Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Αλγόριθμοι Συλλογής Απορριμμάτων για Αυτόματη Διαχείριση Μνήμης

Διπλωματική Εργασία

του

Δημήτρη Χ. Κονόμη

Επιβλέπων: Νικόλαος Παπασπύρου
Αν. Καθηγητής Ε.Μ.Π.

Εργαστήριο Τεχνολογίας Λογισμικού
Αθήνα, Απρίλιος 2015

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Τεχνολογίας Λογισμικού

Αλγόριθμοι Συλλογής Απορριμμάτων για Αυτόματη Διαχείριση Μνήμης

Διπλωματική Εργασία

του

Δημήτρη Χ. Κονόμη

Επιβλέπων: Νικόλαος Παπασπύρου
Αν. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 2η Απριλίου, 2015.

.....	.....	.....
Νικόλαος Παπασπύρου	Κωνσταντίνος Σαγώνας	Κωνσταντίνος Κοντογιάννης
Αν. Καθηγητής Ε.Μ.Π.	Αν. Καθηγητής Ε.Μ.Π.	Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Απρίλιος 2015

.....
Δημήτρης Χ. Κονόμης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Αν και αρχικά περιοριζόταν στη γλώσσα Lisp και τις συναρτησιακές γλώσσες προγραμματισμού, σήμερα η συλλογή απορριμμάτων αποτελεί σημαντικό κομμάτι του συστήματος διαχείρισης μνήμης πολλών σύγχρονων γλωσσών, τόσο προστακτικών όσο και δηλωτικών. Παρά τη φήμη πως επιβραδύνει και διαταράσσει την εκτέλεση διαδραστικών εφαρμογών, οι σύγχρονες τεχνικές υλοποίησης της συλλογής απορριμμάτων έχουν μειώσει την επιβάρυνση της σημαντικά, σε τέτοιο σημείο ώστε να αποτελεί ρεαλιστική επιλογή ακόμη και για παραδοσιακές γλώσσες προγραμματισμού όπως η C.

Παρά την ταχεία ανάπτυξη του μεγέθους μνήμης ακόμη και των πιο φθηνών υπολογιστών, ο αποθηκευτικός χώρος δεν είναι ανεξάντλητος. Όπως όλοι οι περιορισμένοι πόροι απαιτεί προσεκτική συντήρηση και ανακύκλωση. Πολλές γλώσσες προγραμματισμού σήμερα επιτρέπουν την εκχώρηση και την ανάκτηση μνήμης από τον προγραμματιστή για δεδομένα των οποίων η διάρκεια ζωής δεν καθορίζεται από τη λεκτική εμβέλεια τους. Λέμε πώς τα δεδομένα αυτά *εκχωρούνται δυναμικά*. Η διαχείριση της δυναμικής μνήμης μπορεί να γίνεται ρητώς από τον προγραμματιστή, μέσω κλήσεων διαδικασιών ενσωματωμένων στο σύστημα εκτέλεσης ή διαδικασιών βιβλιοθήκης που εκχωρούν αποθηκευτικό χώρο και τον ελευθερώνουν όταν αυτός δεν είναι πλέον απαραίτητος.

Η χειρωνακτική ανάκτηση της δυναμικά διαχειριζόμενης μνήμης συχνά δεν είναι ικανοποιητική. Εναλλακτικά, η ευθύνη αυτής της διαχείρισης μπορεί να μεταβιβασθεί στο σύστημα εκτέλεσης του προγράμματος. Ο προγραμματιστής εξακολουθεί να πραγματοποιεί δυναμικά αιτήματα εκχώρησης μνήμης, χωρίς όμως πλέον να χρειάζεται να καθορίσει πότε αυτή η μνήμη δεν είναι πλέον απαραίτητη: αυτή ανακυκλώνεται αυτόματα. Η *συλλογή απορριμμάτων*, που αποτελεί το κεντρικό θέμα αυτής της εργασίας, είναι ακριβώς η αυτόματη διαχείριση δυναμικά εκχωρούμενου χώρου αποθήκευσης.

Σκοπός αυτής της εργασίας είναι να αποτελέσει σημείο αναφοράς για τους αναγνώστες που ενδιαφέρονται να μάθουν σχετικά με την αυτόματη διαχείριση μνήμης. Το εισαγωγικό κεφάλαιο αρχικά εξετάζει και συγκρίνει την αυτόματη διαχείριση μνήμης και τη ρητή διαχείριση μνήμης, στη συνέχεια παρουσιάζει τις μετρικές με βάση τις οποίες συγκρίνονται τα διάφορα σχήματα συλλογής απορριμμάτων και τέλος ορίζει τις έννοιες του *εκχωρητή*, του *συλλέκτη* και του *τροποποιητή*. Το υπόλοιπο της εργασίας οργανώνεται σε δύο τμήματα. Στο πρώτο αρχικά συζητώνται οι κλασσικές προσεγγίσεις συλλογής απορριμμάτων: *σήμανση και εκκαθάριση*, *σήμανση και συμπύκνωση*, *αντιγραφή* και *καταμέτρηση αναφορών*, και στη συνέχεια γίνεται μία σύγκριση μεταξύ αυτών. Το δεύτερο τμήμα επικεντρώνεται σε πιο προηγμένες τεχνικές συλλογής απορριμμάτων: *γενεαλογική*, *παράλληλη* και *ταυτόχρονη* συλλογή απορριμμάτων. Τέλος δίνεται μια εισαγωγή στη συλλογή απορριμμάτων *πραγματικού-χρόνου*.

Η εργασία αυτή βασίζεται σε μετάφραση και προσαρμογή στα ελληνικά τμημάτων του βιβλίου *The Garbage Collection Handbook: The Art of Automatic Memory Management*,

των R. Jones, A. Hosking και E. Moss [66].

Λέξεις Κλειδιά

γλώσσες προγραμματισμού, αυτόματη διαχείριση μνήμης, συλλογή απορριμμάτων, σήμανση και εκκαθάριση, σήμανση και συμπύκνωση, αντιγραφή, καταμέτρηση αναφορών, γενεαλογική, παράλληλη, ταυτόχρονη, πραγματικού-χρόνου.

Abstract

Whereas it was once confined to the realm of Lisp and functional languages, today garbage collection is an important part of the memory management system of many modern programming languages, imperative as well as declarative. Although garbage collection has had a reputation for sloth and for disrupting interactive programs, modern implementation techniques have reduced its overheads substantially, to the point where garbage collected heaps are a realistic option even for traditional languages like C.

Despite the rapid growth in memory sizes of even the most modest computers, the supply of storage is not inexhaustible. Like all limited resources it requires careful conservation and recycling. Many programming languages today allow the programmer to allocate and reclaim memory for data whose lifetimes are not determined by lexical scope. Such data is said to be *dynamically allocated*. Dynamic memory may be managed explicitly by the programmer through invocations of built-in or library procedures that allocate storage and that dispose or free that storage when it is no longer needed.

Manual reclamation of dynamically managed storage is often unsatisfactory. The alternative is to devolve responsibility for dynamic memory management to the program's run-time system. The programmer must still request dynamically allocated storage to be reserved but no longer needs to determine when that memory is no longer required: it is recycled automatically. Garbage collection, the main topic of this thesis, is precisely this: the automatic management of dynamically allocated storage.

The purpose of this thesis is to become a point of reference for readers interested in learning about automatic memory management. The introductory chapter first examines and compares automatic memory management and explicit memory management, then presents the metrics under which different garbage collection schemes are compared and finally defines the notion of the *allocator*, *collector* and *mutator*. The rest of the thesis is organized in two parts. The first initially discusses the classical garbage collection approaches. We examine *mark-sweep*, *mark-compact*, *copying* and *reference counting* and then give a comparison between them. The second part focuses on more advanced garbage collection techniques. We examine *generational*, *parallel* and *concurrent* garbage collection. Finally, an introduction to *real-time* garbage collection is given.

This thesis is based on the translation and adaptation to Greek of parts of the book *The Garbage Collection Handbook: The Art of Automatic Memory Management*, by R. Jones, A. Hosking and E. Moss [66].

Keywords

programming languages, automatic memory management, garbage collection, mark-sweep, mark-compact, copying, reference counting, generational, parallel, concurrent, real-time.

Ευχαριστίες

Με την παρούσα διπλωματική εργασία ολοκληρώνονται οι σπουδές μου στην Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών του Εθνικού Μετσοβίου Πολυτεχνείου.

Αισθάνομαι τη βαθύτατη ανάγκη να ευχαριστήσω τον επιβλέποντα αυτής της εργασίας καθηγητή Νίκο Παπασπύρου, ο οποίος εκτός από εξαίρετος επιστήμονας και δάσκαλος, είναι και σπουδαίος άνθρωπος. Είναι εκείνος που συνέβαλλε στο να αγαπήσω την Πληροφορική και τις Γλώσσες Προγραμματισμού ειδικότερα.

Παράλληλα θέλω να ευχαριστήσω θερμά και τους καθηγητές μου Κωστή Σαγώνα, Κώστα Κοντογιάννη, Δημήτρη Φωτάκη, Τίμο Σελλή και Στάθη Ζάχο από τους οποίους επίσης διδάχθηκα πάρα πολλά.

Κατά τη διάρκεια των σπουδών μου όμως δεν έμαθα μόνο από τους καθηγητές μου, αλλά και από τους φίλους μου. Ευχαριστώ τον Ηλία, το Ζήση, το Στέφανο, το Νίκο, τον Κωνσταντίνο, το Νίκο, το Διονύση και το Βρεττό.

Last και σίγουρα not least, οφείλω ένα μεγάλο ευχαριστώ στην οικογένειά μου, που, παρά τις όποιες δυσκολίες, με στήριξε και βοήθησε στο να φτάσω εδώ που είμαι σήμερα.

Δημήτρης Χ. Κονόμης

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-TR-1-15, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Τεχνολογίας Λογισμικού, Απρίλιος 2015.

URL: `http://www.softlab.ntua.gr/techrep/`
FTP: `ftp://ftp.softlab.ntua.gr/pub/techrep/`

Περιεχόμενα

Σχήματα

Πίνακες

Αλγόριθμοι

Κεφάλαιο 1

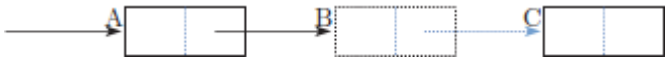
Εισαγωγή

Οι προγραμματιστές στρέφονται ολοένα και περισσότερο σε γλώσσες και συστήματα εκτέλεσης που παρέχουν αυτόματες υπηρεσίες διαχείρισης μνήμης, χάριν των πολλών πλεονεκτημάτων που αυτά προσφέρουν, όπως η αυξημένη ασφάλεια του κώδικα και η δυνατότητα προγραμματισμού σε υψηλό αφαιρετικό επίπεδο που δεν απαιτεί λεπτομερή γνώση του λειτουργικού συστήματος ή/και της αρχιτεκτονικής. Ο Butters [31] διαπιστώνει πως τα οφέλη του διαχειριζόμενου κώδικα είναι ευρέως αποδεκτά. Καθώς η εικονική μηχανή προσφέρει πολλές υπηρεσίες, οι προγραμματιστές χρειάζεται να γράφουν λιγότερο κώδικα. Ο κώδικας είναι ασφαλέστερος αν περάσει επιτυχώς έναν (στατικό) έλεγχο τύπων και αν το σύστημα εκτέλεσης επαληθεύει τον κώδικα καθώς αυτός φορτώνεται, ελέγχει για παραβιάσεις κατά την πρόσβαση σε πόρους (όπως για παράδειγμα δεικτοδότηση εκτός ορίων ενός πίνακα) και διαχειρίζεται αυτόματα τη μνήμη. Το κόστος ανάπτυξης λογισμικού μειώνεται σημαντικά καθώς είναι πλέον ευκολότερη (αν και όχι πάντα εφικτή) η ανάπτυξη εφαρμογών που δύνανται να τρέξουν σε διαφορετικές πλατφόρμες. Τα παραπάνω επιτρέπουν στους προγραμματιστές να αφιερώνουν περισσότερο χρόνο στη λογική των εφαρμογών.

Σχεδόν όλες οι σύγχρονες γλώσσες προγραμματισμού χρησιμοποιούν **δυναμική εκχώρηση μνήμης**. Αυτό επιτρέπει τη δέσμευση και αποδέσμευση μνήμης για ένα αντικείμενο ακόμη και αν το συνολικό μέγεθος αυτού δεν ήταν γνωστό κατά τη διάρκεια της μεταγλώττισης ή η διάρκεια ζωής του ξεπερνά αυτήν της ρουτίνας στην οποία αυτό εκχωρήθηκε. Ένα αντικείμενο που δημιουργείται δυναμικά αποθηκεύεται στο **σωρό** και όχι στη **στοίβα** (στο **εγγράφημα δραστηριοποίησης** της διαδικασίας στην οποία εκχωρήθηκε) είτε **στατικά** (δηλαδή σε κάποια διεύθυνση που είναι γνωστή κατά το χρόνο μεταγλώττισης ή σύνδεσης). Η εκχώρηση μνήμης στο σωρό είναι ιδιαίτερα σημαντική καθώς επιτρέπει στον προγραμματιστή:

- να επιλέξει δυναμικά το μέγεθος νέων αντικειμένων,
- να ορίσει και να χρησιμοποιήσει αναδρομικές δομές δεδομένων όπως συνδεδεμένες λίστες και δυαδικά δένδρα,
- να επιστρέφει αντικείμενα στην καλούσα διαδικασία,
- να επιστρέφει μια συνάρτηση ως αποτέλεσμα μιας συνάρτησης.

Η πρόσβαση στα αντικείμενα του σωρού γίνεται μέσω **αναφορών**. Τυπικά, μια αναφορά είναι ένας **δείκτης** προς το αντικείμενο (και πιο συγκεκριμένα στη διεύθυνση του στη μνήμη).



Σχήμα 1.1: Η πρόωρη διαγραφή ενός αντικειμένου μπορεί να οδηγήσει σε σφάλματα. Εδώ η μνήμη του αντικειμένου B έχει ανακτηθεί. Το ζωντανό αντικείμενο A περιέχει πλέον έναν ξεκρέμαστο δείκτη. Ταυτόχρονα, υπάρχει διαρροή μνήμης: η μνήμη που καταλαμβάνει το αντικείμενο C δεν μπορεί να ανακτηθεί παρότι αυτό δεν είναι πια προσβάσιμο.

1.1 Ρητή αποδέσμευση μνήμης

Κάθε μη τετριμμένο πρόγραμμα το οποίο τρέχει με πεπερασμένη μνήμη στη διάθεσή του, χρειάζεται κατά καιρούς να επανακτήσει τη μνήμη που φιλοξενεί αντικείμενα μη χρήσιμα στους υπολογισμούς. Η μνήμη που χρησιμοποιείται από αντικείμενα του σωρού μπορεί να ανακτηθεί είτε χρησιμοποιώντας **ρητή αποδέσμευση** (όπως για παράδειγμα συμβαίνει με τις συναρτήσεις FREE στη γλώσσα C ή DELETE στη γλώσσα C++), είτε αυτόματα από το **σύστημα εκτέλεσης**, χρησιμοποιώντας καταμέτρηση αναφορών όπως ο Collins [38], ή ένα συλλέκτη απορριμμάτων εξιχνίασης όπως ο McCarthy [81]. Η ρητή αποδέσμευση μνήμης είναι ευάλωτη σε δύο είδη σφαλμάτων.

Πρώτον, η μνήμη ενός αντικειμένου μπορεί να αποδεσμευθεί πρόωρα και ενώ υπάρχουν ακόμη αναφορές προς το αντικείμενο. Μία τέτοια αναφορά ονομάζεται **ξεκρέμαστος δείκτης**. Το αποτέλεσμα που προκύπτει αν ένα πρόγραμμα ακολουθήσει έναν τέτοιο δείκτη είναι απρόβλεπτο. Ο προγραμματιστής δεν έχει κανέναν έλεγχο όσον αφορά το τι συμβαίνει με την αποδεσμευμένη μνήμη: το σύστημα εκτέλεσης μπορεί να επιλέξει, μεταξύ άλλων να την καθαρίσει (εγγράψει με μηδενικά), να την εκχωρήσει για την αποθήκευση ενός νέου αντικειμένου ή να την επιστρέψει στο λειτουργικό σύστημα. Το καλύτερο σενάριο στο οποίο μπορεί ο προγραμματιστής να ελπίζει είναι ο βίαιος τερματισμός του προγράμματος. Είναι ωστόσο πιθανότερο το πρόγραμμα να συνεχίσει την εκτέλεσή του για εκατομμύρια κύκλους πριν τερματιστεί (καθιστώντας την αποσφαλμάτωση επίπονη) ή ακόμη και να ολοκληρώσει επιτυχώς την εκτέλεσή του παράγοντας ωστόσο λανθασμένα αποτελέσματα (γεγονός που ανιχνεύεται δύσκολα).

Δεύτερον, ο προγραμματιστής μπορεί να αποτύχει να αποδεσμεύσει τη μνήμη από ένα αντικείμενο το οποίο δε χρειάζεται πια το πρόγραμμα, οδηγώντας με τον τρόπο αυτό σε **διαρροή μνήμης**. Σε μικρά προγράμματα οι διαρροές μπορούν να αγνοηθούν, ωστόσο σε μεγάλα προγράμματα μπορεί να προκαλέσουν μείωση της επίδοσης (καθώς ο διαχειριστής μνήμης πασχίζει να ικανοποιήσει αιτήματα εκχώρησης) ή ακόμη αποτυχία εκτέλεσης (αν το πρόγραμμα ξεμείνει από μνήμη). Συχνά μάλιστα μία εσφαλμένη αποδέσμευση μπορεί να προκαλέσει ταυτόχρονα τόσο ένα ξεκρέμαστο δείκτη όσο και μια διαρροή.

Τα προγραμματιστικά σφάλματα του τύπου αυτού είναι συνήθη σε περιβάλλοντα όπου δύο ή και περισσότερες διαδικασίες διατηρούν αναφορές προς το ίδιο αντικείμενο. Η κατάσταση είναι ακόμη πιο προβληματική σε περιβάλλοντα ταυτόχρονου προγραμματισμού όπου δύο ή περισσότερα νήματα χειρίζονται δείκτες προς το ίδιο αντικείμενο.

Τι κάνουν λοιπόν οι προγραμματιστές σε μία γλώσσα που δεν υποστηρίζει αυτόματη δυναμική διαχείριση μνήμης; Η ερώτηση αυτή έχει απασχολήσει αρκετούς ερευνητές με την επικρατέστερη άποψη να ορίζει πώς ο προγραμματιστής απαιτείται να είναι συνεπής όσον αφορά τον τρόπο με τον οποίο χειρίζεται την **ιδιοκτησία** των αντικειμένων. Ο Belotsky [16] και άλλοι ερευνητές προτείνουν διαφορετικές στρατηγικές για τη γλώσσα C++. Αρχικά, οι προγραμ-

ματιστές πρέπει να αποφεύγουν τη δέσμευση μνήμης στο σωρό, όποτε αυτό είναι δυνατό. Τα αντικείμενα μπορούν να δεσμεύονται στη στοίβα. Και κατά δεύτερον οι προγραμματιστές θα πρέπει να περνούν σε και να επιστρέφουν από συναρτήσεις/διαδικασίες αντικείμενα κατά τιμή, αντιγράφοντας όλα τα περιεχόμενα της παραμέτρου ή του αποτελέσματος αντί να περνούν δείκτες σε αυτά. Είναι προφανές πώς και οι δύο προσεγγίσεις αποτρέπουν τα σφάλματα που αφορούν τη δέσμευση και αποδέσμευση μνήμης αλλά συνοδεύονται από αυξημένη κίνηση μνήμης και απώλεια της δυνατότητας διαμοιρασμού δεδομένων. Σε μερικές περιπτώσεις επίσης μπορούν να χρησιμοποιηθούν ειδικοί εκχωρητές, οι οποίοι για παράδειγμα μπορούν να διαχειρίζονται μια δεξαμενή αντικειμένων. Στο τέλος της φάσης ενός προγράμματος, ολόκληρη η δεξαμενή μπορεί να αποδεσμεύεται. Στη γλώσσα προγραμματισμού C++ έχουν προταθεί και υλοποιηθεί ειδικές κλάσεις αντικειμένων δεικτών με σκοπό την καλύτερη διαχείριση μνήμης.

Η πληθώρα των διαφορετικών στρατηγικών για ασφαλή ρητή διαχείριση μνήμης δημιουργεί ακόμη ένα πρόβλημα. Ποια προσέγγιση πρέπει να ακολουθήσει ο προγραμματιστής προκειμένου να διαχειριστεί με συνέπεια την ιδιοκτησία των αντικειμένων; Αυτό είναι ιδιαίτερα δύσκολο να απαντηθεί όταν χρησιμοποιείται κώδικας βιβλιοθήκης. Προκύπτουν ερωτήματα σχετικά με το ποια προσέγγιση υιοθετεί ο κώδικας της βιβλιοθήκης και αν όλες οι βιβλιοθήκες που χρησιμοποιεί το πρόγραμμα υιοθετούν την ίδια προσέγγιση.

1.2 Αυτόματη δυναμική διαχείριση μνήμης

Η αυτόματη διαχείριση μνήμης επιλύει πολλά από τα παραπάνω προβλήματα. Η **συλλογή απορριμμάτων** αποτρέπει την εμφάνιση ξεκρέμαστων δεικτών: ένα αντικείμενο ελευθερώνεται μόνο όταν δεν υπάρχει κάποιος δείκτης σε αυτό από προσβάσιμο αντικείμενο. Αντίστροφα, όλα τα μη προσβάσιμα αντικείμενα θα ελευθερωθούν τελικώς από το συλλέκτη. Όλες οι αποφάσεις σχετικά με την ανάκτηση μνήμης ανατίθενται στο συλλέκτη, ο οποίος διαθέτει καθολική γνώση της δομής των αντικειμένων στο σωρό και των νημάτων που έχουν πρόσβαση σε αυτά.

Η διαχείριση μνήμης είναι ένα θέμα μηχανικής λογισμικού. Τα καλώς σχεδιασμένα προγράμματα χτίζονται από ψηφίδες υψηλής συνεκτικότητας και χαμηλής σύζευξης. Η αύξηση της συνεκτικότητας και η μείωση της σύζευξης καθιστά ευκολότερη τη συντήρηση των προγραμμάτων. Ιδανικά, ένας προγραμματιστής θα πρέπει να είναι σε θέση να καταλαβαίνει τη συμπεριφορά μιας ψηφίδας μόνο από τον κώδικα της ψηφίδας ή στη χειρότερη περίπτωση και από τον κώδικα ενός μικρού αριθμού συγγενικών ψηφίδων. Η μείωση της σύζευξης μεταξύ ψηφίδων πρακτικά σημαίνει πώς η συμπεριφορά μιας ψηφίδας δεν εξαρτάται από την υλοποίηση κάποιας άλλης ψηφίδας. Στο πλαίσιο της σωστής διαχείρισης μνήμης, αυτό σημαίνει πώς οι ψηφίδες δεν έχουν γνώση της εσωτερικής λειτουργίας των ψηφίδων που υλοποιούν τη διαχείριση μνήμης. Η χειρωνακτική διαχείριση μνήμης αντίθετα δε συμμορφώνεται με τις αρχές της μηχανικής λογισμικού για ελαχιστοποίηση της επικοινωνίας μεταξύ ψηφίδων. Το κύριο επιχείρημα υπέρ της αυτόματης διαχείρισης μνήμης δεν είναι ότι απλοποιεί τη συγγραφή κώδικα (το οποίο ισχύει) αλλά ότι διαζευγνύει το πρόβλημα της διαχείρισης μνήμης από διαπροσωπείες αντί να το διασπείρει στον κώδικα. Επίσης διευκολύνει την επαναχρησιμοποίηση κώδικα. Γι αυτούς τους λόγους η συλλογή απορριμμάτων αποτελεί, άμεσα ή έμμεσα απαίτηση στο πρότυπο των περισσότερων σύγχρονων γλωσσών προγραμματισμού.

Τονίζουμε ωστόσο πώς η συλλογή απορριμμάτων δεν μπορεί να εξαλείψει πλήρως την εμφάνιση σφαλμάτων που αφορούν τη μνήμη. Οι διαρροές μνήμης αποτελούν ένα από τα πιο συχνά εμφανιζόμενα σφάλματα μνήμης. Παρότι η συλλογή απορριμμάτων τείνει να μειώσει

ActionScript (2000)	Algol-68 (1965)	APL (1964)
AppleScript (1993)	AspectJ (2001)	Awk (1977)
Beta (1983)	C# (1999)	Cyclone (2006)
Managed C++ (2002)	Cecil (1992)	Cedar (1983)
Clean (1984)	CLU (1974)	D (2007)
Dylan (1992)	Dynace (1993)	E (1997)
Eiffel (1986)	Elasti-C (1997)	Emerald (1988)
Erlang (1990)	Euphoria (1993)	F# (2005)
Fortress (2006)	Green (1998)	Go (2010)
Groovy (2004)	Haskell (1990)	Hope (1978)
Icon (1977)	Java (1994)	JavaScript (1994)
Liana (1991)	Limbo (1996)	Lingo (1991)
LotusScript (1995)	Lua (1994)	Mathematica (1987)
MATLAB (1970s)	Mercury (1993)	Miranda (1985)
ML (1990)	Modula-3 (1988)	Oberon (1985)
Objective-C (2007-)	Obliq (1993)	Perl (1986)
Pike (1996)	PHP (1995)	Pliant (1999)
POP-2 (1970)	PostScript (1982)	Prolog (1982)
Python (1991)	Rexx (1979)	Ruby (1993)
Sather (1990)	Scala (2003)	Scheme (1975)
Self (1986)	SETL (1969)	Simula (1964)
SISAL (1983)	Smalltalk (1972)	SNOBOL (1962)
Squeak (1996)	Tcl (1990)	Theta (1994)
VB.NET (2001)	VBScript (1996)	Visual Basic (1991)
VHDL (1987)	X10 (2004)	YAFL (1993)

Πίνακας 1.1: Γλώσσες προγραμματισμού και συλλογή απορριμμάτων. Όλες οι παραπάνω γλώσσες βασίζονται σε συλλογή απορριμμάτων.

την εμφάνιση διαρροών μνήμης, δεν μπορεί να εγγυηθεί την πλήρη εξάλειψή τους. Εάν ένα αντικείμενο δεν είναι πλέον προσβάσιμο από το υπόλοιπο πρόγραμμα, ο συλλέκτης θα ανακτήσει τη μνήμη που αυτό καταλαμβάνει. Εφόσον αυτός είναι ο μόνος τρόπος με τον οποίο ένα αντικείμενο μπορεί να διαγραφεί, δεν μπορούν να προκύψουν ξεκρέμαστοι δείκτες. Επιπλέον, αν η διαγραφή ενός αντικειμένου καθιστά και τα αντικείμενα παιδιά του μη προσβάσιμα, τότε ο συλλέκτης θα ανακτήσει και τη μνήμη που αυτά καταλαμβάνουν. Ωστόσο, ο συλλέκτης δεν μπορεί να βοηθήσει κάπως αν υπάρχει μια δυναμική δομή δεδομένων η οποία συνεχώς αυξάνεται (για παράδειγμα επειδή ο προγραμματιστής εσφαλμένα μόνο προσθέτει σε αυτή δεδομένα χωρίς ποτέ να αφαιρεί) ή αν αυτή είναι μεν προσβάσιμη από το υπόλοιπο πρόγραμμα αλλά αυτό δε θα τη χρησιμοποιήσει ποτέ στο μέλλον.

1.3 Συγκρίνοντας αλγορίθμους συλλογής απορριμμάτων

Η παρούσα εργασία εξετάζει ένα ευρύ φάσμα αλγορίθμων για συλλογή απορριμμάτων, κάθε ένας από τους οποίους έχει σχεδιασθεί λαμβάνοντας υπόψιν διαφορετικές απαιτήσεις όσον αφορά το φορτίο εργασίας, το υλικό και τις επιδόσεις. Δυστυχώς, δεν υπάρχει κάποιος αλγόριθμος που λειτουργεί βέλτιστα σε όλες τις περιπτώσεις. Οι Fitzgerald και Tarditi [52] μελετώντας 6 διαφορετικούς συλλέκτες και 20 benchmarks διαπίστωσαν πώς για κάθε συλλέκτη

υπάρχει τουλάχιστον ένα benchmark το οποίο θα εκτελούνταν τουλάχιστον 15% ταχύτερα με την παρουσία ενός πιο κατάλληλου συλλέκτη. Οι Singer κ.ά. [109] εφαρμόζουν τεχνικές μηχανικής μάθησης προκειμένου να προβλέψουν τη βέλτιστη διαμόρφωση ενός συλλέκτη για ένα συγκεκριμένο πρόγραμμα. Άλλοι ερευνητές, όπως ο Printezis [95] και οι Soman κ.ά. [110] έχουν εξερευνήσει την ιδέα να επιτρέπουν στις εικονικές μηχανές Java να αλλάζουν συλλέκτη καθώς εκτελούνται εάν πιστεύουν, με βάση τα χαρακτηριστικά του υπό εκτέλεση προγράμματος πώς αυτό θα ωφεληθεί από την παρουσία ενός διαφορετικού συλλέκτη. Στην ενότητα αυτή παρουσιάζουμε τις μετρικές βάση των οποίων συγκρίνονται οι διαφορετικοί αλγόριθμοι συλλογής απορριμμάτων. Τονίζουμε ωστόσο πώς οι μετρικές αυτές δεν είναι ανεξάρτητες μεταβλητές: η ρύθμιση μιας παραμέτρου με σκοπό την επίτευξη ενός συγκεκριμένου στόχου ενδέχεται να προκαλέσει άλλες αντιφατικές επιδράσεις.

1.3.1 Ασφάλεια

Ένας αλγόριθμος συλλογής απορριμμάτων πρέπει καταρχήν να είναι **ασφαλής**: δεν πρέπει ποτέ να ανακτήσει τη μνήμη αντικειμένων που είναι ζωντανά. Η εξασφάλιση της ασφάλειας είναι ιδιαίτερα δύσκολη σε ταυτόχρονους συλλέκτες. Επίσης η ασφάλεια της **συντηρητικής συλλογής απορριμμάτων**, όπου ο συλλέκτης δεν έχει καμία απολύτως βοήθεια από το μεταγλωττιστή και το σύστημα εκτέλεσης, είναι ευάλωτη σε συγκεκριμένες βελτιστοποιήσεις του μεταγλωττιστή οι οποίες έχουν ως αποτέλεσμα δεδομένα που δεν είναι δείκτες να φαίνεται ότι είναι.

1.3.2 Ρυθμαπόδοση

Κοινή απαίτηση των χρηστών είναι τα προγράμματά τους να τρέχουν ταχύτερα. Για να συμβαίνει αυτό, πρέπει μεταξύ άλλων, ο υπολογιστικός χρόνος που αφορά συλλογή απορριμμάτων να είναι ο ελάχιστος δυνατός. Στις περισσότερες περιπτώσεις βέβαια, ο χρήστης ενδιαφέρεται το σύνολο της εφαρμογής (τροποποιητής και συλλέκτης) να εκτελείται σε όσο το δυνατόν λιγότερο χρόνο. Στα περισσότερα καλώς σχεδιασμένα συστήματα, ξοδεύεται πολύ περισσότερος χρόνος CPU για την εκτέλεση του τροποποιητή από ότι για την εκτέλεση του συλλέκτη. Έτσι πολλές φορές αξίζει να θυσιαστεί η επίδοση του συλλέκτη χάριν υψηλότερης διεκπεραιωτικής ικανότητας του τροποποιητή.

1.3.3 Πληρότητα και προθυμία

Ιδανικά, η συλλογή απορριμμάτων οφείλει να είναι **πλήρης**: τελικώς όλα τα απορρίμματα στο σωρό θα συλλεγούν. Ωστόσο, αυτό δεν είναι πάντοτε δυνατό ή και επιθυμητό. Όπως θα δούμε και στο κεφάλαιο 5, οι απλοϊκοί αλγόριθμοι καταμέτρησης αναφορών για παράδειγμα αδυνατούν να συλλέξουν αυτοαναφορικές δομές δεδομένων. Επιπλέον, για λόγους επίδοσης, μπορεί να είναι επιθυμητό να μη συλλεγεί ολόκληρος ο σωρός σε κάθε κύκλο συλλογής. Οι γενεαλογικοί συλλέκτες για παράδειγμα, διαχωρίζουν τα αντικείμενα με βάση την ηλικία τους σε δύο ή περισσότερες περιοχές του σωρού τις οποίες ονομάζουν γενεές. Επικεντρώνοντας την προσοχή τους στη νεότερη γενεά, οι γενεαλογικοί συλλέκτες βελτιώνουν τόσο το συνολικό χρόνο εκτέλεσης του συλλέκτη όσο και το μέσο χρόνο παύσης του τροποποιητή για την κάθε ξεχωριστή ενεργοποίηση αυτού.

Οι ταυτόχρονοι συλλέκτες αναμειγνύουν την εκτέλεση τροποποιητή και συλλέκτη με στόχο την αποφυγή ή έστω τη μείωση της χρονικής διάρκειας των παύσεων του προγράμματος χρήστη. Μια συνέπεια αυτού είναι πως τα αντικείμενα που έχουν γίνει απορρίμματα αφού έχει ξεκινήσει ένας κύκλος συλλογής μπορεί να μην ελευθερωθούν παρά στο τέλος του επόμενου κύκλου. Τέτοια αντικείμενα ονομάζονται **αιωρούμενα απορρίμματα**. Σε ένα περιβάλλον λοιπόν όπου συλλέκτης και τροποποιητής εκτελούνται ταυτόχρονα η πληρότητα του συλλέκτη σχετίζεται με την **τελική** συλλογή των απορριμμάτων. Διαφορετικοί αλγόριθμοι μπορεί να διαφέρουν ως προς την **προθυμία** συλλογής τους, οδηγώντας σε συμβιβασμούς χώρου/χρόνου.

1.3.4 Χρόνος παύσης

Μια σημαντική απαίτηση συνήθως είναι η ελαχιστοποίηση της εισβολής του συλλέκτη στην εκτέλεση του προγράμματος. Πολλοί συλλέκτες εισάγουν παύσεις στην εκτέλεση ενός προγράμματος καθώς όλα τα νήματα-τροποποιητές είναι σταματημένα την ώρα που αυτοί συλλέγουν απορρίμματα. Ο όρος **παύση του κόσμου** αναφέρεται ακριβώς στην αναστολή της εκτέλεσης των νημάτων-τροποποιητών κατά τη διάρκεια της εκτέλεσης των νημάτων-συλλεκτών. Είναι εμφανώς επιθυμητό οι **χρόνοι παύσης** να ελαχιστοποιηθούν. Αυτό μπορεί να είναι ιδιαίτερα σημαντικό για διαδραστικές εφαρμογές ή διακομιστές οι οποίοι χειρίζονται συναλλαγές (όπου συνήθως οι ενέργειες πρέπει να διεκπεραιώνονται εντός πολύ στενών χρονικών ορίων). Ωστόσο, οι μηχανισμοί που προσπαθούν να μειώσουν τους χρόνους παύσης έχουν παρενέργειες, όπως θα δούμε στη συνέχεια. Για παράδειγμα οι γενεαλογικοί συλλέκτες προσπαθούν να μειώσουν τους χρόνους παύσης συλλέγοντας συχνά και γρήγορα μία μικρή περιοχή του σωρού που φιλοξενεί νέα αντικείμενα, ενώ συλλέγουν μεγαλύτερες περιοχές του σωρού όπου ζουν παλαιά αντικείμενα σπανιότερα και μόνο αν αυτό χρειασθεί. Ωστόσο, επειδή πρέπει να καταγράφονται οι δείκτες που περνούν τα όρια των περιοχών, η γενεαλογική συλλογή απορριμμάτων επιβάλλει ένα μικρό πρόστιμο στις λειτουργίες εγγραφής δεικτών του τροποποιητή.

Οι παράλληλοι συλλέκτες σταματούν τα νήματα-τροποποιητές και μειώνουν το χρόνο παύσης απασχολώντας πολλά νήματα-συλλέκτες. Οι αυξητικοί και ταυτόχρονοι συλλέκτες επιχειρούν να μειώσουν τους χρόνους παύσης ακόμη περισσότερο εκτελώντας ένα μικρό κβάντο εργασιών συλλογής είτε ανάμεσα στις λειτουργίες του τροποποιητή είτε παράλληλα με αυτές. Και αυτές οι τεχνικές υπαγορεύουν ένα μικρό κόστος στον τροποποιητή προκειμένου να επιτευχθεί ο σωστός συγχρονισμός αυτού με το συλλέκτη.

Η μέγιστη τιμή ή ο μέσος χρόνος παύσης ωστόσο δεν μπορούν να χρησιμοποιηθούν ώστε να ελεγχθεί η πρόοδος του τροποποιητή. Η κατανομή των χρόνων παύσης είναι επίσης μια μετρική που ενδιαφέρει. Στη βιβλιογραφία συναντώνται διάφοροι τρόποι μέτρησης αυτής, όπως η **ελάχιστη χρησιμοποίηση τροποποιητή**, η οποία εισήχθη από τους Cheng και Blelloch [35], και η **φραγμένη χρησιμοποίηση τροποποιητή**, η οποία εισήχθη από τους Sachindran κ.ά. [101]. Και οι δύο μετρικές προσπαθούν να μετρήσουν με ακρίβεια το ελάχιστο κλάσμα του χρόνου εκτέλεσης που δαπανάται για την εκτέλεση του τροποποιητή.

1.3.5 Επιβάρυνση σε χώρο

Ο στόχος ενός διαχειριστή μνήμης είναι η ασφαλής και αποδοτική χρήση του χώρου. Διαφορετικοί διαχειριστές, τόσο ρητοί όσο και αυτόματοι επιβάλλουν διαφορετικά **χωρικά κόστη**. Οι συλλέκτες καταμέτρησης αναφορών για παράδειγμα χρειάζονται χώρο στη μνήμη κάθε

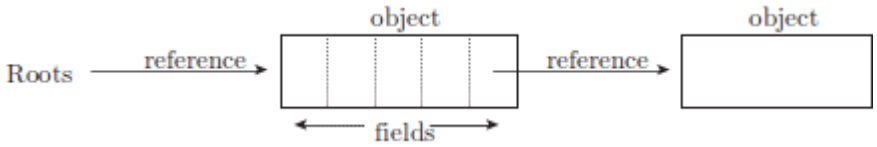
αντικειμένου όπου θα αποθηκεύεται ο μετρητής αναφορών προς αυτό. Οι συλλέκτες σήμανσης και εκκαθάρισης αποθηκεύουν ένα bit/ byte στην επικεφαλίδα ενός αντικειμένου, ενώ κάποιοι συλλέκτες σήμανσης και συμπύκνωσης αποθηκεύουν τη νέα διεύθυνση ενός αντικειμένου σε κάποιο πεδίο στο παλαιό αντίγραφο αυτού. Οι συλλέκτες αντιγραφής διαχωρίζουν το σωρό σε δύο ημιχώρους, μόνο ένας εκ των οποίων είναι διαθέσιμος στον τροποποιητή κάθε χρονική στιγμή: ο άλλος χρησιμεύει ως αντίγραφο ρεζέρβα, όπου ο συλλέκτης θα αντιγράψει τα ζωντανά αντικείμενα στη διάρκεια του επόμενου κύκλου συλλογής. Οι συλλέκτες ενδέχεται επίσης να χρησιμοποιούν βοηθητικές δομές δεδομένων. Οι ταυτόχρονοι συλλέκτες, ή οι γενεαλογικοί συλλέκτες απαιτούν τη χρήση συνόλων ανάμνησης ώστε να καταγράφουν τις τιμές ποιων δεικτών έχει μεταβάλλει στο ενδιάμεσο ο τροποποιητής ή τις διευθύνσεις των διαγενεαλογικών δεικτών αντίστοιχα.

1.3.6 Βελτιστοποιήσεις για ειδικές γλώσσες

Οι αλγόριθμοι συλλογής απορριμμάτων πολλές φορές χαρακτηρίζονται από την εφαρμοσιμότητά τους σε γλώσσες προγραμματισμού διαφορετικών οικογενειών. Οι συναρτησιακές γλώσσες για παράδειγμα προσφέρονται για βελτιστοποιήσεις που σχετίζονται με την αυτόματη διαχείριση μνήμης. Μερικές γλώσσες, όπως η ML διακρίνουν τα δεδομένα σε τροποποιήσιμα και μη τροποποιήσιμα. Αγνές συναρτησιακές γλώσσες όπως η Haskell, επεκτείνουν την παραπάνω ιδέα και δεν αφήνουν τον προγραμματιστή να τροποποιήσει καμία τιμή (τα προγράμματα χαρακτηρίζονται από διαφάνεια αναφορών). Εσωτερικά ωστόσο, ενημερώνουν δομές δεδομένων τουλάχιστον μια φορά, κάτι που δίνει τη δυνατότητα σε γενεαλογικούς συλλέκτες απορριμμάτων να προάγουν πρόθυμα πλήρως υπολογισμένες δομές δεδομένων. Έχουν επίσης προταθεί **πλήρεις** μηχανισμοί για τη συλλογή κύκλων απορριμμάτων με καταμέτρηση αναφορών. Οι δηλωτικές γλώσσες προγραμματισμού ενδεχομένως επιτρέπουν τη χρήση μηχανισμών για αποδοτική διαχείριση του σωρού. Όλα τα δεδομένα που έχουν δημιουργηθεί μετά από ένα σημείο επιλογής σε μία γλώσσα λογικού προγραμματισμού καθίστανται μη προσβάσιμα τη χρονική στιγμή κατά την οποία το πρόγραμμα οπισθοχωρεί στο εν λόγω σημείο. Εάν ο διαχειριστής μνήμης διατάσσει τα αντικείμενα στο σωρό με βάση τη σειρά δημιουργίας τους, η συνολική μνήμη που έχει εκχωρηθεί από το σημείο επιλογής και έπειτα μπορεί να επιστραφεί σε σταθερό χρόνο. Αντίστροφα, διαφορετικά πρότυπα γλωσσών μπορεί να ορίζουν συγκεκριμένες απαιτήσεις από το συλλέκτη. Οι πιο γνωστές αφορούν την ικανότητα του συλλέκτη να χειρίζεται διάφορα είδη δεικτών ή να προκαλεί την οριστικοποίηση νεκρών αντικειμένων.

1.3.7 Κλιμακωσιμότητα και μεταφερσιμότητα

Η ολοένα και περισσότερο αυξανόμενη διάδοση πολυπύρηνων επεξεργαστών σε επιτραπέζιους και φορητούς υπολογιστές πέραν των διακομιστών μεγάλης κλίμακας καθιστά ιδιαίτερα σημαντική την ανάγκη η συλλογή απορριμμάτων να επωφεληθεί από την παρουσία του παράλληλου υλικού. Έτσι συχνά ένας αλγόριθμος συλλογής απορριμμάτων χαρακτηρίζεται ως προς το πώς **κλιμακώνει**. Η λειτουργία ενός αριθμού αλγορίθμων συλλογής απορριμμάτων εξαρτάται από την υποστήριξη που παρέχει το λειτουργικό σύστημα και η αρχιτεκτονική. Αυτό έχει ως αποτέλεσμα η **μεταφερσιμότητα** των αλγορίθμων να μην είναι απαραίτητα εγγυημένη.



Σχήμα 1.2: Ρίζες, αναφορές, αντικείμενα, πεδία.

1.4 Ορολογία

Στην ενότητα αυτή εξηγούμε την ορολογία που χρησιμοποιείται στην εργασία αυτή αλλά και στη βιβλιογραφία.

Ο **σωρός** είναι είτε ένας συνεχόμενος πίνακας από λέξεις μνήμης ή ένα σύνολο από μη συνεχόμενα μπλοκ συνεχόμενων λέξεων. Ένα **αντικείμενο** είναι ένα σύνολο από συνεχόμενες λέξεις μνήμης. Οι επιμέρους λέξεις ενός αντικειμένου αναφέρονται και ως **πεδία**. Σε κάθε πεδίο ενός αντικειμένου αποθηκεύεται ένα βαθμωτό μέγεθος (π.χ. ένας ακέραιος) ή μια **αναφορά**. Μια αναφορά είναι είτε ένας δείκτης προς ένα αντικείμενο του σωρού είτε η ειδική τιμή **null**. Ένα αντικείμενο συνήθως έχει ένα ειδικό πεδίο, την **επικεφαλίδα**, όπου αποθηκεύονται μεταδεδομένα χρήσιμα για το σύστημα εκτέλεσης. Ο σωρός συχνά αναφέρεται και ως **γράφος αντικειμένων**. Ο γράφος είναι κατευθυνόμενος, με κόμβους τα αντικείμενα του σωρού και ακμές τους δείκτες στα πεδία αυτών. Μια ακμή αναπαριστά μια αναφορά από έναν κόμβο προέλευσης ή μία **ρίζα** προς έναν κόμβο προορισμού.

