο χρήστης μπορεί να αποθηκεύσει μια αριθμητική τιμή σε μια t-μεταβλητή ή να επιλέγει να αποθηκεύσει ένα στοιχείο μιας δομής την οποία ο ίδιος όρισε. Στην

περίπτωση αυτή, το σύστημα πρέπει να γνωρίζει απαραίτητα το μέγεθος των δεδομένων που εισάγονται ώστε να τα αποθηκεύσει κατάλληλα. Ένα ακόμη παράδειγμα αποτελεί η ανάγκη δημιουργίας αντιγράφων των δεδομένων των t-μεταβλητών, που διενεργείται από αλγορίθμους STM που διατηρούν δομή επαναφοράς ή επανεκτέλεσης.

Μελλοντικό στόχο αποτελεί η εύρεση εναλλακτικών ιδιοτήτων ορθότητας, πέραν της σειριοποιησιμότητας, και κατάλληλων ιδιοτήτων τερματισμού. Επίσης, αποτελεί ανοιχτό θέμα το κατά πόσο μια συγκεκριμένη ιδιότητα ορθότητας μπορεί να επηρεάζει την ιδιότητα τερματισμού των αλγορίθμων. Αξίζει να σημειωθεί ότι οι ορισμοί των ιδιοτήτων τερματισμού που δόθηκαν στην παρούσα εργασία, δεν καλύπτουν πλήρως τις ανάγκες των αλγορίθμων STM. Συγκεκριμένα στην παρούσα εργασία συζητήθηκε ότι ένας αλγόριθμος STM ικανοποιεί μια ιδιότητα τερματισμού εάν κάθε λειτουργία του ικανοποιεί την ιδιότητα αυτή. Όμως θα ήταν επιθυμητό ο τερματισμός μιας δοσοληψίας να μην σχετίζεται μόνο με τον τερματισμό των λειτουργιών της, αλλά και με τον τρόπο που η ίδια δοσοληψία τερματίζει, δηλαδή ως επιτυχημένη ή μη επιτυχημένη. Έτσι, στους αλγορίθμους STM ενδιαφερόμαστε περισσότερο για τις ιδιότητες που πρέπει να ικανοποιεί ένας αλγόριθμος ώστε να εξασφαλίζεται ότι κάποια δοσοληψία του ολοκληρώνεται επιτυχώς και όχι ότι απλά ολοκληρώνεται. Επομένως, μελλοντικό στόχο αποτελεί ο καθορισμός των ιδιοτήτων αυτών.

Είναι σημαντική η μελέτη και εύρεση κατάλληλων μετρικών πολυπλοκότητας και απόδοσης για τους αλγορίθμους STM, που θα επιστρέψουν τη σύγκριση των ήδη υπαρχόντων αλγορίθμων αλλά και των νέων που θα προταθούν. Επίσης, είναι ενδιαφέρουσα η ανακάλυψη περιορισμών των αλγορίθμων STM και η απόδειξη κάτω ορίων (lower-bound) ή αρνητικών αποτελεσμάτων (impossibility results). Κάτι τέτοιο θα βοηθήσει τους σχεδιαστές συστημάτων STM, να αναγνωρίζουν πότε η λύση ενός ζητήματος είναι βέλτιστη, θα γνωρίζουν προς ποια κατεύθυνση δεν μπορούν να βελτιώσουν τους αλγορίθμους, θα τους επιστρέψει να ανακαλύψουν νέες δυσκολίες και να προτείνουν καινοτόμες λύσεις για την επίτευξη υψηλότερης απόδοσης.

Τελικό μελλοντικό στόχο αποτελεί η σχεδίαση και υλοποίηση νέων μηχανισμών που θα επιτρέπουν στο σύστημα STM να βασίζεται στα ιδιαίτερα χαρακτηριστικά ενός διαμοιραζόμενου αντικειμένου ή μιας εφαρμογής, ώστε να προβλέπει τα προβλήματα συγχρονισμού που μπορούν να προκύψουν και να επιτυγχάνει σημαντικά αυξημένη απόδοση. Για να γίνει αυτό κατανοητό, υπενθυμίζουμε την ανάγκη ταξινόμησης της λίστας ενημέρωσης μιας δοσοληψίας που απαιτείται από τους αλγορίθμους FSTM και NBSTM. Ο αλγόριθμος STM σε ειδικές περιπτώσεις θα μπορούσε να αποφύγει το σημαντικό κόστος ταξινόμησης. Για παράδειγμα εάν ο χρήστης υλοποιούσε τις λειτουργίες μιας συνδεδεμένης λίστας, οι οποίες προσπελάζουν τα στοιχεία της λίστας με συγκεκριμένη σειρά, τότε ο αλγόριθμος STM θα μπορούσε να αποφύγει την ταξινόμηση της λίστας ενημερώσεων και να επιτύχει την ίδια λειτουργικότητα, απλά εξασφαλίζοντας ότι οι t-μεταβλητές διατηρούνται στη λίστα ενημερώσεων με τη σειρά που προσπελάστηκαν για πρώτη φορά από τον χρήστη.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ΑΝΑΦΟΡΕΣ

[1] J. Anderson, Y. Kim and T. Herman. "Shared-Memory Mutual Exclusion: Major Research Trends Since 1986", *Distributed Computing*, 16(2-3): 75-110, September 2003.