Σύμφωνα με τον Dijkstra [44], [45], ένα πρόγραμμα με συλλογή απορριμμάτων διαχωρίζεται στα εξής δύο ημιανεξάρτητα τμήματα:

- Ο **τροποποιητής** εκτελεί τον κώδικα της εφαρμογής, ο οποίος δημιουργεί αντικείμενα και τροποποιεί το γράφο αντικειμένων μεταβάλλοντας αναφορές ώστε αυτές να αναφέρονται σε διαφορετικά αντικείμενα προορισμού. Μια αναφορά μπορεί να περιέχεται στο πεδίο ενός αντικειμένου ή στις ρίζες του προγράμματος (στατικές μεταβλητές, καταχωρητές, στοίβα). Καθώς ο τροποποιητής τροποποιεί τις αναφορές, κάθε αντικείμενο μπορεί να αποσυνδεθεί από τις ρίζες, δηλαδή να μην είναι πλέον προσβάσιμο μέσω μιας ακολουθίας αναφορών από αυτές.
- Ο **συλλέκτης** εκτελεί κώδικα συλλογής απορριμμάτων, ο οποίος ανακαλύπτει μη προσβάσιμα αντικείμενα και ανακτά τη μνήμη που αυτά καταλαμβάνουν.

Ο τροποποιητής μπορεί να είναι είτε **μονονηματικός** είτε **πολυνηματικός**. Το ίδιο ισχύει και για το συλλέκτη.

Εκτός από το σωρό, θεωρούμε πώς υπάρχει ένα (πεπερασμένο) σύνολο ριζών, το οποίο αναπαριστά δείκτες που είναι **άμεσα** προσβάσιμοι από τον τροποποιητή. Κατ' επέκταση, τα αντικείμενα του σωρού στα οποία αναφέρονται οι ρίζες ονομάζονται **αντικείμενα-ρίζες**. Καθώς ο τροποποιητής εκτελείται, οι ρίζες και άρα και ο γράφος αντικειμένων μεταβάλλονται.

Αναφερόμαστε σε έναν κόμβο του σωρού N χρησιμοποιώντας τη διεύθυνσή του στη μνήμη (η οποία δεν αντιστοιχεί απαραίτητα στην πρώτη λέξη του αντικειμένου αλλά ίσως σε κάποιο προκαθορισμένο σημείο ανάμεσα στα δεδομένα και τα μεταδεδομένα του αντικειμένου). Δοθέντος ενός αντικειμένου N, αναφερόμαστε στο i-στό πεδίο του, στο οποίο μπορεί να αποθηκεύεται μία βαθμωτή μεταβλητή ή ένας δείκτης, ως $N[i]$, αντιμετωπίζοντας το αντικείμενο

ως έναν πίνακα από πεδία. Συμβολίζουμε με $|N|$ το πλήθος των πεδίων ενός αντικειμένου, ενώ η αποδιευθυνσιοδότηση ενός μη μηδενικού δείκτη p συμβολίζεται με $*p$. Επιπλέον η διεύθυνση του i-στού πεδίου του αντικειμένου N συμβολίζεται με $\&N[i]$. Επομένως το σύνολο των διευθύνσεων των πεδίων δεικτών ενός αντικειμένου N, συμβολιζόμενο ως $Pointers(N)$ ορίζεται αυστηρά ως:

$$Pointers(N) = \{a | a = \&N[i], \forall i : 0 \leq i < |N| \; where \; N[i] \; is \; a \; pointer\} \tag{1.1}$$

Τέλος, θεωρούμε το σύνολο $Roots$ των ριζών ως ένα ψευδοαντικείμενο και αναφερόμαστε στην i-στή ρίζα ως $Roots[i]$.

Λέμε πώς ένα αντικείμενο είναι **ζωντανό** εάν αυτό πρόκειται να χρησιμοποιηθεί κάποια χρονική στιγμή στο μέλλον από τον τροποποιητή. Ένας συλλέκτης απορριμμάτων είναι ορθός αν και μόνο αν ποτέ δε συλλέγει ζωντανά αντικείμενα. Δυστυχώς ωστόσο, η ζωντάνια είναι μια μη-αποκρίσιμη ιδιότητα των προγραμμάτων: δεν υπάρχει κάποιος τρόπος ώστε να απαντηθεί αν ένα τυχαίο πρόγραμμα θα χρησιμοποιήσει ή όχι στο μέλλον ένα αντικείμενο του σωρού. Το γεγονός πώς ένα πρόγραμμα διαθέτει ένα δείκτη προς ένα αντικείμενο δε σημαίνει απαραίτητα και πώς αυτό θα προσπελαστεί από το πρόγραμμα κάποια στιγμή στο μέλλον. Προσεγγίζουμε τη ζωντάνια με την αποκρίσιμη ιδιότητα της **προσβασιμότητας μέσω δεικτών**. Ένα αντικείμενο N είναι προσβάσιμο από ένα αντικείμενο M, εάν το N μπορεί να προσπελασθεί ακολουθώντας μία ακολουθία δεικτών από κάποιο πεδίο f του M. Επομένως, ένα αντικείμενο είναι προσβάσιμο από τις ρίζες ενός τροποποιητή αν και μόνο αν υπάρχει μια ακολουθία δεικτών που ξεκινάει από κάποια ρίζα και καταλήγει σε αυτό.

Πιο αυστηρά, ορίζουμε τη δυαδική σχέση $\rightarrow_f$ ως εξής. Για κάθε δύο κόμβους M, N, ισχύει πως $M \rightarrow_f N$ αν και μόνο εάν υπάρχει ένα πεδίο $f = \&M[i]$, τέτοιο ώστε $f \in Pointers(M)$ και $*f = N$. Παρόμοια, $Roots \rightarrow_f N$ αν και μόνο αν υπάρχει ένα πεδίο f τέτοιο ώστε $f \in Roots$ και $*f = N$. Λέμε πως ο κόμβος N είναι **άμεσα προσβάσιμος** από τον κόμβο M, και συμβολίζουμε με $M \rightarrow N$ αν υπάρχει κάποιο πεδίο f το οποίο ανήκει στο σύνολο $Pointers(M)$ και είναι τέτοιο ώστε $M \rightarrow_f N$. Με τη βοήθεια των παραπάνω ορισμών, το σύνολο των προσβάσιμων αντικειμένων στο σωρό ορίζεται ως το μεταβατικό κλείσιμο από το σύνολο $Roots$ ως προς την πράξη $\rightarrow$, δηλαδή το ελάχιστο σύνολο:

$$reachable = \{N \in Nodes \mid (\exists r \in Roots : r \rightarrow N) \vee (\exists M \in reachable : M \rightarrow N)\} \tag{1.2}$$

Ένα αντικείμενο του σωρού που είναι μη προσβάσιμο από τις ρίζες δε θα προσπελασθεί ποτέ ξανά από έναν ορθό τροποποιητή. Αντίθετα, ένα προσβάσιμο αντικείμενο μπορεί να προσπελασθεί κάποια στιγμή στο μέλλον. Με τον τρόπο αυτό η ζωντάνια των αντικειμένων όσον αφορά τη συλλογή απορριμμάτων, ορίζεται μέσω της προσβασιμότητας μέσω δεικτών. Τα μη προσβάσιμα αντικείμενα είναι σίγουρα νεκρά και επομένως μπορούν με ασφάλεια να ελευθερωθούν. Αντίθετα, τα προσβάσιμα αντικείμενα μπορεί να είναι ζωντανά και επομένως πρέπει να διατηρηθούν. Παρότι δεν είναι ακριβώς ορθό, θα χρησιμοποιούμε τους όρους **ζωντανό** και **νεκρό** αδιακρίτως με τους όρους **προσβάσιμο** και **μη προσβάσιμο** αντίστοιχα. Τέλος, ο όρος **απόρριμμα** θα χρησιμοποιείται ως συνώνυμος του όρου μη προσβάσιμο.

Ο **εκχωρητής**, ο οποίος μπορεί να θεωρηθεί λειτουργικά κάθετος ως προς το συλλέκτη, υποστηρίζει δύο λειτουργίες: ALLOCATE, η οποία δεσμεύει τη μνήμη που θα καταλάβει ένα αντικείμενο και DEALLOCATE, η οποία επιστρέφει τη μνήμη στον εκχωρητή ώστε αυτός να την επαναχρησιμοποιήσει.

Ορισμένες από τις λειτουργίες που εκτελούν τα νήματα-τροποποιητές ενδιαφέρουν το συλλέκτη: NEW, READ και WRITE. Συγκεκριμένοι διαχειριστές μνήμης μπορεί να επεκτείνουν αυτές τις βασικές λειτουργίες και να τις μετατρέπουν σε **φράγματα**: πράξεις που επιτρέπουν τη σύγχρονη ή ασύγχρονη επικοινωνία με το συλλέκτη. Διακρίνουμε τα φράγματα σε **φράγματα εγγραφής** και **φράγματα ανάγνωσης**.

Αλγόριθμος 1.1 Λειτουργίες τροποποιητή

```
1: function NEW()
2:     return ALLOCATE()

3: function READ(src, i)
4:     return src[i]

5: procedure WRITE(src, i, ref)
6:     src[i] ← ref
```

Στο πλαίσιο της ταυτόχρονης εκτέλεσης νημάτων-τροποποιητών και νημάτων- συλλεκτών, όλοι οι αλγόριθμοι συλλογής απορριμμάτων που εξετάζουμε απαιτούν ορισμένες ακολουθίες εντολών να εκτελούνται **ατομικά**. Για λόγους απλοποίησης αγνοούμε τον εκάστοτε μηχανισμό που εξασφαλίζει την ατομική εκτέλεση τμημάτων κώδικα και απλώς σημειώνουμε τα τελευταία με τη λέξη κλειδί **atomic**.

1.5 Οργάνωση της εργασίας

Στο κεφάλαιο αυτό εξηγούνται οι λόγοι για τους οποίους είναι επιθυμητή η αυτόματη διαχείριση μνήμης και ορίζονται οι έννοιες του εκχωρητή μνήμης, του συλλέκτη απορριμμάτων και του τροποποιητή. Επίσης παρουσιάζονται τα κριτήρια με τα οποία μπορούν να συγκριθούν οι διαφορετικές στρατηγικές συλλογής απορριμμάτων. Το υπόλοιπο της εργασίας οργανώνεται σε δύο μέρη.

Το πρώτο μέρος περιλαμβάνει 5 κεφάλαια. Στα πρώτα 4 εξετάζονται με λεπτομέρεια οι θεμελιώδεις αλγόριθμοι συλλογής απορριμμάτων. Πιο συγκεκριμένα, το κεφάλαιο 2 πραγματεύεται τη συλλογή απορριμμάτων με σήμανση και εκκαθάριση, το κεφάλαιο 3 τη συλλογή απορριμμάτων με σήμανση και συμπύκνωση, το κεφάλαιο 4 τη συλλογή απορριμμάτων με αντιγραφή και το κεφάλαιο 5 τη συλλογή απορριμμάτων με καταμέτρηση αναφορών. Τέλος, το κεφάλαιο 6 συγκρίνει τους παραπάνω θεμελιώδεις αλγορίθμους συλλογής απορριμμάτων.

Το δεύτερο μέρος της εξετάζει προηγμένους αλγορίθμους συλλογής απορριμμάτων και αποτελείται από 4 κεφάλαια. Το κεφάλαιο 7 πραγματεύεται λεπτομερώς τη γενεαλογική συλλογή απορριμμάτων, το κεφάλαιο 8 την παράλληλη συλλογή απορριμμάτων και το κεφάλαιο 9 την ταυτόχρονη συλλογή απορριμμάτων. Τέλος, στο κεφάλαιο 10 δίνεται μια σύντομη εισαγωγή στη συλλογή απορριμμάτων πραγματικού-χρόνου.

Μέρος Ι

Θεμελιώδεις αλγόριθμοι συλλογής απορριμμάτων

Κεφάλαιο 2

Συλλογή απορριμμάτων με σήμανση και εκκαθάριση

Ο ιδανικός στόχος ενός συλλέκτη απορριμμάτων είναι η ανάκτηση του χώρου που καταλαμβάνει κάθε αντικείμενο το οποίο δεν πρόκειται να ξαναχρησιμοποιηθεί από το πρόγραμμα. Κάθε σύστημα αυτόματης διαχείρισης μνήμης είναι επιφορτισμένο με τις εξής υποχρεώσεις:

1. να εκχωρεί χώρο για νέα αντικείμενα,
2. να αναγνωρίζει τα ζωντανά αντικείμενα,
3. να ανακτά το χώρο που καταλαμβάνουν τα νεκρά αντικείμενα.

Οι υποχρεώσεις αυτές δεν είναι ανεξάρτητες. Πιο συγκεκριμένα, ο τρόπος ανάκτησης μνήμης επηρεάζει την εκχώρηση μνήμης. Όπως είδαμε στο κεφάλαιο 1, το πρόβλημα της απόφασης κατά πόσο ένα αντικείμενο είναι πραγματικά ζωντανό είναι μη-αποκρίσιμο. Η προσβασιμότητα μέσω δεικτών δίνει μία υπερ-προσέγγιση του συνόλου των ζωντανών αντικειμένων. Ένα αντικείμενο θεωρείται ζωντανό αν και μόνο αν είναι προσβάσιμο δια μέσου μιας αλυσίδας αναφορών ξεκινώντας από ένα σύνολο (γνωστών) ριζών. Από την άλλη πλευρά, ένα αντικείμενο θεωρείται νεκρό και η μνήμη που αυτό καταλαμβάνει μπορεί να ανακτηθεί, αν δεν είναι προσβάσιμο από καμία τέτοια αλυσίδα αναφορών. Παρότι ένα ζωντανό αντικείμενο ενδέχεται να μην προσπελασθεί ξανά, ένα νεκρό αντικείμενο είναι σίγουρα νεκρό.

Ο πρώτος αλγόριθμος που εξετάζουμε είναι αυτός της **συλλογής με σήμανση και εκκαθάριση** και διατυπώθηκε από τον Collins [81]. Πρόκειται για μια απευθείας υλοποίηση της αναδρομικής ιδιότητας της προσβασιμότητας μέσω δεικτών. Ο αλγόριθμος εκτελείται σε δύο φάσεις: αρχικά ο συλλέκτης διασχίζει το γράφο των αντικειμένων, εκκινώντας από τις ρίζες (καταχωρητές, στοίβα, καθολικές μεταβλητές), δια μέσου των οποίων το πρόγραμμα έχει άμεση πρόσβαση σε αντικείμενα και συνεχίζει ακολουθώντας μεταβλητές-δείκτες και σημαίνοντας κάθε αντικείμενο που συναντά. Η πρώτη αυτή φάση είναι γνωστή και ως **εξιχνίαση**. Στη δεύτερη φάση, αυτήν της **εκκαθάρισης**, ο συλλέκτης εξετάζει κάθε αντικείμενο στο σωρό: κάθε μη-σημασμένο αντικείμενο θεωρείται απόρριμμα και η μνήμη που αυτό καταλαμβάνει απελευθερώνεται.

Ακριβώς επειδή δεν εντοπίζει τα απορρίμματα καθεαυτά, αλλά αντίθετα εντοπίζει τα αντικείμενα που δεν είναι απορρίμματα και έπειτα συμπεραίνει πως τα υπόλοιπα είναι απορρίμματα,

Αλγόριθμος 2.1 Συλλογή με σήμανση και εκκαθάριση: εκχώρηση

```
 1: procedure New()
 2:     ref ← allocate()
 3:     if ref = null then                                  ▷ heap is full
 4:         collect()
 5:         ref ← allocate()
 6:         if ref = null then                              ▷ heap is still full
 7:             error "Out of memory!"
 8:     return ref

 9: procedure collect()
10:     atomic
11:     markFromRoots()
12:     sweep(start, end)
```

ο αλγόριθμος με σήμανση και εκκαθάριση χαρακτηρίζεται πολλές φορές και ως **έμμεσος**. Κάθε φορά που εκτελείται, εντοπίζει από την αρχή τα προσβάσιμα αντικείμενα. Αντίθετα, ένας **άμεσος** αλγόριθμος συλλογής αποφαίνεται κατά πόσο ένα αντικείμενο είναι ζωντανό ή μη από το αντικείμενο καθεαυτό, χωρίς να είναι απαραίτητη η διάσχιση του γράφου των αντικειμένων. Παράδειγμα ενός άμεσου αλγορίθμου αποτελεί ο αλγόριθμος καταμέτρησης αναφορών.

2.1 Ο αλγόριθμος σήμανσης και εκκαθάρισης

Από την πλευρά του συλλέκτη, τα νήματα-τροποποιητές εκτελούν 3 ενδιαφέρουσες ενέργειες τις New, Read και Write, τις οποίες κάθε αλγόριθμος συλλογής ορίζει με το δικό του τρόπο. Η διαπροσωπεία του συλλέκτη με τον τροποποιητή είναι ιδιαίτερα απλή. Αν δεν είναι δυνατή η εκχώρηση μνήμης για ένα αντικείμενο σε ένα νήμα-τροποποιητή, καλείται ο συλλέκτης και το αίτημα εκχώρησης επαναλαμβάνεται (αλγόριθμος 2.1). Η παρουσία της λέξης κλειδί **atomic** τονίζει πώς η εκτέλεση του συλλέκτη πραγματοποιείται με παύση του κόσμου, δηλαδή χωρίς να εκτελούνται ταυτόχρονα νήματα- τροποποιητές. Αν και πάλι το αίτημα δεν μπορεί ικανοποιηθεί, η διαθέσιμη μνήμη στο σωρό έχει εξαντληθεί. Ενώ τις περισσότερες φορές αυτό αποτελεί σφάλμα εκτέλεσης, σε ορισμένες γλώσσες προγραμματισμού η συνάρτηση New δύναται να εγείρει μία εξαίρεση την οποία ο προγραμματιστής μπορεί να πιάσει και να χειριστεί.

Πριν ξεκινήσει την εξιχνίαση του γράφου αντικειμένων, ο συλλέκτης πρέπει να αρχικοποιήσει τη λίστα εργασιών (μεταβλητή *worklist*) με τα αντικείμενα εκκίνησης της εξερεύνησης (διαδικασία markFromRoots στον αλγόριθμο 2.2). Κάθε αντικείμενο-ρίζα σημαίνεται και τοποθετείται στη λίστα εργασιών. Η σήμανση κωδικοποιείται συνήθως με την τιμή ενός bit/byte το οποίο αποθηκεύεται είτε στην κεφαλίδα του αντικειμένου είτε σε ένα ξεχωριστό bitmap. Αν ένα αντικείμενο δεν περιέχει δείκτες, απλώς σημαίνεται και δεν εισάγεται στη λίστα εργασιών, αφού αυτό δεν είναι απαραίτητο καθώς δεν έχει παιδιά. Για να ελαχιστοποιηθεί το μέγεθος της λίστας εργασιών, η markFromRoots καλεί αμέσως τη διαδικασία mark. Εναλλακτικά, μπορεί να είναι επιθυμητή η όσο το δυνατόν γρηγορότερη σάρωση των ριζών κάθε νήματος-τροποποιητή. Για παράδειγμα ένας ταυτόχρονος συλλέκτης μπορεί να διακόπτει

Αλγόριθμος 2.2 Συλλογή με σήμανση και εκκαθάριση: σήμανση

```
 1: procedure markFromRoots()
 2:     initialize(worklist)
 3:     for all fld ∈ Roots do
 4:         ref ← *fld
 5:         if ref ≠ null and not isMarked(ref) then
 6:             setMarked(ref)
 7:             add(worklist, ref)
 8:             mark()

 9: procedure initialize()
10:     worklist ← empty

11: procedure mark()
12:     while not isEmpty(worklist) do
13:         ref ← remove(worklist)
14:         for all fld ∈ Pointers(ref) do
15:             child ← *fld
16:             if child ≠ null and not isMarked(child) then
17:                 setMarked(child)
18:                 add(worklist, child)
```

την εκτέλεση ενός νήματος-τροποποιητή για πολύ μικρό χρονικό διάστημα ώστε να σαρώσει τη στοίβα του και να συνεχίσει με την εξερεύνηση του γράφου ενώ το τελευταίο εκτελείται.

Η λίστα εργασιών μπορεί να υλοποιηθεί ως μία στοίβα, οδηγώντας έτσι σε μία κατά βάθος διάσχιση του γράφου των αντικειμένων. Αν μάλιστα τα bit σήμανσης βρίσκονται αποθηκευμένα μαζί με τα αντικείμενα (π.χ. στην κεφαλίδα των τελευταίων), τότε τα επόμενα αντικείμενα που πρόκειται να εξεταστούν, αυτά που είναι ήδη σημασμένα, βρίσκονται με μεγάλη πιθανότητα στην κρυφή μνήμη. Η τοπικότητα των αναφορών μπορεί να επιδράσει σε μεγάλο βαθμό στην επίδοση του συλλέκτη.

Η σήμανση του γράφου των αντικειμένων υλοποιείται από έναν απλό βρόχο **while**. Ο δείκτης προς ένα αντικείμενο εξάγεται από τη λίστα εργασιών και οι δείκτες του αντικειμένου αυτού, εφόσον δε δείχνουν σε σημασμένα αντικείμενα εισάγονται σε αυτή, μέχρις ότου η λίστα εργασιών αδειάσει. Σε αυτήν την εκδοχή της mark κάθε αντικείμενο προς το οποίο υπάρχει αναφορά στη λίστα εργασιών είναι σημασμένο. Ο τερματισμός της διαδικασίας mark, επομένως και της markFromRoots εξασφαλίζεται από το γεγονός πως ένας δείκτης σε ήδη σημασμένο αντικείμενο δεν εισάγεται στη λίστα εργασιών. Κατά την επιστροφή της markFromRoots, κάθε αντικείμενο που είναι προσβάσιμο από τις ρίζες του προγράμματος έχει σημανθεί. Κάθε μη σημασμένο αντικείμενο είναι απόρριμμα και η μνήμη που αυτό καταλαμβάνει μπορεί να απελευθερωθεί.

Η φάσης της εκκαθάρισης υλοποιείται από τη διαδικασία sweep. Σαρώνεται ο σωρός και είτε αφαιρείται η σήμανση από ένα σημασμένο αντικείμενο, είτε ελευθερώνεται η μνήμη που είχε εκχωρηθεί σε ένα μη σημασμένο αντικείμενο. Ας παρατηρήσουμε πως το κόστος αφαίρεσης της σήμανσης μπορεί να αποφευχθεί αν η σημασία των bit/byte σήμανσης εναλλάσσεται μεταξύ δύο διαδοχικών συλλογών.

Η συλλογή με σήμανση και εκκαθάριση επιβάλλει ορισμένους περιορισμούς όσον αφορά την

Αλγόριθμος 2.3 Συλλογή με σήμανση και εκκαθάριση: εκκαθάριση

```
1: procedure SWEEP(start, end)
2:     scan ← start
3:     while scan < end do
4:         if ISMARKED(scan) then
5:             UNSETMARKED(scan)
6:         else
7:             FREE(scan)
8:         scan ← NEXTOBJECT(scan)
```

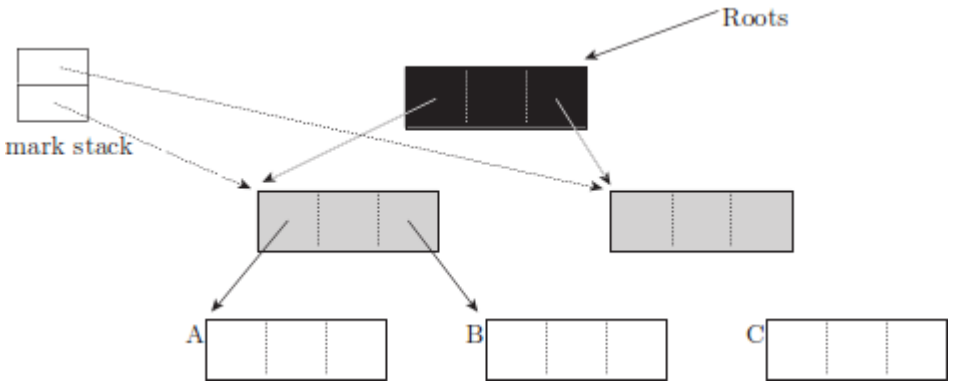
οργάνωση του σωρού. Πρώτον, δε μετακινεί αντικείμενα. Αυτό έχει ως αποτέλεσμα ο διαχειριστής μνήμης να πρέπει να είναι ιδιαίτερα προσεκτικός προκειμένου ο σωρός να μην κατακερματισθεί σε τέτοιο βαθμό ώστε ο εκχωρητής να μην μπορεί πλέον να ικανοποιεί αιτήματα, καθώς αυτό θα έχει ως αποτέλεσμα είτε τη δραματική αύξηση της συχνότητας κλήσης του συλλέκτη είτε, στη χειρότερη περίπτωση την παντελή αδυναμία εκχώρησης μνήμης. Επιπλέον, ο εκκαθαριστής πρέπει να μπορεί να προσδιορίσει τη θέση κάθε κόμβου στο σωρό ακόμη και υπό την παρουσία παραγεμίσματος που εισάγεται από τις απαιτήσεις ευθυγράμμισης που έχουν οι περισσότερες σύγχρονες αρχιτεκτονικές. Κατά συνέπεια, το έργο της συνάρτησης NEXTOBJECT ενδέχεται να είναι σημαντικά πιο πολύπλοκο από μια απλή πρόσθεση της διεύθυνσης ενός αντικειμένου με το μέγεθος αυτού.

2.2 Η τριχρωματική αφαίρεση

Είναι ιδιαίτερα βολική η ύπαρξη ενός τρόπου περιγραφής της κατάστασης των αντικειμένων κατά τη διάρκεια της συλλογής (αν έχουν σημανθεί ή όχι, αν βρίσκονται στη λίστα εργασιών κλπ). Η **τριχρωματική αφαίρεση** είναι ένας χρήσιμος χαρακτηρισμός των συλλεκτών εξιχνίασης ο οποίος επιτρέπει την επιχειρηματολογία σχετικά με την ορθότητα ενός συλλέκτη σε όρους αναλλοίωτων, την ισχύ των οποίων πρέπει αυτός να εξασφαλίζει.

Σύμφωνα με την τριχρωματική αφαίρεση, η συλλογή εξιχνίασης διαιρεί το γράφο αντικειμένων σε **μαύρα** (ζωντανά) και **λευκά** (πιθανώς νεκρά) αντικείμενα. Αρχικά, κάθε αντικείμενο είναι λευκό, ενώ την πρώτη φορά που ανακαλύπτεται κατά την εξερεύνηση χρωματίζεται **γκρι**. Χρωματίζεται δε μαύρο όταν έχει σαρωθεί και έχουν εντοπισθεί τα παιδιά του. Διαισθητικά, ένα αντικείμενο είναι μαύρο αν ο συλλέκτης έχει ολοκληρώσει την επεξεργασία του και γκρι αν ο συλλέκτης το έχει ανακαλύψει αλλά δεν έχει τελειώσει ακόμη την επεξεργασία του (ή χρειάζεται να το επεξεργασθεί ξανά). Σε αναλογία με το χρωματισμό των αντικειμένων, ένα πεδίο δείκτης μπορεί επίσης να χαρακτηρισθεί ως προς το χρώμα του: γκρι όταν ο συλλέκτης το ανακαλύπτει για πρώτη φορά και μαύρο όταν ο συλλέκτης έχει τελειώσει την εξερεύνηση του υπογράφου που το έχει ως ρίζα. Σύμφωνα με τον Pirinen [93], η αναλογία αυτή επιτρέπει την αντιμετώπιση των ριζών του τροποποιητή σαν αυτός να ήταν αντικείμενο. Ένας γκρι τροποποιητής έχει ρίζες οι οποίες δεν έχουν σαρωθεί ακόμα από το συλλέκτη. Ένας μαύρος τροποποιητής έχει ρίζες οι οποίες έχουν σαρωθεί από το συλλέκτη και δε χρειάζεται να εξετασθούν ξανά. Η εξιχνίαση προοδεύει στο σωρό με την μετακίνηση του **μετώπου κύματος** (των γκρι αντικειμένων) του συλλέκτη που ξεχωρίζει τα μαύρα αντικείμενα από τα λευκά μέχρις ότου όλα τα προσβάσιμα αντικείμενα έχουν χρωματισθεί μαύρα.

Ο αλγόριθμος της συλλογής με σήμανση και εκκαθάριση διατηρεί μια σημαντική αναλλοίωτη:



Σχήμα 2.1: Σήμανση με τριχρωματική αφαίρεση. Τα μαύρα αντικείμενα καθώς και τα παιδιά αυτών έχουν επεξεργασθεί από το συλλέκτη. Ο συλλέκτης γνωρίζει την ύπαρξη των γκρι αντικειμένων αλλά δεν έχει ολοκληρώσει την επεξεργασία τους. Ο συλλέκτης δεν έχει επισκεφθεί ακόμη τα λευκά αντικείμενα (και μερικά δε τα επισκεφθεί ποτέ).

στο τέλος κάθε επανάληψης του βρόχου σήμανσης δεν υπάρχουν αναφορές από μαύρα προς λευκά αντικείμενα. Επομένως κάθε λευκό προσβάσιμο αντικείμενο είναι προσβάσιμο από κάποιο γκρι αντικείμενο. Η μη διατήρηση της αναλλοίωτης διακινδυνεύει τη μη σήμανση ενός ζωντανού απογόνου ενός μαύρου αντικειμένου (και κατ' επέκταση τη λανθασμένη εκκαθάριση αυτού) καθώς ο συλλέκτης δεν επανεξετάζει μαύρα αντικείμενα. Η τριχρωματική αφαίρεση είναι ιδιαίτερα χρήσιμη κατά τη θεώρηση αλγορίθμων για ταυτόχρονη συλλογή απορριμμάτων, όπου τα νήματα-τροποποιητές εκτελούνται ταυτόχρονα με τα νήματα-συλλέκτες.

2.3 Σήμανση με χρήση bitmap

Ένα bit σήμανσης συνήθως αποθηκεύεται στην επικεφαλίδα του αντικειμένου. Εναλλακτικά, τα bits σήμανσης μπορούν να αποθηκεύονται σε ένα ξεχωριστό bitmap στην άκρη του σωρού που συσχετίζει κάθε ένα bit με κάθε πιθανή διεύθυνση στη μνήμη όπου μπορεί να αποθηκευθεί ένα αντικείμενο. Ο απαιτούμενος χώρος για την αποθήκευση του bitmap εξαρτάται από τις απαιτήσεις ευθυγράμμισης της εικονικής μηχανής. Μπορεί να χρησιμοποιηθεί είτε ένα καθολικό bitmap, είτε αν ο σωρός είναι οργανωμένος σε μπλοκ, ένα bitmap ανά μπλοκ. Η τελευταία οργάνωση έχει ως πλεονέκτημα το γεγονός πως δε σπαταλάται χώρος στην περίπτωση που ο σωρός δεν είναι συνεχόμενος. Ο πίνακας bitmap για κάθε μπλοκ μπορεί να αποθηκευθεί σε μία σταθερή θέση του μπλοκ, κάτι το οποίο ωστόσο διακινδυνεύει την υποβάθμιση της επίδοσης, καθώς τα bitmap θα ανταγωνίζονται για τα ίδια σύνολα σε μία συσχετιστική κρυφή μνήμη. Επιπλέον, η πρόσβαση στο bitmap αυτόματα σημαίνει το άγγιγμα της αντίστοιχης σελίδας. Για να αντιμετωπισθεί το πρώτο πρόβλημα, η θέση ενός bitmap στο μπλοκ μπορεί να μεταβάλλεται και να προκύπτει για παράδειγμα ως αποτέλεσμα της εφαρμογής μιας συνάρτησης κατακερματισμού στη διεύθυνση του μπλοκ. Εναλλακτικά, το bitmap μπορεί να αποθηκευθεί εκτός του μπλοκ, χρησιμοποιώντας όμως έναν πίνακα που δεικτοδοτείται από μπλοκ (με πιθανή και πάλι χρήση κάποιας συνάρτησης κατακερματισμού). Η τεχνική αυτή αντιμετωπίζει και τα δύο προβλήματα ταυτόχρονα.

Τα bitmap αρκούν στην περίπτωση που η συλλογή απασχολεί μόνο ένα νήμα-σημαντή. Διαφορετικά, η ενημέρωση ενός bit σε ένα bitmap είναι ευάλωτη στην απώλεια ενημερώσεων, ενώ αντίθετα η ενημέρωση ενός bit στην επικεφαλίδα ενός αντικειμένου διακινδυνεύει απλώς

την εγγραφή της ίδιας τιμής εις διπλούν. Πολλοί συλλέκτες χρησιμοποιούν πίνακες bytemap για να καταστήσουν τις καταστάσεις συναγωνισμού ακίνδυνες. Εναλλακτικά, η τροποποίηση ενός bit σε ένα bitmap πρέπει να γίνεται με τη χρήση συγχρονισμένων λειτουργιών. Στην πράξη η κατάσταση είναι επίσης περίπλοκη όσον αφορά μεμονωμένα bits της επικεφαλίδας σε συστήματα που επιτρέπουν ταυτόχρονη εκτέλεση συλλέκτη και τροποποιητή, καθώς οι λέξεις της επικεφαλίδας φιλοξενούν μεταξύ άλλων και δεδομένα τα οποία μπορεί να προσπελάσει ο τροποποιητής, όπως κλειδώματα και κωδικούς κατακερματισμού. Μια προσεκτική σχεδίαση μπορεί να τοποθετήσει τα παραπάνω δεδομένα σε διαφορετικές λέξεις της επικεφαλίδας και να εξαλείψει την ανάγκη για ατομική τροποποίηση των bits σήμανσης αυτής.

Η χρήση bitmap σήμανσης παρουσιάζει έναν αριθμό από πλεονεκτήματα. Ένα bitmap αποθηκεύει τα bit σήμανσης πιο πυκνά σε σύγκριση με την αποθήκευσή τους στις επικεφαλίδες των αντικειμένων. Η χρήση bitmap σήμανσης κατά τη συλλογή με σήμανση και εκκαθάριση συνεπάγεται πως η φάση της σήμανσης δεν τροποποιεί κανένα αντικείμενο, παρά διαβάζει τα πεδία δείκτες των ζωντανών αντικειμένων. Με εξαίρεση τη φόρτωση του πεδίου περιγραφητή τύπου, κανένα άλλο τμήμα αντικειμένων που δεν περιέχουν δείκτες δεν προσπελάζεται. Η φάση της εκκαθάρισης δε θα διαβάσει από ούτε και θα γράψει σε κάποιο ζωντανό αντικείμενο, παρότι μπορεί να ενημερώσει πεδία αντικειμένων απορριμμάτων στο πλαίσιο ανάκτησης της μνήμης που αυτά καταλαμβάνουν (για παράδειγμα για να εισάγει ένα αντικείμενο σε μια ελεύθερη λίστα). Συνεπώς η σήμανση με χρήση bitmap έχει την τάση να τροποποιεί λιγότερες λέξεις μνήμης, με αποτέλεσμα ο αριθμός των βρώμικων μπλοκ κρυφής μνήμης που πρέπει να αντιγραφούν στην κύρια μνήμη να μειώνεται.

Η σήμανση με τη χρήση bitmap υιοθετήθηκε αρχικά για ένα συντηρητικό συλλέκτη που σχεδίασαν οι Boehm και Wieser [28] για την παροχή αυτόματης διαχείρισης μνήμης σε υλοποιήσεις μη συνεργατικών γλωσσών όπως η C και η C++. Σε συστήματα με ακριβή πληροφορία τύπων η ταυτοποίηση κάθε θέσης μνήμης που περιέχει ένα δείκτη είναι ακριβής ανεξάρτητα από το αν αυτή αφορά το εσωτερικό κάποιου αντικειμένου, τη στοίβα ή κάποια καθολική μεταβλητή. Οι **συντηρητικοί** συλλέκτες αντίθετα δεν λαμβάνουν αυτό το επίπεδο υποστήριξης από το μεταγλωττιστή ή το σύστημα εκτέλεσης και για αυτό προβαίνουν σε συντηρητικές αποφάσεις όσον αφορά την ταυτοποίηση δεικτών. Εάν η τιμή που βρίσκεται σε μια θέση μνήμης μοιάζει αρκετά με δείκτη, ο συλλέκτης αποφασίζει πως πράγματι είναι. Είναι εμφανές πως η συντηρητική συλλογή απορριμμάτων ενδέχεται εσφαλμένα να ερμηνεύσει μια θέση μνήμης ως δείκτη, κάτι που έχει δύο συνέπειες όσον αφορά την ασφάλεια. Πρώτον, ο συλλέκτης δεν επιτρέπεται να μεταβάλλει το περιεχόμενο καμίας θέσης μνήμης στο χώρο διευθύνσεων του τροποποιητή (συμπεριλαμβανομένων των ριζών). Αυτό αποκλείει αμέσως αλγορίθμους που μετακινούν αντικείμενα, καθώς αυτό θα απαιτούσε την ενημέρωση κάθε αναφοράς προς ένα μετακινηθέν αντικείμενο. Επίσης αποκλείει την αποθήκευση των bits σήμανσης στις επικεφαλίδες των αντικειμένων καθώς ένα "αντικείμενο" για το οποίο γίνεται λόγος μπορεί να μην είναι αντικείμενο αν η πρόσβαση σε αυτό έγινε από την εσφαλμένη θεώρηση μιας τιμής ως δείκτη. Η τροποποίηση της τιμής ενός bit μπορεί να καταστρέψει τα δεδομένα του χρήστη. Δεύτερον, η ελαχιστοποίηση της πιθανότητας ο τροποποιητής να προσπελάζει δεδομένα του συλλέκτη είναι πολύ χρήσιμη. Η προσθήκη μιας επικεφαλίδας στην αρχή ενός αντικειμένου είναι πιο ριψοκίνδυνη από την αποθήκευση των μεταδεδομένων του συλλέκτη όπως τα bits σήμανσης σε μία ξεχωριστή δομή δεδομένων.

Σύμφωνα με τον Boehm [25], ένα επιπλέον κίνητρο για τη σήμανση με χρήση bitmap είναι η ανάγκη για ελαχιστοποίηση της σελιδοποίησης που προκαλείται από τη δράση του συλλέκτη. Στα σύγχρονα συστήματα πάντως η πρόκληση της παραμικρής σελιδοποίησης από το συλλέκτη θεωρείται απαράδεκτη. Το ερώτημα είναι κατά πόσο η σήμανση με χρήση bitmap μπορεί να βελτιώσει την επίδοση της κρυφής μνήμης. Σύμφωνα με τους Hayes [58] και Jones

και Ryder [68] τα αντικείμενα έχουν μια τάση να ζουν και να πεθαίνουν σε συστάδες. Πολλοί εκχωρητές προσπαθούν να τοποθετήσουν τα αντικείμενα σε γειτονικές θέσεις μνήμης. Η εκκαθάριση με χρήση bitmap παρουσιάζει δύο πλεονεκτήματα. Πρώτον, επιτρέπει το μαζικό έλεγχο των bits/bytes σήμανσης για τα αντικείμενα μιας συστάδας, καθώς είτε όλα θα έχουν τη λογική τιμή 0 είτε όλα θα έχουν τη λογική τιμή 1. Άμεση συνέπεια του γεγονότος αυτού είναι η ευκολία προσδιορισμού μέσω του bitmap του κατά πόσο ένα μπλοκ μνήμης αποτελείται μόνο από απορρίμματα και άρα μπορεί να επιστραφεί ολόκληρο στον εκχωρητή.

Οι Garner κ.ά. [56] χρησιμοποιούν μια υβριδική προσέγγιση, συσχετίζοντας κάθε μπλοκ των ξεχωριστών ελεύθερων λιστών από μπλοκ διαφορετικού μεγέθους με ένα byte και ταυτόχρονα αποθηκεύοντας ένα bit στην επικεφαλίδα κάθε αντικειμένου. Το byte έχει την τιμή 0xFF αν και μόνο αν το αντίστοιχο μπλοκ έχει ένα τουλάχιστον ζωντανό αντικείμενο. Με τον τρόπο αυτό το byte-map επιτρέπει στον εκκαθαριστή να εντοπίζει εύκολα τα μπλοκ που δεν έχουν κανένα ζωντανό αντικείμενο και στη συνέχεια να τα ανακυκλώνει ολόκληρα. Το βασικό πλεονέκτημα της προσέγγισης αυτής είναι πως τόσο το bit στην επικεφαλίδα ενός αντικειμένου όσο και το αντίστοιχο byte του bytemap που αφορά το μπλοκ στο οποίο αυτό ζει μπορούν να εγγράφονται χωρίς τη χρήση συγχρονισμένων λειτουργιών.

Αλγόριθμος 2.4 Συλλογή με σήμανση και εκκαθάριση: σήμανση με bitmap (Printezis & Detlefs)

```
1:  procedure MARK()
2:      cur ← NEXTINBITMAP()
3:      while cur < HeapEnd do
4:          ADD(worklist, cur)
5:          MARKSTEP(cur)
6:          cur ← NEXTINBITMAP()

7:  procedure MARKSTEP(start)
8:      while not ISEMPTY(worklist) do
9:          ref ← REMOVE(worklist)
10:         for all fld ∈ Pointers(ref) do
11:             child ← *fld
12:             if child ≠ null and not ISMARKED(child) then
13:                 SETMARKED(child)
14:                 if child < start then
15:                     ADD(worklist, child)
```

Οι Printezis και Detlefs [97] χρησιμοποιούν bitmap για να μειώσουν τον απαιτούμενο χώρο για τις στοίβες σήμανσης σε έναν σχεδόν-ταυτόχρονο, γενεαλογικό συλλέκτη. Αρχικά, ως συνήθως οι ρίζες του τροποποιητή σημαίνονται με την εγγραφή της λογικής τιμής 1 ορισμένων bits στο bitmap. Στη συνέχεια, το νήμα-σημαντής σαρώνει γραμμικά το bitmap αναζητώντας ζωντανά αντικείμενα. Ο αλγόριθμος 2.4 διατηρεί την ακόλουθη αναλλοίωτη: τα σημασμένα αντικείμενα που βρίσκονται σε χαμηλότερες διευθύνσεις από την τιμή του δείκτη *cur* στη διαδικασία MARK είναι μαύρα, ενώ τα αντικείμενα που βρίσκονται σε υψηλότερες διευθύνσεις είναι γκρι. Όταν εντοπίζεται το επόμενο ζωντανό (σημασμένο) αντικείμενο, αυτό ωθείται στη στοίβα και ο έλεγχος περνά στη διαδικασία MARKSTEP. Η τελευταία εκτελεί έναν βρόχο **while** προκειμένου να αποκαταστήσει την ακόλουθη αναλλοίωτη: αντικείμενα εξωθούνται από τη στοίβα και τα παιδιά τους σημαίνονται αναδρομικά μέχρις ότου αυτή εκκενωθεί. Αν ένα αντικείμενο βρίσκεται σε χαμηλότερη διεύθυνση μνήμης από την τρέχουσα τιμή

του δείκτη *cur*, αυτό εισάγεται στη στοίβα σήμανσης, αλλιώς η επεξεργασία του αναβάλλεται για αργότερα. Η βασική διαφορά αυτού του αλγορίθμου και του αλγορίθμου 2.2 αφορά στην εισαγωγή των παιδιών ενός αντικειμένου στη στοίβα σήμανσης. Τα αντικείμενα σημαίνονται αναδρομικά μόνο αν αυτά βρίσκονται πίσω από το μαύρο μέτωπο κύματος που κινείται γραμμικά στο σωρό. Παρότι η πολυπλοκότητα του αλγορίθμου αυτού είναι γραμμική ως προς το μέγεθος του χώρου που συλλέγεται, στην πράξη η αναζήτηση σε ένα bitmap είναι φθηνή.

2.4 Οκνηρή εκκαθάριση

Η χρονική πολυπλοκότητα της σήμανσης είναι $O(L)$, όπου L το συνολικό μέγεθος των ζωντανών αντικειμένων του σωρού, ενώ η χρονική πολυπλοκότητα της εκκαθάρισης $O(H)$, όπου H το μέγεθος του σωρού. Καθώς $H > L$, εκ πρώτης όψεως φαίνεται πώς το κόστος της εκκαθάρισης κυριαρχεί στο κόστος του αλγορίθμου συλλογής με σήμανση και εκκαθάριση, κάτι που στην πράξη δεν παρατηρείται. Το "κυνήγι" δεικτών στη φάση της σήμανσης οδηγεί σε τελείως μη προβλέψιμα μοτίβα πρόσβασης στη μνήμη, ενώ αντίθετα η συμπεριφορά της εκκαθάρισης είναι πολύ πιο προβλέψιμη. Επιπροσθέτως, το κόστος της εκκαθάρισης ενός αντικειμένου είναι πολύ μικρότερο από το κόστος της εξερεύνησης του υπογράφου των αντικειμένων με ρίζα αυτό. Η προφόρτωση αντικειμένων αποτελεί έναν τρόπο βελτίωσης της συμπεριφοράς της εκκαθάρισης όσον αφορά την κρυφή μνήμη. Για να αποφύγουν τον κατακερματισμό της μνήμης οι εκχωρητές πολλών διαχειριστών μνήμης που υποστηρίζουν συλλέκτες με σήμανση και εκκαθάριση τοποθετούν αντικείμενα του ίδιου μεγέθους σε συνεχόμενες διευθύνσεις μνήμης, οδηγώντας με τον τρόπο αυτό σε σταθερό βήμα μοτίβου πρόσβασης μνήμης κατά την εκκαθάριση μπλοκ από αντικείμενα του ίδιου μεγέθους. Αυτή η προσέγγιση δεν επιτρέπει μόνο προφόρτωση λογισμικού αλλά είναι ιδανική και για τους μηχανισμούς προφόρτωσης υλικού που διαθέτουν οι σύγχρονοι επεξεργαστές.

Μπορούν οι χρόνοι παύσης των νημάτων-τροποποιητών για την εκτέλεση της φάσης της εκκαθάρισης να ελαττωθούν ή και να εξαλειφθούν πλήρως; Παρατηρούμε δύο ιδιότητες των αντικειμένων και των bit σήμανσης αυτών. Πρώτον, από τη στιγμή που ένα αντικείμενο γίνεται απόρριμμα, παραμένει απόρριμμα: δεν μπορεί να προσπελασθεί ή να αναστηθεί από κάποιο νήμα-τροποποιητή. Δεύτερον, τα νήματα-τροποποιητές δεν έχουν πρόσβαση στα bit σήμανσης. Επομένως η εκκαθάριση μπορεί να ανατεθεί σε ξεχωριστά νήματα-συλλέκτες, τα οποία εκτελούνται ταυτόχρονα με τα νήματα-τροποποιητές. Ο Hughes [64] προτείνει μια απλούστερη λύση, την **οκνηρή εκκαθάριση**. Η τεχνική ουσιαστικά πληρώνει το κόστος της εκκαθάρισης σε δόσεις με την εκκαθάριση να πραγματοποιείται από τον εκχωρητή. Στην απλούστερη εκδοχή η συνάρτηση ALLOCATE αυξάνει το δείκτη εκκαθάρισης μέχρις ότου βρει επαρκή χώρο σε μια ακολουθία μη σημασμένων αντικειμένων. Ωστόσο, η εκκαθάριση ενός μπλοκ κάθε φορά με πολλά αντικείμενα είναι πιο αποδοτική.

Ο αλγόριθμος 2.5 επενεργεί σε ένα μπλοκ κάθε φορά. Είναι σύνηθες οι εκχωρητές να τοποθετούν στο ίδιο μπλοκ ισομεγέθη αντικείμενα. Κάθε κλάση μεγέθους έχει ένα ή και περισσότερα μπλοκ εκχώρησης μνήμης καθώς και μία λίστα ανάκτησης (μεταβλητή $reclaimList$) από μπλοκ που δεν έχουν ακόμη εκκαθαρισθεί. Ως συνήθως ο συλλέκτης σημαίνει όλα τα ζωντανά αντικείμενα στο σωρό αλλά αντί να εκκαθαρίσει πρόθυμα ολόκληρο το σωρό, επιστρέφει τα εντελώς κενά από ζωντανά αντικείμενα μπλοκ στον εκχωρητή. Τα υπόλοιπα μπλοκ προστίθενται στην κατάλληλη λίστα ανάκτησης με βάση το μέγεθος των αντικειμένων που φιλοξενούν. Με τη λήξη της φάσης διακοπής του κόσμου τα νήματα-τροποποιητές επανεκκινούνται. Η συνάρτηση ALLOCATE αρχικά επιχειρεί την απόκτηση μιας ελεύθερης θέσης από

Αλγόριθμος 2.5 Συλλογή με σήμανση και εκκαθάριση: οκνηρή εκκαθάριση σε οργανωμένο κατά μπλοκ σωρό

```
1:  procedure COLLECT()
2:      atomic
3:      MARKFROMROOTS()
4:      for all block in blocks do
5:          if not ISMARKED(block) then          ▷ no objects marked in this block?
6:              ADD(blockAllocator, block)       ▷ return block to block allocator
7:          else
8:              ADD(reclaimList, block)

9:  function ALLOCATE(sz)
10:     atomic
11:     result ← REMOVE(sz)                      ▷ allocate from size class for sz
12:     if result = null then                    ▷ if no free slots for this size...
13:         LAZYSWEEP(sz)                        ▷ sweep a little
14:         result ← REMOVE(sz)
15:     return result                            ▷ if still null, collect

16: procedure LAZYSWEEP(sz)
17:     repeat
18:         block ← NEXTBLOCK(reclaimList, sz)
19:         if block ≠ null then
20:             SWEEP(start(block), end(block))
21:             if SPACEFOUND(block) then
22:                 return
23:     until block = null                       ▷ reclaim list for this size class is empty
24:     ALLOWSLOW(sz)

25: procedure ALLOCSLOW(sz)                      ▷ allocation slow path
26:     block ← ALLOCATEBLOCK()
27:     if block ≠ null then                     ▷ from the block allocator
28:         INITIALIZE(block, sz)
```

την αντίστοιχη κλάση μεγέθους. Αν η προσπάθεια αποτύχει, καλείται ο οκνηρός εκκαθαριστής ο οποίος εκκαθαρίζει ένα ή περισσότερα εναπομείναντα μπλοκ της αντίστοιχης λίστας *reclaimList* μέχρις ότου το αίτημα μπορεί να ικανοποιηθεί. Τι συμβαίνει στην περίπτωση όπου η λίστα των όχι ακόμη εκκαθαρισμένων μπλοκ είναι κενή ή δεν υπάρχει ελεύθερη θέση σε κανένα από τα εκκαθαρισμένα μπλοκ; Η απάντηση είναι πως ο εκκαθαριστής προσπαθεί να αποκτήσει ένα ολόκληρο ελεύθερο μπλοκ από έναν κατώτερου επιπέδου εκχωρητή μπλοκ. Το φρέσκο μπλοκ στη συνέχεια αρχικοποιείται με ρύθμιση των μεταδεδομένων αυτού (όπως για παράδειγμα με τη δημιουργία ενός bytemap σήμανσης). Αν πάλι δεν υπάρχουν διαθέσιμα φρέσκα μπλοκ, απαιτείται η κλήση του συλλέκτη.

Υπάρχει ένα λεπτό ζήτημα όσον αφορά την οκνηρή εκκαθάριση σε έναν οργανωμένο κατά μπλοκ σωρό. Ο Hughes [64] δουλεύει με έναν συνεχόμενο σωρό και επομένως εξασφαλίζει πως ο εκχωρητής θα εκκαθαρίσει κάθε αντικείμενο πριν ξεμείνει από χώρο και καλέσει το συλλέκτη. Ωστόσο, η οκνηρή εκκαθάριση ξεχωριστών κλάσεων για μπλοκ διαφορετικού μεγέθους δεν μπορεί να το εξασφαλίσει, καθώς είναι σχεδόν σίγουρο πως ο εκκαθαριστής θα εξαντλήσει μια κλάση μεγέθους (και όλα τα άδεια μπλοκ αυτής) πριν προχωρήσει στην εκκαθάριση μπλοκ που ανήκουν σε διαφορετικές κλάσεις μεγέθους. Το γεγονός αυτό οδηγεί σε δύο προβλήματα. Πρώτον, αντικείμενα-απορρίμματα σε μη εκκαθαρισμένα μπλοκ δεν αποδεσμεύονται, οδηγώντας σε διαρροή μνήμης. Αν το μπλοκ περιλαμβάνει ένα ζωντανό αντικείμενο, η παραπάνω διαρροή είναι αβλαβής καθώς τα αντικείμενα του μπλοκ δε θα ανακυκλωθούν ούτως ή άλλως πριν ο συλλέκτης πραγματοποιήσει ένα αίτημα για αντικείμενο της κλάσης μεγέθους στην οποία ανήκει το μπλοκ. Δεύτερον, αν όλα τα αντικείμενα σε ένα μπλοκ διαδοχικά μετατραπούν σε απορρίμματα, έχει χαθεί η δυνατότητα ανάκτησης όλου του μπλοκ μονομιάς.

Η απλούστερη λύση είναι η ολοκλήρωση της εκκαθάρισης όλων των μπλοκ του σωρού πριν την έναρξη της σήμανσης. Ωστόσο, μπορεί να είναι προτιμότερο να δοθούν σε ένα μπλοκ περισσότερες ευκαιρίες οκνηρής εκκαθάρισής του. Οι Garner κ.ά. [56] πληρώνουν ένα μικρό ποσό διαρροής για να αποφύγουν το κόστος της πρόθυμης εκκαθάρισης στο σύστημα Jikes RVM/MMTk του Blackburn κ.ά. [20]. Πιο συγκεκριμένα, αντί ενός bit, χρησιμοποιούν ένα φραγμένο ακέραιο για τη σήμανση των αντικειμένων. Αυτό συνήθως δεν επιφέρει επιπλέον κόστος σε χώρο καθώς υπάρχει χώρος παραπάνω από ένα bit αν οι σημάνσεις αποθηκεύονται στις επικεφαλίδες των αντικειμένων και συχνά ξεχωριστές δομές σήμανσης χρησιμοποιούν byte αντί για bit. Κάθε κύκλος συλλογής αυξάνει την τιμή σήμανσης κατά $1 \mod 2^K$, όπου K το μέγεθος της λέξης σήμανσης σε bits, με την τιμή σήμανσης να επανέρχεται προφανώς στο 0 σε περίπτωση υπερχείλισης. Με τον τρόπο αυτό, ο συλλέκτης μπορεί να ξεχωρίσει ένα αντικείμενο που έχει σημανθεί στον τρέχοντα κύκλο συλλογής από ένα αντικείμενο που είχε σημανθεί σε κάποιον προηγούμενο κύκλο συλλογής. Μόνο τα αντικείμενα με την τρέχουσα τιμή σήμανσης θεωρούνται σημασμένα. Η μηδενική τιμή σήμανσης είναι ασφαλής καθώς, ακριβώς πριν την εμφάνισή της, κάθε ζωντανό αντικείμενο στο σωρό είναι είτε μη σημασμένο (η μνήμη του εκχωρήθηκε μετά τον προηγούμενο κύκλο συλλογής), είτε έχει τη μέγιστη τιμή σήμανσης. Κάθε αντικείμενο που είναι σημασμένο με την επόμενη τιμή σήμανσης πρέπει να έχει σημανθεί κάποιο πολλαπλάσιο του 2^K αριθμό συλλογών πριν: επομένως είναι αιωρούμενο απόρριμμα και δε σημαίνεται από το νήμα-σημαντή. Η πιθανή προκύπτουσα διαρροή αντιμετωπίζεται σε ένα βαθμό με τη σήμανση ολόκληρων μπλοκ. Οποτεδήποτε ο MMTk συλλέκτης σημαίνει ένα αντικείμενο, σημαίνει επίσης και το αντίστοιχο μπλοκ. Αν κανένα από τα αντικείμενα ενός μπλοκ δεν είναι σημασμένο με την τρέχουσα τιμή σήμανσης, τότε ούτε και το μπλοκ είναι και συνεπώς μπορεί να ανακτηθεί ολόκληρο, όπως στον αλγόριθμο 2.5. Λαμβάνοντας υπόψη την τάση των αντικειμένων να ζουν και να πεθαίνουν σε συστάδες, αναμένει κανείς πως η παραπάνω τεχνική είναι αποδοτική.

Η οκνηρή εκκαθάριση προσφέρει έναν αριθμό πλεονεκτημάτων. Αρχικά έχει καλή τοπικότητα:

μία θέση μνήμης ενός πρώην απορρίμματος τείνει να χρησιμοποιηθεί σύντομα μετά την εκκαθάριση του τελευταίου. Επιπλέον, η αλγοριθμική πολυπλοκότητα της συλλογής με σήμανση και εκκαθάριση είναι πλέον γραμμική ως προς το μέγεθος των ζωντανών αντικειμένων του σωρού, ίδια με την πολυπλοκότητα της συλλογής με αντιγραφή που εξετάζουμε στο κεφάλαιο 4. Συγκεκριμένα, ο Boehm [24] επισημαίνει πως η συλλογή με σήμανση και οκνηρή εκκαθάριση παρουσιάζει βέλτιστες επιδόσεις στις ίδιες συνθήκες με τη συλλογή με αντιγραφή: όταν το μεγαλύτερο μέρος του σωρού είναι άδειο, καθώς η αναζήτηση μη σημασμένων αντικειμένων από την οκνηρή εκκαθάριση θα ολοκληρωθεί γρήγορα.

2.5 Θέματα προς εξέταση

Παρά το γεγονός πώς είναι ο παλαιότερος αλγόριθμος συλλογής απορριμμάτων, υπάρχουν διάφοροι λόγοι για τους οποίους η συλλογή με σήμανση και εκκαθάριση παραμένει ακόμα και σήμερα ελκυστική επιλογή.

2.5.1 Επιβάρυνση τροποποιητή

Η απλούστερη εκδοχή του αλγορίθμου δεν επιβάλλει καμία επιβάρυνση στις λειτουργίες Read και Write του τροποποιητή, σε αντίθεση για παράδειγμα με τη συλλογή με καταμέτρηση αναφορών που εξετάζουμε στο κεφάλαιο 5. Ωστόσο η συλλογή με σήμανση και εκκαθάριση συχνά χρησιμοποιείται ως βασικός αλγόριθμος σε πιο εξεζητημένους συλλέκτες που απαιτούν κάποιας μορφής συγχρονισμό μεταξύ συλλέκτη και τροποποιητή. Τόσο οι γενεαλογικοί συλλέκτες (κεφάλαιο 7) όσο και οι ταυτόχρονοι και αυξητικοί συλλέκτες (κεφάλαιο 9) απαιτούν από τον τροποποιητή να ενημερώνει το συλλέκτη οποτεδήποτε τροποποιεί δείκτες.

2.5.2 Ρυθμαπόδοση

Σε συνδυασμό με την οκνηρή εκκαθάριση, η συλλογή με σήμανση και εκκαθάριση προσφέρει υψηλή ρυθμαπόδοση. Η φάση της σήμανσης είναι σχετικά φθηνή και κυριαρχείται από την καταδίωξη δεικτών. Χρειάζεται απλώς να θέσει ένα bit/byte για κάθε ζωντανό αντικείμενο που ανακαλύπτει, σε αντίθεση με τη συλλογή με αντιγραφή (κεφάλαιο 4) ή τη συλλογή με σήμανση και συμπύκνωση (κεφάλαιο 3) που αντιγράφουν ή μετακινούν αντικείμενα. Από την άλλη πλευρά, όπως και οι περισσότεροι συλλέκτες εξιχνίασης που εξετάζουμε στα πρώτα κεφάλαια αυτής της εργασίας, η συλλογή με σήμανση και εκκαθάριση απαιτεί την διακοπή της εκτέλεσης των νημάτων-τροποποιητών κατά τη διάρκεια εκτέλεσης του συλλέκτη. Ο χρόνος παύσης ενός κύκλου συλλογής εξαρτάται πρωτίστως από το πρόγραμμα που εκτελεί ο τροποποιητής και την είσοδο αυτού και μπορεί εύκολα σε μερικές περιπτώσεις να φθάσει σε διάρκεια και μερικά δευτερόλεπτα.

2.5.3 Απαιτήσεις σε χώρο

Η συλλογή με σήμανση και εκκαθάριση έχει σαφώς καλύτερη διαχείριση χώρου από τη συλλογή με αντιγραφή. Ενδεχομένως να έχει καλύτερη διαχείριση χώρου και από τη συλλογή με καταμέτρηση αναφορών. Τα bits σήμανσης συχνά αποθηκεύονται χωρίς κόστος στις επικεφαλίδες των αντικειμένων. Εναλλακτικά, αν χρησιμοποιείται ένα bitmap η επιβάρυνση σε

χώρο εξαρτάται από τις απαιτήσεις ευθυγράμμισης της αρχιτεκτονικής. Η συλλογή με καταμέτρηση αναφορών από την άλλη πλευρά απαιτεί τη χρήση μίας λέξης μνήμης σε κάθε επικεφαλίδα αντικειμένου για την αποθήκευση του μετρητή αναφορών του τελευταίου. Η συλλογή με αντιγραφή έχει ακόμη χειρότερη διαχείριση του χώρου αφού διαχωρίζει το σωρό σε δύο ισομεγέθεις ημιχώρους εκ των οποίων μόνο ο ένας είναι ανά πάσα στιγμή διαθέσιμος στον τροποποιητή. Από την άλλη πλευρά, οι συλλέκτες που δεν συμπυκνώνουν το σωρό, όπως η συλλογή με σήμανση και εκκαθάριση αλλά και η συλλογή με καταμέτρηση αναφορών απαιτούν από το διαχειριστή μνήμης τη χρήση πιο περίπλοκων εκχωρητών. Οι επιπλέον δομές δεδομένων που απαιτούνται για την υποστήριξη τέτοιων συλλεκτών προσθέτουν μια μη αμελητέα επιβάρυνση. Επιπρόσθετα η συλλογή χωρίς συμπύκνωση συχνά συνοδεύεται από κατακερματισμό της μνήμης.

Ο αλγόριθμος συλλογής απορριμμάτων με σήμανση και εκκαθάριση είναι όπως εξηγήσαμε ένας αλγόριθμος εξιχνίασης. Ως τέτοιος, οφείλει να αναγνωρίσει πρώτα όλα τα ζωντανά αντικείμενα του σωρού πριν ανακτήσει τη μνήμη που χρησιμοποιείται από νεκρά αντικείμενα. Η διαδικασία αυτή είναι ακριβή και θα πρέπει να εκτελείται όσο πιο σπάνια γίνεται. Αυτό σημαίνει πώς πρέπει να αφιερωθεί ειδικός χώρος στο σωρό για τη λειτουργία των συλλεκτών εξιχνίασης. Αν τα ζωντανά αντικείμενα καταλαμβάνουν ένα μεγάλο μέρος του σωρού και οι εκχωρητές εκχωρούν πολύ συχνά μνήμη ο συλλέκτης με σήμανση και εκκαθάριση θα καλείται πολύ συχνά. Ο Jones [67] επισημαίνει πώς για σωρούς μεσαίου και μεγάλου μεγέθους το ποσοστό του σωρού που αφιερώνεται στον ειδικό χώρο για τις λειτουργίες του συλλέκτη μπορεί να κυμαίνεται από 20% έως και 50%. Οι Hertz και Berger [60] πάντως δείχνουν πώς η αυτόματη διαχείριση μνήμης προγραμμάτων στη γλώσσα Java με τη χρήση συλλογής με σήμανση και εκκαθάριση ενδέχεται να απαιτήσει έναν σωρό ακόμη και μερικές φορές μεγαλύτερο σε μέγεθος προκειμένου να πετύχει την ίδια ρυθμαπόδοση με την χειρωνακτική διαχείριση μνήμης αυτών με ρητή αποδέσμευση.

2.5.4 Μετακίνηση ή όχι;

Η μη μετακίνηση αντικειμένων παρουσιάζει πλεονεκτήματα και μειονεκτήματα. Το βασικό πλεονέκτημα που προκύπτει από τη μη μετακίνηση αντικειμένων είναι πώς καθιστά τη συλλογή με σήμανση και εκκαθάριση κατάλληλη για χρήση σε περιβάλλοντα όπου δεν υπάρχει συνεργασία μεταξύ μεταγλωττιστή της γλώσσας και συλλέκτη απορριμμάτων. Χωρίς έγκυρη πληροφορία τύπων, οι ρίζες του τροποποιητή και πεδία δείκτες αντικειμένων δεν γίνεται να ενημερωθούν με νέες διευθύνσεις μετακινηθέντων αντικειμένων. Αυτό συμβαίνει λόγω της έλλειψης βεβαιότητας σχετικά με το αν μια μεταβλητή ή ένα πεδίο στο εσωτερικό ενός αντικείμενου είναι δείκτης.

Η ασφάλεια σε μη συνεργατικά συστήματα που διαχειρίζονται από έναν συντηρητικό συλλέκτη αποτρέπει τον τελευταίο από την τροποποίηση δεδομένων του χρήστη (συμπεριλαμβανομένων των επικεφαλίδων αντικειμένων). Επιπλέον ενθαρρύνει την ξεχωριστή αποθήκευση των μεταδεδομένων του συλλέκτη από τα μεταδεδομένα του χρήστη και του συστήματος εκτέλεσης ώστε να μειωθεί η πιθανότητα τροποποίησης των πρώτων από τον τροποποιητή. Για τους παραπάνω λόγους είναι επιθυμητή η αποθήκευση των bit/byte σήμανσης σε πίνακες bitmap και όχι στις επικεφαλίδες των αντικειμένων.

Το πρόβλημα με τη μετακίνηση αντικειμένων είναι πώς σε εφαρμογές που τρέχουν για πολλή ώρα, ο σωρός τείνει να κατακερματίζεται. Ο Robson [98],[99] αποδεικνύει πώς οι εκχωρητές σε συστήματα διαχείρισης μνήμης που χρησιμοποιούν μη μετακινούντα συλλέκτη απαιτούν $O(\log \frac{max}{min})$ περισσότερο χώρο από τον ελάχιστο δυνατό, όπου min και max το ελάχιστο και

μέγιστο δυνατό μέγεθος αντικειμένων αντίστοιχα. Συνεπώς ένας συλλέκτης ο οποίος δε συμπυκνώνει το σωρό με μεγάλη πιθανότητα καλείται συχνότερα από έναν που το συμπυκνώνει. Ακόμη, όπως σημειώθηκε πιο πάνω, ένα ποσοστό από 20% έως και 50% του σωρού πρέπει να αφιερώνεται αποκλειστικά στη λειτουργία ενός συλλέκτη εξιχνίασης ώστε να αποφευχθεί η αυξημένη συχνότητα κλήσης του τελευταίου.

Για να ελαχιστοποιηθεί η μείωση της επίδοσης λόγω του κατακερματισμού, πολλές εμπορικές υλοποιήσεις συλλεκτών οι οποίες διαχειρίζονται μια περιοχή του σωρού με σήμανση και εκκαθάριση περιοδικά χρησιμοποιούν και διαφορετικό αλγόριθμο όπως αυτόν με σήμανση και συμπύκνωση για να εξαλείψουν τον κατακερματισμό. Συγκεκριμένα αυτό απαιτείται αν η εφαρμογή δε διατηρεί σταθερό ρυθμό μεγεθών αντικειμένων ή δεσμεύει μνήμη για πολλά μεγάλα αντικείμενα. Αν αρχίσει να δεσμεύει μνήμη για μεγαλύτερα αντικείμενα από ότι πριν, ενδέχεται να προκύψουν πολλές μικρές τρύπες στο σωρό που δε χρησιμοποιούνται πλέον για τη δέσμευση μνήμης νέων αντικειμένων του ίδιου (μεγαλύτερου) μεγέθους. Αντίστροφα, αν αρχίσει να δεσμεύει μνήμη για μικρότερα αντικείμενα από ότι πριν, τα τελευταία μπορεί να φιλοξενούνται σε κενά που προηγουμένως καταλαμβάνονταν από μεγαλύτερα αντικείμενα οδηγώντας έτσι σε σπατάλη του αχρησιμοποίητου χώρου των κενών. Οι Dimpsey κ.ά. [46], όπως και οι Blackburn και McKinley [21] ωστόσο παρατηρούν πώς η προσεκτική διαχείριση του σωρού μπορεί να ελαττώσει την τάση για κατακερματισμό της μνήμης επωφελούμενη της τάσης των αντικειμένων να ζουν και να πεθαίνουν σε συστάδες.

Κεφάλαιο 3

Συλλογή απορριμμάτων με σήμανση και συμπύκνωση

Η τάση τους να κατακερματίζουν τη μνήμη είναι το βασικό πρόβλημα των μη μετακινούντων συλλεκτών απορριμμάτων. Παρότι μπορεί να υπάρχει ελεύθερη μνήμη στο σωρό, αυτή είναι πολύ πιθανόν να είναι κατακερματισμένη σε πολλά μικρά τμήματα και όχι συνεχόμενη. Συνεπώς είναι πιθανόν ο εκχωρητής να μην ικανοποιήσει ένα αίτημα παρότι υπάρχει ελεύθερη μνήμη στο σωρό καθώς αυτή είναι κατακερματισμένη.

Η **συλλογή απορριμμάτων με σήμανση και συμπύκνωση** επινοήθηκε προς λύσιν του προβλήματος του κατακερματισμού. Το σημαντικό πλεονέκτημα της μεθόδου είναι πως απλοποιεί το έργο του εκχωρητή. Ο σωρός αποτελείται από δύο συνεχόμενα τμήματα: το πρώτο φιλοξενεί αντικείμενα ενώ το δεύτερο είναι ελεύθερο και χρησιμοποιείται από τον εκχωρητή για την ικανοποίηση αιτημάτων μνήμης. Τα δύο αυτά τμήματα διαχωρίζονται από ένα δείκτη, ο οποίος σηματοδοτεί ταυτόχρονα το τέλος του χρησιμοποιούμενου τμήματος και την αρχή του ελεύθερου τμήματος. Συνεπώς, αν υπάρχει ελεύθερη μνήμη (πρακτικά αν ο εν λόγω δείκτης δεν έχει ξεπεράσει το τέλος του σωρού), ο εκχωρητής επιστρέφει την τιμή του δείκτη και κατόπιν τον ενημερώνει αυξάνοντας τον κατά το μέγεθος του αιτήματος. Η συλλογή με σήμανση και συμπύκνωση συγκεντρώνει τα προσβάσιμα αντικείμενα στο άκρο του σωρού, ο οποίος είναι ενιαίος. Αντίθετα, η συλλογή με αντιγραφή την οποία εξετάζουμε στο κεφάλαιο 4 μετακινεί τα προσβάσιμα αντικείμενα από έναν ημιχώρο σε έναν άλλο ημιχώρο του σωρού.

Οι συλλέκτες με σήμανση και συμπύκνωση λειτουργούν σε φάσεις. Η πρώτη φάση είναι πάντα αυτή της σήμανσης, την οποία και εξετάσαμε στο προηγούμενο κεφάλαιο. Τη φάση της σήμανσης διαδέχονται οι φάσεις της συμπύκνωσης, οι οποίες μετακινούν αντικείμενα και ενημερώνουν τις μεταβλητές δείκτες που δείχνουν σε προσβάσιμα αντικείμενα με τις νέες διευθύνσεις αυτών. Το πλήθος των φάσεων, η σειρά με την οποία αυτές εκτελούνται καθώς και ο τρόπος με τον οποίο μετακινούνται τα αντικείμενα διαφέρει από αλγόριθμο σε αλγόριθμο. Ο τρόπος μάλιστα με τον οποίο συμπυκνώνονται τα αντικείμενα επηρεάζει την τοπικότητα του τροποποιητή.

Κάθε μετακινών συλλέκτης μπορεί να οργανώσει τα αντικείμενα στο σωρό με έναν από τους ακόλουθους 3 τρόπους, οπότε και χαρακτηρίζεται ως:

1. **Τυχαίος**: Τα αντικείμενα μετακινούνται ανεξαρτήτως της αρχικής τους σειράς ή του αν δείχνουν το ένα στο άλλο.

2. **Γραμμικοποιών**: Τα αντικείμενα τοποθετούνται με τέτοιο τρόπο ώστε να είναι κοντά με σχετικά προς αυτά αντικείμενα στο μέγιστο βαθμό. Δύο αντικείμενα είναι σχετικά αν το ένα δείχνει στο άλλο ή αν είναι αδέρφια σε μια δομή δεδομένων κ.ο.κ.

3. **Ολισθαίνων**: Τα αντικείμενα "ολισθαίνουν" στο ένα άκρο του σωρού, με αποτέλεσμα να διατηρείται η αρχική τους διάταξη στο σωρό.

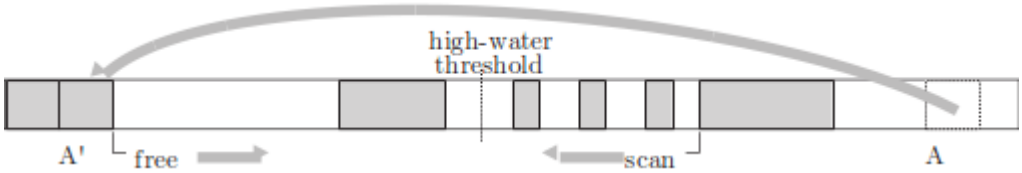
Οι περισσότεροι συλλέκτες με σήμανση και συμπύκνωση που απαντώνται στη βιβλιογραφία είναι τυχαίοι ή ολισθαίνοντες. Οι συλλέκτες αυτών των κατηγοριών έχουν απλή υλοποίηση και εξαιρετικές επιδόσεις σε ταχύτητα. Το μειονέκτημά τους είναι πώς οδηγούν σε μικρή έως καθόλου τοπικότητα για τον τροποποιητή, καθώς αντικείμενα αντιστοιχίζονται σε διαφορετικά μπλοκ της κρυφής μνήμης ή ακόμα και σε διαφορετικές σελίδες της εικονικής μνήμης. Οι Abuaiadh κ.ά. σε πρόσφατη μελέτη τους, [2] επαληθεύουν πως η τυχαία συμπύκνωση του σωρού μπορεί να οδηγήσει σε δραστική μείωση της διεκπεραιωτικής ικανότητας του τροποποιητή. Όλοι οι σύγχρονοι συλλέκτες με σήμανση και συμπύκνωση είναι ολισθαίνοντες και έτσι δεν παρεμβαίνουν στην τοπικότητα του τροποποιητή.

Οι τυχαίοι συλλέκτες διαχειρίζονται αντικείμενα ενός συγκεκριμένου μεγέθους ή συμπυκνώνουν ξεχωριστά αντικείμενα διαφορετικών μεγεθών. Η συμπύκνωση μπορεί να απαιτεί δύο ή τρία περάσματα στο σωρό. Ακόμη ενδέχεται να χρειάζεται η προσθήκη μιας επιπλέον λέξης στην επικεφαλίδα ενός αντικειμένου όπου θα αποθηκεύονται πληροφορίες σχετικά με τη νέα διεύθυνση αυτού. Το κόστος σε χώρο από την προσθήκη αυτή μπορεί να είναι σημαντικό σε έναν γενικού σκοπού διαχειριστή μνήμης. Τέλος, οι συλλέκτες με συμπύκνωση μπορεί να επιβάλλουν επιπρόσθετους περιορισμούς όσον αφορά τις μεταβλητές δείκτες. Για παράδειγμα, προς ποια κατεύθυνση επιτρέπεται αυτοί να δείχνουν; Επιτρέπονται οι εσωτερικοί δείκτες;

Στη συνέχεια εξετάζουμε διάφορα είδη συμπύκνωσης. Σε κάθε περίπτωση, ο συλλέκτης καλείται ως εξής:

Αλγόριθμος 3.1 Συλλογή με σήμανση και συμπύκνωση

```
1: procedure COLLECT()
2:     atomic
3:     MARKFROMROOTS()
4:     COMPACT()
```



Σχήμα 3.1: Ο αλγόριθμος Two-Finger του Edward. Ζωντανά αντικείμενα από την κορυφή του σωρού μετακινούνται σε ελεύθερα κενά στη βάση του σωρού. Εδώ το αντικείμενο στη θέση A έχει μετακινηθεί στη θέση A'. Ο αλγόριθμος τερματίζει όταν οι δείκτες $scan$ και $free$ συναντηθούν.

3.1 Συμπύκνωση με δύο δείκτες

Ο αλγόριθμος Two-Finger [104] εκτελεί τη συμπύκνωση σε δύο φάσεις και συμπυκνώνει τα προσβάσιμα αντικείμενα σε τυχαία σειρά. Είναι βέλτιστος δε στην ειδική περίπτωση όπου ο τροποποιητής χρησιμοποιεί τα αντικείμενα του ίδιου σταθερού μεγέθους. Η ιδέα του είναι απλή: δοθέντος του όγκου των προσβάσιμων αντικειμένων στην προς συμπύκνωση περιοχή, γνωρίζουμε ποιο θα είναι το όριο της περιοχής αυτής μετά την συμπύκνωση. Ζωντανά αντικείμενα πάνω από αυτό το κατώφλι αντιγράφονται στα κατάλληλα κενά κάτω από το κατώφλι.

Ο αλγόριθμος 3.2 διατηρεί δύο δείκτες, τους $free$ και $scan$, οι οποίοι αρχικά δείχνουν στην αρχή και το τέλος της προς συμπύκνωση περιοχής. Στη διάρκεια της πρώτης φάσης, ο δείκτης $free$ αυξάνεται μέχρις ότου συναντήσει ένα αντικείμενο που είναι ελεύθερο (μη-σημασμένο), ενώ ο δείκτης $scan$ μειώνεται μέχρις ότου συναντήσει ένα ζωντανό αντικείμενο (σημασμένο). Αν οι δύο δείκτες συναντηθούν, η πρώτη αυτή φάση ολοκληρώνεται.

Αλλιώς, το αντικείμενο στη θέση $scan$ αντιγράφεται στη θέση $free$, στο παλιό αντίγραφο στη θέση $scan$ αποθηκεύεται η νέα διεύθυνση (την τιμή του δείκτη $free$ εκείνη τη χρονική στιγμή) του αντικειμένου και η διαδικασία επαναλαμβάνεται. Είναι εμφανές πως η ποιότητα της συμπύκνωσης εξαρτάται πρωτίστως από το εάν το μέγεθος του αντικειμένου στη θέση $scan$ είναι το ίδιο ή σχεδόν το ίδιο με το μέγεθος του αντικειμένου στη θέση $free$: αν αυτό δε συμβαίνει, η μνήμη μπορεί και πάλι να είναι κατακερματισμένη. Γι αυτόν το λόγο ο αλγόριθμος είναι βέλτιστος στην περίπτωση που χρησιμοποιούνται αντικείμενα ίδιου (σταθερού) μεγέθους.

Μετά το πέρας της πρώτης φάσης, ο δείκτης $free$ δείχνει ακριβώς στο σύνορο μεταξύ του χρησιμοποιούμενου και του ελεύθερου τμήματος του σωρού. Η δεύτερη φάση αναλαμβάνει να ενημερώσει τις τιμές των δεικτών προς τα μετακινηθέντα αντικείμενα: Οι νέες διευθύνσεις αυτών βρίσκονται αποθηκευμένες στις παλιές τους θέσεις κατά τη μετακίνησή τους στη διάρκεια της πρώτης φάσης.

Στα πλεονεκτήματα του αλγορίθμου περιλαμβάνονται η απλότητα και η ταχύτητα. Επίσης δεν παρουσιάζει επιβάρυνση σε χώρο, αφού η νέα διεύθυνση ενός αντικειμένου γράφεται στην παλιά του θέση μόνο αφότου έχει ολοκληρωθεί η αντιγραφή του και συνεπώς καμία πληροφορία δεν καταστρέφεται. Ο αλγόριθμος υποστηρίζει εσωτερικούς δείκτες. Τέλος, το μοτίβο πρόσβασης στη μνήμη είναι σχετικά προβλέψιμο, κάτι που σημαίνει πως η προφόρτωση (υλικού ή/και λογισμικού) μπορεί να οδηγήσει το συλλέκτη σε καλή συμπεριφορά ως προς την κρυφή μνήμη. Ωστόσο, το βασικό μειονέκτημα παραμένει: τα αντικείμενα τοποθετούνται σε τυχαία σειρά στο σωρό και αυτό καταστρέφει την όποια τοπικότητα του τροποποιητή.

Αλγόριθμος 3.2 Συλλογή με σήμανση και συμπύκνωση: ο αλγόριθμος συμπύκνωσης Two-Finger

```
1:  procedure COMPACT()
2:      RELOCATE(HeapStart, HeapEnd)
3:      UPDATEREFERENCES(HeapStart, free)

4:  procedure RELOCATE(start, end)
5:      free ← start
6:      scan ← end
7:      while free < scan do
8:          while ISMARKED(free) do
9:              UNSETMARKED(free)
10:             free ← free + SIZE(free)                    ▷ find next hole
11:         while not ISMARKED(scan) and scan > free do
12:             scan ← scan − SIZE(scan)                    ▷ find previous live object
13:         if scan > free then
14:             UNSETMARKED(scan)
15:             MOVE(scan, free)
16:             scan ← free                  ▷ leave forwarding address (destructively)
17:             free ← free + SIZE(free)
18:             scan ← scan − SIZE(scan)

19: procedure UPDATEREFERENCES(start, end)
20:     for all fld in Roots do          ▷ update roots that pointed to moved objects
21:         ref ← ∗fld
22:         if ref > end then
23:             ∗fld ← ref           ▷ use the forwarding address left in the first pass
24:     scan ← start
25:     while scan < end do                              ▷ update fields in live region
26:         for all fld in Pointers(scan) do
27:             ref ← ∗fld
28:             if ref > end then
29:                 ∗fld ← ref       ▷ use the forwarding address left in the first pass
30:         scan ← scan + SIZE(scan)                                    ▷ next object
```

3.2 Ο αλγόριθμος Lisp 2

Ο επόμενος αλγόριθμος που εξετάζουμε συμπυκνώνει τα αντικείμενα διατηρώντας την αρχική τους διάταξη και είναι ο Lisp 2. Ο αλγόριθμος 3.3 χρησιμοποιείται ευρύτατα, είτε στην αρχική του μορφή, είτε παραλλαγμένος ώστε να εκτελείται παράλληλα, όπως από τους Flood κ.ά. [53]. Μπορεί να χρησιμοποιηθεί σε εφαρμογές όπου ο σωρός φιλοξενεί αντικείμενα διαφορετικών μεγεθών και παρότι πραγματοποιεί τρία περάσματα στο σωρό, κάθε ένα από αυτά κάνει λίγη δουλειά. Παρότι γενικά οι αλγόριθμοι συλλογής με σήμανση και συμπύκνωση φημίζονται για τη χαμηλή διεκπεραιωτική τους ικανότητα, οι Cohen και Nicolau [37] διαπίστωσαν πως επρόκειτο για το γρηγορότερο από τους αλγορίθμους που συνέκριναν. Το βασικό του μειονέκτημα είναι πως απαιτεί μία επιπλέον λέξη στην επικεφαλίδα ενός αντικειμένου για την αποθήκευση της νέας διεύθυνσης στην οποία αυτό θα μετακινηθεί. Το πεδίο αυτό μπορεί να χρησιμοποιηθεί για την αποθήκευση του bit σήμανσης.

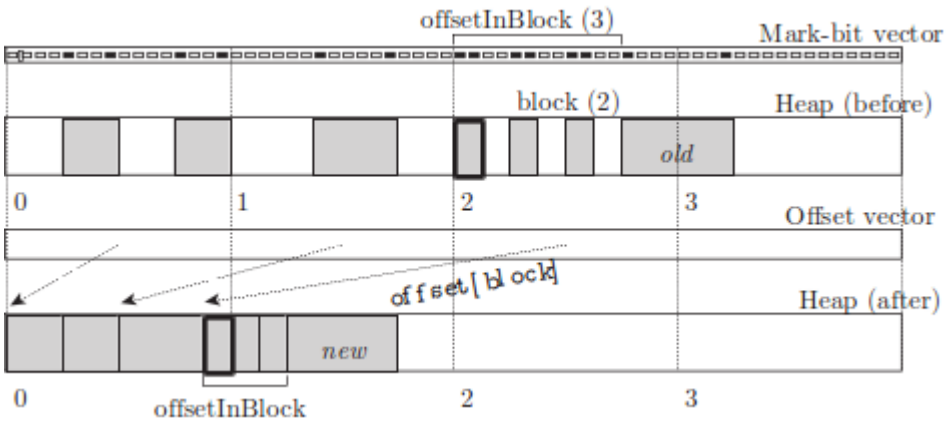
Το πρώτο πέρασμα του αλγορίθμου (μετά από τη φάση της σήμανσης) υπολογίζει τη διεύθυνση στην οποία θα μετακινηθεί κάθε ζωντανό αντικείμενο και αποθηκεύει την τιμή αυτή στο πεδίο $forwardingAddress$ αυτού. Η διαδικασία COMPUTELOCATIONS, η οποία πραγματοποιεί το πρώτο πέρασμα, δέχεται τρία ορίσματα: την αρχική και τελική διεύθυνση του προς συμπκύκνωση τμήματος του σωρού (περιοχή προέλευσης) καθώς και την αρχική διεύθυνση του τμήματος του σωρού όπου θα μετακινηθούν τα αντικείμενα (περιοχή προορισμού). Παρότι συνήθως η περιοχή προορισμού ταυτίζεται με την περιοχή προέλευσης, αν η συμπύκνωση εκτελείται παράλληλα, κάθε νήμα-συμπυκνωτής δύναται να διατηρεί τις δικές του διακριτές περιοχές. Η διαδικασία COMPUTELOCATIONS διατηρεί δύο δείκτες στο σωρό: ο δείκτης *scan* διατρέχει σειριακά το σωρό, εξετάζοντας κάθε αντικείμενο (σημασμένο και μή) στην περιοχή προέλευσης, ενώ ο δείκτης $free$ δείχνει στην πρώτη ελεύθερη θέση στην περιοχή προορισμού. Αν ένα αντικείμενο είναι ζωντανό, τότε θα μετακινηθεί (τελικά) στη θέση $free$ και έτσι η τρέχουσα τιμή του δείκτη free αποθηκεύεται στο πεδίο $forwardingAddress$ του αντικειμένου και κατόπιν αυξάνεται κατά το μέγεθος αυτού. Αν πάλι είναι νεκρό, απλώς αγνοείται.

Το δεύτερο πέρασμα πραγματοποιείται από τη διαδικασία UPDATEREFERENCES, κατά την εκτέλεση της οποίας ενημερώνονται οι ρίζες των νημάτων-τροποποιητών καθώς και τα πεδία δείκτες σημασμένων αντικειμένων ώστε να δείχνουν στις νέες διευθύνσεις των αντικειμένων προς τα οποία δείχνουν, χρησιμοποιώντας την τιμή που αποθηκεύθηκε στο πεδίο $forwarding\text{-}Address$ των τελευταίων στη διάρκεια του πρώτου περάσματος.

Στο τρίτο και τελευταίο πέρασμα, η διαδικασία RELOCATE μετακινεί κάθε ζωντανό (σημασμένο) αντικείμενο στη νέα του διεύθυνση.

Αλγόριθμος 3.3 Συλλογή με σήμανση και συμπύκνωση: ο αλγόριθμος συμπύκνωσης Lisp 2

```
1: procedure COMPACT()
2:     COMPUTELOCATIONS(HeapStart, HeapEnd, HeapStart)
3:     UPDATEREFERENCES(HeapStart, HeapEnd)
4:     RELOCATE(HeapStart, HeapEnd)

5: procedure COMPUTELOCATIONS(start, end, toRegion)
6:     scan ← start
7:     free ← toRegion
8:     while scan < end do
9:         if ISMARKED(scan) then
10:            FORWARDINGADDRESS(scan) ← free
11:            free ← free + SIZE(scan)
12:        scan ← scan + SIZE(scan)

13: procedure UPDATEREFERENCES(start, end)
14:    for all fld in Roots do
15:        ref ← *fld
16:        if ref ≠ null then
17:            *fld ← FORWARDINGADDRESS(ref)
18:    scan ← start
19:    while scan < end do
20:        if ISMARKED(scan) then
21:            for all fld in Pointers(scan) do
22:                if *fld ≠ null then
23:                    *fld ← FORWARDINGADDRESS(*fld)
24:        scan ← scan+ SIZE(scan)

25: procedure RELOCATE(start, end)
26:    scan ← start
27:    while scan < end do
28:        if ISMARKED(child) then
29:            dest ← SETMARKED(child)
30:            MOVE(scan, dest)
31:            UNSETMARKED(dest)
32:        scan ← scan + SIZE(scan)
```



Σχήμα 3.2: Ο σωρός (πριν και μετά τη συμπύκνωση) και τα μεταδεδομένα που χρησιμοποιεί ο αλγόριθμος Compressor. Τα bits του διανύσματος σήμανσης υποδεικνύουν την αρχή και το τέλος κάθε ζωντανού αντικειμένου. Οι λέξεις στο διάνυσμα μετατόπισης αποθηκεύουν τη διεύθυνση προς την οποία θα μετακινηθεί το πρώτο ζωντανό αντικείμενο του αντίστοιχου μπλοκ. Οι διευθύνσεις προώθησης δεν αποθηκεύονται αλλά υπολογίζονται όταν αυτό είναι απαραίτητο από τα διανύσματα σήμανσης και μετατόπισης.