[2] B. Bloom. "Space/time trade-offs in hash coding with allowable errors", *Communications of the ACM*, 13(7): 422-426, July 1970.

[3] D. Dice, O. Shalev and N. Shavit. "Transactional Locking II", In *Proceedings of the 20th International Symposium on Distributed Computing (DISC)*, September 2006.

[4] D. Dice and N. Shavit. "What Really Makes Transactions Fast", In *Proceedings of the 1st ACM SIGPLAN Workshop on Transactional Computing (TRANSACT)*, June 2006.

[5] M. Fomitchev and E. Ruppert. "Lock-Free Linked Lists and Skip Lists", In *Proceedings of the 23rd annual ACM symposium on Principles of distributed computing (PODC)*, pages 50-59, July 2004.

[6] K. Fraser. *Practical Lock Freedom*, Phd Thesis, University of Cambridge, September 2003.

[7] R. Guerraoui and M. Kapalka. "On Obstruction-Free Transactions", In *Proceedings of the 20th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 304-313, June 2008.

[8] M. Herlihy. "Wait-Free Synchronization", In *ACM Transactions on Programming Languages and Systems*, 13(1):124-149, January 1991.

[9] M. Herlihy, J. Eliot and B. Moss. "Transactional Memory: Architectural Support for Lock-Free Data Structures", In *20th Annual Symposium on Computer Architecture*, pages 289-300, May 1993.

[10] M. Herlihy and E. Koskinen. "Transactional Boosting: a Methodology for Highly-Concurrent Transactional Objects", In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 207-216, February 2008.

[11] M. Herlihy, V. Luchangco and M. Moir. "Obstruction-Free Synchronization: Double-Ended Queues as an Example", In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, pages 522-529, May 2003.

[12] M. P. Herlihy, V. Luchangco, M. Moir and W. N. Scherer III. "Software Transactional Memory for Dynamic-Sized Data Structures", In *Proceedings of 22nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 92-101, July 2003.

[13] M. P. Herlihy and J. M. Wing. "Linearizability: a Correctness Condition for Concurrent Objects", *ACM Transactions on Programming Languages and Systems*, 12(3): 463-492, June 1990.

[14] V. J. Marathe, W. N. Scherer III and M. L. Scott. "Design Tradeoffs in Modern Software Transactional Memory Systems", In *Proceedings of 7th Workshop on Languages, Compilers, and Run-Time Support For Scalable Systems*, pages 1-7, October 2004.

[15] V. J. Marathe, W. N. Scherer III and M. L. Scott. "Adaptive Software Transactional Memory", In *Proceedings of the 19th International Symposium on Distributed Computing (DISC)*, pages 354-368, September 2005.

[16] M. Michael. "High Performance Dynamic Lock-Free Hash Tables and List-Based Sets", In *Proceedings of the 14th annual ACM symposium on Parallel algorithms and architectures (SPAA)*, pages 73-82, August 2002

[17] B. Saha, A.-R. Adl-Tabatabai, R. L. Hundson, C. C. Minh and B. Hertzberg. "McRT-STM: a High Performance Software Transactional Memory System for a Multi-Core Runtime", In *Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 187-197, March 2006.

[18] M. L. Scott. "Sequential Specification of Transactional Memory Semantics", In *Proceedings of the 1st ACM SIGPLAN Workshop on Transactional Computing (TRANSACT)*, June 2006.

[19] N. Shavit and D. Touitou. "Software Transactional Memory", *Distributed Computing*, Special Issue(10), pages 99-116, February 1997. (also, In *Proceedings of 14th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 204-213, August 1995).

[20] M. F. Spear, M. M. Michael and C. von Praun. "RingSTM: Scalable Transactions with a Single Atomic Instruction", In *Proceedings of the 20th Annual Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 275-284, June 2008.

[21] J. D. Valois. "Lock-Free Linked Lists Using Compare-And-Swap", In *Proceedings of the 14th annual ACM symposium on Principles of distributed computing (PODC)*, pages 214-222, August 1995.

ΣΥΝΤΟΜΟ ΒΙΟΓΡΑΦΙΚΟ

Ο Κοσμάς Ελευθέριος γεννήθηκε το 1984 στην Θεσσαλονίκη. Αποφοίτησε από το 2° Ενιαίο Λύκειο Κατερίνης το 2001 και την ίδια χρονιά εισήχθη στο προπτυχιακό πρόγραμμα σπουδών του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων. Ολοκλήρωσε τις προπτυχιακές του σπουδές τον Ιούνιο του 2005 και έπειτα από τον Φεβρουάριο του 2006 παρακολουθεί το μεταπτυχιακό πρόγραμμα σπουδών του ίδιου Τμήματος. Τα ευρευνητικά του ενδιαφέροντα εστιάζονται στον τομέα των Παράλληλων και Κατανεμημένων Συστημάτων.