3.3 Συμπύκνωση με ένα πέρασμα

Εάν θέλουμε να μειώσουμε τον αριθμό των περασμάτων στο σωρό σε δύο (ένα για σήμανση και ένα για ολίσθηση αντικειμένων), τότε πρέπει να αποθηκεύσουμε τις διευθύνσεις προορισμού σε ένα βοηθητικό πίνακα ο οποίος διατηρείται κατά τη διάρκεια της συμπύκνωσης. Οι αλγόριθμοι με σήμανση και συμπύκνωση που σχεδίασαν οι Abuaiadh κ.ά. [2] καθώς και οι Kermany και Petrank [73] επιτυγχάνουν υψηλές επιδόσεις σε πολυεπεξεργαστικά συστήματα ακολουθώντας αυτήν ακριβώς την τεχνική. Ο πρώτος αλγόριθμος είναι παράλληλος (ο συλλέκτης χρησιμοποιεί πολλαπλά νήματα-συλλέκτες ενώ τα νήματα-τροποποιητές είναι σταματημένα), ενώ ο δεύτερος δύναται να διαμορφωθεί τόσο για ταυτόχρονη εκτέλεση (επιτρέποντας ταυτόχρονη εκτέλεση των νημάτων- συλλεκτών και των νημάτων-τροποποιητών) όσο και για αυξητική εκτέλεση (όπου η εκτέλεση του τροποποιητή αναστέλλεται περιοδικά για ένα βραχύ χρονικό διάστημα κατά το οποίο επιτρέπεται στο συλλέκτη η εκτέλεση ενός μικρού κβάντου εργασιών).

Στην ενότητα αυτή εξετάζουμε την απλή εκδοχή του αλγορίθμου, όπου και υποθέτουμε πώς διακόπτεται η εκτέλεση των νημάτων-τροποποιητών κατά την εκτέλεση του συλλέκτη.

Η λειτουργία του αλγορίθμου στηρίζεται κυρίως στη χρήση βοηθητικών πινάκων ή διανυσμάτων. Όπως και πολλοί συλλέκτες, ο αλγόριθμος Compressor χρησιμοποιεί ως διάνυσμα σήμανσης ένα bitmap όπου κάθε bit αντιστοιχεί σε μία λέξη μνήμης. Η φάση της σήμανσης θέτει σε λογικό 1 τα bit που αντιστοιχούν στην πρώτη και τελευταία λέξη ενός ζωντανού αντικειμένου. Εξετάζοντας τη δομή αυτή, ο συλλέκτης είναι σε θέση να προσδιορίσει κατά τη διάρκεια της συμπύκνωσης το μέγεθος ενός οιουδήποτε (ζωντανού) αντικειμένου.

Επιπλέον, ο αλγόριθμος χρησιμοποιεί έναν πίνακα όπου αποθηκεύει τις διευθύνσεις προορισμού των αντικειμένων. Καθώς η αποθήκευση της διεύθυνσης κάθε ενός αντικειμένου ξεχωριστά θα ήταν απαγορευτική, ο αλγόριθμος διαιρεί το σωρό σε μικρά, ισομεγέθη μπλοκ με τυπικό μέγεθος 256 ή 512 bytes. Ο πίνακας $offset$ αποθηκεύει τη διεύθυνση προορισμού

του πρώτου αντικειμένου για κάθε μπλοκ. Οι νέες διευθύνσεις των υπολοίπων ζωντανών αντικειμένων μπορούν να υπολογισθούν με χρήση του πίνακα $offset$ και του διανύσματος σήμανσης. Ομοίως, δοθείσης μιας αναφοράς προς ένα αντικείμενο, μπορούμε να υπολογίσουμε το μπλοκ στο οποίο αντιστοιχεί και επομένως να εξάγουμε τη διεύθυνση προορισμού από την κατάλληλη καταχώριση στον πίνακα $offset$ και τα bit σήμανσης για αυτό το μπλοκ. Αυτό επιτρέπει στον αλγόριθμο να αντικαταστήσει τα πολλαπλά περάσματα στο σωρό για μετακίνηση αντικειμένων και επιδιόρθωση δεικτών με ένα απλό και γρήγορο πέρασμα του διανύσματος σήμανσης για την κατασκευή του πίνακα $offset$, καθώς και άλλο ένα πέρασμα για τη μετακίνηση αντικειμένων και την ενημέρωση των δεικτών συμβουλευόμενος τις βοηθητικές αυτές δομές.

Αλγόριθμος 3.4 Συλλογή με σήμανση και εκκαθάριση: ο αλγόριθμος συμπύκνωσης Compressor

```
1:  procedure COMPACT()
2:      computeLocations(HeapStart, HeapEnd, HeapStart)
3:      updateReferencesRelocate(HeapStart, HeapEnd)

4:  procedure computeLocations(start, end, toRegion)
5:      for b ← 0 to numBits(start, end) − 1 do
6:          if b mod BITS_IN_BLOCK = 0 then              ▷ crossed boundary?
7:              offset[block] ← loc                      ▷ first object will be moved to loc
8:              block ← block + 1
9:          if bitmap[b] = MARKED then
10:             loc ← loc + BYTES_PER_BIT                ▷ advance by size of live objects

11: procedure newAddress(old)
12:     block ← getBlockNum(old)
13:     return offset[block] + offsetInBlock(old)

14: procedure updateReferencesRelocate(start, end)
15:     for all fld in Roots do
16:         ref ← *fld
17:         if ref ≠ null then
18:             *fld ← newAddress(ref)
19:     scan ← start
20:     while scan < end do
21:         scan ← nextMarkedObject(scan)                ▷ use the bitmap
22:         for all fld in Pointers(scan) do             ▷ update references
23:             ref ← *fld
24:             if ref ≠ null then
25:                 *fld ← newAddress(ref)
26:         dest ← newAddress(scan)
27:         move(scan, dest)
```

Αφού η ρουτίνα computeLocations υπολογίσει το πίνακα $offset$, η ρουτίνα updateReferencesRelocate ενημερώνει αρχικά τους δείκτες των αντικειμένων ριζών. Ο αλγόριθμος Lisp 2 ξεχωρίζει τα περάσματα της μετακίνησης αντικειμένων και της ενημέρωσης δεικτών, καθώς οι πληροφορίες της μετακίνησης αποθηκεύονται στο σωρό και χάνονται κατά

τη μετακίνηση αντικειμένων. Αντίθετα, ο αλγόριθμος Compressor μετακινεί τα ζωντανά αντικείμενα και ενημερώνει τους δείκτες σε ένα μόνο πέρασμα, καθώς οι νέες διευθύνσεις των αντικειμένων υπολογίζονται εύκολα από τον πίνακα $offset$ και το bitmap σήμανσης και δε χρειάζεται να αποθηκευθούν στο σωρό. Η ρουτίνα NEWADDRESS υπολογίζει τη νέα διεύθυνση ενός ζωντανού αντικειμένου. Αρχικά υπολογίζει το μπλοκ στο οποίο βρίσκεται το αντικείμενο και χρησιμοποιεί την τιμή για τη δεικτοδότηση του πίνακα $offset$ ώστε να βρει τη διεύθυνση προορισμού του **πρώτου** ενεργού αντικειμένου στο συγκεκριμένο μπλοκ. Η νέα διεύθυνση του αντικειμένου προκύπτει προσθέτοντας στην προηγούμενη τιμή το μέγεθος σε λέξεις των ζωντανών αντικειμένων που προηγούνται του αντικειμένου στο μπλοκ, δεδομένο που υπολογίζει συμβουλευόμενη το bitmap που αντιστοιχεί στο εν λόγω μπλοκ.

3.4 Θέματα προς εξέταση

3.4.1 Είναι απαραίτητη η συμπύκνωση;

Η συλλογή με σήμανση και εκκαθάριση έχει μικρότερες απαιτήσεις σε μνήμη από ότι άλλες τεχνικές όπως η συλλογή με αντιγραφή. Επιπλέον, καθώς δεν μετακινεί αντικείμενα, ένας συλλέκτης σήμανσης και εκκαθάρισης αρκείται στο να εντοπίσει τα προσβάσιμα από τις ρίζες αντικείμενα και να τα σημάνει: δεν τα μεταβάλλει. Περιβάλλοντα όπου το σύστημα εκτέλεσης δε μπορεί να παρέχει έγκυρη πληροφορία τύπων ώστε να προσδιοριστούν με βεβαιότητα οι μεταβλητές δείκτες δυσχεραίνουν τον προσδιορισμό και τη σήμανση αντικειμένων που είναι προσβάσιμα από τις ρίζες του τροποποιητή. Παρόλα αυτά, το σημαντικότερο μειονέκτημα ενός συλλέκτη σήμανσης και εκκαθάρισης είναι η τάση του να κατακερματίζει τη μνήμη. Η χρήση προηγμένων τεχνικών για την εκχώρηση μνήμης όπως η χρήση ξεχωριστών συνδεδεμένων ελεύθερων λιστών για αντικείμενα διαφορετικού μεγέθους μειώνει την πιθανότητα για κατακερματισμό της μνήμης μόνο σε εφαρμογές με μικρές απαιτήσεις σε μεγάλα αντικείμενα και με σχετικά μη μεταβλητή αναλογία μεγεθών αντικειμένων. Σε μία τυπική εφαρμογή η οποία τρέχει για πολύ χρόνο και στην οποία χρησιμοποιούνται αντικείμενα διαφόρων μεγεθών, ο κατακερματισμός που επιφέρει ένας μη-μετακινών συλλέκτης αποτελεί πηγή προβλημάτων. Γι αυτόν το λόγο, οι περισσότερες εμπορικές υλοποιήσεις της εικονικής μηχανής της γλώσσας Java χρησιμοποιούν μετακινούντες συλλέκτες που συμπυκνώνουν το σωρό.

3.4.2 Επίδραση στη ρυθμαπόδοση

Η συλλογή με σήμανση και συμπύκνωση ωστόσο τείνει να είναι πιο αργή από τη συλλογή με σήμανση και εκκαθάριση ή τη συλλογή με αντιγραφή. Ο λόγος είναι πως με εξαίρεση τον αλγόριθμο Compressor, ο συλλέκτης πραγματοποιεί πολλαπλά περάσματα στο σωρό, με το κάθε πέρασμα να είναι ακριβό. Ο Printezis [95] και οι Soman κ.ά. [110] προτείνουν τη χρήση προσαρμοστικής συλλογής απορριμμάτων: ο συλλέκτης εκτελεί τον αλγόριθμο με σήμανση και εκκαθάριση και προσφεύγει στον αλγόριθμο με σήμανση και συμπύκνωση όταν οι μετρικές κατακερματισμού υποδεικνύουν πώς η αλλαγή θα είναι επικερδής.

3.4.3 Μακρόβια αντικείμενα

Δεν είναι ασυνήθιστο για αντικείμενα που ζουν πολύ ή είναι αθάνατα (δηλαδή ζουν καθ' όλη τη διάρκεια εκτέλεσης του τροποποιητή) να συσσωρεύονται στην αρχή του σωρού από μετακινούντες συλλέκτες. Οι συλλέκτες αντιγραφής διαχειρίζονται μη αποδοτικά τα αντικείμενα

αυτά, αντιγράφοντάς τα συνέχεια από τον ένα ημιχώρο στον άλλον. Από την άλλη πλευρά, οι γενεαλογικοί συλλέκτες (τους οποίους εξετάζουμε στο κεφάλαιο 7), τα προωθούν σε διαφορετικό τμήμα του σωρού, το οποίο και συλλέγεται σπανίως. Η γενεαλογική συλλογή απορριμμάτων ωστόσο δεν ενδείκνυται πάντα και ιδιαίτερα όταν ο σωρός είναι σχεδόν γεμάτος. Επίσης δε λύνει το πρόβλημα αν τα αντικείμενα βρίσκονται ήδη αποθηκευμένα στην παλαιότερη γενεά. Η συλλογή με σήμανση και συμπύκνωση μπορεί να επιλέξει τη μη συμπύκνωση αυτού του "ιζήματος". Ο Hanson [57] πρώτος παρατήρησε την εμφάνιση του προβλήματος στο σύστημα SITBOL, το οποίο αποτελούσε μία υλοποίηση της γλώσσας SNOBOL4 για αρχιτεκτονική DEC-10. Ο Hanson πρότεινε τη δυναμική μέτρηση του μεγέθους του "ιζήματος" και τη μη συλλογή αυτού εκτός και αν είναι τελείως απαραίτητη, με τίμημα ένα μικρό ποσοστό κατακερματισμού.

3.4.4 Τοπικότητα

Όπως είδαμε, οι συλλέκτες σήμανσης και συμπύκνωσης μπορεί να διατηρούν τη διάταξη των αντικειμένων στο σωρό είτε να τα αναδιατάσσουν τυχαία. Παρότι οι δεύτεροι είναι ταχύτεροι και χωρίς κόστος σε χώρο, η τοπικότητα του τροποποιητή ενδέχεται να καταστραφεί τελείως από την τυχαία διάταξη των αντικειμένων. Η ολισθαίνουσα συμπύκνωση έχει τέλος ένα ακόμη πλεονέκτημα: η ανάκτηση της συνολικής μνήμης που καταλαμβάνεται από αντικείμενα μετά από ένα συγκεκριμένο όριο στο σωρό δύναται να ανακτηθεί αμέσως με μία απλή μείωση του δείκτη $free$.

3.4.5 Περιορισμοί

Στη βιβλιογραφία έχει προταθεί μια πληθώρα αλγορίθμων συλλογής με σήμανση και συμπύκνωση. Πολλοί από αυτούς ωστόσο έχουν ιδιότητες μη επιθυμητές και σε αρκετές περιπτώσεις απαράδεκτες. Ένα από τα ζητήματα αφορά το κόστος σε χώρο που επιβάλλει η ανάγκη αποθήκευσης των διευθύνσεων προώθησης (παρότι αυτό είναι μικρότερο από το αντίστοιχο κόστος της συλλογής με αντιγραφή). Μερικοί αλγόριθμοι συμπύκνωσης επιβάλλουν περιορισμούς στον τροποποιητή. Απλοί αλγόριθμοι όπως ο Two-Finger μπορούν να διαχειριστούν μόνο αντικείμενα του ίδιου μεγέθους. Εφόσον λοιπόν είναι δυνατός ο διαχωρισμός των αντικειμένων με βάση το μέγεθός τους, σε ποιο βαθμό είναι απαραίτητη η συμπύκνωση; Τέλος, οι περισσότεροι αλγόριθμοι συμπύκνωσης αποκλείουν τη χρήση εσωτερικών δεικτών: ο αλγόριθμος Two-Finger αποτελεί εξαίρεση.

Κεφάλαιο 4

Συλλογή απορριμμάτων με αντιγραφή

Η συλλογή με σήμανση και εκκαθάριση έχει σχετικά μικρό κόστος, ωστόσο ενδέχεται η μνήμη να κατακερματίζεται. Η συλλογή απορριμμάτων πρέπει να καταλαμβάνει τον ελάχιστο δυνατό χρόνο εκτέλεσης σε ένα καλώς διαμορφωμένο σύστημα και καθώς το κόστος των εργασιών μνήμης του τροποποιητή κυριαρχεί έναντι αυτού του συλλέκτη, η εκχώρηση μνήμης πρέπει να είναι γρήγορη. Η συλλογή με σήμανση και συμπύκνωση εξαλείφει τον κατακερματισμό της μνήμης και υποστηρίζει πολύ γρήγορη εκχώρηση μνήμης με τίμημα την αύξηση του χρόνου συλλογής αφού ο σωρός διατρέχεται πολλές φορές. Στο παρόν κεφάλαιο παρουσιάζεται η **συλλογή απορριμμάτων με αντιγραφή**, η οποία επινοήθηκε από τους Fenichel και Yochelson [51] και Cheney [34]. Η συλλογή με αντιγραφή συμπυκνώνει το σωρό επιτρέποντας γρήγορη εκχώρηση μνήμης και επίσης πραγματοποιεί μόνο ένα πέρασμα στο σωρό. Το βασικό μειονέκτημα είναι πως το μέγεθος της διαθέσιμης μνήμης στο σωρό μειώνεται στο μισό.

4.1 Αντιγραφή μεταξύ ημιχώρων

Η βασική εκδοχή ενός συλλέκτη αντιγραφής, η οποία φαίνεται στον αλγόριθμο 4.2 διαιρεί το σωρό σε δύο ισομεγέθεις **ημιχώρους**, το **χώρο-από** και το **χώρο-προς**. Εάν υπάρχει διαθέσιμη μνήμη για ένα αντικείμενο, αυτή εκχωρείται στο χώρο-προς αυξάνοντας την τιμή ενός δείκτη $free$ (αλγόριθμος 4.1). Διαφορετικά, ο ρόλος των ημιχώρων εναλλάσσεται πριν ο συλλέκτης αντιγράψει όλα τα ζωντανά αντικείμενα από τον πρώην χώρο-προς (και νυν χώρο-από) στο νυν χώρο-προς (πρώην χώρο-από). Στο τέλος της συλλογής, όλα τα ζωντανά αντικείμενα είναι τοποθετημένα σε ένα πυκνό πρόθεμα του χώρου-προς. Ο συλλέκτης αγνοεί το χώρο-από και τα αντικείμενα που βρίσκονται σε αυτόν μέχρι την επόμενη συλλογή. Στην πράξη βέβαια, οι περισσότεροι συλλέκτες αντιγραφής μηδενίζουν το χώρο-από για ασφάλεια κατά την αρχικοποίηση του επόμενου κύκλου συλλογής.

Μετά την αρχικοποίηση, οι συλλέκτες αντιγραφής αντιγράφουν τα αντικείμενα ρίζες στο χώρο-προς. Τα αντικείμενα που έχουν αντιγραφεί αλλά όχι ακόμα εξεταστεί είναι γκρι. Κάθε πεδίο δείκτης ενός γκρι αντικειμένου περιέχει είτε την τιμή **null** είτε μία αναφορά προς ένα αντικείμενο του χώρου-από. Η διαδικασία SCAN ενημερώνει κάθε πεδίο-δείκτη ενός γκρι αντικειμένου, ο οποίος δείχνει σε κάποιο αντικείμενο του χώρου-από, με τη διεύθυνση του αντιγράφου του αντικειμένου στο χώρο-προς.

Αλγόριθμος 4.1 Συλλογή με αντιγραφή: αρχικοποίηση και εκχώρηση

```
1:  procedure CREATESEMISPACES()
2:      tospace ← HeapStart
3:      extent ← (HeapEnd − HeapStart)/2          ▷ size of a semispace
4:      top ← fromspace
5:      free ← tospace

6:  function ALLOCATE(size)
7:      atomic
8:      result ← free
9:      newfree ← result + size
10:     if newFree > top then
11:         return null                           ▷ signal "memory exhausted"
12:     free ← newfree
13:     return result
```

Πιο συγκεκριμένα, για κάθε πεδίο δείκτη του ορίσματος της, καλεί τη διαδικασία PROCESS, η οποία με τη σειρά της, αν το όρισμα της δεν περιέχει την τιμή **null**, καλεί τη συνάρτηση FORWARD με όρισμα το όρισμα με το οποίο αυτή κλήθηκε. Η συνάρτηση FORWARD ελέγχει αν το αντικείμενο αναφοράς του ορίσματος της έχει ήδη αντιγραφεί στο χώρο-προς, και αν ναι, τότε επιστρέφει τη διεύθυνση του αντιγράφου. Αν όχι, καλεί τη συνάρτηση COPY η οποία εκτελεί τις εξής ενέργειες: αποθηκεύει στη μεταβλητή $toRef$ την τιμή του δείκτη $free$, αυξάνει το δείκτη $free$ κατά το μέγεθος του αντικειμένου (όπως ακριβώς συμβαίνει και κατά την εκχώρηση μνήμης), μετακινεί (αντιγράφει) το αντικείμενο στη θέση $toRef$, αποθηκεύει στο πεδίο $forwardingAddress$ του αντικειμένου τη διεύθυνση του αντιγράφου του $toRef$ και τέλος, πριν επιστρέψει τη μεταβλητή $toRef$, την προσθέτει στη λίστα εργασιών. Η αποθήκευση της διεύθυνσης ενός αντικειμένου στο πεδίο $forwardingAddress$ του αντιγράφου του στο χώρο-από από τη συνάρτηση COPY εξασφαλίζει πως η συλλογή με αντιγραφή διατηρεί την τοπολογία των ζωντανών αντικειμένων κατά την αντιγραφή τους από το χώρο-από στο χώρο-προς. Σε αντίθεση με τους περισσότερους συλλέκτες με σήμανση και συμπύκνωση, ένας συλλέκτης με αντιγραφή δεν απαιτεί κανένα επιπλέον πεδίο στην επικεφαλίδα ενός αντικειμένου. Οποιοδήποτε πεδίο στο αντίγραφο ενός αντικειμένου του χώρου-προς στο χώρο-από μπορεί να αποθηκεύσει τη διεύθυνση του αντικειμένου στο χώρο-προς, καθώς το αντίγραφο αυτό δεν πρόκειται να χρησιμοποιηθεί ξανά μετά το πέρας της συλλογής. Το γεγονός αυτό καθιστά τη συλλογή με αντιγραφή κατάλληλη ακόμη και για αντικείμενα χωρίς επικεφαλίδα.

Όπως και οι περισσότερες τεχνικές συλλογής απορριμμάτων με εξιχνίαση, έτσι και η συλλογή με αντιγραφή διατηρεί μια λίστα εργασιών την οποία και επεξεργάζεται μέχρις ότου αυτή εκκενωθεί. Η λίστα εργασιών δύναται να υλοποιηθεί με διαφορετικούς τρόπους, οδηγώντας σε διαφορετική σειρά διάσχισης του γράφου αντικειμένων και διαφορετικές απαιτήσεις σε χώρο. Οι Fenichel και Yochelson [51] υλοποιούν τη λίστα εργασιών ως μία απλή βοηθητική στοίβα, όπως κάνουν οι συλλέκτες με σήμανση και εκκαθάριση. Η αντιγραφή τελειώνει όταν η στοίβα είναι άδεια.

Ο αλγόριθμος του Cheney [34] χρησιμοποιεί τα γκρι αντικείμενα του χώρου-προς ως μία ουρά (αλγόριθμος 4.3). Η μόνη επιπλέον μεταβλητή που χρησιμοποιεί είναι ο δείκτης $scan$, ο οποίος κάθε χρονική στιγμή αναφέρεται στο επόμενο μη σαρωμένο αντικείμενο. Όταν οι δύο ημιχώροι εναλλάσσονται οι δείκτες $scan$ και $free$ αρχικοποιούνται ώστε να δείχνουν στην αρχή του χώρου-προς (η διαδικασία INITIALIZE εγγράφει το δείκτη $scan$ με την τιμή

Αλγόριθμος 4.2 Συλλογή με αντιγραφή: αντιγραφή ημιχώρων

```
 1: procedure COLLECT()
 2:     FLIP()
 3:     INITIALIZE(worklist)                                   ▷ empty
 4:     for all fld in Roots do                                ▷ copy the roots
 5:         PROCESS(fld)
 6:     while not ISEMPTY(worklist) do                         ▷ copy transitive closure
 7:         ref ← REMOVE(worklist)
 8:         SCAN(ref)

 9: procedure FLIP()                                            ▷ switch semispaces
10:     fromspace, tospace ← tospace, fromspace
11:     top ← tospace + extent
12:     free ← tospace

13: procedure SCAN(ref)
14:     for all fld in Pointers(Roots) do
15:         PROCESS(fld)

16: procedure PROCESS(fld)          ▷ update field with reference to tospace replica
17:     fromRef ← *fld
18:     if fromRef ≠ null then
19:         *fld ← FORWARD(fromRef)                            ▷ update with tospace reference

20: function FORWARD(fromRef)
21:     toRef ← FORWARDINGADDRESS(fromRef)
22:     if toRef = null then                                   ▷ not copied (not marked)
23:         toRef ← COPY(fromRef)
24:     return toRef

25: function COPY(fromRef)          ▷ copy object and return forwarding address
26:     toRef ← free
27:     free ← free + SIZE(fromRef)
28:     MOVE(fromRef, toRef)
29:     FORWARDINGADDRESS(fromRef) ← toRef                     ▷ mark
30:     ADD(worklist, toRef)
31:     return toRef
```

του δείκτη $free$, ο οποίος στη διαδικασία FLIP ενημερώθηκε με την διεύθυνση της αρχής του ημιχώρου-προς). Αφού αντιγραφούν τα αντικείμενα ρίζες, η λίστα εργασιών, δηλαδή το σύνολο των γκρι αντικειμένων αποτελείται ακριβώς από τα αντιγεγραμμένα και όχι ακόμη σαρωμένα αντικείμενα στο χώρο-προς μεταξύ των δεικτών $scan$ και $free$. Αυτή ακριβώς είναι και η αναλλοίωτη που διατηρείται κατά τη διάρκεια της συλλογής. Ο δείκτης $scan$ αυξάνεται καθώς πεδία αντικειμένων του χώρου-προς σαρώνονται και ενημερώνονται. Η συλλογή ολοκληρώνεται όταν η λίστα εργασιών είναι κενή: όταν ο δείκτης $scan$ συναντήσει το δείκτη $free$. Η υλοποίηση είναι πολύ απλή: η συνάρτηση ISEMPTY συγκρίνει τις τιμές των δεικτών $scan$ και $free$, η συνάρτηση REMOVE απλώς επιστρέφει την τιμή του δείκτη $scan$ και τέλος η διαδικασία ADD δεν προβαίνει σε καμία ενέργεια.

Αλγόριθμος 4.3 Συλλογή με αντιγραφή: λίστα εργασιών του Cheney

```
1:  procedure INITIALIZE()
2:      scan ← free

3:  function ISEMPTY(worklist)
4:      return scan = free

5:  function REMOVE(worklist)
6:      ref ← scan
7:      scan ← scan + SIZE(scan)
8:      return ref

9:  procedure ADD(worklist, ref)
10:     /* nop */
```

4.2 Σειρά διάταξης και τοπικότητα

Η τοπικότητα τόσο του τροποποιητή όσο και του συλλέκτη μπορεί να έχει σημαντική επίδραση στην επίδοση ενός προγράμματος. Όπως είδαμε στο προηγούμενο κεφάλαιο και παρατηρούν οι Abuaiadh κ.ά. [2], ο συλλέκτης μπορεί να βλάψει την τοπικότητα του τροποποιητή και επομένως και την επίδοση του τελευταίου εάν μετακινεί αντικείμενα σε τυχαίες νέες θέσεις χωρίς να λαμβάνει υπόψη σχέσεις μεταξύ δεικτών ή την αρχική διάταξη των αντικειμένων. Ωστόσο, απαιτείται ο συμβιβασμός ανάμεσα στα οφέλη που προκύπτουν από την τοπικότητα του τροποποιητή και τη συχνότητα κλήσης του συλλέκτη. Ας συγκρίνουμε για παράδειγμα τη συλλογή με σήμανση και εκκαθάριση και τη συλλογή με αντιγραφή. Η συλλογή με σήμανση και εκκαθάριση έχει στη διάθεσή της διπλάσιο χώρο στο σωρό από ότι η συλλογή με αντιγραφή και αν υποθέσουμε πως οι υπόλοιπες παράμετροι είναι ίδιες καταλήγουμε στο συμπέρασμα πως η χρήση της πρώτης τεχνικής θα πραγματοποιήσει μισό αριθμό συλλογών. Οι Blackburn κ.ά. [18] διαπιστώνουν πως πράγματι η συλλογή με σήμανση και εκκαθάριση παρουσιάζει καλύτερες συνολικά επιδόσεις από τη συλλογή με αντιγραφή όταν ο σωρός είναι σχεδόν γεμάτος. Αντίθετα, οι Blackburn και McKinley [17] βρίσκουν πως σε μεγάλους σωρούς, τα οφέλη τοπικότητας που προκύπτουν από τη σειριακή εκχώρηση μνήμης υπερνικούν την αποδοτικότητα χώρου της συλλογής με σήμανση και εκκαθάριση, έχοντας ως αποτέλεσμα καλύτερους ρυθμούς αστοχιών σε όλα τα επίπεδα της κρυφής μνήμης.

Οι Blackburn κ.ά. [18] αντιγράφουν τα αντικείμενα κατά βάθος. Αντίθετα, ο Cheney διασχίζει

το γράφο των αντικειμένων κατά πλάτος. Παρότι η διάσχιση υλοποιείται ως μία γραμμική (και συνεπώς προβλέψιμη) σάρωση της λίστας εργασιών των γκρι αντικειμένων του χώρου-προς, η αντιγραφή αντικειμένων κατά πλάτος επιδρά δυσμενώς στην τοπικότητα του τροποποιητή καθώς τείνει να ξεχωρίζει γονείς και παιδιά.

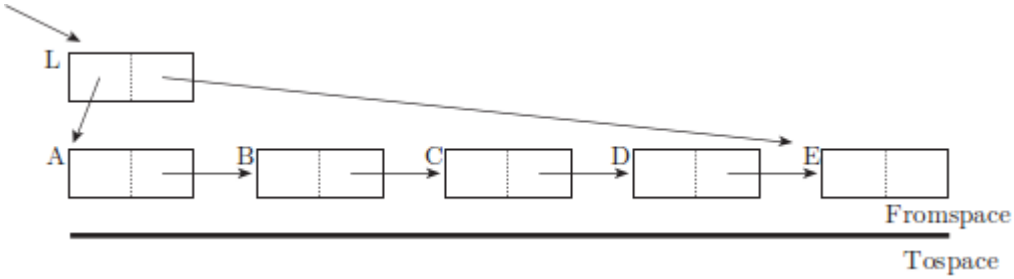
Ο White [118] ήταν ο πρώτος που συνέστησε τη χρήση του συλλέκτη προς βελτίωση της επίδοσης του τροποποιητή. Τόσο η συλλογή με αντιγραφή όσο και η συλλογή με σήμανση και συμπύκνωση μετακινούν αντικείμενα, επηρεάζοντας με μεγάλη πιθανότητα την τοπικότητα του τροποποιητή. Η ολισθαίνουσα μετακίνηση θεωρείται η βέλτιστη όσον αφορά τη συλλογή με σήμανση και συμπύκνωση καθώς διατηρεί τη διάταξη των αντικειμένων που έχει εγκαθιδρύσει ο εκχωρητής. Η συντηρητική αυτή πολιτική είναι σίγουρα ασφαλής. Είναι όμως και βέλτιστη; Η συλλογή με σήμανση και συμπύκνωση, η οποία συμπυκνώνει το σωρό επιτόπου είτε μετακινώντας ζωντανά αντικείμενα σε τρύπες είτε ολισθαίνοντας τα στο ένα άκρο του σωρού, δεν έχει πολλές δυνατότητες τροποποίησης της διάταξης των αντικειμένων στο σωρό προς όφελος της τοπικότητας του τροποποιητή. Από την άλλη πλευρά, οποιοσδήποτε αλγόριθμος συλλογής ο οποίος μετακινεί αντικείμενα σε μία φρέσκια περιοχή του σωρού χωρίς να καταστρέφει τα αυθεντικά δεδομένα μπορεί να τα αναδιατάξει προκειμένου να βελτιώσει την επίδοση του τροποποιητή.

Δυστυχώς υπάρχουν δύο λόγοι για τους οποίους δεν μπορούμε να βρούμε μία βέλτιστη διάταξη των αντικειμένων που να ελαχιστοποιεί τον αριθμό αστοχιών κρυφής μνήμης του προγράμματος. Πρώτον, ο συλλέκτης δεν μπορεί να γνωρίζει το μοτίβο πρόσβασης των μελλοντικών προσβάσεων σε αντικείμενα. Ακόμη χειρότερα, οι Petrank και Rawitz [92] αποδεικνύουν πως το πρόβλημα της τοποθέτησης είναι NP-complete: δεδομένης της τέλειας γνώσης των μελλοντικών προσβάσεων, δεν υπάρχει αποδοτικός αλγόριθμος υπολογισμού μιας βέλτιστης τοποθέτησης.

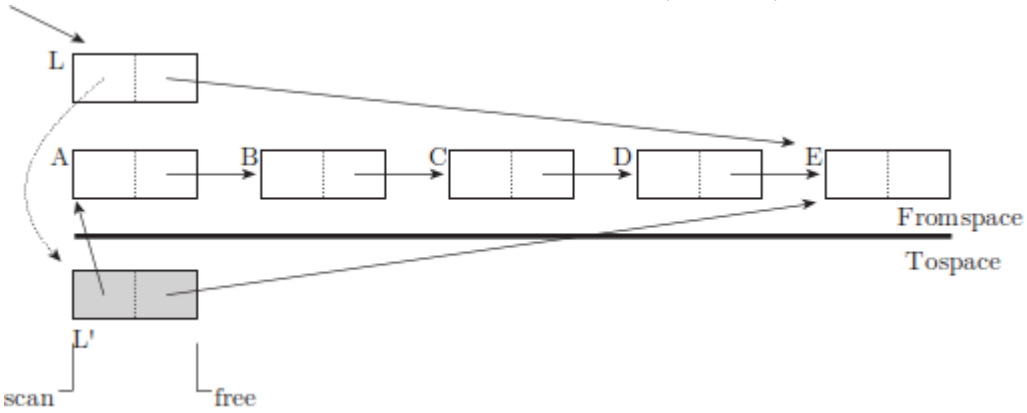
Αλγόριθμος 4.4 Συλλογή με αντιγραφή: σχεδόν κατά-βάθος αντιγραφή (Moon)

```
1:  procedure INITIALIZE(worklist)
2:      scan ← free
3:      partialScan ← free

4:  procedure ISEMPTY(worklist)
5:      return scan = free                                  ▷ as per Cheney

6:  procedure REMOVE(worklist)
7:      if partialScan < free then
8:          ref ← partialScan                               ▷ prefer secondary scan
9:          partialScan ← partialScan + SIZE(partialScan)
10:     else
11:         ref ← scan                                      ▷ primary scan
12:         scan ← scan + SIZE(scan)
13:     return ref

14: procedure ADD(worklist, ref) ▷ secondary scan on the most recently allocated page
15:     partialScan ← MAX(partialScan, startOfPage(ref))
```

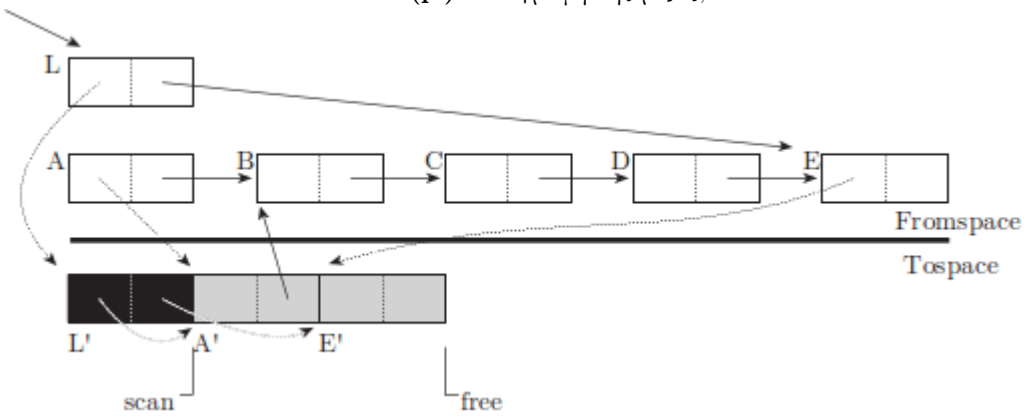
Η μοναδική λύση στο πρόβλημα είναι η χρήση ευριστικών. Μια πιθανή προσέγγιση είναι η χρήση της συμπεριφοράς του παρελθόντος για την πρόβλεψη της συμπεριφοράς του μέλλο-



(α′) Ο χώρος-προς πριν τη συλλογή

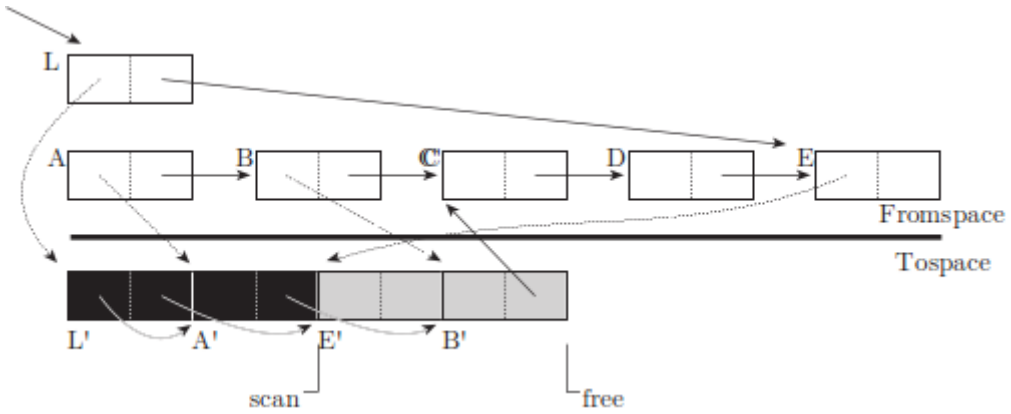


(β′) Αντιγραφή της ρίζας, L

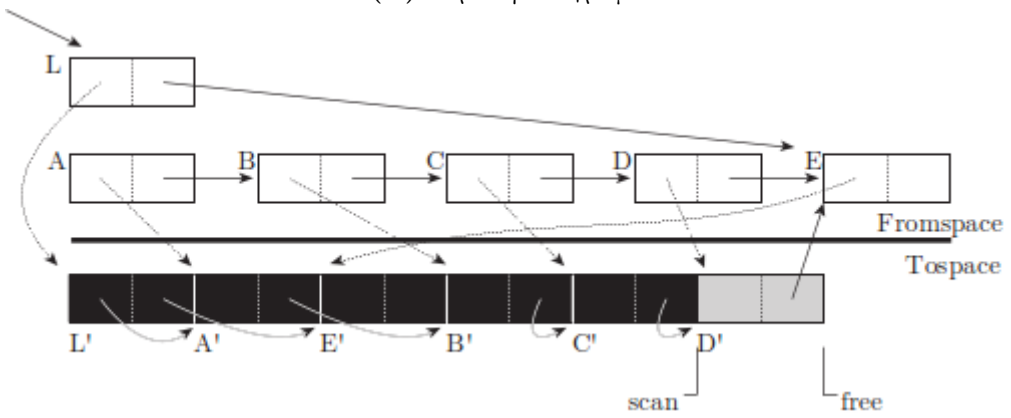


(γ′) Σάρωση του αντιγράφου του L

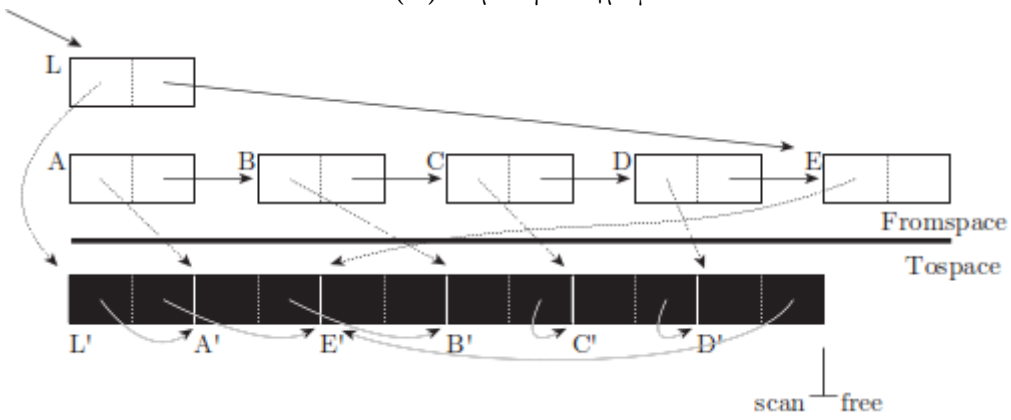
Σχήμα 4.1: Συλλογή με αντιγραφή: ένα παράδειγμα



(δ′) Σάρωση αντιγράφου του κ.ο.κ



(ε′) Σάρωση αντιγράφου του C



(στ′) Σάρωση αντιγράφου του D. $scan = free$ και η συλλογή ολοκληρώνεται.

Σχήμα 4.1: Συλλογή με αντιγραφή: ένα παράδειγμα (συνέχεια)

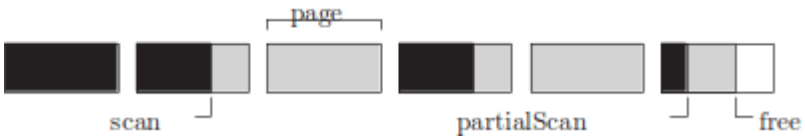
ντος. Ορισμένοι ερευνητές χρησιμοποιούν είτε στατιστική ανάλυση βασιζόμενοι στην υπόθεση των Calder κ.ά. [32] πως τα προγράμματα συμπεριφέρονται παρόμοια για διαφορετικές εισόδους, είτε online δειγματοληψία βασιζόμενοι στην υπόθεση των Chilimbi κ.ά. [36] πως η συμπεριφορά του προγράμματος παραμένει η ίδια από μία περίοδο στην επόμενη. Μια άλλη ευριστική είναι η διατήρηση της διάταξης εκχώρησης, όπως κάνει η ολισθαίνουσα συμπύκνωση. Μια τρίτη στρατηγική αφορά την προσπάθεια τοποθέτησης παιδιών δίπλα σε έναν από τους γονείς τους, αφού ο μόνος τρόπος πρόσβασης ενός παιδιού είναι η φόρτωση μιας αναφοράς από κάποιον από τους γονείς αυτού. Ο αλγόριθμος του Cheney υλοποιεί κατά πλάτος αντιγραφή τείνοντας όμως να τοποθετεί κοντά μακρινά ξαδέρφια και όχι γονείς με τα παιδιά τους.

Οι αρχικές μελέτες του πώς μπορεί να επωφεληθεί η τοπικότητα του τροποποιητή από τη σειρά αντιγραφής των αντικειμένων επικεντρώνονταν στην ελαχιστοποίηση των σφαλμάτων σελίδας: ο απώτερος στόχος είναι η τοποθέτηση σχετιζόμενων αντικειμένων στην ίδια σελίδα. Ο Stamos [112], [111] και ο Blau [22] διαπίστωσαν μέσω προσομοιώσεων πως παρότι η κατά βάθος αντιγραφή είναι πιο επικερδής από την κατά πλάτος αντιγραφή, αυτή οδηγεί σε χειρότερη συμπεριφορά όσον αφορά τα σφάλματα σελίδων σε σχέση με την αρχική διάταξη δημιουργίας των αντικειμένων. Οι Wilson κ.ά. [120] ωστόσο υποστηρίζουν πως οι εν λόγω προσομοιώσεις δε λαμβάνουν υπόψη τους την τοπολογία των πραγματικών προγραμμάτων των γλωσσών Lisp και Smalltalk, τα οποία τείνουν να δημιουργούν δένδρα με μεγάλο πλάτος και μικρό βάθος: οι δενδρικές δομές υλοποιούν πίνακες κατακερματισμού και έχουν σχεδιασθεί με στόχο την καλύτερη διασπορά των κλειδιών προς αποφυγή συγκρούσεων.

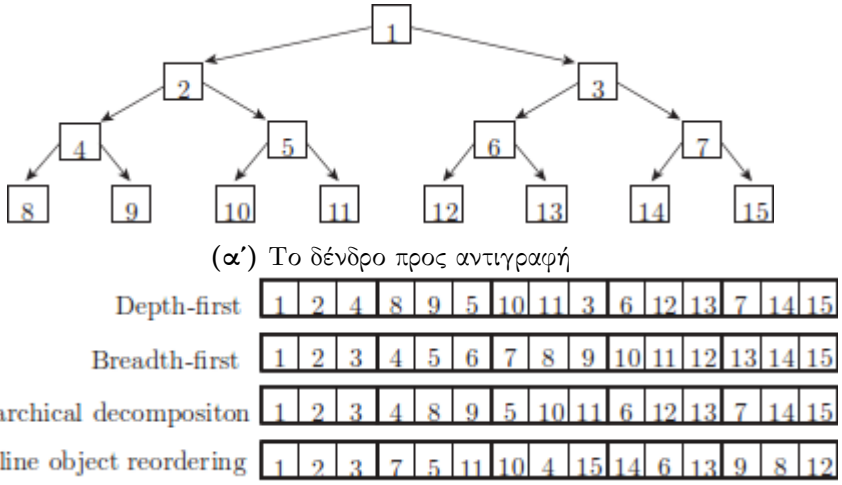
Ο αλγόριθμος των Fenichel και Yochelson υλοποιεί την κατά βάθος διάσχιση με τη χρήση μιας βοηθητικής στοίβας. Είναι πάντως δυνατή η επίτευξη μιας σχεδόν κατά πλάτος διάσχισης χωρίς τη χρήση βοηθητικής στοίβας. Ο Moon [83] τροποποίησε τον αλγόριθμο του Cheney για να πετύχει μία σχεδόν κατά βάθος διάσχιση. Ο αλγόριθμος του Moon (αλγόριθμος 4.4) χρησιμοποιεί εκτός του αρχικού δείκτη *scan* έναν επιπλέον δείκτη, τον *partialScan*. Οποτεδήποτε αντιγράφεται ένα αντικείμενο, ο αλγόριθμος του Moon εκκινεί μια δευτερεύουσα σάρωση στην τελευταία σελίδα του χώρου-προς η οποία δεν έχει ακόμη σαρωθεί πλήρως. Όταν αυτό συμβεί, η κύρια σάρωση συνεχίζει από την πρώτη σελίδα που δεν έχει σαρωθεί πλήρως. Στην πράξη, η λίστα εργασιών υλοποιείται ως ένα ζεύγος ουρών Cheney. Αυτή η **ιεραρχική αποσύνθεση** πλεονεκτεί έναντι της κλασσικής κατά βάθος εξερεύνησης όσον αφορά την τοποθέτηση γονέων και παιδιών στην ίδια σελίδα.

Το μειονέκτημα του αλγορίθμου του Moon είναι πως αντικείμενα μπορεί να σαρώνονται δύο φορές, αφού αυτός καταγράφει μόνο ένα ζεύγος δεικτών σάρωσης και κατά συνέπεια ξεχνά τα μπλοκ μεταξύ των θέσεων *scan* και $free$ που έχουν ήδη μερικώς σαρωθεί. Πράγματι, όπως διαπίστωσαν οι Wilson κ.ά. [120] το 30% των αντικειμένων ενδέχεται να σαρωθεί εις διπλούν. Οι τελευταίοι τροποποίησαν τον αλγόριθμο του Moon παρέχοντας σε κάθε σελίδα τους δικούς της δείκτες $free$ και *scan*, με τη λίστα εργασιών να περιλαμβάνει πλέον τα μπλοκ που έχουν σαρωθεί μερικώς. Αυτό σημαίνει πώς η κύρια σάρωση δεν χρειάζεται να επισκεφθεί ξανά αντικείμενα σελίδων που έχει ήδη επεξεργασθεί η δευτερεύουσα σάρωση.

Η προσέγγιση του Moon είναι στατική και δε λαμβάνει υπόψη τη συμπεριφορά των ξεχωριστών εφαρμογών. Είναι εμφανές ωστόσο πως τα οφέλη από την αναδιάταξη των αντικειμένων εξαρτώνται από τη συμπεριφορά του τροποποιητή. Οι Lam κ.ά. [74] διαπίστωσαν πως ο αλγόριθμος του Moon παρουσιάζει ευαισθησία όσον αφορά τη δομή και την ποικιλία των διαφόρων δομών δεδομένων και εμφανίζει απογοητευτικές επιδόσεις για μη δενδροειδείς δομές. Οι Siegwart και Hirzel [108] επίσης παρατήρησαν πως ένας παράλληλος συλλέκτης με ιεραρχική αποσύνθεση ωφελεί σημαντικά την επίδοση μόνο ορισμένων benchmarks.



Σχήμα 4.2: Προσεγγιστική αντιγραφή κατά βάθος (Moon). Κάθε μπλοκ αναπαριστά μια σελίδα. Ως συνήθως, τα σαρωμένα πεδία είναι μαύρα, ενώ τα πεδία που έχουν αντιγραφεί αλλά όχι σαρωθεί ακόμη γκρι. Ο ελεύθερος χώρος απεικονίζεται με λευκό χρώμα.



(α′) Το δένδρο προς αντιγραφή

(β′) Τοποθέτηση αντικειμένων στο σωρό μετά την αντιγραφή

Σχήμα 4.3: Αντιγραφή ενός δένδρου με διαφορετικές σειρές διάσχισης. Κάθε γραμμή δείχνει πώς η αντίστοιχη σειρά διάσχισης τοποθετεί τα αντικείμενα στο χώρο-προς, υποθέτοντας πώς κάθε σελίδα χωράει τρία αντικείμενα.

Οι Huang κ.ά. [63] παρακολουθούν δυναμικά το προφίλ μιας εφαρμογής και προσπαθούν να αντιγράψουν "θερμά" πεδία αντικειμένων δίπλα με τους γονείς αυτών. Αυτό το σχήμα **online επαναδιάταξης αντικειμένων** καθώς και η λειτουργία του φαίνονται στον αλγόριθμο 4.5 και το σχήμα 4.3 αντίστοιχα. Ο βασικός βρόχος σάρωσης του αλγορίθμου επεξεργάζεται πρώτα τη λίστα "θερμών" αντικειμένων και μετά τη λίστα "ψυχρών" αντικειμένων. Η υλοποίηση του μηχανισμού δειγματοληψίας ως τμήμα ενός προσαρμοστικού δυναμικού μεταγλωττιστή επιτρέπει τη φθηνή ταυτοποίηση των 'θερμών' αντικειμένων. Η υλοποίηση του Huang κ.ά. επίσης λαμβάνει υπόψη τις αλλαγές της συμπεριφοράς του τροποποιητή ανάμεσα στις διάφορες φάσεις λειτουργίας του, επιτρέποντας στη "θερμότητα" των πεδίων να φθίνει και να δειγματοληφθεί ξανά. Τέλος, ο Huang κ.ά. διαπίστωσαν πως η τεχνική της αντιγραφής με online αναδιάταξη αντικειμένων εμφανίζει συγκρίσιμες ή και καλύτερες επιδόσεις από στατικές τεχνικές όπως η αντιγραφή κατά πλάτος.

Αλγόριθμος 4.5 Συλλογή με αντιγραφή: online αναδιάταξη αντικειμένων

```
1:  procedure COLLECT()
2:      atomic
3:      FLIP()
4:      INITIALIZE(hotList, coldList)
5:      for all fld in Roots do
6:          ADVICEPROCESS(fld)
7:      repeat
8:          while not ISEMPTY(hotList) do
9:              ADVICESCAN(remove(hotList))
10:         while not ISEMPTY(coldList) do
11:             ADVICEPROCESS(remove(coldList))
12:     until ISEMPTY(hotList)

13: procedure INITIALIZE(hotList, coldList)
14:     hotList ← empty
15:     coldList ← empty

16: procedure ADVICEPROCESS(fld)
17:     fromRef ← *fld
18:     if fromRef ≠ null then
19:         *fld ← FORWARD(fromRef)

20: procedure ADVICESCAN(obj)
21:     for all fld in Pointers(obj) do
22:         if ISHOT(fld) then
23:             ADVICEPROCESS(fld)
24:         else
25:             ADD(coldList, fld)
```

4.3 Θέματα προς εξέταση

4.3.1 Εκχώρηση μνήμης

Η εκχώρηση μνήμης σε έναν συμπυκνωμένο σωρό είναι γρήγορη επειδή είναι απλή. Συνήθως απλώς απαιτεί τον έλεγχο έναντι ενός άνω ορίου (του σωρού ή ενός μπλοκ) και την υλοποίηση ενός δείκτη $free$. Εάν ο σωρός είναι οργανωμένος σε μπλοκ και όχι συνεχόμενος, ο έλεγχος ενίοτε θα αποτυγχάνει στην οποία περίπτωση πρέπει να δεσμευθεί ένα καινούριο μπλοκ. Η συχνότητα εμφάνισης της αργής αυτής περίπτωσης θα εξαρτηθεί από το πώς μεταβάλλεται ο μέσος όρος μεγέθους των δημιουργούμενων αντικειμένων και το μέγεθος του μπλοκ. Η σειριακή εκχώρηση μνήμης επίσης είναι κατάλληλη για χρήση σε πολυνηματικές εφαρμογές, καθώς σε κάθε νήμα-τροποποιητή μπορεί να δοθεί ένας τοπικός απομονωτής εκχώρησης χωρίς να χρειάζεται ο συγχρονισμός με άλλα νήματα. Η οργάνωση αυτή είναι απλή και απαιτεί τη χρήση μικρού αριθμού μεταδεδομένων, σε αντίθεση με τοπικά σχήματα εκχώρησης για μη μετακινούντες συλλέκτες όπου κάθε νήμα-τροποποιητής μπορεί να χρειάζεται τις δικές του δομές δεδομένων για την εκχώρηση μνήμης με ξεχωριστές ελεύθερες λίστες για αντικείμενα διαφορετικού μεγέθους.

Ο κώδικας που υλοποιεί τη σειριακή εκχώρηση είναι σύντομος και επιπλέον παρουσιάζει καλή συμπεριφορά όσον αφορά την κρυφή μνήμη καθώς η εκχώρηση προχωράει σειριακά στο σωρό. Παρότι ο συνδυασμός της σειριακής εκχώρησης, της μικρής διάρκειας ζωής αντικειμένων και της οργάνωσης του σωρού σε ημιχώρους συνεπάγεται πως με μεγάλη πιθανότητα η επόμενη θέση μνήμης που θα εκχωρηθεί είναι αυτή που έχει χρησιμοποιηθεί λιγότερο πρόσφατα, οι μηχανισμοί προφόρτωσης των σύγχρονων επεξεργαστών συνήθως καλύπτουν τη λανθάνουσα καθυστέρηση που προκύπτει. Εάν όμως η συμπεριφορά αυτή συγκρούεται με την πολιτική αντικατάστασης της λιγότερο πρόσφατα χρησιμοποιημένης σελίδας (LRU) του λειτουργικού συστήματος σε βαθμό ώστε ο ρυθμός εναλλαγής σελίδων να χειροτερεύει την επίδοση, τότε χρειάζεται επανεξέταση της διαμόρφωσης του συστήματος διαχείρισης μνήμης. Η ικανοποιητική εκτέλεση της εφαρμογής μπορεί να απαιτεί περισσότερη φυσική μνήμη ή αλλαγή της πολιτικής συλλογής.

Οι Blackburn κ.ά. [18] κατά την πειραματική μελέτη ενός μικρού σχετικά benchmark διαπίστωσαν πώς παρότι η σειριακή εκχώρηση υπερτερεί κατά 11% έναντι της εκχώρησης με ελεύθερες λίστες, η εκχώρηση καθεαυτή αποτελεί μόλις το 10% του συνολικού χρόνου εκτέλεσης μιας εφαρμογής. Επομένως η διαφορά κόστους ανάμεσα στις δύο τεχνικές μπορεί να είναι αμελητέα. Ωστόσο το κυρίαρχο κόστος της δημιουργίας ενός αντικειμένου συνήθως αφορά την αρχικοποίηση του αντικειμένου και όχι την εκχώρηση μνήμης. Επιπλέον, τα αντικείμενα έχουν παρόμοιους κύκλους ζωής σε πολλές εφαρμογές. Ο τροποποιητής δημιουργεί έναν αριθμό από σημασιολογικά συνδεδεμένα αντικείμενα περίπου την ίδια χρονική περίοδο, τα επεξεργάζεται και τα εγκαταλείπει μαζικώς. Σε τέτοιες εφαρμογές η χρήση συμπυκνωμένου σωρού προσφέρει υψηλή τοπική χωρικότητα αφού τα σχετιζόμενα αντικείμενα τοποθετούνται στην ίδια σελίδα ή ακόμη και στο ίδιο μπλοκ κρυφής μνήμης αν είναι μικρά. Μια τέτοια διάταξη τείνει εμφανώς να οδηγήσει σε χαμηλότερο ρυθμό αστοχιών κρυφής μνήμης σε σύγκριση με την περίπτωση όπου η μνήμη των συσχετιζόμενων αντικειμένων εκχωρείται μέσω ξεχωριστών ελεύθερων λιστών.

4.3.2 Χώρος και τοπικότητα

Το άμεσο μειονέκτημα της συλλογής με αντιγραφή ημιχώρων είναι η ανάγκη διατήρησης ενός δεύτερου ημιχώρου, ο οποίος μερικές φορές καλείται και **εφεδρικό αντίγραφο**. Δεδομένου ενός προϋπολογισμού μνήμης και αγνοώντας τις δομές δεδομένων που είναι απαραίτητες για τη λειτουργία του συλλέκτη η συλλογή με αντιγραφή προσφέρει μόνο το μισό χώρο του σωρού σε σχέση με άλλες τεχνικές συλλογής απορριμμάτων. Το αποτέλεσμα είναι πως οι συλλέκτες αντιγραφής θα πραγματοποιήσουν περισσότερους κύκλους συλλογής από ότι άλλοι συλλέκτες. Εάν αυτό θα οδηγήσει σε καλύτερη ή χειρότερη επίδοση εξαρτάται από τους διάφορους συμβιβασμούς μεταξύ συλλέκτη και τροποποιητή, τα χαρακτηριστικά της εφαρμογής και το μέγεθος της διαθέσιμης μνήμης σωρού.

Η απλή ασυμπτωτική ανάλυση πολυπλοκότητας μπορεί να προτιμήσει τη συλλογή με αντιγραφή έναντι της συλλογής με σήμανση και εκκαθάριση. Έστω M το συνολικό μέγεθος του σωρού και L το συνολικό μέγεθος των ζωντανών αντικειμένων. Η συλλογή με αντιγραφή πρέπει να αντιγράψει και σαρώσει τα ζωντανά αντικείμενα καθώς και να ενημερώσει τους δείκτες αυτών. Η συλλογή με σήμανση και εκκαθάριση από την άλλη πλευρά πρέπει να ανακαλύψει όλα τα ζωντανά αντικείμενα και στη συνέχεια να εκκαθαρίσει όλο το σωρό. Ο Jones [67] ορίζει τη χρονική πολυπλοκότητα για τη συλλογή με αντιγραφή και για τη συλλογή με σήμανση και εκκαθάριση ως:

$$t_{copy} = cL \tag{4.1}$$
$$t_{MS} = mL + sM \tag{4.2}$$

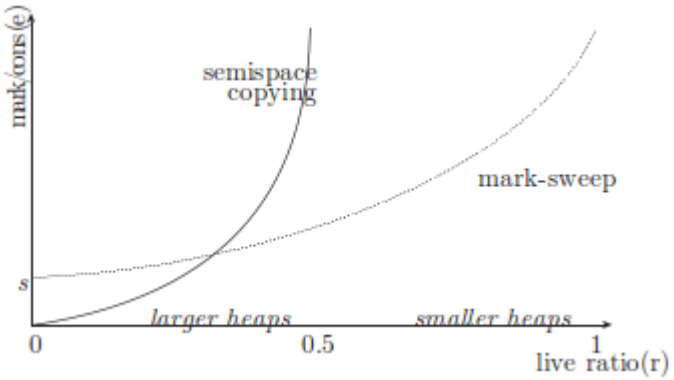
Το ποσό της μνήμης που ανακτάται από κάθε συλλογή είναι αντίστοιχα:

$$m_{copy} = \frac{M}{2} - L \tag{4.3}$$
$$m_{MS} = M - L \tag{4.4}$$

Έστω τώρα $r = \frac{L}{M}$ το ποσοστό της ζωντανής μνήμης, το οποίο υποθέτουμε πώς είναι σταθερό. Η αποδοτικότητα ενός αλγορίθμου μπορεί να περιγραφεί από το λόγο mark/cons, e, ο οποίος αντιπροσωπεύει την εργασία του συλλέκτη ανά μονάδα ανακτημένης μνήμης.

$$e_{copy} = \frac{2cr}{1 - 2r} \tag{4.5}$$
$$e_{MS} = \frac{mr + s}{1 - r} \tag{4.6}$$

Η συλλογή με αντιγραφή μπορεί να είναι πιο αποδοτική από τη συλλογή με σήμανση και εκκαθάριση αν ο σωρός είναι αρκετά μεγάλος και το r αρκετά μικρό. Αυτή η απλουστευμένη ανάλυση ωστόσο αγνοεί διάφορα ζητήματα. Οι σύγχρονοι συλλέκτες με σήμανση και εκκαθάριση συνήθως χρησιμοποιούν οκνηρή εκκαθάριση, κάτι που μειώνει τη σταθερά s και κατ' επέκταση το λόγο e_{MS}. Η ανάλυση της πολυπλοκότητας πρέπει να αντιμετωπισθεί με μεγάλη προσοχή καθώς συνήθως αγνοεί λεπτομέρειες υλοποίησης παρότι οι Hertz και Berger [60] επιβεβαιώνουν πειραματικά τη μορφή των καμπυλών (όπως ότι το κόστος της συλλογής



Σχήμα 4.4: Ρυθμοί mark/cons για συλλογή με σήμανση και εκκαθάριση και συλλογή με αντιγραφή.

με σήμανση και εκκαθάριση είναι αντιστρόφως ανάλογο προς το μέγεθος του σωρού). Οι πραγματικές λεπτομέρειες υλοποίησης παρότι είναι σημαντικές για πραγματικούς συλλέκτες δεν λαμβάνονται υπόψη από αναλύσεις πολυπλοκότητας. Παράδειγμα μιας τέτοιας λεπτομέρειας αποτελεί η θετική επίδραση στην τοπικότητα του τροποποιητή της χρήσης σειριακής εκχώρησης μνήμης.

Επομένως η σειριακή εκχώρηση μνήμης τείνει να τοποθετεί αντικείμενα που προσπελάζονται ταυτόχρονα σε διαδοχικές θέσεις μνήμης, κάτι που συμβάλλει στη βελτίωση του ρυθμού αστοχιών κρυφής μνήμης του τροποποιητή. Ωστόσο η συλλογή με αντιγραφή αναδιατάσσει τα επιζώντα αντικείμενα στο σωρό. Παρότι η συλλογή με αντιγραφή κατά Cheney δεν χρειάζεται βοηθητική στοίβα για την καθοδήγηση της εξιχνίασης, η διάσχιση κατά πλάτος τείνει να διαχωρίζει τους γονείς από τα παιδιά τους. Η ιεραρχική αποσύνθεση προσφέρει έναν καλό συμβιβασμό μεταξύ της πληρωμής του κόστους της βοηθητικής στοίβας και της βελτίωσης της διάταξης των αντικειμένων στο σωρό. Ωστόσο παρότι η προσεκτική αναδιάταξη των αντικειμένων μπορεί να είναι ωφέλιμη για μερικά προγράμματα, σε ορισμένες περιπτώσεις η επίδρασή της μπορεί να αποδειχθεί αμελητέα. Τα περισσότερα αντικείμενα έχουν μικρή διάρκεια ζωής και δεν επιβιώνουν ούτε από μία συλλογή. Επιπλέον, όπως παρατηρούν οι Blackburn και McKinley [17], πολλές εφαρμογές εστιάζουν τις προσβάσεις στη μνήμη και ιδιαίτερα τις εγγραφές σε αυτά τα νέα αντικείμενα. Η πολιτική διάσχισης ενός συλλέκτη δεν μπορεί προφανώς να επηρεάσει την ιδιότητες τοπικότητας αντικειμένων που δεν μετακινούνται.

4.3.3 Μετακίνηση αντικειμένων

Η επιλογή ενός συλλέκτη αντιγραφής εξαρτάται μεταξύ άλλων από το αν επιτρέπεται η μετακίνηση αντικειμένων καθώς και από το κόστος της μετακίνησης. Σε μερικά περιβάλλοντα δεν επιτρέπεται η μετακίνηση αντικειμένων. Ένας από τους λόγους είναι η έλλειψη πληροφορίας τύπων που καθιστά μη ασφαλή την τροποποίηση ενός πεδίου που είναι πιθανόν δείκτης σε ένα αντικείμενο. Ένας διαφορετικός λόγος είναι πώς μπορεί μια αναφορά προς ένα αντικείμενο να έχει μεταβιβασθεί ως σε μη διαχειρίσιμο κώδικα (για παράδειγμα ως όρισμα σε κάποια κλήση συστήματος). Επιπλέον το πρόβλημα της εύρεσης δεικτών πολύ συχνά είναι απλούστερο συλλογή με σήμανση και εκκαθάριση από ότι στη συλλογή με αντιγραφή. Αρκεί η ανακάλυψη τουλάχιστον μιας αναφοράς προς ένα ζωντανό αντικείμενο όταν χρησιμοποιείται ένας μη μετακινών συλλέκτης. Από την άλλη πλευρά, ένας μετακινών συλλέκτης πρέπει να εντοπίσει και ενημερώσει όλες τις αναφορές προς ένα μετακινηθέν αντικείμενο. Όπως θα δούμε και στο

κεφάλαιο 9, το γεγονός αυτό καθιστά την ταυτόχρονη συλλογή με μετακίνηση σημαντικά δυσκολότερη από την ταυτόχρονη συλλογή χωρίς μετακίνηση καθώς όλες οι ενημερώσεις των αναφορών προς κάθε αντικείμενο πρέπει να εμφανίζονται ως ατομικές.

Η αντιγραφή ορισμένων αντικειμένων είναι ακριβή. Παρότι η αντιγραφή ενός μικρού αντικειμένου κοστίζει περισσότερο από τη σήμανση αυτού, το κόστος και η λανθάνουσα καθυστέρηση της αντιγραφής συχνά κρύβονται από το κόστος της παρακολούθησης δεικτών και της ανακάλυψης πληροφορίας τύπων. Από την άλλη πλευρά η επαναλαμβανόμενη αντιγραφή μεγάλων αντικειμένων μπορεί να οδηγήσει σε φτωχές επιδόσεις. Μια λύση είναι η αποφυγή της αντιγραφής τους και η ανάθεση της διαχείρισής τους σε έναν μη μετακινούντα συλλέκτη. Μια άλλη λύση είναι η αντιγραφή τους να πραγματοποιείται εικονικά και όχι φυσικά.

Κεφάλαιο 5

Συλλογή απορριμμάτων με καταμέτρηση αναφορών

Οι αλγόριθμοι που έχουμε εξετάσει μέχρι στιγμής είναι έμμεσοι: κάθε ένας από αυτούς διασχίζει τον γράφο των αντικειμένων ώστε να αναγνωρίσει τα ζωντανά αντικείμενα. Σε αυτό το κεφάλαιο εξετάζουμε την τέταρτη και τελευταία θεμελιώδη τεχνική συλλογής απορριμμάτων, αυτήν της καταμέτρησης αναφορών. Η **συλλογή απορριμμάτων με καταμέτρηση αναφορών**, η οποία οφείλεται στον Collins [38], δεν επισκέπτεται όλο το σωρό προκειμένου να εντοπίσει τα ζωντανά αντικείμενα και να συμπεράνει πώς όλα τα υπόλοιπα είναι απορρίμματα, αλλά επιδρά απευθείας στα αντικείμενα καθώς αναφορές σε αυτά δημιουργούνται ή καταστρέφονται.

Ο αλγόριθμος διατηρεί μια απλή αναλλοίωτη: ένα αντικείμενο θεωρείται ζωντανό αν και μόνο αν ο αριθμός των αναφορών σε αυτό είναι αυστηρά θετικός. Η συλλογή με καταμέτρηση αναφορών συσχετίζει λοιπόν έναν μετρητή αναφορών με κάθε διαχειριζόμενο αντικείμενο. Ο μετρητής αυτός συνήθως αποθηκεύεται σε ένα πεδίο στην επικεφαλίδα του αντικειμένου.

Ο αλγόριθμος 5.1 αποτελεί την απλούστερη εκδοχή της συλλογής με καταμέτρηση αναφορών. Η διαδικασία WRITE αυξάνει το μετρητή αναφορών του νέου αντικειμένου στο οποίο αναφέρεται ένας δείκτης και μειώνει το μετρητή αναφορών του παλαιού αντικειμένου προς το οποίο αυτός αναφερόταν. Η διαδικασία αυτή καλείται ακόμη και για την ενημέρωση τοπικών μεταβλητών. Κατά την έξοδο μιας διαδικασίας, τέλος, η WRITE καλείται για να ενημερώσει με την τιμή **null** τις τοπικές μεταβλητές αυτής. Οι διαδικασίες ADDREFERENCE και DELETEREFERENCE αυξάνουν και μειώνουν αντίστοιχα το μετρητή αναφορών του αντικειμένου ορίσματος τους. Μόλις ένας μετρητής αναφορών ενός αντικειμένου μηδενισθεί, τότε πριν η μνήμη αυτού ελευθερωθεί, μειώνονται κατά 1 οι μετρητές αναφορών των αντικειμένων στα οποία δείχνουν τα πεδία-δείκτες του.

Η μέθοδος WRITE αποτελεί ένα παράδειγμα ενός φράγματος εγγραφής. Για κάθε φράγμα εγγραφής, ο μεταγλωττιστής παράγει μία ακολουθία κώδικα πριν και μετά την εγγραφή ενός δείκτη. Όπως θα δούμε και αργότερα, τα φράγματα εγγραφής είναι απαραίτητα στην περίπτωση που οι συλλέκτες δε θεωρούν τη ζωντάνια ολόκληρου του γράφου αντικειμένων ατομικά ως προς τον τροποποιητή. Οι ταυτόχρονοι καθώς και οι γενεαλογικοί συλλέκτες αποτελούν παραδείγματα τέτοιων συλλεκτών. Σε κάθε περίπτωση, τα φράγματα εγγραφής εγγυώνται τη διατήρηση της αναλλοίωτης του εκάστοτε αλγορίθμου συλλογής.

Αλγόριθμος 5.1 Συλλογή με κατάμετρηση αναφορών: απλοϊκή καταμέτρηση αναφορών

```
1:  function New()
2:      ref ← allocate()
3:      if ref = null then
4:          error "Out of memory!"
5:      rc(ref) ← 0
6:      return ref

7:  procedure Write(src, i, ref)
8:      atomic
9:      addReference(ref)
10:     deleteReference(src[i])
11:     src[i] ← ref

12: procedure addReference(ref)
13:     if ref ≠ null then
14:         rc(ref) ← rc(ref) + 1

15: procedure deleteReference(ref)
16:     if ref ≠ null then
17:         rc(ref) ← rc(ref) − 1
18:         if rc(ref) = 0 then
19:             for all fld in Pointers(ref) do
20:                 deleteReference(*fld)
21:             free(ref)
```

5.1 Πλεονεκτήματα & Μειονεκτήματα

Υπάρχουν αρκετοί λόγοι για τους οποίους η συλλογή με καταμέτρηση αναφορών αποτελεί ελκυστική επιλογή. Αρχικά, το κόστος της διαχείρισης μνήμης κατανέμεται στις λειτουργίες εγγραφής του τροποποιητή, χωρίς να απαιτείται η παύση της εκτέλεσης αυτού. Η μνήμη ενός αντικειμένου απελευθερώνεται αμέσως μόλις πάψει να υπάρχει καμία αναφορά προς αυτό, κάτι που καθιστά τον αλγόριθμο κατάλληλο για λειτουργία σε εφαρμογές όπου ο σωρός είναι σχεδόν γεμάτος. Αντίθετα, οι συλλέκτες με εξιχνίαση, χρειάζονται κάποιο χώρο στο σωρό ώστε να λειτουργήσουν. Επιπρόσθετα, καθώς η καταμέτρηση αναφορών επιδρά απευθείας στα αντικείμενα προέλευσης και προορισμού των δεικτών, η τοπικότητα του συλλέκτη δε μπορεί να είναι χειρότερη από την τοπικότητα του τροποποιητή. Ακόμη η συλλογή με καταμέτρηση αναφορών μπορεί να υλοποιηθεί χωρίς καμία βοήθεια από το σύστημα εκτέλεσης ή το μεταγλωττιστή, καθώς δεν απαιτεί τη γνώση των ριζών του προγράμματος. Τέλος, όπως παρατηρούν οι Rodrigues και Jones [100] η καταμέτρηση αναφορών μπορεί να ανακτήσει μνήμη ακόμα και αν κάποια τμήματα του συστήματος δεν είναι προσβάσιμα, κάτι που έχει μεγάλη σημασία σε κατανεμημένα συστήματα.

Για όλους τους παραπάνω λόγους, η συλλογή με καταμέτρηση αναφορών έχει υιοθετηθεί από πολλά συστήματα λογισμικού, ανάμεσα στα οποία περιλαμβάνονται: υλοποιήσεις γλωσσών προγραμματισμού (αρχικές εκδόσεις Smalltalk και Lisp, awk, perl, python), εφαρμογές όπως το Photoshop, συστήματα διαχείρισης εγγράφων και συστήματα διαχείρισης αρχείων λειτουργικών συστημάτων. Επίσης, βιβλιοθήκες για ασφαλή ανάκτηση μνήμης αντικειμένων είναι διαθέσιμες για γλώσσες των οποίων το πρότυπο δεν επιβάλλει (ακόμη) αυτόματη διαχείριση μνήμης. Οι βιβλιοθήκες αυτές χρησιμοποιούν έξυπνους δείκτες για την πρόσβαση σε αντικείμενα. Οι έξυπνοι δείκτες υπερφορτώνουν κατασκευαστές και τελεστές όπως η ανάθεση, είτε για να επιβάλλουν αποκλειστική ιδιοκτησία στα αντικείμενα είτε για να παρέχουν καταμέτρηση αναφορών. Ο Edelson [49] ωστόσο παρατηρεί πώς οι έξυπνοι δείκτες έχουν διαφορετική σημασιολογία από τους "πρωτόγονους" δείκτες που προσπαθούν να μοντελοποιήσουν.

Ωστόσο, τη συλλογή απορριμμάτων με καταμέτρηση αναφορών χαρακτηρίζουν και διάφορα μειονεκτήματα.

Πρώτον, η καταμέτρηση αναφορών επιβάλλει ένα χρονικό κόστος στον τροποποιητή. Σε αντίθεση με τους αλγορίθμους εξιχνίασης που εξετάσαμε στα προηγούμενα κεφάλαια, η καταμέτρηση αναφορών επανορίζει όλες τις λειτουργίες Read και Write δεικτών προκειμένου να διαχειριστεί τους μετρητές αναφορών. Ακόμη και μια απλή διάσχιση μιας λίστας απαιτεί μία αύξηση και μία μείωση ενός μετρητή αναφοράς για κάθε αντικείμενο αυτής. Από άποψη επίδοσης, η επιβάρυνση λειτουργιών που ενημερώνουν καταχωρητές και κομμάτια της στοίβας είναι απαγορευτική. Γι αυτόν ακριβώς το λόγο, ο απλοϊκός αλγόριθμος που παρουσιάστηκε είναι πρακτικά μη εφαρμόσιμος για χρήση σε ένα γενικού σκοπού υψηλών επιδόσεων διαχειριστή μνήμης.

Δεύτερον, τόσο οι ενέργειες που αφορούν διαχείριση των μετρητών αναφορών όσο και οι φορτώσεις και αποθηκεύσεις δεικτών πρέπει να είναι ατομικές ώστε να αποφευχθούν καταστάσεις συναγωνισμού μεταξύ των νημάτων-τροποποιητών οι οποίες θα μπορούσαν να οδηγήσουν σε πρώιμη αποδέσμευση της μνήμης ενός αντικειμένου. Η εξασφάλιση της ατομικότητας μόνο της λειτουργίας ενημέρωσης του μετρητή αναφοράς δεν επαρκεί. Κάποιες βιβλιοθήκες έξυπνων δεικτών απαιτούν ιδιαίτερα προσεκτική μεταχείριση προκειμένου να αποφευχθούν οι καταστάσεις συναγωνισμού. Για παράδειγμα, η βιβλιοθήκη έξυπνων δεικτών Boost για τη γλώσσα C++, η οποία παρέχει και λειτουργίες καταμέτρησης αναφορών, νήματα που εκτε-

λούνται ταυτόχρονα μπορεί είτε να διαβάζουν τον ίδιο μοιραζόμενο δείκτη ταυτόχρονα είτε να μεταβάλλουν διαφορετικά στιγμιότυπα αυτού. Η βιβλιοθήκη εξασφαλίζει όμως την ατομικότητα μόνο των λειτουργιών αύξησης/μείωσης του μετρητή αναφορών. Ο συνδυασμός ανάγνωσης/εγγραφής ενός δείκτη με την ενημέρωση ενός μετρητή αναφοράς δεν αποτελεί μία μοναδική ατομική πράξη.

Τρίτον, η απλοϊκή μέτρηση αναφορών μετατρέπει λειτουργίες ανάγνωσης-μόνο σε λειτουργίες που απαιτούν εγγραφή στη μνήμη (ώστε να ενημερωθούν οι μετρητές αναφορών). Όμοια, απαιτείται η ανάγνωση και εγγραφή του παλαιού αντικειμένου αναφοράς ενός δείκτη πριν αυτός αλλάξει τιμή. Αυτές οι εγγραφές "μολύνουν" την κρυφή μνήμη και επιφέρουν επιπλέον κίνηση μεταξύ μνήμης και επεξεργαστή.

Τέταρτον, η συλλογή με καταμέτρηση αναφορών αδυνατεί να συλλέξει κυκλικές δομές δεδομένων (δομές δεδομένων που περιλαμβάνουν αναφορές στον εαυτό τους). Ακόμη και αν μία τέτοια δομή είναι απομονωμένη στο γράφο των αντικειμένων (δεν υπάρχει δηλαδή μονοπάτι προς αυτή από τις ρίζες), οι μετρητές αναφορών των συνιστωσών της δε θα μηδενισθούν ποτέ. Οι Bacon και Rajan [11] διαπιστώνουν πώς αυτοαναφορικές δομές (διπλά συνδεδεμένες λίστες, κυκλικές ουρές, δένδρα με δείκτες προς τη ρίζα κ.ά.) απαντώνται συχνά στις εφαρμογές, παρότι η συχνότητά τους μπορεί να ποικίλλει σημαντικά από εφαρμογή σε εφαρμογή.

Πέμπτον, στη χειρότερη περίπτωση, ο αριθμός των αναφορών σε ένα αντικείμενο μπορεί να ισούται με το πλήθος των αντικειμένων του σωρού. Αυτό συνεπάγεται πώς ο μετρητής αναφορών θα καταλαμβάνει μια ολόκληρη λέξη μνήμης. Λαμβάνοντας υπόψη την διαπίστωση των Dieckmann και Hölzle [43] και Blackburn κ.ά. [19] πως το μέσο μέγεθος αντικειμένων στις αντικειμενοστραφείς γλώσσες είναι μικρό (στη Java 20-64 bytes) καθώς το γεγονός πώς τα κελιά cons στη Lisp χρειάζονται συνήθως 2-3 λέξεις, η επιβάρυνση σε χώρο μπορεί να είναι σημαντική.

Τέλος, η συλλογή απορριμμάτων με καταμέτρηση αναφορών μπορεί επίσης να προκαλέσει παύσεις. Τη στιγμή που διαγράφεται ο δείκτης στην κεφαλή μιας μεγάλης δομής δεδομένων, η διαδικασία DELETEREFERENCE θα πρέπει να ελευθερώσει αναδρομικά τη μνήμη όλων των αντικειμένων-απογόνων της ρίζας. Ο Boehm [26] μάλιστα ισχυρίζεται πώς οι παύσεις αυτές μπορεί ακόμα και να ξεπεράσουν τις αντίστοιχες παύσεις από συλλέκτες εξιχνίασης.

Στη συνέχεια εξετάζουμε προηγμένους αλγορίθμους συλλογής απορριμμάτων με καταμέτρηση αναφορών.

5.2 Καταμέτρηση αναφορών με αναβολή

Τα περισσότερα συστήματα υψηλών επιδόσεων με καταμέτρηση αναφορών, όπως για παράδειγμα των Blackburn και McKinley [17] χρησιμοποιούν την τεχνική της **καταμέτρησης αναφορών με αναβολή**. Η συντριπτική πλειοψηφία των φορτώσεων δεικτών αφορά τοπικές και προσωρινές μεταβλητές, δηλαδή τους καταχωρητές και τη στοίβα. Οι Deutsch και Bobrow [42] έδειξαν πώς μπορούν να αποφευχθούν οι ενέργειες καταμέτρησης αναφορών που αφορούν αυτές τις λειτουργίες, ενημερώνοντας το μετρητή αναφορών ενός αντικειμένου μόνο όταν ο δείκτης προς αυτό αποθηκεύεται στο πεδίο κάποιου αντικειμένου του σωρού. Οι λειτουργίες καταμέτρησης αναφορών που αφορούν τους καταχωρητές και τη στοίβα αναβάλλονται. Με την τεχνική αυτή, οι μετρητές αναφορών των τοπικών μεταβλητών παύουν να είναι αξιόπιστοι: δεν είναι πλέον ασφαλής η αποδέσμευση ενός αντικειμένου αν ο μετρητής αναφορών του είναι μηδενικός. Η ανακύκλωση των αντικειμένων απορριμμάτων απαιτεί την

περιοδική διόρθωση των μετρητών κατά τη διάρκεια παύσεων του κόσμου. Ο Ungar [114] δείχνει πως οι παύσεις αυτές είναι σημαντικά μικρότερες σε σχέση με τις αντίστοιχες ενός συλλέκτη σήμανσης και εκκαθάρισης.

Ο αλγόριθμος 5.2 φορτώνει τις αναφορές σε αντικείμενα χρησιμοποιώντας, την απλή, χωρίς φράγμα υλοποίηση READ που ορίστηκε στο κεφάλαιο 1. Παρόμοια, οι ρίζες ενημερώνονται με αναφορές χρησιμοποιώντας μία μη φραγμένη εντολή STORE. Αντίθετα, οι εγγραφές σε αντικείμενα του σωρού πρέπει να είναι φραγμένες. Στην περίπτωση αυτή, η αύξηση ενός μετρητή αναφοράς γίνεται κανονικά ως συνήθως. Ωστόσο, αν η μείωση του μετρητή αναφορών του αντικειμένου στο οποίο έδειχνε ένας δείκτης έχει ως αποτέλεσμα το μηδενισμό του μετρητή, το φράγμα εγγραφής WRITE προσθέτει το αντικείμενο (μια αναφορά σε αυτό) σε ένα πίνακα μηδενικών μετρητών (μεταβλητή *zct*) αντί να το απελευθερώσει αμέσως. Ο πίνακας αυτός μπορεί να υλοποιηθεί με διαφόρους τρόπους, χρησιμοποιώντας ένα bitmap όπως ο Baden [12] ή ένα hashtable όπως οι Deutsch και Bobrow [42]. Η ανακύκλωση ενός αντικειμένου με μηδενικό μετρητή αναφορών σε αυτό το σημείο δεν είναι ασφαλής, καθώς ενδέχεται να υπάρχει μια αναφορά σε αυτό από τη στοίβα του προγράμματος. Διαισθητικά, ο πίνακας μηδενικών μετρητών περιλαμβάνει αναφορές σε αντικείμενα με μηδενικό μετρητή αναφορών τα οποία μπορεί όμως και να είναι ζωντανά.

Ωστόσο, τα αντικείμενα απορρίμματα πρέπει κάποια στιγμή να συλλεγούν αν δε θέλουμε το πρόγραμμα να ξεμείνει από μνήμη. Περιοδικά, για παράδειγμα όταν ο εκχωρητής αποτύχει να επιστρέψει μνήμη στη συνάρτηση NEW, διακόπτεται η λειτουργία των νημάτων-τροποποιητών και τα αντικείμενα του πίνακα μηδενικών μετρητών εξετάζονται ώστε να ελεγχθεί εάν ο πραγματικός μετρητής αναφορών τους είναι μηδενικός.

Ένα τέτοιο αντικείμενο μπορεί να είναι ζωντανό μόνο αν αναφέρεται σε αυτό κάποια ρίζα του τροποποιητή. Ο απλούστερος τρόπος ανακάλυψης τέτοιων αντικειμένων είναι η σάρωση των ριζών και η σήμανση των αντικειμένων αυτών δια μέσου της αύξησης των μετρητών αναφορών τους. Σε αυτό το σημείο ο συλλέκτης είναι σίγουρος πως ένα αντικείμενο με μηδενικό μετρητή αναφορών δεν είναι προσβάσιμο και άρα είναι απόρριμμα. Μπορούμε πλέον να διατρέξουμε ξανά τον πίνακα μηδενικών μετρητών, αποδεσμεύοντας τα αντικείμενα με μηδενικό μετρητή αναφορών ως συνήθως. Τέλος, οι λειτουργίες σήμανσης αντιστρέφονται: διατρέχονται οι ρίζες του τροποποιητή και μειώνονται κατά 1 οι μετρητές αναφορών των αντικειμένων στα οποία πιθανώς δείχνουν. Αν ο μετρητής αναφορών ενός αντικειμένου μηδενισθεί, αυτό καταχωρείται εκ νέου στον πίνακα μηδενικών μετρητών.

Η καταμέτρηση αναφορών με αναβολή αφαιρεί το κόστος της μεταχείρισης μετρητών αναφορών των τοπικών μεταβλητών από τον τροποποιητή. Ο Ungar [114] και ο Baden [12] έχουν ισχυρισθεί πως μπορεί να μειώσει το κόστος των τροποποιήσεων δεικτών έως και 80%. Δεδομένης της σημασίας της τοπικότητας, μπορούμε να υποθέσουμε πως το πλεονέκτημα της καταμέτρησης αναφορών με αναβολή έναντι της απλοϊκής καταμέτρησης αναφορών όσον αφορά την επίδοση θα είναι ακόμη μεγαλύτερο στο σύγχρονο υλικό. Ωστόσο, οι επιδιορθώσεις των μετρητών αναφορών λόγω ενημερώσεων πεδίων- δεικτών αντικειμένων πρέπει να πραγματοποιούνται πρόθυμα και ατομικά.

5.3 Καταμέτρηση αναφορών με συγκέντρωση

Η καταμέτρηση αναφορών με αναβολή αφορά το κόστος της εφαρμογής λειτουργιών καταμέτρησης αναφορών σε τοπικές μεταβλητές. Το πρόβλημα του πώς να μειωθεί το κόστος από την καταμέτρηση αναφορών προκύπτει από την ενημέρωση δεικτών που αποθηκεύονται σε

Αλγόριθμος 5.2 Συλλογή με καταμέτρηση αναφορών: καταμέτρηση αναφορών με αναβολή

```
1:  function New()
2:      ref ← allocate()
3:      if ref = null then
4:          collect()
5:          ref ← allocate()
6:          if ref = null then
7:              error "Out of memory!"
8:      rc(ref) ← 0
9:      add(zct, ref)
10:     return ref

11: procedure write(src, i, ref)
12:     if src = Roots then
13:         src[i] ← ref
14:     else
15:         atomic
16:         addReference(ref)
17:         remove(zct, ref)
18:         deleteReferenceToZCT(src[i])
19:         src[i] ← ref

20: procedure deleteReferenceToZCT(ref)
21:     if ref ≠ null then
22:         rc(ref) ← rc(ref) −1
23:         if rc(ref) = 0 then
24:             add(zct, ref)                          ▷ defer freeing

25: procedure collect()
26:     atomic
27:     for all fld in Roots do                        ▷ mark the stacks
28:         addReference(∗fld)
29:     sweepZCT()
30:     for all fld in Roots do                        ▷ unmark the stacks
31:         deleteReferenceToZCT(∗fld)

32: procedure sweepZCT()
33:     while not isEmpty(zct) do
34:         ref ← remove(zct)
35:         if rc(ref) = 0 then                        ▷ now reclaim garbage
36:             for all fld in Pointers(ref) do
37:                 deleteReference(∗fld)
38:             free(ref)
```

πεδία αντικειμένων. Οι Levanoni και Petrank [76] παρατηρούν πως, για κάθε χρονική περίοδο και κάθε αντικείμενο, μόνο οι τιμές στην αρχή και το τέλος της περιόδου έχουν σημασία και οι ενδιάμεσες τιμές μπορούν να αγνοηθούν. Επομένως αρκετές καταστάσεις ενός αντικειμένου μπορούν να συγκεντρωθούν σε αυτές τις δύο. Για παράδειγμα, έστω ένα αντικείμενο X, οποίο περιλαμβάνει ένα δείκτη f, ο οποίος αρχικά αναφέρεται στο αντικείμενο O_0 και ενημερώνεται συνέχεια ώστε να δείχνει στα αντικείμενα $O_1, O_2, \ldots O_3$. Αυτό θα είχε ως αποτέλεσμα την ακόλουθη αλληλουχία ενεργειών καταμέτρησης αναφορών:

$$\mathrm{rc}(O_0)\text{–}, \boxed{\mathrm{rc}(O_1)\text{++}, \mathrm{rc}(O_1)\text{–}}, \boxed{\mathrm{rc}(O_2)\text{++}, \ldots}, \mathrm{rc}(O_n)\text{++}.$$

Οι ενδιάμεσες λειτουργίες αλληλοαναιρούνται και μπορούν να παραλειφθούν. Οι Levanoni και Petrank [76] τις αγνοούν αντιγράφοντας αντικείμενα σε έναν τοπικό χώρο καταγραφής πριν την πρώτη τροποποίηση του στη διάρκεια μιας εποχής. Τη στιγμή που ενημερώνεται ο δείκτης ενός αντικειμένου, αυτό καταγράφεται, δηλαδή η διεύθυνσή του και οι τρέχουσες τιμές των δεικτών που περιέχει αποθηκεύονται σε έναν τοπικό απομονωτή ενημερώσεων (αλγόριθμος 5.3). Το τροποποιημένο αντικείμενο σημειώνεται ως βρώμικο.

Η διαδικασία LOG επιχειρεί να αποφύγει την παρουσία διπλών εγγραφών στον τοπικό χώρο καταγραφής ενός νήματος-τροποποιητή προσαρτώντας αρχικά τις τιμές των πεδίων δεικτών του αντικειμένου στο χώρο καταγραφής. Στη συνέχεια αφού ελέγξει πώς το αντικείμενο ακόμη δεν είναι βρώμικο, οριστικοποιεί την καταγραφή γράφοντας το αντικείμενο στο χώρο καταγραφής, σημειωμένο κατάλληλα ώστε να μπορεί να αναγνωρισθεί ως καταγραφή αντικειμένου και όχι ως καταγραφή πεδίου αντικειμένου. Τέλος, ενημερώνεται ο εσωτερικός δρομέας του χώρου καταγραφής. Το αντικείμενο σημειώνεται ως βρώμικο με την εγγραφή ενός δείκτη στην καταγραφή του στην επικεφαλίδα του.

Παρότι μια κατάσταση συναγωνισμού μπορεί να οδηγήσει στην καταγραφή του στιγμιοτύπου ενός αντικειμένου σε περισσότερους του ενός τοπικούς απομονωτές, ο αλγόριθμος εγγυάται πώς οι καταγραφές θα είναι πανομοιότυπες μεταξύ τους και επομένως δεν έχει σημασία σε ποια από όλες δείχνει η επικεφαλίδα του αντικειμένου. Ανάλογα βέβαια με το μοντέλο συνέπειας μνήμης του επεξεργαστή, αυτό το φράγμα εγγραφής μπορεί να μην απαιτεί λειτουργίες συγχρονισμού.

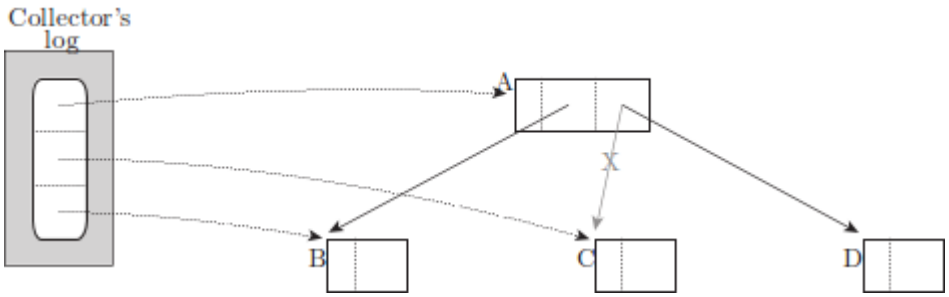
Στην αρχή κάθε κύκλου συλλογής, ο αλγόριθμος 5.4 διακόπτει κάθε νήμα-τροποποιητή, μεταφέρει τον απομονωτή ενημερώσεων αυτού στο χώρο καταγραφής του συλλέκτη και δεσμεύει ένα νέο απομονωτή ενημερώσεων. Η διαδικασία PROCESSREFERENCECOUNTS ελέγχει αν ένα αντικείμενο είναι ακόμη βρώμικο πριν ενημερώσει τους μετρητές αναφορών. Οι μετρητές αναφορών των παιδιών ενός αντικειμένου πριν την πρώτη του τροποποίηση στην τρέχουσα εποχή μειώνονται, ενώ αντίστοιχα οι μετρητές αναφορών των τρεχόντων παιδιών του αυξάνονται. Σε ένα απλό σύστημα, κάθε αντικείμενο με μηδενικό μετρητή αναφορών θα μπορούσε να αποδεσμευθεί αναδρομικά. Ωστόσο, αν η καταμέτρηση αναφορών σε τοπικές μεταβλητές αναβάλλεται, ή αν για λόγους επίδοσης ο αλγόριθμος δεν εγγυάται πώς θα διαχειριστεί όλες τις αυξήσεις πριν τις μειώσεις, μπορούμε απλά να θυμηθούμε ένα αντικείμενο με μηδενικό μετρητή αναφορών. Ο αλγόριθμος επίσης καθαρίζει το αντικείμενο (αφαιρεί τη σήμανση dirty) ούτως ώστε αυτό να μην επανεξετασθεί στη διάρκεια του τρέχοντος κύκλου συλλογής. Οι δείκτες στα παλαιά παιδιά ενός αντικειμένου μπορούν να βρεθούν από το στιγμιότυπο αυτού στο χώρο καταγραφής, ενώ οι δείκτες στα τρέχοντα παιδιά του από το αντικείμενο καθεαυτό. Οι Paz και Petrank σημειώνουν [88] πως υπάρχει η δυνατότητα για προφόρτωση αντικειμένων ή μετρητών αναφορών κατά την εκτέλεση των βρόχων των διαδικασιών DECREMENTOLD και INCREMENTNEW.

Αλγόριθμος 5.3 Συλλογή με καταμέτρηση αναφορών: φράγμα εγγραφής για καταμέτρηση αναφορών με αναβολή

```
1:  me ← myThreadId

2:  procedure WRITE(src, i, ref)
3:      ref ← ALLOCATE()
4:      if not DIRTY(src) then
5:          LOG(src)
6:      src[i] ← ref

7:  procedure LOG(obj)
8:      for all fld in Pointers(obj) do
9:          if *fld ≠ null then
10:             APPEND(updates[me], *fld)
11:     if not DIRTY(obj) then
12:         slot ← APPENDANDCOMMIT(updates[me], obj)
13:         SETDIRTY(obj, slot)

14: function DIRTY(obj)
15:     return LOGPOINTER(obj) ≠ CLEAN

16: procedure SETDIRTY(obj, slot)
17:     LOGPOINTER(obj) ← slot          ▷ address of entry for obj in updates[me]
```

Αλγόριθμος 5.4 Συλλογή με καταμέτρηση αναφορών: ενημέρωση μετρητών αναφορών για καταμέτρηση αναφορών με συγκέντρωση

```
 1: procedure COLLECT()
 2:     COLLECTBUFFERS()
 3:     PROCESSREFERENCECOUNTS()
 4:     SWEEPZCT()

 5: procedure COLLECTBUFFERS()
 6:     collectorLog ← []
 7:     for all t in Threads do
 8:         collectorLog ← collectorLog + updates[t]

 9: procedure PROCESSREFERENCECOUNTS()
10:     for all entry in collectorLog do
11:         obj ← OBJFROMLOG(entry)
12:         if DIRTY(obj) then                    ▷ do not process duplicates
13:             LOGPOINTER(obj) ← CLEAN
14:             INCREMENTNEW(obj)
15:             DECREMENTOLD(entry)

16: procedure DECREMENTOLD(entry)
17:     for all fld in Pointers(entry) do         ▷ use the values in the collector's log
18:         child ← *fld
19:         if child ≠ null then
20:             rc(child ← rc(child) − 1
21:             if rc(child) = 0 then
22:                 ADD(zct, child)

23: procedure INCREMENTNEW(obj)
24:     for all fld in Pointers(obj) do           ▷ use the values in the object
25:         child ← *fld
26:         if child ≠ null then
27:             rc(child) ← rc(child) + 1
```



Σχήμα 5.1: Καταμέτρηση αναφορών με συγκέντρωση. Αν το αντικείμενο A τροποποιήθηκε στην προηγούμενη εποχή, για παράδειγμα μέσω της ενημέρωσης του δείκτη προς το αντικείμενο C με ένα δείκτη προς το αντικείμενο D, τα πεδία δεικτών του αντικειμένου A έχουν αντιγραφεί στο χώρο καταγραφής. Το παλαιό αντικείμενο αναφοράς C μπορεί να βρεθεί από το χώρο καταγραφής του συλλέκτη, ενώ το νέο αντικείμενο αναφοράς D κατευθείαν από το αντικείμενο A.

Ο συνδυασμός της καταμέτρησης αναφορών με αναβολή και συγκέντρωση αφαιρεί το κόστος της καταμέτρησης αναφορών από τον τροποποιητή. Το πλεονέκτημα αυτό ωστόσο δεν έρχεται χωρίς κόστος. Έχουμε επανεισάγει παύσεις στη λειτουργία του τροποποιητή (παρότι αυτές αναμένεται να είναι συντομότερες από τις αντίστοιχες ενός συλλέκτη εξιχνίασης) και επίσης αυξήσει τις απαιτήσεις σε μνήμη για τη διατήρηση των απομονωτών και του πίνακα μηδενικών μετρητών.

5.4 Κυκλική Καταμέτρηση Αναφορών

Καθώς οι μετρητές αναφορών των αντικειμένων μιας κυκλικής δομής δεδομένων έχουν τιμή τουλάχιστον 1, η απλή καταμέτρηση αναφορών δεν μπορεί από μόνη της να συλλέξει τέτοιες δομές. Οι κύκλοι ωστόσο εμφανίζονται συχνά και δημιουργούνται τόσο από τους προγραμματιστές εφαρμογών όσο και το σύστημα εκτέλεσης.

Η απλούστερη προσέγγιση στο πρόβλημα είναι ο συνδυασμός της καταμέτρησης αναφορών με συνήθη, εφεδρική συλλογή εξιχνίασης. Η ελπίδα είναι πώς τα περισσότερα αντικείμενα δε θα είναι προσβάσιμα από κύκλους και επομένως θα απελευθερώνονται αμέσως από την καταμέτρηση αναφορών. Οι εναπομένουσες κυκλικές δομές συλλέγονται από το συλλέκτη εξιχνίασης. Η ιδέα αυτή απλά μειώνει τη συχνότητα εκτέλεσης της συλλογής εξιχνίασης. Σε επίπεδο γλώσσας, οι Friedman και Wise [55] παρατήρησαν πώς κύκλοι μπορούν να δημιουργηθούν μόνο σε αγνές συναρτησιακές γλώσσες από αναδρομικούς ορισμούς και επομένως μπορούν να αντιμετωπιστούν ειδικά δεδομένης της παρατήρησης ορισμένων περιορισμών.

Πολλοί ερευνητές έχουν προτείνει την ειδική αντιμετώπιση των λειτουργιών εγγραφής δεικτών που κλείνουν κύκλους. Μεταξύ αυτών είναι οι Friedman και Wise, [55], ο BrownBridge [30], ο Salkild [102]; οι Pepels κ.ά. [90] καθώς και ο Axford [6].

Οι πιο ευρέως υιοθετημένοι μηχανισμοί για το χειρισμό κυκλικών δομών με καταμέτρηση αναφορών ωστόσο χρησιμοποιούν μία τεχνική γνωστή και ως **δοκιμαστική διαγραφή**. Η βασική ιδέα αφορά στο ότι δεν είναι απαραίτητο ο εφεδρικός συλλέκτης εξιχνίασης να επισκεφθεί ολόκληρο τον γράφο των ζωντανών αντικειμένων. Αντίθετα, αυτός μπορεί να εστιάσει την προσοχή του σε εκείνα τα τμήματα του γράφου στα οποία η διαγραφή ενός δείκτη μπορεί να έχει δημιουργήσει έναν κύκλο απορριμμάτων. Ας παρατηρήσουμε πως:

- Σε κάθε κύκλο απορριμμάτων, οι μη μηδενικοί μετρητές αναφορών οφείλονται σε εσωτερικούς δείκτες.
- Οι κύκλοι απορριμμάτων μπορεί να προκύψουν μόνο από τη διαγραφή κάποιου δείκτη που αφήνει το μετρητή αναφορών κάποιου αντικειμένου θετικό.

Οι αλγόριθμοι μερικής εξιχνίασης εκμεταλλεύονται τις παραπάνω παρατηρήσεις επισκεπτόμενοι μόνο το υπογράφημα αντικειμένων που έχει ως ρίζα κάποιο αντικείμενο το οποίο υποπτεύονται πως είναι απόρριμμα. Οι αλγόριθμοι αυτοί διαγράφουν δοκιμαστικά κάθε δείκτη που συναντούν μέσω της μείωσης προσωρινά του μετρητή αναφορών του αντικειμένου στο οποίο αυτός αναφέρεται. Με τον τρόπο αυτό, πρακτικά αφαιρούν τη συμβολή των εσωτερικών δεικτών στην κυκλικότητα της δομής. Αν ο μετρητής αναφορών ενός αντικειμένου εξακολουθεί να είναι μη μηδενικός μετά από τη διαδικασία αυτή, τότε αυτό θα πρέπει να είναι προσβάσιμο από κάποιο αντικείμενο εκτός του υπογράφου και συνεπώς το αντικείμενο δεν είναι απόρριμμα. Επιπλέον ούτε το μεταβατικό κλείσιμο του αντικειμένου ως προς την προσβασιμότητα μέσω δεικτών περιέχει απορρίμματα.

Οι Bacon κ.ά. [8], οι Bacon και Rajan [11] και οι Paz κ.ά. [87] επινόησαν τον αλγόριθμο Recycler που υποστηρίζει ταυτόχρονη κυκλική καταμέτρηση αναφορών. Στην παρούσα φάση παρουσιάζουμε τη σύγχρονη εκδοχή του αλγορίθμου, αναβάλλοντας την παρουσίαση της ασύγχρονης για το κεφάλαιο 9. Ο αλγόριθμος 5.5 δρα σε 3 περάσματα:

1. Αρχικά, εξιχνιάζει υπογράφους ξεκινώντας από αντικείμενα που αναγνωρίζονται ως πιθανά μέλη κύκλων απορριμμάτων, μειώνοντας τους μετρητές αναφορών που οφείλονται σε εσωτερικούς δείκτες (διαδικασία MARKCANDIDATES). Τα επισκεπτόμενα αντικείμενα χρωματίζονται γκρι.
2. Στη συνέχεια ελέγχει το μετρητή αναφορών κάθε τέτοιου αντικειμένου: αν αυτός είναι θετικός, το αντικείμενο είναι προσβάσιμο από κάποιο αντικείμενο που δεν ανήκει στον υπό εξιχνίαση υπογράφο και επομένως η δράση του πρώτου περάσματος αναιρείται (διαδικασία SCAN) χρωματίζοντας τα ζωντανά γκρι αντικείμενα μαύρα. Τα μη ζωντανά γκρι αντικείμενα χρωματίζονται λευκά.
3. Τελικώς, όλα τα αντικείμενα ενός υπογράφου που είναι ακόμη λευκά είναι απορρίμματα και ελευθερώνονται.

Με μαύρο χρώμα σημειώνονται τα ενεργά αντικείμενα ενώ με λευκό τα απορρίμματα. Με γκρι χρώμα σημειώνονται τα αντικείμενα που είναι πιθανά μέλη ενός κύκλου απορριμμάτων, ενώ με μωβ τα αντικείμενα που είναι πιθανώς ρίζες ενός κύκλου απορριμμάτων.

Η διαγραφή μιας οποιασδήποτε πλην της τελευταίας αναφοράς σε ένα αντικείμενο ενδέχεται να απομονώσει έναν κύκλο απορριμμάτων. Σε αυτήν την περίπτωση, ο αλγόριθμος 5.5 χρωματίζει μωβ το αντικείμενο και το προσθέτει στη λίστα των υποψήφιων μελών κύκλων απορριμμάτων. Διαφορετικά το αντικείμενο είναι απόρριμμα και ο μετρητής αναφορών του οφείλει να είναι μηδενικός. Η διαδικασία RELEASE επαναφέρει το χρώμα του σε μαύρο, επεξεργάζεται αναδρομικά τα παιδιά του και αν δεν είναι υποψήφια ρίζα κάποιου κύκλου απορριμμάτων, το ελευθερώνει. Η ανάκτηση αντικειμένων από το σύνολο υποψηφίων ριζών κύκλων απορριμμάτων αναβάλλεται μέχρις ότου εκτελεσθεί η διαδικασία MARKCANDIDATES.

Η διαδικασία MARKCANDIDATES εξετάζει κάθε αντικείμενο στο σύνολο υποψηφίων ριζών κύκλων απορριμμάτων. Αν ένα αντικείμενο είναι ακόμη μωβ (δηλαδή δεν έχει προστεθεί κάποια

Αλγόριθμος 5.5 Συλλογή με καταμέτρηση αναφορών: ο αλγόριθμος Recycler

```
1:  function New()
2:      ref ← allocate()
3:      if ref = null then
4:          collect()                                   ▷ the cycle collector
5:          ref ← allocate()
6:          if ref = null then
7:              error "Out of memory!"
8:      rc(ref) ← 0
9:      return ref

10: procedure addReference(ref)
11:     if ref ≠ null then
12:         rc(ref) ← rc(ref) + 1
13:         colour(ref) ← black                         ▷ cannot be in a garbage cycle

14: procedure deleteReference(ref)
15:     if ref ≠ null then
16:         rc(ref) ← rc(ref) − 1
17:         if rc(ref) = 0 then
18:             release(ref)
19:         else
20:             candidate(ref)                          ▷ might isolate a garbage cycle

21: procedure release(ref)
22:     for all fld in Pointers(ref) do
23:         deleteReference(fld)
24:     colour(ref) ← black                             ▷ objects on the free-list are black
25:     if not ref in candidates then                   ▷ deal with candidates later
26:         free(ref)

27: procedure candidate(ref)                            ▷ colour as a candidate and add to the set
28:     if colour(ref) ≠ purple then
29:         colour(ref) ← purple
30:         candidates ← candidates ∪ ref

31: procedure collect()
32:     atomic
33:     for all ref in candidates do
34:         scan(ref)
35:     collectCandidates()
```

Αλγόριθμος 5.5 Συλλογή με καταμέτρηση αναφορών: ο αλγόριθμος Recycler (συνέχεια)

```
36: procedure markCandidates()
37:     for all ref in candidates do
38:         if colour(ref) = purple then
39:             markGrey(ref)
40:         else
41:             remove(candidates, ref)
42:             if colour(ref) = purple and rc(ref) = 0 then
43:                 free(ref)

44: procedure markGrey(ref)
45:     if colour(ref) ≠ grey then
46:         colour(ref) ← grey
47:         for all fld in Pointers(ref) do
48:             child ← *fld
49:             if child ≠ null then
50:                 rc(ref) ← rc(ref) − 1                    ▷ trial deletion
51:                 markGrey(child)

52: procedure scan(ref)
53:     if colour(ref) = grey then
54:         if rc(ref) > 0 then
55:             scanBlack(ref)              ▷ there must be an external reference
56:         else
57:             colour(ref) ← white                          ▷ looks like garbage...
58:             for all fld in Pointers(ref) do              ▷ ...so continue
59:                 child ← *fld
60:                 if child ≠ null then
61:                     scan(child)

62: procedure scanBlack(ref)          ▷ repair the reference counts of live data
63:     colour(ref) ← white
64:     for all fld in Pointers(ref) do
65:         child ← *fld
66:         if child ≠ null then
67:             rc(ref) ← rc(ref) + 1                        ▷ undo the trial deletion
68:             if colour(child) ≠ black then
69:                 scanBlack(child)
```

Αλγόριθμος 5.5 Συλλογή με καταμέτρηση αναφορών: ο αλγόριθμος Recycler (συνέχεια)

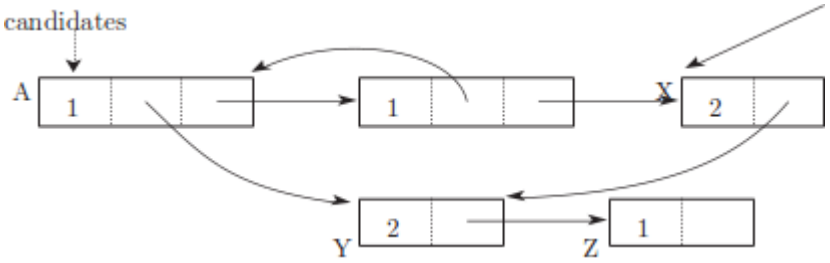
```
70: procedure COLLECTCANDIDATES()
71:     while not ISEMPTY(candidates) do
72:         ref ← REMOVE(candidates)
73:         COLLECTWHITE(ref)

74: procedure COLLECTWHITE(ref)
75:     if colour(ref) = white and not ref in candidates then
76:         colour(ref) ← black                    ▷ free-list objects are black
77:         for all fld in Pointers(ref) do
78:             child ← *fld
79:             if child ≠ null then
80:                 COLLECTWHITE(child)
81:         FREE(ref)
```

αναφορά σε αυτό από τη στιγμή που εισήχθη στο σύνολο), το μεταβατικό κλείσιμο αυτού ως προς την προσβασιμότητα μέσω δεικτών χρωματίζεται γκρι. Διαφορετικά αφαιρείται από το σύνολο και στην περίπτωση που είναι μαύρο και ο μετρητής αναφορών του μηδενικός, ελευθερώνεται. Η διαδικασία MARKGREY χρωματίζει γκρι τον υπογράφο που έχει ως ρίζα το εν λόγω αντικείμενο και αφαιρεί από τους μετρητές αναφορών των αντικειμένων του υπογράφου τη συμβολή των εσωτερικών δεικτών.

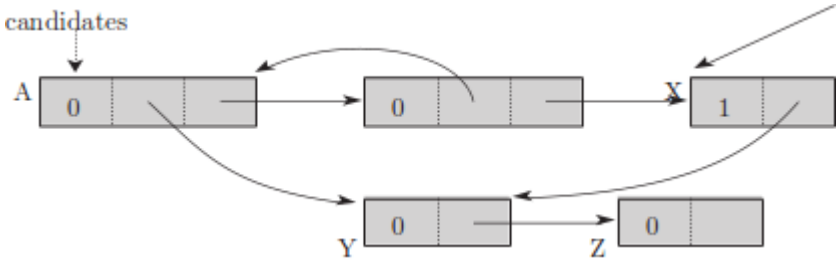
Στη δεύτερη φάση της συλλογής, εξετάζονται οι μετρητές αναφορών κάθε αντικειμένου που είναι υποψήφια ρίζα ενός κύκλου απορριμμάτων καθώς και των αντικειμένων που ανήκουν στο μεταβατικό κλείσιμο αυτού ως προς την προσβασιμότητα μέσω δεικτών. Αν ο μετρητής αναφορών ενός αντικειμένου βρεθεί θετικός, τότε σίγουρα υπάρχει εξωτερική αναφορά προς αυτό. Στην περίπτωση αυτή, η διαδικασία SCANBLACK αναιρεί την επίδραση της διαδικασίας MARKGREY αυξάνοντας το μετρητή αναφορών και χρωματίζοντας το αντικείμενο μαύρο. Αντίθετα, αν ο μετρητής αναφορών είναι μηδενικός, η διαδικασία SCAN χρωματίζει το αντικείμενο λευκό και εξετάζει αναδρομικά τα παιδιά του. Στο σημείο αυτό δε μπορούμε με βεβαιότητα να υποθέσουμε πως ένα λευκό αντικείμενο είναι απόρριμμα, καθώς αυτό μπορεί να επανεξετασθεί αργότερα από κλήση της διαδικασίας SCANBLACK με όρισμα κάποιο άλλο αντικείμενο του υπογράφου.

Τέλος, η τρίτη φάση, η οποία υλοποιείται από τη διαδικασία COLLECTWHITE αποδεσμεύει τα λευκά αντικείμενα. Επαναληπτικά, μέχρις ότου αδειάσει το σύνολο υποψηφίων ριζών κύκλων απορριμμάτων αφαιρείται από αυτό ένα αντικείμενο και εξετάζεται το χρώμα του. Αν είναι λευκό, το αντικείμενο ελευθερώνεται (χρωματίζεται μαύρο) και στη συνέχεια εξετάζονται αναδρομικά τα παιδιά του. Η διαδικασία COLLECTWHITE δεν επεξεργάζεται αντικείμενα-παιδιά που τυχαίνει να βρίσκονται στο σύνολο των υποψηφίων ριζών κύκλων απορριμμάτων: η ελευθέρωσή τους πραγματοποιείται σε επόμενη επανάληψη του βρόχου της διαδικασίας COLLECTCANDIDATES.

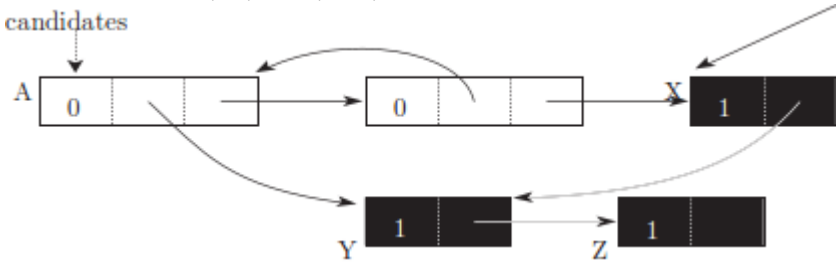
Ο αλγόριθμος Recycler μπορεί να βελτιστοποιηθεί περαιτέρω με την στατική αναγνώριση πως αντικείμενα συγκεκριμένων κλάσεων δε μπορούν να είναι μέλη κύκλων απορριμμάτων (όπως για παράδειγμα αντικείμενα που δεν περιέχουν δείκτες, αλλά όχι μόνο) και ως εκ τούτου δε χρειάζεται να μπουν στο σύνολο των υποψήφιων ριζών κύκλων απορριμμάτων. Τα αντικείμενα αυτά χρωματίζονται πράσινα και όχι μαύρα. Οι Bacon και Rajan [11] διαπιστώνουν πώς αυτή η βελτιστοποίηση μειώνει κατά μία τάξη μεγέθους το μέγεθος του συνόλου υποψηφίων ριζών



(α′) Πριν τη διαδικασία markGrey.



(β′) Μετά τη markGrey, όλα τα αντικείμενα που είναι προσβάσιμα από ένα αντικείμενο υποψήφιο μέλος κύκλου απορριμμάτων έχουν σημανθεί γκρι και η επίδραση των εσωτερικών δεικτών αυτού του γκρι υπογράφου έχει αφαιρεθεί. Ο μετρητής αναφορών του αντικειμένου X, το οποίο είναι ακόμη προσβάσιμο, έχει μη μηδενική τιμή.



(γ′) Μετά τη scan, όλα τα προσβάσιμα αντικείμενα είναι μαύρα και οι μετρητές αναφορών αυτών έχουν επιδιορθωθεί και αντικατοπτρίζουν ζωντανές αναφορές.

Σχήμα 5.2: Κυκλική καταμέτρηση αναφορών. Το πρώτο πεδίο κάθε αντικειμένου αποθηκεύει το μετρητή αναφορών του.

κύκλων απορριμμάτων.

5.5 Θέματα προς εξέταση

Η καταμέτρηση αναφορών είναι ελκυστική για την προθυμία συλλογής αντικειμένων απορριμμάτων αλλά και τις ιδιότητες τοπικότητας αυτής. Η απλοϊκή καταμέτρηση αναφορών μπορεί να ανακτήσει τη μνήμη από ένα αντικείμενο αμέσως μόλις αφαιρείται ο τελευταίος δείκτης που αναφέρεται σε αυτό. Η λειτουργία της εμπλέκει μόνο τους παλαιούς και νέους στόχους των δεικτών που εγγράφονται ή διαβάζονται, σε αντίθεση με τη συλλογή εξιχνίασης που επισκέπτεται κάθε ζωντανό αντικείμενο στο σωρό. Ωστόσο, τα πλεονεκτήματα της μεθόδου είναι ταυτόχρονα και τα μειονεκτήματά της. Καθώς δεν μπορεί να ελευθερώσει ένα αντικείμενο μέχρις ότου κανένας δείκτης να μην αναφέρεται σε αυτό, δεν μπορεί να συλλέξει κύκλους απορριμμάτων. Επιπλέον, προσθέτει ένα μικρό κόστος σε κάθε λειτουργία Read και Write του τροποποιητή, επιβαρύνοντας με τον τρόπο αυτό τη ρυθμαπόδοση του τελευταίου περισσότερο σε σχέση με τη συλλογή εξιχνίασης. Επιπλέον, οι πολυνηματικές εφαρμογές απαιτούν τον αυστηρό συγχρονισμό των τροποποιήσεων των μετρητών αναφορών και των ενημερώσεων δεικτών. Τέλος, η συλλογή απορριμμάτων με καταμέτρηση αναφορών αυξάνει το μέγεθος των αντικειμένων.

5.5.1 Περιβάλλον εκτέλεσης

Παρά τα παραπάνω ζητήματα, ο αποκλεισμός της καταμέτρησης αναφορών χωρίς περαιτέρω σκέψη είναι εσφαλμένος. Ασφαλώς και δεν είναι κατάλληλη ως τμήμα του διαχειριστή μνήμης μιας εικονικής μηχανής γενικού σκοπού, ειδικά αν τα εμπλεκόμενα αντικείμενα είναι μικρά, εμφανίζονται συχνά κύκλοι και ο ρυθμός λειτουργιών εγγραφής δεικτών από τον τροποποιητή είναι υψηλός. Ωστόσο, υπάρχουν περιβάλλοντα στα οποία η χρήση καταμέτρησης αναφορών είναι κατάλληλη. Η καταμέτρηση αναφορών αποδεικνύεται αποδοτική σε περιβάλλοντα όπου οι χρόνοι ζωής των περισσότερων αντικειμένων είναι αρκετά απλοί ώστε αυτά να διαχειρίζονται ρητώς. Μπορεί επίσης να περιορισθεί στη διαχείριση ενός μικρότερου αριθμού πόρων με πιο σύνθετες σχέσεις ιδιοκτησίας. Συχνά οι πόροι αυτοί είναι μεγάλα αντικείμενα με αποτέλεσμα το κόστος της προσθήκης στην επικεφαλίδα μιας λέξης για την αποθήκευση του μετρητή αναφορών να είναι αμελητέο. Δεδομένα όπως bitmap ψηφιακών εικόνων δεν περιλαμβάνουν δείκτες και συνεπώς είναι αδύνατη η εμφάνιση κύκλων απορριμμάτων. Επιπρόσθετα, η καταμέτρηση αναφορών μπορεί να υλοποιηθεί ως τμήμα μιας βιβλιοθήκης και όχι ως τμήμα του συστήματος εκτέλεσης, δίνοντας με τον τρόπο αυτό πλήρη έλεγχο στον προγραμματιστή όσον αφορά τη χρήση της. Στην περίπτωση αυτή βέβαια απαιτείται ιδιαίτερη προσοχή από τον προγραμματιστή, ο οποίος πρέπει να εξασφαλίσει την απουσία καταστάσεων συναγωνισμού μεταξύ λειτουργιών εγγραφής δεικτών και ενημερώσεων μετρητών αναφορών.

5.5.2 Προηγμένες τεχνικές

Εξεζητημένοι αλγόριθμοι καταμέτρησης αναφορών προσφέρουν λύσεις για τα περισσότερα από τα προβλήματα που αντιμετωπίζει η απλοϊκή καταμέτρηση αναφορών, επιβάλλοντας ωστόσο παύση του κόσμου, όπως συμβαίνει και στη συλλογή εξιχνίασης.

Η μνήμη αντικειμένων ενός κύκλου απορριμμάτων μπορεί να ανακτηθεί από ένα εφεδρικό συλλέκτη εξιχνίασης, ή χρησιμοποιώντας την τεχνική της δοκιμαστικής διαγραφής. Και στις

δύο περιπτώσεις, η εκτέλεση του τροποποιητή αναστέλλεται κατά τη διάρκεια συλλογής των κύκλων απορριμμάτων.

Παρότι στη χειρότερη περίπτωση το πεδίο ενός αντικειμένου όπου αποθηκεύεται ο μετρητής αναφορών πρέπει να είναι τόσο μεγάλο ώστε να χωράει το μέγιστο πλήθος δεικτών, οι περισσότερες εφαρμογές διατηρούν μικρό αριθμό αναφορών προς τα περισσότερα αντικείμενα. Συχνά, είναι δυνατό να κλαπούν μερικά bits από μια λέξη της επικεφαλίδας που χρησιμοποιείται για την αποθήκευση ενός κλειδώματος ή ενός κωδικού κατακερματισμού. Είναι βέβαια επίσης σύνηθες να υπάρχουν πολλές αναφορές προς μερικά αντικείμενα.

Η επιβάρυνση στη ρυθμαπόδοση του τροποποιητή μπορεί να μειωθεί παραλείποντας μερικές τροποποιήσεις δεικτών και μειώνοντας το κόστος άλλων. Η καταμέτρηση αναφορών με αναβολή αγνοεί τις εγγραφές από τον τροποποιητή των τοπικών μεταβλητών. Αυτό επιτρέπει στους μετρητές αναφορών των αντικειμένων που είναι προσβάσιμα από τις ρίζες να έχουν μικρότερη τιμή από πραγματική και αποτρέπει την πρόωρη ανάκτηση της μνήμης που αυτά καταλαμβάνουν. Η καταμέτρηση αναφορών με συγκέντρωση λαμβάνει υπόψη την κατάσταση ενός αντικειμένου μόνο στην αρχή και στο τέλος μιας εποχής, αγνοώντας τις λειτουργίες εγγραφής δεικτών στο ενδιάμεσο χρονικό διάστημα. Κατά κάποιο τρόπο είναι μια αυτόματη βελτιστοποίηση: αφαιρεί τις περιττές ενδιάμεσες επιδιορθώσεις των μετρητών αναφορών. Ωστόσο και πάλι, τόσο η καταμέτρηση αναφορών με αναβολή όσο και η καταμέτρηση αναφορών με συγκέντρωση επιβάλλουν την παύση του κόσμου για ένα χρονικό διάστημα ώστε να επιδιορθωθούν οι μετρητές αναφορών. Επιπλέον, προσθέτουν μία επιβάρυνση στις απαιτήσεις σε χώρο της απλοϊκής καταμέτρησης αναφορών για την αποθήκευση είτε του πίνακα μηδενικών μετρητών αναφορών είτε των απομονωτών καταγραφής ενημερώσεων.

Ένα επιπρόσθετο πλεονέκτημα των προηγμένων αυτών τεχνικών συλλογής με καταμέτρηση αναφορών είναι πώς κλιμακώνουν καλά με μεγάλους σωρούς. Το κόστος τους είναι ανάλογο μόνο προς τον αριθμό των εγγραφών δεικτών και όχι του όγκου των δεδομένων ζωντανών αντικειμένων.

Κεφάλαιο 6

Σύγκριση αλγορίθμων συλλογής απορριμμάτων

Στα προηγούμενα κεφάλαια, παρουσιάσαμε αναλυτικά τους 4 θεμελιώδεις αλγορίθμους συλλογής απορριμμάτων. Σε αυτό το κεφάλαιο τους συγκρίνουμε με περισσότερη λεπτομέρεια. Τους εξετάζουμε με δύο διαφορετικούς τρόπους. Πρώτον, αναλύουμε τα κριτήρια με βάση τα οποία μπορούμε να αξιολογήσουμε τους αλγορίθμους καθώς επίσης και τα πλεονεκτήματα και μειονεκτήματα που προκύπτουν από τη χρήση διαφορετικών προσεγγίσεων σε διαφορετικές καταστάσεις. Στη συνέχεια παρουσιάζουμε τους αφηρημένους αλγορίθμους συλλογής με εξιχνίαση και συλλογής με καταμέτρηση αναφορών, όπως αυτοί εισήχθησαν από τους Bacon κ.ά. [10]. Η παρουσίαση των αλγορίθμων σε αυτό το αφηρημένο πλαίσιο αποκαλύπτει πως, παρότι οι αλγόριθμοι επιδεικνύουν επιφανειακές διαφορές, μοιράζονται μια βαθειά και αξιοσημείωτη ομοιότητα.

6.1 Ρυθμαπόδοση

Η συνολική ρυθμαπόδοση των εφαρμογών έρχεται με μεγάλη πιθανότητα πρώτη στις λίστες επιθυμιών των χρηστών. Αυτή πιθανόν αποτελεί τον πρωταρχικό στόχο για μια μαζική εφαρμογή ή για ένα διακομιστή ιστού όπου οι παύσεις είναι ανεκτές ή κρύβονται από άλλα στοιχεία του συστήματος, όπως οι καθυστερήσεις δικτύου. Παρότι είναι σημαντικό οι ενέργειες της συλλογής απορριμμάτων να εκτελούνται όσο το δυνατόν ταχύτερα, η χρήση ενός γρηγορότερου συλλέκτη δε σημαίνει απαραίτητα πώς ένας υπολογισμός θα εκτελεσθεί γρηγορότερα. Σε ένα καλώς διαμορφωμένο σύστημα, η συλλογή απορριμμάτων θα καταλαμβάνει ένα πολύ μικρό ποσοστό του συνολικού χρόνου εκτέλεσης. Αν το τίμημα που πρέπει να πληρωθεί για ταχύτερη συλλογή είναι ένας μεγαλύτερος "φόρος" στις λειτουργίες του τροποποιητή, είναι εξίσου πιθανό ο χρόνος εκτέλεσης της εφαρμογής να αυξηθεί αντί να μειωθεί. Η επιβάρυνση των λειτουργιών του τροποποιητή μπορεί να είναι άμεση ή έμμεση. Παραδείγματα άμεσης επιβάρυνσης είναι η εκτέλεση φραγμάτων εγγραφής και ανάγνωσης, όπως αυτά που απαιτεί η συλλογή απορριμμάτων με καταμέτρηση αναφορών. Ωστόσο, η επίδοση του τροποποιητή μπορεί να επηρεασθεί και έμμεσα, για παράδειγμα επειδή ένας συλλέκτης αντιγραφής έχει αναδιατάξει τα αντικείμενα με τέτοιον τρόπο ώστε να επηρεάζει δυσμενώς τη συμπεριφορά της εφαρμογής ως προς την κρυφή μνήμη. Η αποφυγή (οποτεδήποτε αυτό είναι εφικτό) λειτουργιών συγχρονισμού είναι εξίσου σημαντική. Δυστυχώς, οι τροποποιήσεις των μετρητών

αναφορών πρέπει να συγχρονίζονται προκειμένου να μη χάνονται ενημερώσεις. Η καταμέτρηση αναφορών με αναβολή ή συγκέντρωση δύναται να εξαλείψει ένα σημαντικό μέρος του παραπάνω κόστους συγχρονισμού.

Μπορούμε επίσης να εξετάσουμε την πολυπλοκότητα των διαφόρων αλγορίθμων συλλογής. Για τη συλλογή απορριμμάτων με σήμανση και εκκαθάριση, πρέπει να συμπεριλάβουμε το κόστος των φάσεων της εξιχνίασης (σήμανσης) και της εκκαθάρισης, ενώ το κόστος της συλλογής με αντιγραφή εξαρτάται μόνο από την εξιχνίαση. Η εξιχνίαση απαιτεί την επίσκεψη κάθε **ζωντανού** αντικειμένου, ενώ η εκκαθάριση απαιτεί την επίσκεψη κάθε αντικειμένου (ζωντανού ή νεκρού). Μπορεί σε αυτό το σημείο κάποιος να υποθέσει πώς το κόστος της συλλογής με σήμανση και εκκαθάριση είναι μεγαλύτερο από το κόστος της συλλογής με αντιγραφή. Ωστόσο, ο αριθμός των εκτελούμενων εντολών κατά την επίσκεψη ενός αντικειμένου είναι μικρότερος κατά τη συλλογή με σήμανση και εκκαθάριση από ότι κατά τη συλλογή με αντιγραφή. Η τοπικότητα παίζει εξίσου σημαντικό ρόλο και τεχνικές προφόρτωσης μπορεί να χρησιμοποιηθούν για να μειωθούν οι αστοχίες κρυφής μνήμης. Ωστόσο το ερώτημα του κατά πόσο τέτοιες τεχνικές μπορούν να εφαρμοσθούν στη συλλογή με αντιγραφή χωρίς να χαθεί το πλεονέκτημα της τοπικότητας που παρέχει η κατά-βάθος αντιγραφή παραμένει ακόμη ανοιχτό. Τέλος, ο συνδυασμός σήμανσης και οκνηρής εκκαθάρισης επιφέρει το μέγιστο κέρδος στις συνθήκες εκείνες όπου και η αντιγραφή αποδίδει βέλτιστα: όταν τα ζωντανά αντικείμενα καταλαμβάνουν ένα μικρό ποσοστό του σωρού.

6.2 Χρόνος παύσης

Η εξασφάλιση μικρών χρόνων παύσης είναι σημαντική όχι μόνο για διαδραστικές εφαρμογές αλλά και για άλλες, όπως για παράδειγμα εφαρμογές δοσοληψιών όπου οι τυχόν καθυστερήσεις μπορεί να οδηγήσουν σε συγκέντρωση μεγάλου φορτίου ανεκτέλεστης εργασίας. Οι συλλέκτες εξιχνίασης που έχουμε εξετάσει μέχρι τώρα λειτουργούν με παύση του κόσμου. Οι χρόνοι παύσης για την εκτέλεση συλλογής απορριμμάτων ήταν πολύ μεγάλοι στα αρχικά συστήματα και μπορούν ακόμη και σε σύγχρονες αρχιτεκτονικές να διακόψουν την εκτέλεση μεγάλων εφαρμογών έως και ένα δευτερόλεπτο. Το άμεσο πλεονέκτημα της συλλογής απορριμμάτων με καταμέτρηση αναφορών είναι η αποφυγή τέτοιων παύσεων μέσω της κατανομής του κόστους των λειτουργιών διαχείρισης μνήμης στις λειτουργίες εγγραφής του τροποποιητή. Ωστόσο, όπως είδαμε, η αποφυγή των παύσεων δεν επιτυγχάνεται πάντα σε υψηλών επιδόσεων συστήματα καταμέτρησης αναφορών. Αρχικά, η διαγραφή της τελευταίας αναφοράς προς μία πλούσια σε δείκτες δομή οδηγεί σε αναδρομικές τροποποιήσεις μετρητών αναφορών και ανάκτηση μνήμης αντικειμένων. Παρότι ευτυχώς δεν υπάρχει ανταγωνισμός μεταξύ των τροποποιήσεων αντικειμένων-απορριμμάτων, οι τελευταίες μπορεί να προκαλέσουν ανταγωνισμό στα μπλοκ κρυφής μνήμης που περιλαμβάνουν τα αντικείμενα-απορρίμματα. Ακόμη είδαμε πώς η καταμέτρηση αναφορών με αναβολή και η καταμέτρηση αναφορών με συγκέντρωση εισάγουν μια παύση του κόσμου κατά την οποία διορθώνονται οι μετρητές αναφορών ζωντανών αντικειμένων και ανακτάται η μνήμη νεκρών αντικειμένων προς τα οποία υπάρχει αναφορά στον πίνακα μηδενικών μετρητών αναφορών.

6.3 Χώρος

Όλοι οι αλγόριθμοι συλλογής απορριμμάτων εισάγουν χωρικά κόστη, με διάφορους παράγοντες να συμβάλλουν σε αυτό. Ένας αλγόριθμος μπορεί να πληρώνει ένα κόστος ανά αντι-

κείμενο, όπως π.χ. για τα πεδία μετρητών αναφορών. Οι συλλέκτες αντιγραφής χρειάζονται επιπρόσθετο χώρο στο σωρό για τη διατήρηση ενός εφεδρικού χώρου αντιγραφής, ο οποίος πρέπει για ασφάλεια να έχει τέτοιο μέγεθος ώστε να χωράνε όλα τα ζωντανά αντικείμενα. Οι μη μετακινούντες συλλέκτες αντιμετωπίζουν το πρόβλημα του κατακερματισμού της μνήμης του σωρού, που μειώνει το μέγεθος της διαθέσιμης στην εφαρμογή μνήμης. Ο χώρος εκτός σωρού όπου αποθηκεύονται διάφορα μεταδεδομένα δεν πρέπει να αγνοηθεί. Οι συλλέκτες εξιχνίασης χρειάζονται χώρο για στοίβες σήμανσης, bitmap σήμανσης και άλλες βοηθητικές δομές δεδομένων. Κάθε διαχειριστής μνήμης που δε χρησιμοποιεί συμπύκνωση, θα χρησιμοποιήσει χώρο για τις δομές δεδομένων του (όπως για παράδειγμα στην περίπτωση που η εκχώρηση μνήμης γίνεται με ξεχωριστές ελεύθερες λίστες για αντικείμενα διαφορετικών μεγεθών). Τέλος, προκειμένου ένας συλλέκτης εξιχνίασης ή ένας συλλέκτης καταμέτρησης αναφορών με αναβολή να μην καλείται πολύ συχνά, πρέπει να διατεθεί σε αυτόν ειδικός χώρος στο σωρό για την προσωρινή φιλοξενία απορριμμάτων. Αντίθετα, η συλλογή απορριμμάτων με απλοϊκή καταμέτρηση αναφορών ελευθερώνει ένα αντικείμενο αμέσως μόλις αυτό αποσυνδεθεί από το γράφο των ζωντανών αντικειμένων. Εκτός από το προφανές πλεονέκτημα της αποτροπής της συσσώρευσης απορριμμάτων στο σωρό, η μνήμη που ανακτάται με μεγάλη πιθανότητα χρησιμοποιείται σύντομα μετά την ανάκτησή της με αποτελέσμα να βελτιώνεται η επίδοση της κρυφής μνήμης. Μάλιστα, σε ορισμένες περιπτώσεις είναι πιθανό ο μεταγλωττιστής να εντοπίσει πότε ένα αντικείμενο καθίσταται μη προσβάσιμο και να επαναχρησιμοποιήσει τη μνήμη που αυτό καταλαμβάνει άμεσα, χωρίς μεσολάβηση του διαχειριστή μνήμης (συλλέκτη).

Οι συλλέκτες πρέπει να είναι όχι μόνο πλήρεις (να ανακτούν τη μνήμη όλων των αντικειμένων απορριμμάτων **τελικώς**) αλλά και πρόθυμοι, δηλαδή να ανακτούν άμεσα τη μνήμη όλων των αντικειμένων-απορριμμάτων **σε κάθε κύκλο συλλογής**. Ωστόσο, αρκετοί σύγχρονοι συλλέκτες υψηλών επιδόσεων θυσιάζουν την αμεσότητα για χάρη της επίδοσης και επιτρέπουν σε μερικά αντικείμενα-απορρίμματα να αιωρούνται στο σωρό μεταξύ δύο διαδοχικών κύκλων συλλογής. Η συλλογή απορριμμάτων με καταμέτρηση αναφορών τέλος δεν είναι πλήρης, αφού αδυνατεί να συλλέξει κύκλους απορριμμάτων χωρίς την εξιχνίαση του γράφου αντικειμένων.

6.4 Υλοποίηση

Η υλοποίηση των αλγορίθμων συλλογής απορριμμάτων και ειδικότερα των αλγορίθμων ταυτόχρονης συλλογής απορριμμάτων είναι δύσκολη. Η διαπροσωπεία ανάμεσα στο συλλέκτη και το μεταγλωττιστή είναι ιδιαίτερα σημαντική. Λάθη στην υλοποίηση του συλλέκτη συχνά εμφανίζονται πολύ αργά (πιθανόν αρκετούς κύκλους συλλογής αργότερα), καθώς ο τροποποιητής επιχειρεί την αποδεικτοδότηση μιας αναφοράς που δεν είναι πλέον έγκυρη. Συνεπώς οι συλλέκτες απορριμμάτων πρέπει να είναι ταυτόχρονα εύρωστοι και γρήγοροι.

Ένα πλεονέκτημα των μη μετακινούντων συλλεκτών σήμανσης και εκκαθάρισης είναι η απλότητα της διαπροσωπείας μεταξύ τροποποιητή και συλλέκτη: ο τελευταίος καλείται όταν η διαθέσιμη μνήμη έχει εξαντληθεί και ο εκχωρητής αδυνατεί να ικανοποιήσει αιτήματα. Η βασική πολυπλοκότητα αυτής της διαπροσωπείας έγκειται στον καθορισμό των ριζών, αναζητώντας δείκτες στις καθολικές μεταβλητές, τους καταχωρητές και τη στοίβα. Από την άλλη πλευρά, το έργο των μετακινούντων συλλεκτών αντιγραφής και των συλλεκτών σήμανσης και συμπύκνωσης είναι σημαντικά πιο πολύπλοκο. Ένας μετακινών συλλέκτης πρέπει να εντοπίσει και να ενημερώσει **όλους** τους δείκτες προς ένα ζωντανό αντικείμενο, ενώ ένας μη μετακινών συλλέκτης αρκείται στο να ταυτοποιήσει **τουλάχιστον ένα** δείκτη προς ένα ζωντανό αντικείμενο χωρίς να χρειάζεται να μεταβάλλει την τιμή του. Οι γνωστοί και ως **συντηρητικοί** συλλέκτες μπορούν να ανακτήσουν μνήμη που καταλαμβάνεται από αντικείμενα-απορρίμματα

χωρίς ακριβή γνώση της στοίβας του τροποποιητή ή της διάταξης των αντικειμένων στο σωρό. Αντίθετα πραγματοποιούν συντηρητικές (ασφαλείς) υποθέσεις σχετικά με το κατά πόσο μια τιμή είναι δείκτης ή όχι. Καθώς οι μη μετακινούντες συλλέκτες δεν ενημερώνουν τις τιμές δεικτών, ο μόνος κίνδυνος από την εσφαλμένη ταυτοποίηση μιας τιμής ως δείκτη προς ένα αντικείμενο του σωρού αφορά στην εισαγωγή μιας διαρροής μνήμης (η τιμή του δείκτη δεν τροποποιείται).

Η συλλογή απορριμμάτων με καταμέτρηση αναφορών έχει τα πλεονεκτήματα και τα μειονεκτήματα που απορρέουν από την υψηλή σύζευξή της με τον τροποποιητή. Στα θετικά συγκαταλέγεται το γεγονός πώς μπορεί να υλοποιηθεί ως μία βιβλιοθήκη, επιτρέποντας στον προγραμματιστή να επιλέξει ποια αντικείμενα θα διαχειρίζονται αυτόματα με καταμέτρηση αναφορών και ποια ρητώς. Στα αρνητικά από την άλλη πλευρά περιλαμβάνονται η εισαγωγή επιβάρυνσης στις λειτουργίες εγγραφής του τροποποιητή και της επιβάρυνσης συγχρονισμού, που επιβάλλει η εξασφάλιση της ορθότητας των τροποποιήσεων μετρητών αναφορών.

Η επίδοση οποιασδήποτε μοντέρνας γλώσσας προγραμματισμού που κάνει εκτεταμένη χρήση δυναμικώς εκχωρηθέντων δεδομένων εξαρτάται σε ένα μεγάλο βαθμό από το διαχειριστή μνήμης. Τα κρίσιμα τμήματα κώδικα αφορούν την εκχώρηση μνήμης, τα φράγματα εγγραφής και ανάγνωσης του τροποποιητή και τους εσωτερικούς βρόχους του συλλέκτη. Σε κάθε περίπτωση, η έξοδος του μεταγλωττιστή έχει σημασία και ο κώδικας σε assembly πρέπει να εξετάζεται προσεκτικά. Η επίδραση που έχει ο κώδικας στη κρυφή μνήμη επηρεάζει επίσης σε μεγάλο βαθμό την επίδοση.

6.5 Προσαρμοστικά συστήματα

Τα εμπορικά συστήματα συχνά προσφέρουν τη δυνατότητα επιλογής ανάμεσα σε διαφορετικούς αλγορίθμους συλλογής απορριμμάτων, ο καθένας εκ των οποίων έρχεται με μια πληθώρα επιλογών διαμόρφωσης. Αρκετοί ερευνητές έχουν προτείνει τη δυναμική προσαρμογή των συστημάτων εκτέλεσης στο περιβάλλον τους. Το σύστημα εκτέλεσης της γλώσσας Java των Soman κ.ά. [110] προσαρμόζεται δυναμικά αλλάζοντας τον αλγόριθμο συλλογής κατά την εκτέλεση του προγράμματος με βάση το μέγεθος της διαθέσιμης μνήμης του σωρού. Το σύστημα τους χρησιμοποιεί είτε off-line στατιστική ανάλυση για την επισημείωση των προγραμμάτων με το βέλτιστο συνδυασμό συλλέκτη/μέγεθος σωρού, είτε αλλάζει τον αλγόριθμο συλλογής συγκρίνοντας την τρέχουσα χρησιμοποίηση χώρου με το μέγιστο μέγεθος σωρού. Οι Singer κ.ά. [109] από την άλλη πλευρά χρησιμοποιούν τεχνικές μηχανικής μάθησης με στόχο την πρόβλεψη του βέλτιστου συλλέκτη από τις στατικές ιδιότητες ενός προγράμματος.

6.6 Ενοποιημένη θεωρία συλλογής απορριμμάτων

Στα προηγούμενα 4 κεφάλαια, εξετάσαμε δύο κατηγορίες αλγορίθμων συλλογής απορριμμάτων: τους **άμεσους** (καταμέτρηση αναφορών) και τους **έμμεσους** (σήμανση-εκκαθάριση, σήμανση-συμπύκνωση και αντιγραφή). Οι Bacon κ.ά. [10] δείχνουν πως υπάρχουν πολλές ομοιότητες ανάμεσα στους αλγορίθμους των δύο κατηγοριών. Το αφηρημένο πλαίσιο τους επιτρέπει τη διατύπωση μιας ευρείας ποικιλίας διαφορετικών αλγορίθμων συλλογής απορριμμάτων δίνοντας έμφαση ακριβώς στις ομοιότητες και τις διαφορές τους.

6.6.1 Αφηρημένη συλλογή απορριμμάτων

Ο Bacon κ.ά. αρχικά παρατηρούν πως η συλλογή απορριμμάτων μπορεί να διατυπωθεί ως ένας υπολογισμός σταθερού σημείου, που αναθέτει μετρητές αναφορών $\rho(n)$ σε κόμβους $n \in Nodes$. Πιο συγκεκριμένα:

$$\begin{aligned} &\forall ref \in Nodes : \\ &\rho(ref) = |\{fld \in Roots : *fld = ref\}| \\ &\qquad + |\{fld \in Pointers(n) : n \in Nodes \wedge \rho(n) > 0 \wedge *fld = ref\}| \end{aligned} \tag{6.1}$$

Μετά την ανάθεση των μετρητών αναφορών, διατηρούνται οι κόμβοι με μη μηδενικό μετρητή αναφορών και οι υπόλοιποι συλλέγονται. Οι μετρητές αναφορών δε χρειάζεται να είναι ακριβείς και αρκεί να έχουν μια τιμή που αποτελεί ασφαλή προσέγγιση της πραγματικής. Οι αφηρημένοι αλγόριθμοι συλλογής απορριμμάτων εκτελούν τέτοιους υπολογισμούς σταθερού σημείου χρησιμοποιώντας μια λίστα εργασιών W από αντικείμενα προς επεξεργασία. Οι αλγόριθμοι τερματίζουν τη χρονική στιγμή κατά την οποία η λίστα αυτή αδειάζει.

6.6.2 Συλλογή απορριμμάτων με εξιχνίαση

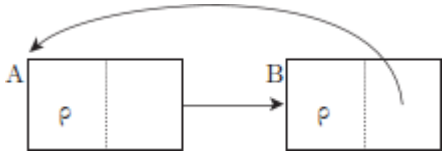
Η αφαίρεση παρουσιάζει τη συλλογή εξιχνίασης ως μία μορφή συλλογής καταμέτρησης αναφορών. Η αφηρημένη συλλογή εξιχνίασης παρουσιάζεται στον αλγόριθμο 6.1. Οι μετρητές αναφορών όλων των κόμβων είναι αρχικά μηδενικοί. Στο τέλος κάθε κύκλου συλλογής η διαδικασία sweepTracing μηδενίζει εκ νέου τους μετρητές αναφορών των κόμβων και η διαδικασία New αρχικοποιεί το μετρητή αναφορών ενός καινούριου κόμβου στο μηδέν. Η διαδικασία collectTracing συσσωρεύει στη λίστα εργασιών W όλους τους μη μηδενικούς δείκτες από το σύνολο των ριζών χρησιμοποιώντας τη διαδικασία rootsTracing και στη συνέχεια περνά τη λίστα εργασιών W στη διαδικασία scanTracing.

Η συλλογή συνεχίζει με εξιχνίαση του γράφου ώστε να ανακαλυφθούν όλοι οι προσβάσιμοι από τις ρίζες κόμβοι. Η διαδικασία scanTracing εξιχνιάζει αντικείμενα από τη λίστα εργασιών W και ανανεώνει το μετρητή αναφορών κάθε κόμβου αυξάνοντας τον κατά 1 κάθε φορά που συναντά τον κόμβο. Την πρώτη φορά που ανακαλύπτεται ένας προσβάσιμος κόμβος src (οπότε και $\rho(src) = 1$), ο συλλέκτης εξετάζει αναδρομικά όλες τις εξερχόμενες ακμές του κόμβου, σαρώνοντας τα πεδία του και προσθέτοντας (δείκτες προς) τα παιδιά στη λίστα εργασιών W.

Κατά την έξοδο από το βρόχο **while** ο μετρητής αναφορών κάθε ζωντανού κόμβου ισούται με το πλήθος των εισερχόμενων σε αυτόν ακμών. Η διαδικασία sweepTracing ελευθερώνει μη χρησιμοποιούμενους κόμβους και μηδενίζει εκ νέου τους μετρητές αναφορών των υπόλοιπων κόμβων για τον επόμενο κύκλο συλλογής. Μία πρακτική υλοποίηση μπορεί να κωδικοποιήσει το μετρητή αναφορών ενός κόμβου μόνο με ένα bit και να καταγράφει έτσι αν ο κόμβος έχει ανακαλυφθεί ή όχι. Το bit σήμανσης λειτουργεί επομένως ως μία προσέγγιση του πραγματικού μετρητή αναφορών.

Η συλλογή εξιχνίασης υπολογίζει την ελάχιστη λύση σταθερού σημείου της εξίσωσης 6.1: οι μετρητές αναφορών των κόμβων είναι οι ελάχιστοι δυνατοί που την ικανοποιούν.

Μπορούμε επίσης να ερμηνεύσουμε τους αφηρημένους αλγορίθμους συλλογής απορριμμάτων λαμβάνοντας υπόψη την τριχρωματική αφαίρεση που συζητήσαμε στην ενότητα 2.2. Στον



Σχήμα 6.1: Ένας απλός κύκλος.

αλγόριθμο 6.1, κόμβοι με μηδενικό μετρητή αναφορών είναι λευκοί, ενώ κόμβοι με μη μηδενικό μετρητή αναφορών είναι μαύροι. Η μετάβαση του χρώματος ενός κόμβου από λευκό σε μαύρο (μέσω γκρι) πραγματοποιείται την πρώτη φορά που αυτός ανακαλύπτεται και σαρώνεται. Τελικώς, η αφηρημένη συλλογή εξιχνίασης διαμερίζει τους κόμβους του σωρού σε μαύρους (ζωντανούς) και λευκούς (νεκρούς).

6.6.3 Συλλογή απορριμμάτων με καταμέτρηση αναφορών

Η αφηρημένη συλλογή απορριμμάτων με καταμέτρηση αναφορών παρουσιάζεται στον αλγόριθμο 6.2, όπου και φαίνεται πως οι λειτουργίες καταμέτρησης αναφορών απομονώνονται από τις διαδικασίες INC και DEC αντί να εφαρμόζονται αμέσως. Ο συλλέκτης απορριμμάτων, που υλοποιείται από τη διαδικασία COLLECTCOUNTING, εφαρμόζει τις αναβεβλημένες αυξήσεις I μέσω της διαδικασίας APPLYINCREMENTS και τις αναβεβλημένες μειώσεις D μέσω της διαδικασίας SCANCOUNTING.

Ο τροποποιητής, μέσω της διαδικασίας WRITE, αποθηκεύει μία αναφορά προς έναν καινούριο κόμβο προορισμού dst στο πεδίο $src[i]$. Πριν πραγματοποιήσει την ενημέρωση $src[i] \leftarrow dst$, απομονώνει μια αύξηση του μετρητή αναφορών του καινούριου κόμβου προορισμού (INC(dst)) καθώς και μία μείωση του μετρητή αναφορών του παλαιού κόμβου προορισμού (DEC($src[i]$)). Κάθε κύκλος συλλογής ξεκινά εφαρμόζοντας τις αναβεβλημένες αυξήσεις μετρητών αναφορών, με τις αναβεβλημένες μειώσεις να εφαρμόζονται στην επόμενη φάση. Κατά την έναρξη εκτέλεσης της διαδικασίας SCANCOUNTING οι μετρητές αναφορών έχουν μεγαλύτερη τιμή από την πραγματική τους. Επομένως η διαδικασία μειώνει το μετρητή αναφορών ενός κόμβου της λίστας εργασιών καθώς τον επισκέπτεται. Κάθε κόμβος προέλευσης src, του οποίου ο μετρητής αναφορών $\rho(src)$ πέφτει στο 0, σε αυτή τη φάση αντιμετωπίζεται ως απόρριμμα και τα παιδιά του προστίθενται στη λίστα εργασιών. Τελικώς, η διαδικασία SWEEPCOUNTING ελευθερώνει τους κόμβους-απορρίμματα.

Οι αλγόριθμοι εξιχνίασης και καταμέτρησης αναφορών είναι ταυτόσημοι, αν εξαιρέσει κανείς μικρές διαφορές. Κάθε ένας έχει μια διαδικασία σάρωσης: η διαδικασία SCANTRACING εφαρμόζει αυξήσεις στους μετρητές αναφορών ενώ η διαδικασία SCANCOUNTING εφαρμόζει μειώσεις. Και στις δύο περιπτώσεις η αναδρομική συνθήκη ελέγχει αν ένας μετρητής αναφορών έχει μηδενισθεί. Τέλος, κάθε ένας έχει μια διαδικασία εκκαθάρισης, που ελευθερώνει το χώρο που καταλαμβάνεται από κόμβους-απορρίμματα.

Η καταμέτρηση αναφορών είναι περίπλοκη όταν υπάρχουν κύκλοι στο γράφο αντικειμένων. Το τετριμμένο παράδειγμα του σχήματος 6.1 δείχνει έναν απλό απομονωμένο κύκλο, όπου υποθέτοντας πως ο κόμβος A έχει μηδενικό μετρητή αναφορών οδηγούμαστε στο συμπέρασμα πως και ο κόμβος B έχει μηδενικό μετρητή αναφορών. Αν όμως θεωρήσουμε πως ο μετρητής αναφορών του κόμβου A έχει την τιμή 1, συμπεραίνουμε πως και ο μετρητής αναφορών του κόμβου B έχει την τιμή 1.

Γενικότερα, στους υπολογισμούς σταθερών σημείων ενδέχεται να υπάρχουν παραπάνω από

Αλγόριθμος 6.1 Αφηρημένη συλλογή εξιχνίασης

```
1:  procedure collectTracing()
2:      atomic
3:      rootsTracing(W)
4:      scanTracing(W)
5:      sweepTracing()

6:  procedure scanTracing(W)
7:      while not isEmpty(W) do
8:          src ← remove(W)
9:          ρ(src) ← ρ(src) + 1                          ▷ shade src
10:         if ρ(src) = 1 then                  ▷ src was white, now grey
11:             for all fld in Pointers(src) do
12:                 ref ← *fld
13:                 if ref ≠ null then
14:                     W ← W + [ref]

15: procedure sweepTracing()
16:     for all node in Nodes do
17:         if ρ(node) = 0 then                          ▷ node is white
18:             free(node)
19:         else                                         ▷ node is black
20:             ρ(node) ← 0                      ▷ reset node to white

21: function New()
22:     ref ← allocate()
23:     if ref = null then
24:         collectTracing()
25:         ref ← allocate()
26:         if ref = null then
27:             error "Out of memory!"
28:     ρ(ref) ← 0                                       ▷ node is white
29:     return ref

30: procedure rootsTracing(R)
31:     for all fld in Roots do
32:         ref ← *fld
33:         if ref ≠ null then
34:             R ← R + [ref]
```

Αλγόριθμος 6.2 Αφηρημένη συλλογή καταμέτρησης αναφορών

```
 1: procedure collectCounting(I, D)
 2:     atomic
 3:     applyIncrements(I)
 4:     scanCounting(D)
 5:     sweepCounting()

 6: procedure scanCounting(W)
 7:     while not isEmpty(W) do
 8:         src ← remove(W)
 9:         ρ(src) ← ρ(src) − 1
10:         if ρ(src) = 0 then
11:             for all fld in Pointers(src) do
12:                 ref ← *fld
13:                 if ref ≠ null then
14:                     W ← W + [ref]

15: procedure sweepCounting()
16:     for all node in Nodes do
17:         if ρ(node) = 0 then
18:             free(node)

19: function New()
20:     ref ← allocate()
21:     if ref = null then
22:         collectCounting()
23:         ref ← allocate()
24:         if ref = null then
25:             error "Out of memory!"
26:     ρ(ref) ← 0
27:     return ref

28: procedure dec(ref)
29:     if ref ≠ null then
30:         D ← D + [ref]

31: procedure inc(ref)
32:     if ref ≠ null then
33:         I ← I + [ref]

34: procedure Write(src, i, dst)
35:     inc(dst)
36:     dec(src[i])
37:     src[i] ← dest

38: procedure applyIncrements(I)
39:     while not isEmpty(I) do
40:         ref ← remove(I)
41:         ρ(ref) ← ρ(ref) + 1
```

μία λύσεις. Στην περίπτωση του σχήματος 6.1 έχουμε $Nodes = \{A, B\}$ και $Roots = \{\}$. Υπάρχουν δύο λύσεις σταθερού σημείου της εξίσωσης 6.1 για αυτόν τον απλό γράφο: το ελάχιστο σταθερό σημείο $\rho(A) = \rho(B) = 0$ και το μέγιστο σταθερό σημείο $\rho(A) = \rho(B) = 1$. Η συλλογή εξιχνίασης υπολογίζει το ελάχιστο σταθερό σημείο, ενώ η συλλογή με καταμέτρηση αναφορών το μέγιστο σταθερό σημείο, με αποτέλεσμα να μην μπορεί να συλλέξει κύκλους απορριμμάτων. Η διαφορά μεταξύ των δύο λύσεων είναι ακριβώς το σύνολο των κόμβων που είναι προσβάσιμοι από κύκλους απορριμμάτων.

Μέρος ΙΙ

Προηγμένοι αλγόριθμοι συλλογής απορριμμάτων

Κεφάλαιο 7

Γενεαλογική συλλογή απορριμμάτων

Ο στόχος ενός συλλέκτη απορριμμάτων είναι η εύρεση νεκρών αντικειμένων και εν συνεχεία η αποδέσμευση του χώρου που αυτά καταλαμβάνουν. Οι συλλέκτες εξιχνίασης (και ειδικότερα οι συλλέκτες αντιγραφής) παρουσιάζουν καλές επιδόσεις όταν ο σωρός περιλαμβάνει σχετικά λίγα ζωντανά αντικείμενα. Δε μεταχειρίζονται ωστόσο αποδοτικά αντικείμενα με μακρά διάρκεια ζωής, τα οποία είτε σημαίνουν και ξε-σημαίνουν είτε αντιγράφουν από τον ένα ημιχώρο στον άλλο διαρκώς. Είδαμε στο κεφάλαιο 3 πώς αντικείμενα με μακρά διάρκεια ζωής τείνουν να συγκεντρώνονται στο κάτω μέρος του σωρού όταν αυτός διαχειρίζεται από ένα συλλέκτη με σήμανση και συμπύκνωση και πώς πολλοί συλλέκτες αποφεύγουν τη συμπύκνωση αυτού του τμήματος. Παρότι αυτή η βελτιστοποίηση αφαιρεί το κόστος της μετακίνησης αυτών, τα εν λόγω αντικείμενα πρέπει να εξετάζονται και τα πεδία-δείκτες αυτών να ενημερώνονται σε κάθε κύκλο συλλογής.

Οι γενεαλογικοί συλλέκτες απορριμμάτων επεκτείνουν την παραπάνω ιδέα αγνοώντας τα παλαιότερα αντικείμενα οποτεδήποτε αυτό είναι δυνατό. Οι συλλέκτες αυτής της κατηγορίας βασίζονται στην ασθενή γενεαλογική υπόθεση πως τα περισσότερα αντικείμενα πεθαίνουν νέα. Επικεντρώνοντας την προσοχή τους στα νεότερα αντικείμενα, προσπαθούν να μεγιστοποιήσουν τον ελεύθερο χώρο που ανακτούν με το λιγότερο δυνατό κόπο. Κατηγοριοποιούν τα αντικείμενα με βάση την ηλικία τους σε γενεές, με την κάθε γενιά να ζει συνήθως σε ένα ξεχωριστό τμήμα του σωρού. Οι γενεές συλλέγονται σε αύξουσα σειρά ως προς την ηλικία τους, ενώ τα αντικείμενα που επιβιώνουν μετά από αρκετούς κύκλους συλλογής προάγονται σε παλαιότερες γενεές.

Οι περισσότεροι συλλέκτες αυτής της κατηγορίας διαχειρίζονται τις νεότερες γενεές με αντιγραφή. Ο απαιτούμενος χρόνος που δαπανάται για τη νεότερη γενεά, γνωστή και ως βρεφοκομείο εξαρτάται από το μέγεθός της. Ρυθμίζοντας επομένως το μέγεθος αυτό, μπορούμε να ελέγξουμε το χρόνο παύσης για τη συλλογή μιας γενεάς. Οι χρόνοι παύσης για τη συλλογή των νεότερων γενεών σε εφαρμογές που τρέχουν σε ένα σύγχρονο μηχάνημα και επαληθεύουν την ασθενή γενεαλογική υπόθεση, είναι της τάξης των χιλιοστών του δευτερολέπτου. Θεωρώντας ακόμη πώς η συλλογή εκτελείται σχετικά αραιά, η δράση ενός γενεαλογικού συλλέκτη περνάει σχεδόν απαρατήρητη στον τροποποιητή.

Αν ένας γενεαλογικός συλλέκτης εκτιμήσει πώς η συλλογή μόνο της νεότερης γενιάς δε θα ανακυκλώσει αρκετό χώρο, χρειάζεται να συλλεχθεί ολόκληρος ο σωρός. Συνεπώς η γενεαλογική συλλογή απορριμμάτων βελτιώνει μόνο την αναμενόμενη τιμή του χρόνου παύσης

και όχι την χείριστη περίπτωση. Δεν επαρκεί από μόνη της για ένα σύστημα πραγματικού χρόνου.

Η **γενεαλογική συλλογή απορριμμάτων** δεν είναι τελείως δωρεάν. Αρχικά η συνεχόμενη συλλογή μόνο των νεότερων γενεών δεν μπορεί να εντοπίσει απορρίμματα που ζουν σε παλαιότερες γενεές. Επιπλέον, για να μπορεί να συλλέξει μόνο μια γενεά, αγνοώντας τις υπόλοιπες, ο συλλέκτης θα πρέπει με κάποιο τρόπο να σημειώνει τους δείκτες που διασχίζουν τα σύνορα μεταξύ των γενεών.

7.1 Πώς μετράται ο χρόνος;

Για να προάγει ένας γενεαλογικός συλλέκτης αντικείμενα από μία γενεά σε μία παλαιότερη, απαιτείται να μπορεί να προσδιορίζει πόσο παλαιά είναι αυτά. Επομένως χρειάζεται τόσο ένα μηχανισμό μέτρησης του χρόνου όσο και ένα μηχανισμό μέτρησης της ηλικίας των αντικειμένων. Στη βιβλιογραφία συναντώνται κυρίως δύο πιθανές μετρικές: το συνολικό μέγεθος της δεσμευμένης μνήμης και ο χρόνος που έχει περάσει από την έναρξη της εκτέλεσης της εφαρμογής. Η μετρική του χρόνου καθορίζει το προφίλ της εφαρμογής. Απαντάει σε ερωτήσεις που αφορούν το για πόσο τρέχει μια εφαρμογή καθώς και ποιοι είναι οι χρόνοι παύσης αυτής και πώς αυτοί είναι κατανεμημένοι. Απαντήσεις στις ερωτήσεις αυτές μπορούν να χρησιμοποιηθούν ώστε να αποφανθεί κανείς σχετικά με την αποκρισιμότητα της εφαρμογής και το κατά πόσο η παύση αυτής για συλλογή απορριμμάτων ενοχλεί έναν διαδραστικό χρήστη. Από την άλλη πλευρά, εσωτερικά, η διαφορά σε bytes της συνολικής δεσμευμένης μνήμης στο σωρό μεταξύ των χρονικών στιγμών της γέννησης και του θανάτου ενός αντικειμένου αντικατοπτρίζει καλύτερα τη χρονική διάρκεια ζωής αυτού. Είναι επίσης ένα μέγεθος ανεξάρτητο της αρχιτεκτονικής. Τέλος, αποτελεί στοιχείο των απαιτήσεων σε μνήμη μιας εφαρμογής και είναι στενά συνδεδεμένο με τη συχνότητα με την οποία θα κληθεί ο συλλέκτης. Η μέτρηση του χρόνου με όρους bytes είναι ιδιαίτερα επίπονη διαδικασία σε πολυεπεξεργαστικά περιβάλλοντα όπου εκτελούνται ταυτόχρονα πολλά νήματα. Η χρήση ενός απλού καθολικού μετρητή του συνολικού μεγέθους της δεσμευμένης μνήμης ενδέχεται να παραπλανήσει το συλλέκτη σχετικά με την ηλικία ενός αντικειμένου, καθώς μια μεγάλη τιμή του μπορεί να έχει προκύψει από την εκχώρηση μνήμης σε νήματα άσχετα με το αντικείμενο. Οι πραγματικές υλοποιήσεις γενεαλογικών συλλεκτών απορριμμάτων μοντελοποιούν την ηλικία ενός αντικειμένου ως τον αριθμό των συλλογών από τις οποίες αυτό έχει επιβιώσει.

7.2 Γενεαλογικές υποθέσεις

Η ασθενής γενεαλογική υπόθεση, σύμφωνα με την οποία τα περισσότερα αντικείμενα πεθαίνουν νέα, φαίνεται γενικώς έγκυρη, ανεξαρτήτως της γλώσσας προγραμματισμού. Οι Foderaro και Fateman [54] διαπίστωσαν πώς σε ένα πακέτο υπολογιστικής άλγεβρας γραμμένο στη γλώσσα MacLisp, το 98% της μνήμης που ανακτούσε ένας κύκλος συλλογής είχε εκχωρηθεί μετά το πέρας του προηγούμενου κύκλου. Ο Zorn [123] ανέφερε πως το ποσοστό των αντικειμένων στη γλώσσα Common Lisp τα οποία δεν επιβίωναν μέχρι τα 10 KB εκχώρησης κυμαίνεται μεταξύ 50% και 90%. Αντίστοιχα συμπεράσματα ισχύουν και για τις συναρτησιακές γλώσσες προγραμματισμού. Οι Sansom και Simon Peyton Jones [103], παρατήρησαν, πώς στη Haskell, ένα ποσοστό μεταξύ 75% και 95% των αντικειμένων του σωρού πεθαίνουν πριν την ηλικία των 10 KB και πώς μόνο το 5% των αντικειμένων ζουν μετά το 1 MB. Ο Appel παρατήρησε πώς στην Standard ML/NJ, σε κάθε κύκλο συλλογής, το 98%

της μνήμης κάθε γενεάς ελευθερώνεται, ενώ οι Stefanovic και Moss [113] βρήκαν πώς μόνο ένα ποσοστό από 2% έως 8% αντικειμένων του σωρού επιβίωναν μετά το κατώφλι των 100 KB.

Η υπόθεση ισχύει και για πολλά προγράμματα γραμμένα σε αντικειμενοστρεφείς γλώσσες. Ο Ungar [117] βρήκε πως λιγότερο από το 7% των αντικειμένων στη Smalltalk ζουν πέραν των 140 KB. Οι Dieckmann και Hölzle [43] ανέφεραν πως ο όγκος των ζωντανών Java αντικειμένων στη σουίτα benchmark SPECjvm98 που επιβίωναν μετά τα 100 KB εκχώρησης κυμαινόταν μεταξύ 1% και 40% και πως λιγότερο από το 21 % ζούσε πέραν του 1 MB παρότι το ποσοστό διέφερε σημαντικά από εφαρμογή σε εφαρμογή. Οι Blackburn κ.ά [19] βρήκαν πως κατά μέσο όρο ένα ποσοστό μικρότερο του 9% των αντικειμένων που εκχωρούνταν στις εφαρμογές των σουιτών benchmark SPECjvm98 και DaCapo ζούσε και μετά τα 4 MB παρότι υπήρχε μεγάλη απόκλιση ανάμεσα στα διαφορετικά benchmark. Αυτό βέβαια ήταν ένα άνω φράγμα του ποσοστού των αντικειμένων που ζούσαν πέραν των 4 MB, καθώς μερικά επιζώντα αντικείμενα μπορεί να γλίτωναν καθώς εκχωρούνταν σχεδόν αμέσως μετά από μια συλλογή της νεότερης γενεάς. Οι Jones και Ryder [68] βρήκαν πως οι διάρκειες ζωής των αντικειμένων εφαρμογών Java ακολουθούσαν τη διωνυμική κατανομή: ένα ποσοστό μεταξύ 65% και 96% των αντικειμένων στις εφαρμογές της σουίτας benchmark DaCapo δεν επιβίωναν μετά τα 64 KB, ενώ ένα ποσοστό μεταξύ 3% και 16% ήταν αθάνατα ή ζούσαν και πέραν των 4 MB. Ακόμη και στις προστακτικές γλώσσες προγραμματισμού όπου δεν υπάρχει αυτόματη διαχείριση μνήμης, η διάρκεια ζωής πολλών αντικειμένων είναι μικρή. Οι Barrett και Zorn [14] ανέφεραν πώς παραπάνω από το 50% των εκχωρηθέντων στο σωρό αντικειμένων πέθαιναν κάποια στιγμή πριν τα 10 KB και λιγότερο από το 10% επιβίωνε πέραν των 32 KB.

Από την άλλη πλευρά, υπάρχουν εμφανώς λιγότερες ενδείξεις που να επιβεβαιώνουν την **ισχυρή γενεαλογική υπόθεση** του Hayes [58], σύμφωνα με την οποία ακόμη και για τα αντικείμενα που δε δημιουργήθηκαν πρόσφατα, ισχύει πώς τα νεότερα αντικείμενα θα έχουν μικρότερο ρυθμό επιβίωσης από τα παλαιότερα. Το απλό μοντέλο της ασθενούς γενεαλογικής υπόθεσης περιγράφει ικανοποιητικά τη γενική συμπεριφορά των αντικειμένων. Ωστόσο, εξαιρώντας τα αντικείμενα που πεθαίνουν νέα, η δημογραφία των αντικειμένων σε μία ευρύτερη χρονική κλίμακα είναι πιο πολύπλοκη. Οι χρόνοι ζωής των αντικειμένων δεν είναι τυχαίοι. Όπως παρατηρούν οι Dieckmann και Hölzle [43] και οι Jones και Ryder [68], επειδή τα προγράμματα λειτουργούν σε φάσεις, τα αντικείμενα έχουν την τάση να ζουν σε συστάδες και να πεθαίνουν όλα μαζί ταυτόχρονα. Επιπλέον, ένας σημαντικό πλήθος αντικειμένων μπορεί να μην πεθάνει ποτέ. Κάποιοι ερευνητές έχουν ακόμη ισχυρισθεί πώς ενδέχεται να υπάρχει ένας συσχετισμός μεταξύ της διάρκειας ζωής των αντικειμένων και του μεγέθους τους. Οι απόψεις πάντως σχετικά το κατά πόσο ο ισχυρισμός είναι αληθής διαφέρουν ανάμεσα στους Caudill και Wirfs-Brock [33], τους Ungar και Jackson [115] και Barrett και Zorn [14].

7.3 Γενεές και οργάνωση σωρού

Μια ευρεία γκάμα στρατηγικών έχει προταθεί στη βιβλιογραφία για την οργάνωση των γενεών. Οι γενεαλογικοί συλλέκτες μπορεί να χρησιμοποιούν δύο ή και περισσότερες γενεές, οι οποίες μπορεί να διαχωρίζονται φυσικά ή λογικά. Το μέγεθος μιας γενεάς μπορεί να είναι σταθερό και φραγμένο ή μπορεί να απαιτείται συμβιβασμός μεταξύ των μεγεθών των διαφόρων υποχώρων του σωρού. Μια γενεά μπορεί να είναι επίπεδη στο εσωτερικό της ή να περιλαμβάνει έναν αριθμό από υποχώρους που χαρακτηρίζονται με βάση την ηλικία και είναι γνωστοί ως **κάδοι**. Μια γενεά ενδέχεται επίσης να περιέχει το δικό της υποχώρο για μεγάλα αντικείμενα. Τέλος, κάθε γενεά μπορεί να μεταχειρίζεται από διαφορετικό αλγόριθμο.

Οι πρωταρχικοί στόχοι της γενεαλογικής συλλογής απορριμμάτων είναι η μείωση των χρόνων παύσης και η βελτίωση της διεκπεραιωτικής ικανότητας. Υποθέτοντας πώς η διαχείριση της νεότερης γενεάς έχει ανατεθεί σε ένα συλλέκτη αντιγραφής, οι αναμενόμενοι χρόνοι παύσης εξαρτώνται κατά κύριο λόγο από τον όγκο των ζωντανών αντικειμένων που επιβιώνουν από μία **ελάσσονα** συλλογή της γενεάς αυτής, ο οποίος με τη σειρά του εξαρτάται από το μέγεθος αυτής. Αν ωστόσο το μέγεθος του βρεφοκομείου είναι πολύ μικρό και συνεπώς η συλλογή πολύ γρήγορη, το ποσό της μνήμης που θα ελευθερωθεί θα είναι μικρό, καθώς τα αντικείμενα του βρεφοκομείου δεν έχουν επαρκή χρόνο μέχρι να πεθάνουν. Το γεγονός αυτό ενδέχεται να έχει πολλές ανεπιθύμητες επιπτώσεις.

Πρώτον, θα αυξηθεί σημαντικά η συχνότητα συλλογής της νεότερης γενεάς. Παράλληλα, το κόστος της αντιγραφής, το οποίο είναι ανάλογο του πλήθους των αντικειμένων που επιβιώνουν αναμένεται να αυξηθεί επίσης, δεδομένου ότι τα αντικείμενα θα έχουν λιγότερο χρόνο ώστε να πεθάνουν. Επίσης, κάθε κύκλος συλλογής απαιτεί την αναστολή των νημάτων-τροποποιητών και τη σάρωση της στοίβας κάθε νήματος-τροποποιητή.

Δεύτερον, η παλαιότερη γενεά αναμένεται να γεμίσει πολύ γρήγορα με συνέπεια να χρειαστεί να συλλεγεί και αυτή συντομότερα. Υψηλοί ρυθμοί προώθησης θα προκαλέσουν αύξηση της συχνότητας συλλογής της παλαιότερης γενεάς ή και όλων των γενεών. Επιπροσθέτως, η πρόωρη προώθηση αντικειμένων αυξάνει την πιθανότητα της εμφάνισης "οικογενειοκρατίας", καθώς αντικείμενα που έχουν εγκατασταθεί μόνιμα στην παλαιά γενιά διατηρούν εν ζωή τους νεκρούς απογόνους τους στη νέα γενεά, προκαλώντας μία τεχνητή αύξηση του ρυθμού επιβίωσης.

Τρίτον, υπάρχουν σημαντικές ενδείξεις πως τα προσφάτως δημιουργηθέντα αντικείμενα τροποποιούνται συχνότερα από ότι τα παλαιότερα. Αν τα αντικείμενα αυτά προωθηθούν πρόωρα, ο υψηλός ρυθμός επεξεργασίας τους θα προσθέσει επιπλέον πίεση στις λειτουργίες εγγραφής του τροποποιητή. Αυτό είναι ιδιαίτερα ανεπιθύμητο, ειδικά στην περίπτωση όπου το κόστος του φράγματος εγγραφής είναι υψηλό.

Τέλος, η προώθηση αντικειμένων προκαλεί την αραίωση των δεδομένων επεξεργασίας ενός προγράμματος. Η ταυτόχρονη ελαχιστοποίηση του χρόνου διάρκειας των μικρών συλλογών καθώς και του πλήθους των **μειζόνων** και πιο ακριβών συλλογών από τη μία πλευρά, και η αποφυγή της σημαντικής επιβάρυνσης του τροποποιητή με ενέργειες που αφορούν στην αυτόματη διαχείριση μνήμης από την άλλη, είναι οι δύο αντικρουόμενοι στόχοι μεταξύ των οποίων η γενεαλογική οργάνωση προσπαθεί να επιτύχει συμβιβασμό.

7.4 Πολλαπλές γενεές

Η προσθήκη περισσότερων γενεών αποτελεί μία απάντηση στο δίλημμα του πώς να διατηρηθούν μικροί χρόνοι παύσης για τις συλλογές του βρεφοκομείου ενώ ταυτόχρονα να αποφευχθούν οι πλήρεις συλλογές λόγω του ότι η παλαιότερη γενεά γεμίζει γρήγορα. Ο ρόλος των ενδιάμεσων γενεών αφορά το φιλτράρισμα εκείνων των αντικειμένων που έχουν επιβιώσει από τη συλλογή της νεότερης γενεάς και δε ζουν πολύ ακόμα. Εάν ένας συλλέκτης προωθεί όλα τα ζωντανά αντικείμενα **μαζικά** από τη νεότερη γενεά, στους επιζώντες θα περιλαμβάνονται και τα προσφάτως δημιουργηθέντα αντικείμενα παρά το ότι αυτά αναμένεται να πεθάνουν σύντομα. Με τη χρήση πολλαπλών γενεών, το μέγεθος της μικρότερης γενεάς δύναται να κρατηθεί αρκετά μικρό ώστε να ικανοποιούνται οι απαιτήσεις που αφορούν στους χρόνους παύσης χωρίς να αυξάνεται ο όγκος των δεδομένων που πεθαίνουν στην παλαιότερη γενεά σύντομα μετά την προαγωγή τους σε αυτή.

Η χρήση πολλαπλών γενεών μπορεί να έχει πολλαπλά μειονεκτήματα. Τα περισσότερα συστήματα συλλέγουν όλες τις νέες γενεές πριν συλλέξουν την παλαιότερη. Η τεχνική προσφέρει το πλεονέκτημα πως απαιτείται η καταγραφή μόνο των δεικτών από αντικείμενα μιας παλιάς γενεάς προς αντικείμενα μιας νέας. Δυστυχώς όμως οι δείκτες αυτοί εμφανίζονται σπανιότερα από ότι οι αντίστοιχοι δείκτες με αντίστροφη κατεύθυνση. Παρότι ο απαιτούμενος χρόνος για τη συλλογή μιας ενδιάμεσης γενεάς είναι μικρότερος από τον αντίστοιχο χρόνο για τη συλλογή ολόκληρου του σωρού, οι χρόνοι παύσης θα είναι μεγαλύτεροι από τους αντίστοιχους χρόνους παύσης της συλλογής της νεότερης γενιάς μόνο. Οι γενεαλογικοί συλλέκτες πολλαπλών γενεών επίσης είναι πιο σύνθετοι στην υλοποίηση και μπορεί να εισάγουν επιπρόσθετα κόστη: το κρίσιμο από άποψη επίδοσης τμήμα του κώδικα που υλοποιεί την εξερεύνηση του γράφου των αντικειμένων καλείται να ξεχωρίζει ανάμεσα σε πολλές γενεές και όχι μόνο δύο (κάτι το οποίο συνήθως πραγματοποιείται με έναν απλό έλεγχο έναντι μιας διεύθυνσης, η οποία μπορεί να είναι μία σταθερά μεταγλώττισης). Η αύξηση των γενεών επίσης αναμένεται να αυξήσει το πλήθος των δημιουργούμενων διαγενεαλογικών δεικτών, γεγονός το οποίο με τη σειρά του ενδέχεται να αυξήσει την πίεση στο φράγμα εγγραφής του τροποποιητή. Τέλος, η οργάνωση του σωρού σε πολλαπλές γενεές τείνει να αυξάνει το μέγεθος του συνόλου ριζών των νεότερων γενεών καθώς προάγονται αντικείμενα τα οποία δε θα είχαν προαχθεί αν ο χώρος των ενδιάμεσων γενεών είχε χρησιμοποιηθεί για την αύξηση του μεγέθους της νεότερης γενεάς.

Παρότι πολλοί αρχικοί γενεαλογικοί συλλέκτες για τις γλώσσες Smalltalk και Lisp λειτουργούσαν με πολλές γενεές, σχεδόν όλοι οι σύγχρονοι γενεαλογικοί συλλέκτες για αντικειμενοστρεφείς γλώσσες χρησιμοποιούν μόνο δύο. Σύμφωνα με τους Marlow κ.ά. [80], ακόμη και όταν ο σωρός οργανώνεται σε περισσότερες των δύο γενεών, όπως για παράδειγμα συμβαίνει στις υλοποιήσεις συναρτησιακών γλωσσών όπου οι ρυθμοί γέννησης και θνησιμότητας αντικειμένων είναι ιδιαίτερα υψηλοί, δύο γενεές είναι διαθέσιμες από προεπιλογή. Σε αυτές τις περιπτώσεις, μηχανισμοί στο εσωτερικό των γενεών και ειδικότερα της νεότερης μπορούν να χρησιμοποιηθούν για έλεγχο του ρυθμού προαγωγής των αντικειμένων.

7.5 Καταγραφή ηλικίας

Η καταγραφή της ηλικίας των αντικειμένων είναι άρρηκτα συνδεδεμένη με την πολιτική προαγωγής τους. Η χρήση πολλαπλών γενεών παρέχει ένα μη ακριβή μηχανισμό καταμέτρησης της ηλικίας των αντικειμένων. Στη συνέχεια εξετάζουμε τρόπους με τους οποίους η νεότερη γενιά μπορεί να δομηθεί ούτως ώστε να ελέγχεται ο ρυθμός προαγωγής.

Η απλούστερη οργάνωση είναι κάθε γενεά εκτός από την παλαιότερη να υλοποιηθεί ως ένας απλός ημιχώρος. Οποτεδήποτε συλλέγεται η νεότερη γενεά, τα επιζώντα αντικείμενα προάγονται **μαζικά** στην επόμενη γενεά. Η στρατηγική αυτή χαρακτηρίζεται από απλότητα και βέλτιστη χρησιμοποίηση του χώρου του σωρού που φιλοξενεί τη νεότερη γενεά. Δεν απαιτείται η καταγραφή της ηλικίας για κάθε αντικείμενο ξεχωριστά, ούτε και υπάρχει η ανάγκη για τη διατήρηση ενός εφεδρικού ημιχώρου αντιγραφής για κάθε γενεά (εκτός ίσως για την παλαιότερη, αν αυτή διαχειρίζεται από συλλογή με αντιγραφή). Οι Blackburn κ.ά. [20] αναφέρουν πώς πο γενεαλογικοί συλλέκτες που χρησιμοποιούνται από το διαχειριστή μνήμης MMTk στην εικονική μηχανή για τη γλώσσα Java προάγουν με αυτόν τον τρόπο τα αντικείμενα. Ωστόσο, οι Barrett και Zorn [14] διαπίστωσαν πώς η μαζική προαγωγή κάθε ζωντανού αντικειμένου (σε ένα σύστημα Lisp) ενδέχεται να προκαλέσει ρυθμούς προαγωγής από 50% έως και 100% υψηλότερους συγκριτικά με μία στρατηγική όπου τα αντικείμενα προάγονται μόνο αφού έχουν επιβιώσει από περισσότερες της μιας ελασσόνων συλλογών.

Η προαγωγή αντικειμένων μπορεί να καθυστερήσει με τη δόμηση μιας γενεάς σε δύο ή περισσότερους **ημιχώρους γήρανσης**. Η τεχνική αυτή επιτρέπει στα αντικείμενα να αντιγράφονται πολλές φορές από το χώρο-από στο χώρο-προς εντός μιας γενεάς πριν αυτά προαχθούν στην αμέσως επόμενη. Ο συλλέκτης των Lieberman και Hewitt [79] συλλέγει μια γενεά πολλές φορές πριν τελικά προαγάγει μαζικά τα αντικείμενα που επιβιώνουν. Σε κάθε περίπτωση, είτε όλα τα ζωντανά αντικείμενα του χώρου-από αντιγράφονται στο χώρο-προς εντός της γενεάς είτε προάγονται στην επόμενη γενεά, ανάλογα με την ηλικία της γενεάς συνολικά. Ενώ αυτή η διευθέτηση προσφέρει στα παλαιότερα αντικείμενα της γενεάς περισσότερο χρόνο για να πεθάνουν, τα νεότερα αντικείμενα ενδέχεται να προαχθούν πρόωρα.

Η εικονική μηχανή ExactVM επίσης υλοποίησε τη νεότερη από τις δύο γενεές ως ένα ζεύγος ημιχώρων γήρανσης ωστόσο ήλεγχε την προαγωγή κάθε αντικειμένου ξεχωριστά, χρησιμοποιώντας 5 bits από την επικεφαλίδα δύο λέξεων αυτού για να καταγράφει την ηλικία του. Παρότι η τεχνική αυτή αποτρέπει την πρόωρη προαγωγή των νεότερων αντικειμένων, προσθέτει μια λειτουργία πρόσθεσης κατά την επεξεργασία κάθε ζωντανού αντικειμένου της νεότερης γενεάς.

Η οργάνωση μιας γενεάς ως **συστοιχία κάδων** επιτρέπει ένα λεπτότερο ηλικιακό διαχωρισμό των αντικειμένων χωρίς την ξεχωριστή αποθήκευση της ηλικίας για κάθε αντικείμενο. Μια γενεά διαιρείται σε ένα αριθμό από κάδους (υποχώρους) και τα αντικείμενα μεταβιβάζονται από τον ένα κάδο στον επόμενο σε κάθε συλλογή. Τα αντικείμενα από τον παλαιότερο κάδο προωθούνται στην αμέσως επόμενη γενεά. Με αυτήν την οργάνωση, σε ένα σύστημα με n κάδους, ένα αντικείμενο δεν προωθείται στην επόμενη γενεά αν δεν έχει επιβιώσει από n συλλογές. Ο μεταγλωττιστής Glasgow Haskell Compiler (ghc) επιτρέπει αυθαίρετο πλήθος κάδων εντός μιας γενεάς. Ο Shaw [105] διαιρεί επιπλέον κάθε κάδο σε ένα ζεύγος ημιχώρων και τα αντικείμενα που επιβιώνουν από τη συλλογή αντιγράφονται μεταξύ κάθε ζεύγους b φορές πριν μετακινηθούν στον επόμενο κάδο. Με τον τρόπο αυτό, ένα αντικείμενο προάγεται στην επόμενη γενεά αν έχει επιβιώσει από $2b - 1$ έως και $2b$ συλλογές. Καθώς μάλιστα οι γενεές είναι συνεχόμενες, ο γηραιότερος κάδος μπορεί να συγχωνευτεί με την παλαιά γενεά καθυστερώντας την προαγωγή μέχρις ότου ο χώρος-προς αυτού γειτνιάσει με την παλαιά γενεά. Εκείνη τη χρονική στιγμή, ολόκληρος ο κάδος προάγεται με απλή ρύθμιση του συνόρου μεταξύ των γενεών.

Είναι σημαντικό να καταλάβει κανείς τις διαφορές μεταξύ κάδων και γενεών. Και οι δύο διαχωρίζουν αντικείμενα με βάση την ηλικία τους, ωστόσο διαφορετικές γενεές συλλέγονται με διαφορετική συχνότητα ενώ όλοι οι κάδοι στο εσωτερικό μιας γενεάς συλλέγονται ταυτόχρονα. Επιπλέον, καθώς μια γενεά ενδέχεται να συλλέγεται αργότερα από ότι μια άλλη, είναι απαραίτητη η καταγραφή δεικτών από αντικείμενα που ανήκουν σε μία παλαιά γενεά προς αντικείμενα που ανήκουν σε μία νέα γενεά. Αντίθετα, δεν είναι απαραίτητη η καταγραφή δεικτών από αντικείμενα ενός κάδου προς αντικείμενα ενός άλλου κάδου στο εσωτερικό μιας γενεάς. Με τη διαίρεση της νεότερης γενεάς σε κάδους και την ελάττωση της πρόωρης προαγωγής, η πίεση στο φράγμα εγγραφής μπορεί να ελαττωθεί με ταυτόχρονο έλεγχο της προαγωγής χωρίς να απαιτείται η αποθήκευση της ηλικίας ανά αντικείμενο.

Όλες οι παραπάνω οργανώσεις χαρακτηρίζονται από σπατάλη χώρου καθώς ο μισός χώρος μιας γενεάς δεσμεύεται για αντιγραφή. Ο Ungar [117] οργάνωσε τη νέα γενεά σε ένα μεγάλο χώρο δημιουργίας (εκχώρησης) ο οποίος είναι γνωστός και ως **εδέμ** και σε δύο μικρότερους **ημιχώρους επιβίωσης**, το χώρο-από επιβίωσης και το χώρο-προς επιβίωσης. Ως συνήθως, τα αντικείμενα εκχωρούνται στην περιοχή της εδέμ η οποία συλλέγεται μαζί με τον χώρο-προς επιβίωσης σε κάθε μικρή συλλογή. Τα ζωντανά αντικείμενα της εδέμ προάγονται στο χώρο-προς επιβίωσης, ενώ τα ζωντανά αντικείμενα του χώρου-από επιβίωσης είτε αντιγράφονται

στο χώρο-προς επιβίωσης είτε προάγονται στην επόμενη γενεά, ανάλογα με την ηλικία τους. Η οργάνωση αυτή βελτιώνει τη χρησιμοποίηση του διαθέσιμου χώρου καθώς το μέγεθος της εδέμ είναι κατά πολύ μεγαλύτερο από το συνολικό μέγεθος των ημιχώρων επιβίωσης. Για παράδειγμα, στην εικονική μηχανή HotSpot για τη γλώσσα Java η προεπιλεγμένη αναλογία είναι 32:1. Η πολιτική προαγωγής δεν ορίζει ένα αυστηρό όριο ηλικίας, αλλά αντίθετα προσπαθεί να διατηρήσει την περιοχή των ημιχώρων επιβίωσης μισογεμάτη.

7.6 Προσαρμογή στη συμπεριφορά του προγράμματος

Η προσαρμογή της διαμόρφωσης ενός διαγενεαλογικού συλλέκτη στη συμπεριφορά του τροποποιητή είναι απαραίτητη καθώς οι κατανομές των χρόνων ζωής των αντικειμένων δεν είναι τυχαίες αλλά ούτε και στατικές. Τα αληθινά προγράμματα συνήθως λειτουργούν σε φάσεις και υπάρχει μια ευρεία ποικιλία κοινών μοτίβων συμπεριφοράς. Η ύπαρξη ενός συνόλου από ζωντανά αντικείμενα που συσσωρεύονται σταδιακά σε μια γενεά του σωρού και στη συνέχεια πεθαίνουν ταυτόχρονα εμφανίζεται συχνά στην πράξη. Εναλλακτικά, αφού διαβούν μια συγκεκριμένη ηλικία, τα αντικείμενα αυτά μπορεί να συνεχίσουν να ζουν για πολύ καιρό. Μια μη αυστηρή συμμόρφωση της δημογραφίας προς την ασθενή γενεαλογική υπόθεση μπορεί να δημιουργήσει προβλήματα στο συλλέκτη. Εάν ένας μεγάλος όγκος δεδομένων που ζει αρκετά, ώστε να προαχθεί σε μια παλαιότερη γενεά, πεθάνει σύντομα μετά την προαγωγή, η επίδοση θα μειωθεί. Οι Ungar και Jackson [115], [116] έχουν προτείνει διάφορους μηχανισμού ελέγχου της πολιτικής προαγωγής.

Η προσαρμογή των συλλεκτών απορριμμάτων στη συμπεριφορά του τροποποιητή είναι γενικότερα επιθυμητή και χρησιμοποιείται με στόχο τόσο τη μείωση των αναμενόμενων χρόνων παύσης όσο και τη βελτίωση της συνολικής ρυθμαπόδοσης. Ο απλούστερος μηχανισμός δρομολόγησης του συλλέκτη ορίζει την κλήση του τελευταίου οποτεδήποτε ο εκχωρητής ξεμένει από μνήμη. Ένας γενεαλογικός διαχειριστής μνήμης ωστόσο μπορεί να ελέγξει τους αναμενόμενους χρόνους παύσης προσαρμόζοντας κατάλληλα το μέγεθος της νεότερης γενεάς: μια μικρότερη γενεά μειώνει τον όγκο των αντικειμένων που θα διασωθούν αλλά και τον όγκο των αντικειμένων που θα εκκαθαρισθούν κατά τη συλλογή αυτής. Το μέγεθος της νεότερης γενεάς επηρεάζει επίσης το ρυθμό προαγωγής αντικειμένων μεταξύ δύο διαφορετικών γενεών. Αν η γενεά είναι πολύ μικρή και δεν παρέχει στα αντικείμενα αρκετό χρόνο για να πεθάνουν, ο ρυθμός προαγωγής θα είναι υψηλότερος. Αν αντίθετα η νεότερη γενεά έχει πολύ μεγάλο μέγεθος, το διάστημα μεταξύ δύο διαδοχικών συλλογών θα είναι μεγαλύτερο και ένα μικρότερο ποσοστό αντικειμένων θα επιβιώσει και θα φθάσει στην παλαιότερη γενεά.

7.6.1 Συλλογή απορριμμάτων κατά Appel

Ο Appel [4] εισήγαγε έναν προσαρμοστικό γενεαλογικό συλλέκτη απορριμμάτων για τη γλώσσα προγραμματισμού Standard ML, ο οποίος δοθέντος ενός προϋπολογισμού μνήμης, αφιερώνει το μέγιστο δυνατό χώρο στη νεότερη γενεά μη χρησιμοποιώντας σταθερά μεγέθη για τις γενεές του σωρού. Το σχήμα αυτό είναι σχεδιασμένο ειδικά για περιβάλλοντα όπου η θνησιμότητα των νέων αντικειμένων είναι υψηλή: στη γλώσσα ML, τυπικά μόλις το 2% των αντικειμένων της νέας γενεάς επιβίωνε από έναν κύκλο συλλογής. Ο σωρός διαιρείται σε τρεις περιοχές: την παλαιά γενεά, ένα εφεδρικό αντίγραφο και τη νέα γενεά. Η συλλογή της νεότερης γενεάς προάγει όλους τους νέους επιζώντες στο τέλος της παλαιάς γενεάς. Μετά τη συλλογή, ο χώρος που δεν χρειάζεται για αντικείμενα της παλαιάς γενεάς διαιρείται

ισομερώς και προς δημιουργία του εφεδρικού αντιγράφου και μιας καινούριας νέας γενεάς. Αν ο χώρος που εκχωρείται στη νέα γενεά πέσει κάτω από ένα ορισμένο κατώφλι, συλλέγεται ολόκληρος ο σωρός.

Το πλεονέκτημα της συλλογής κατά Appel είναι πως με τη δυναμική προσαρμογή του μεγέθους του εφεδρικού αντιγράφου, προσφέρεται καλή χρησιμοποίηση μνήμης και μειώνεται το πλήθος απαιτούμενων κύκλων συλλογής σε σύγκριση με διαμορφώσεις που χρησιμοποιούν μαζική προαγωγή αντικειμένων και σταθερό μέγεθος για τη νέα γενεά. Ωστόσο, απαιτείται προσοχή ώστε να αποφευχθεί η υπερβολική αύξηση της συχνότητας κλήσης του συλλέκτη. Benchmarks με υψηλούς ρυθμούς εκχώρησης και με χαμηλό ρυθμό προαγωγής αντικειμένων εμφανίζονται συχνά στην πράξη. Αυτό μπορεί να οδηγήσει σε μία κατάσταση όπου ο χώρος του σωρού που καταλαμβάνει η νέα γενεά συρρικνώνεται σε τέτοιο βαθμό ώστε η συχνότητα των μικρών συλλογών να είναι υπερβολικά υψηλή αλλά ο όγκος των δεδομένων που προάγονται να μην είναι επαρκής για την πυροδότηση μιας μεγάλης συλλογής. Για να αντιμετωπισθεί το πρόβλημα, η παλαιά γενεά πρέπει να συλλέγεται οποτεδήποτε το μέγεθος της νέας γενεάς πέσει κάτω από ένα ορισμένο ελάχιστο κατώφλι.

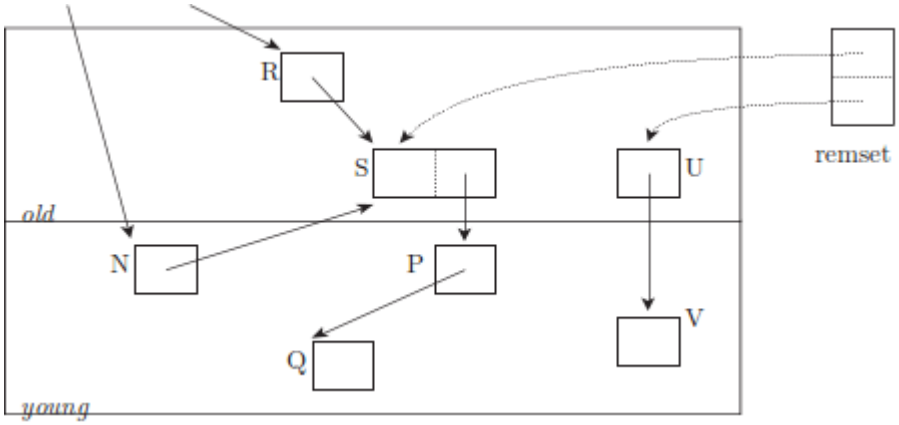
7.6.2 Αναδραστικός έλεγχος προαγωγής

Άλλα σχήματα για τον έλεγχο του ρυθμού προαγωγής σχετίζονται πιο άμεσα με στόχους που αφορούν τους χρόνους παύσης. Οι Ungar και Jackson [115], [116] προτείνουν τη **δημογραφική προαγωγή με ανάδραση** ώστε να εξομαλύνουν τις παύσεις μακράς διάρκειας που επιφέρει η προαγωγή αντικειμένων που πεθαίνουν σύντομα μετά την προαγωγή. Ο όγκος των δεδομένων που προάγονται σε μία συλλογή χρησιμοποιείται για την πρόβλεψη της χρονικής διάρκειας του επόμενου κύκλου συλλογής και για να επιταχύνει ή επιβραδύνει την προαγωγή.

Παρότι ο μηχανισμός αυτός μπορεί να ελέγξει τους ρυθμούς προαγωγής, αδυνατεί να υποβιβάσει αντικείμενα από μια παλαιότερη γενεά σε μία νεότερη. Οι Barrett και Zorn [15] μεταβάλλουν το σύνορο μεταξύ δύο γενεών και προς τις δύο κατευθύνσεις. Η τεχνική τους χαρακτηρίζεται από το κόστος της επιβεβλημένης καταγραφής περισσότερων δεικτών, καθώς η θέση του διαγενεαλογικού συνόρου δεν μπορεί να προβλεφθεί στατικά.

7.7 Διαγενεαλογικοί δείκτες

Οι ρίζες μιας γενεάς πρέπει να ανακαλυφθούν πριν αυτή συλλεχθεί. Οι ρίζες για μια γενεά δεν περιλαμβάνουν μόνο τους δείκτες που βρίσκονται σε καταχωρητές, στη στοίβα και σε καθολικές μεταβλητές αλλά και δείκτες προς αντικείμενα της γενεάς από αντικείμενα που ζουν σε κάποια άλλη περιοχή του σωρού η οποία δε συλλέγεται ταυτόχρονα με τη γενεά. Αυτές οι περιοχές τυπικά περιλαμβάνουν τις παλαιότερες γενεές αλλά και περιοχές εκτός του γενεαλογικού σωρού όπως χώροι που φιλοξενούν πολύ μεγάλα αντικείμενα και χώροι που δε συλλέγονται ποτέ, όπως οι χώροι που αποθηκεύουν αθάνατα αντικείμενα και πιθανώς κώδικα. Όπως έχουμε εξηγήσει, οι διαγενεαλογικοί δείκτες δημιουργούνται είτε με την αρχικοποίηση εγγραφών κατά τη δημιουργία των αντικειμένων, είτε από ενημερώσεις πεδίων δεικτών αντικειμένων από τον τροποποιητή είτε τέλος κατά την μετακίνηση αντικειμένων σε διαφορετικές γενεές. Γενικά, οι δείκτες αυτοί πρέπει να εντοπίζονται τη στιγμή της δημιουργίας τους και να καταγράφονται ώστε να μπορούν να χρησιμοποιηθούν ως ρίζες κατά τη συλλογή μιας γενεάς. Κάθε δείκτης που πρέπει να καταγράφεται καλείται συνήθως και **ενδιαφέρων δείκτης**.



Σχήμα 7.1: Διαγενεαλογικοί δείκτες. Η διατήρηση των ζωντανών αντικειμένων της νέας γενεάς χωρίς την εξερεύνηση όλου του σωρού, απαιτεί ένα μηχανισμό και μια δομή δεδομένων για την καταγραφή των αντικειμένων S και U που περιέχουν δείκτες προς αντικείμενα της νέας γενεάς.

7.7.1 Σύνολα ανάμνησης

Οι δομές δεδομένων που χρησιμοποιούνται για την καταγραφή διαγενεαλογικών δεικτών ονομάζονται **σύνολα ανάμνησης**. Τα σύνολα ανάμνησης καταγράφουν τη θέση προέλευσης διαφόρων δεικτών μεταξύ διαφορετικών χώρων του σωρού. Καταγράφεται η προέλευση και όχι ο προορισμός ενός δείκτη για δύο λόγους. Πρώτον, επιτρέπεται σε ένα μετακινούντα συλλέκτη να ενημερώσει το πεδίο προέλευσης με τη νέα διεύθυνση ενός αντικειμένου που έχει αντιγραφεί η προαχθεί. Δεύτερον ένα πεδίο προέλευσης κάποιου δείκτη μπορεί να ενημερωθεί πολλές φορές στο διάστημα ανάμεσα σε δύο διαδοχικούς κύκλους συλλογής και έτσι αν ο συλλέκτης θυμάται την προέλευση του δείκτη, εξασφαλίζεται πως αυτός επεξεργάζεται το πραγματικό αντικείμενο προς το οποίο αναφέρεται ο δείκτης τη στιγμή της προέλευσης και όχι τα αντικείμενα προορισμούς ενδιάμεσων παρωχημένων τιμών του αυτού. Επομένως το σύνολο ανάμνησης για κάθε γενεά αποθηκεύει τις θέσεις εκείνες στις οποίες βρίσκεται ένας πιθανόν ενδιαφέρων δείκτης προς κάποιο αντικείμενο της γενεάς. Οι διάφορες υλοποιήσεις των συνόλων ανάμνησης διαφέρουν ως προς την ακρίβεια με την οποία καταγράφουν τέτοιες θέσεις. Η επιλογή της ακρίβειας προσπαθεί να επιτύχει ένα συμβιβασμό μεταξύ της επιβάρυνσης του τροποποιητή, του απαιτούμενου χώρου αποθήκευσης των συνόλων ανάμνησης και του κόστους επεξεργασίας τους από το συλλέκτη.

Εμφανώς είναι επιθυμητός ο εντοπισμός και η καταγραφή όσο το δυνατόν λιγότερων δεικτών. Οι ενημερώσεις δεικτών από το συλλέκτη κατά τη μετακίνηση αντικειμένων εντοπίζονται εύκολα. Οι εγγραφές δεικτών από τον τροποποιητή μπορούν να ανιχνευθούν από ένα φράγμα εγγραφής λογισμικού, το οποίο μπορεί να εισαχθεί αυτόματα από το μεταγλωττιστή πριν από κάθε λειτουργία εγγραφής ενός δείκτη. Αυτό βέβαια δεν είναι εφικτό αν ο μεταγλωττιστής της γλώσσας είναι μη συνεργατικός. Στην περίπτωση αυτή, οι θέσεις των λειτουργιών εγγραφής μπορούν συχνά να καθορισθούν από το διαχειριστή εικονικής μνήμης του λειτουργικού συστήματος.

Η συχνότητα των ενημερώσεων δεικτών διαφέρει μεταξύ των διαφορετικών γλωσσών προγραμματισμού και των υλοποιήσεων αυτών. Ο Zorn εφαρμόζοντας στατική ανάλυση σε μια σουίτα προγραμμάτων σε Lisp [124], υπολόγισε τη συχνότητα των ενημερώσεων δεικτών από 13% έως και 15%, ενώ ο Appel υπολόγισε μία χαμηλότερη στατική συχνότητα της τάξης του

3% στη γλώσσα Lisp [3], και μία δυναμική συχνότητα χρόνου εκτέλεσης της τάξης του 1% για τη γλώσσα ML [4]. Οι Dieckmann και Hölzle [43] βρήκαν τέλος πώς τα προγράμματα σε γλώσσα Java μπορεί να διαφέρουν σημαντικά ως προς τη συχνότητα των ενημερώσεων δεικτών (το ποσοστό των προσβάσεων στο σωρό που αφορούσε αποθηκεύσεις τιμών σε πεδία δείκτες κυμαινόταν από 6% έως και 70%).

Αν τα πλαίσια αυτά σαρώνονται ως μέρος του συνόλου ριζών σε κάθε συλλογή, είναι δυνατή η ανακάλυψη των θέσεων τους που περιέχουν δείκτες. Επιπλέον, αν ο μεταγλωττιστής μπορεί να ταυτοποιήσει τις λειτουργίες εγγραφής στη στοίβα, τότε δε χρειάζεται να τοποθετήσει φράγματα εγγραφής πριν από αυτές. Επιπρόσθετα, πολλοί δείκτες αναφέρονται σε αντικείμενα της ίδιας διαμέρισης. Παρότι οι αποθηκεύσεις πιθανώς εντοπίζονται, οι δείκτες δεν είναι ενδιαφέροντες από γενεαλογική άποψη και δε χρειάζεται να καταγραφούν.

Εάν επιβληθεί μια τάξη ως προς τη σειρά συλλογής των διαφόρων γενεών, το πλήθος των διαγενεαλογικών δεικτών που πρέπει να καταγραφούν μπορεί να μειωθεί ακόμη περισσότερο. Εξασφαλίζοντας πώς οποτεδήποτε συλλέγεται μια γενεά συλλέγονται και όλες οι νεότερες γενεές από αυτή, μόνο οι δείκτες από παλαιά αντικείμενα προς νέα αντικείμενα πρέπει να καταγραφούν. Πολλές λειτουργίες εγγραφής δεικτών αφορούν την αρχικοποίηση πεδίων αντικειμένων που μόλις έχουν δημιουργηθεί. Εξ ορισμού, οι δείκτες αυτοί αναφέρονται σε γηραιότερα αντικείμενα. Δυστυχώς πολλές γλώσσες διαχωρίζουν την εκχώρηση μνήμης για ένα αντικείμενο από την αρχικοποίηση των πεδίων του τελευταίου, καθιστώντας δύσκολη τη διάκριση των λειτουργιών εγγραφής δεικτών που δεν αφορούν αρχικοποιήσεις και πιθανόν δημιουργούν δείκτες από μια παλαιά γενεά προς μία νέα γενεά. Άλλες γλώσσες παρέχουν περισσότερη υποστήριξη στο μεταγλωττιστή όσον αφορά την αναγνώριση των λειτουργιών εγγραφής δεικτών που δε χρειάζονται κάποιο φράγμα εγγραφής. Η πλειοψηφία των λειτουργιών εγγραφής δεικτών σε μία αγνή οκνηρή γλώσσα συναρτησιακού προγραμματισμού όπως η Haskell κάνει τους δείκτες να αναφέρονται σε γηραιότερα αντικείμενα. Η γλώσσα ML απαιτεί από τον προγραμματιστή να σημειώσει ρητά τις τροποποιήσιμες μεταβλητές: οι λειτουργίες εγγραφής αυτών των μεταβλητών είναι η μόνη πηγή δημιουργίας δεικτών από αντικείμενα μιας παλαιάς γενεάς προς αντικείμενα μιας νέας γενεάς. Το σκηνικό στις αντικειμενοστρεφείς γλώσσες προγραμματισμού από την άλλη όπως για παράδειγμα στη Java είναι πιο πολύπλοκο. Η φιλοσοφία προγραμματισμού εδώ βασίζεται στην ενημέρωση της κατάστασης αντικειμένων, κάτι που οδηγεί φυσικά στη δημιουργία περισσότερων δεικτών από αντικείμενα μιας νέας γενεάς προς αντικείμενα μιας παλαιάς γενεάς.

Σε σωρούς με πολλαπλές ανεξάρτητα συλλεγόμενες γενεές απαιτείται διαφορετική τεχνική φιλτραρίσματος δεικτών. Για παράδειγμα ένας συλλέκτης μπορεί να εφαρμόσει ευριστικές για να αποφασίσει ποιο χώρο θα συλλέξει, δίνοντας προτεραιότητα στους χώρους με το μικρότερο πλήθος ζωντανών αντικειμένων. Στην περίπτωση αυτή το φράγμα εγγραφής πρέπει να καταγράψει δείκτες και προς τις δύο κατευθύνσεις. Καθώς αυτή η σχεδίαση αναμένεται να αυξήσει το πλήθος των διαγενεαλογικών δεικτών προς καταγραφή, είναι βέλτιστη για χρήση σε μια υλοποίηση όπου το μέγεθος του συνόλου ανάμνησης είναι ανεξάρτητο του πλήθους των ενθυμούμενων δεικτών.

7.8 Διαχείριση χώρου

Η νεότερη γενεά συνήθως διαχειρίζεται με αντιγραφή. Τα επιζώντα αντικείμενα αντιγράφονται είτε σε ένα φρέσκο ημιχώρο στην ίδια γενεά είτε σε μία παλαιότερη γενεά. Οι συλλογές των νέων γενεών αναμένεται να είναι συχνές και σύντομες σε διάρκεια, λαμβάνοντας υπόψη την

ασθενή γενεαλογική υπόθεση. Από την άλλη πλευρά, οι συλλογές των παλαιότερων γενεών αναμένεται να είναι λιγότερο συχνές αλλά να διαρκούν πολύ, καθώς μαζί με αυτές πρέπει να συλλεγούν και οι νεότερες γενεές προκειμένου να αποφευχθεί το κόστος χρήσης ενός φράγματος εγγραφής που καταγράφει δείκτες και προς τις δύο κατευθύνσεις. Συνήθως μια συλλογή της παλαιότερης γενεάς θα συλλέξει όλες τις περιοχές του σωρού εκτός ίσως από την περιοχή όπου φιλοξενούνται αθάνατα αντικείμενα (παρότι οι δείκτες της περιοχής αυτής αντιμετωπίζονται ως ρίζες και ενδέχεται να ενημερωθούν). Μια πλήρης συλλογή του σωρού δεν απαιτεί τη χρήση συνόλων ανάμνησης, εκτός ίσως για τοποθεσίες της περιοχής αθάνατων αντικειμένων στην περίπτωση που αυτή δε σαρώνεται.

Έχει προταθεί μία ευρεία γκάμα στρατηγικών διαχείρισης της παλαιότερης γενεάς. Η συλλογή με αντιγραφή μεταξύ ημιχώρων δεν είναι η καλύτερη επιλογή ειδικά αν λάβει κανείς υπόψη πως και η νεότερη γενεά συλλέγεται με την ίδια τεχνική: η απαίτηση της συλλογής για την διατήρηση ενός εφεδρικού χώρου αντιγραφής καταλήγει να δαπανά το μισό χώρο στο σωρό, κάτι που έχει ως αποτέλεσμα την αύξηση της συχνότητας των συλλογών όλων των γενεών. Επίσης τα μακρόβια αντικείμενα μετακινούνται συνέχεια. Οι Blackburn κ.ά ισχυρίζονται [18] πώς η συλλογή με σήμανση και εκκαθάριση προσφέρει καλύτερη χρησιμοποίηση του διαθέσιμου χώρου, ειδικά σε μικρούς σωρούς. Το μειονέκτημα της συλλογής με σήμανση και εκκαθάριση, η οποία δεν μετακινεί τα ζωντανά αντικείμενα, αφορά στη μείωση της επίδοσης από τον πιθανό κατακερματισμό. Η λύση είναι η προσθήκη μιας φάσης συμπύκνωσης της παλαιάς γενεάς οποτεδήποτε ο κατακερματισμός επηρεάζει δυσμενώς την επίδοση. Η συμπύκνωση επίσης μπορεί να διαχειρισθεί τα μακρόβια αντικείμενα με ειδικό τρόπο, όπως είδαμε στο κεφάλαιο 3.

Οι γενεαλογικοί συλλέκτες σχεδόν πάντοτε διαχωρίζουν τις γενεές τους φυσικά και όχι εικονικά. Αυτό απαιτεί τη διαχείριση των νεότερων γενεών από συλλογή με αντιγραφή. Ένας προσαρμοστικός συλλέκτης αντιγραφής σαν και αυτόν του Appel για παράδειγμα συντηρητικά απαιτεί ο εφεδρικός χώρος αντιγραφής να έχει το ίδιο μέγεθος με την περιοχή που συλλέγεται καθώς στη χειρότερη περίπτωση όλα τα αντικείμενα μπορεί να επιβιώσουν. Στην πράξη βέβαια τα περισσότερα αντικείμενα δεν επιβιώνουν από μια συλλογή της νέας γενεάς.

Οι McGachey κ.ά. [82] ισχυρίζονται πώς η χρησιμοποίηση του χώρου μπορεί να βελτιωθεί με τη διατήρηση ενός μικρότερου μεγέθους εφεδρικού χώρου αντιγραφής και την εναλλαγή μεταξύ συλλογής με αντιγραφή σε συλλογή με σήμανση και συμπύκνωση οποτεδήποτε το μέγεθος του χώρου αυτού γίνει πολύ μικρό.

7.9 Αφηρημένη γενεαλογική συλλογή απορριμμάτων

Στην ενότητα αυτή εξετάζουμε πώς μπορούμε να εντάξουμε τη γενεαλογική συλλογή απορριμμάτων στο αφαιρετικό πλαίσιο συλλογής που εξετάσαμε στην ενότητα 6.6. Μία αφηρημένη παρουσίαση ενός συμβατικού, γενεαλογικού, δύο γενεών, μαζικής προαγωγής συλλέκτη της νεότερης γενεάς δίνεται στον αλγόριθμο 7.1.

Σε αναλογία με τους προηγούμενους αφηρημένους αλγορίθμους συλλογής απορριμμάτων, ο αλγόριθμος διατηρεί ένα πολυσύνολο I από αναβεβλημένες αυξήσεις μετρητών αναφορών αντικειμένων της νέας γενεάς. Ένα σύνολο ανάμνησης περιλαμβάνει όλα τα πεδία δείκτες με προέλευση την παλαιά γενεά και προορισμό τη νέα. Το πολυσύνολο I περιλαμβάνει ακριβώς τους διαγενεαλογικούς δείκτες προς αντικείμενα της νέας γενεάς και γι' αυτόν ακριβώς το λόγο η διαδικασία DECNURSERY αφαιρεί αναφορές από αυτό όταν η τιμή ενός ενθυμούμενου

Αλγόριθμος 7.1 Αφηρημένη γενεαλογική συλλογή απορριμμάτων: ρουτίνες συλλέκτη

```
1:  procedure COLLECTNURSERY(I)
2:      atomic
3:      ROOTSNURSERY(I)
4:      SCANNURSERY(I)
5:      SWEEPNURSERY()

6:  procedure SCANNURSERY(W)
7:      while not ISEMPTY(W) do
8:          src ← REMOVE(W)
9:          ρ(src) ← ρ(src) + 1                          ▷ shade src
10:         if ρ(src) = 1 then                           ▷ src was white, now grey
11:             for all fld in Pointers(src) do
12:                 ref ← *fld
13:                 if ref ∈ Nursery then
14:                     W ← W + [ref]

15: procedure SWEEPNURSERY()
16:     while not ISEMPTY(Nursery) do
17:         node ← REMOVE(Nursery)                       ▷ en masse promotion
18:         if ρ(node) = 0 then                          ▷ node is white
19:             FREE(node)

20: procedure ROOTSNURSERY(I)
21:     for all fld in Roots do
22:         ref ← *fld
23:         if ref ≠ null and ref ∈ Nursery then
24:             I ← I + [ref]
```

διαγενεαλογικού δείκτη μεταβάλλεται. Τελικώς αν ένα αντικείμενο n της νέας γενεάς εμφανίζεται στο I, τότε διατηρείται από το γενεαλογική συλλέκτη. Το πλήθος εμφάνισης αυτού μάλιστα ισούται με το μετρητή αναφορών του, χωρίς να συμπεριλαμβάνονται οι αναφορές από αντικείμενα της νέας γενεάς. Ένας αλγόριθμος συλλογής εξιχνίασης συνοψίζει σε ένα μόνο bit σήμανσης την ισχύ της πρότασης $n \in I$.

Όταν καλείται η διαδικασία COLLECTNURSERY, το πολυσύνολο I περιέχει τα αντικείμενα της νέας γενεάς με μηδενικό μετρητή αναφορών, λαμβάνοντας υπόψη αναφορές μόνο από αντικείμενα της παλαιάς γενεάς. Μετά την προσθήκη αναφορών προς αντικείμενα της νέας γενεάς από τη διαδικασία ROOTNURSERY, η νέα γενεά εξιχνιάζεται από τους κόμβους του πολυσυνόλου I (διαδικασία SCANNURSERY) και στη συνέχεια εκκαθαρίζεται (διαδικασία SWEEPNURSERY), όπου τα επιζώντα αντικείμενα προάγονται μαζικά στην παλαιά γενεά και τα μη προσβάσιμα αντικείμενα με μηδενικό μετρητή αναφορών ελευθερώνονται.

Αλγόριθμος 7.2 Αφηρημένη γενεαλογική συλλογή απορριμμάτων: ρουτίνες τροποποιητή

```
 1: function NEW()
 2:     ref ← ALLOCATE()
 3:     if ref = null then
 4:         COLLECTNURSERY(I)
 5:         ref ← ALLOCATE()
 6:         if ref = null then
 7:             COLLECT()                    ▷ tracing, counting, or other full-heap GC
 8:             ref ← ALLOCATE()
 9:             if ref = null then
10:                 error "Out of memory!"
11:     ρ(ref) ← 0                           ▷ node is black
12:     Nursery ← Nursery ∪ ref              ▷ allocate in Nursery
13:     return ref
14: procedure INCNURSERY(node)
15:     if ref ∈ Nursery then
16:         I ← I + [node]

17: procedure DECNURSERY(node)
18:     if ref ∈ Nursery then
19:         I ← I − [node]

20: procedure WRITE(src, i, ref)
21:     if src ≠ Roots and src ∉ Nursery then
22:         INCNURSERY(ref)
23:         DECNURSERY(src[i])
24:     src[i] ← dest
```

7.10 Θέματα προς εξέταση

Η γενεαλογική συλλογή απορριμμάτων έχει αποδειχθεί αποδοτική, προσφέροντας σημαντικές βελτιώσεις επίδοσης για μια ευρεία γκάμα εφαρμογών. Μειώνοντας το μέγεθος της νεότερης

γενεάς και επικεντρώνοντας τις προσπάθειες συλλογής σε αυτήν, οι αναμενόμενοι χρόνοι παύσης μπορούν να ελαττωθούν σε τέτοιο σημείο ώστε να περνούν σχεδόν απαρατήρητοι σε διάφορα περιβάλλοντα. Η τεχνική της γενεαλογικής συλλογής απορριμμάτων μπορεί επίσης να αυξήσει τη γενική ρυθμαπόδοση του συλλέκτη με δύο τρόπους. Πρώτον, μειώνει τη συχνότητα επεξεργασίας των μακρόβιων αντικειμένων και επομένως όχι μόνο μειώνει την υπολογιστική προσπάθεια αλλά επιπλέον δίνει στα αντικείμενα αυτά περισσότερο χρόνο για να πεθάνουν (και έτσι να μην χρειαστεί να εξιχνιαστούν οι υπογράφοι των οποίων είναι ρίζες). Δεύτερον, οι γενεαλογικοί συλλέκτες συνήθως εκχωρούν μνήμη για την αποθήκευση νέων αντικειμένων σειριακά σε μία περιοχή βρεφοκομείου. Η σειριακή εκχώρηση έχει ως αποτέλεσμα τη βελτίωση του ρυθμού αστοχιών κρυφής μνήμης καθώς το μοτίβο πρόσβασης στη μνήμη είναι προβλέψιμο και επιπλέον οι περισσότερες εγγραφές αφορούν τα νεότερα αντικείμενα.

Η γενεαλογική συλλογή απορριμμάτων ωστόσο δεν είναι πανάκεια. Η αποδοτικότητά της εξαρτάται άμεσα από την κατανομή της διάρκειας των αντικειμένων μιας εφαρμογής. Αν η μεγάλη πλειοψηφία των αντικειμένων δεν πεθαίνει νέα, η γενεαλογική συλλογή απορριμμάτων δεν είναι κατάλληλη.

Επιπλέον, η γενεαλογική συλλογή απορριμμάτων βελτιώνει μόνο τους αναμενόμενους χρόνους παύσης. Τελικώς, ο συλλέκτης πρέπει να συλλέξει όλο το σωρό και συνεπώς η γενεαλογική συλλογή από μόνη της δεν μπορεί να επιλύσει το πρόβλημα της μείωσης του χρόνου παύσης **χειρότερης περίπτωσης**, ο οποίος αυξάνεται υπερβολικά πολύ σε μεγάλους σωρούς. Συνεπώς, η γενεαλογική συλλογή απορριμμάτων δεν μπορεί να παράσχει τις απαιτούμενες εγγυήσεις ενός σκληρού συστήματος πραγματικού χρόνου όπου οι προθεσμίες πρέπει πάντοτε να τηρούνται.

Η υλοποίηση της συλλογής απορριμμάτων είναι απλούστερη αν υπάρχει η δυνατότητα μετακίνησης αντικειμένων προς διάκριση των παλαιών αντικειμένων από τα νέα. Ο φυσικός διαχωρισμός δεν προσφέρει μόνο το πλεονέκτημα της τοπικότητας που συζητήθηκε παραπάνω, αλλά επίσης πιο αποδοτικούς ελέγχους χώρου, οι οποίοι είναι απαραίτητοι στο φράγμα εγγραφής αλλά και κατά την εξιχνίαση της νεότερης γενεάς. Τα αντικείμενα πάντως μπορούν να διαχωρίζονται με βάση την ηλικία τους και εικονικά, χρησιμοποιώντας πιθανώς ένα bit στην επικεφαλίδα τους ή σε κάποιο bitmap.

Οι γενεαλογικοί συλλέκτες εγείρουν μια πληθώρα διαφορετικών ερωτημάτων όσον αφορά τόσο τις επιλογές υλοποίησης όσο και τις απαιτήσεις του τελικού χρήστη μιας εφαρμογής. Η υλοποίηση γενεαλογικών συλλεκτών περιλαμβάνει πολύ περισσότερες σχεδιαστικές επιλογές από την επιλογή του μεγέθους του σωρού.

Η πρώτη απόφαση υλοποίησης συνήθως αφορά το αν ο σωρός θα οργανωθεί σε δύο ή περισσότερες γενεές. Η απάντηση εξαρτάται πρωτίστως από τις κατανομές της διάρκειας ζωής των εφαρμογών για τις οποίες ο συλλέκτης θα παρέχει υποστήριξη. Εάν ένα σημαντικό ποσοστό των αντικειμένων αναμένεται να επιβιώσουν από τη νεότερη γενεά αλλά να πεθάνουν σύντομα μετά την προαγωγή τους, τότε ίσως αξίζει η προσθήκη ενδιάμεσων γενεών. Τα περισσότερα συστήματα αυτόματης διαχείρισης μνήμης με γενεαλογική συλλογή απορριμμάτων ωστόσο από προεπιλογή περιλαμβάνουν δύο γενεές και μια αθάνατη γενεά. Η χρήση πολλαπλών γενεών προσπαθεί να λύσει το πρόβλημα της πρόωρης προαγωγής αντικειμένων. Υπάρχουν και άλλοι τρόποι επίλυσής του.

Αρχικά, ο ρυθμός προαγωγής εξαρτάται από το μέγεθος της νέας γενεάς: μια μεγαλύτερη νέα γενεά προσφέρει περισσότερο χρόνο σε ένα αντικείμενο για να πεθάνει. Μερικοί γενεαλογικοί συλλέκτες επιτρέπουν στο χρήστη να ρυθμίσει ένα σταθερό μέγεθος για τη νεότερη γενεά. Άλλοι πάλι επιτρέπουν στη νεότερη γενεά να αυξάνεται δυναμικά μέχρις ότου καταλάβει όλο

το σωρό (εκτός από ήδη δεσμευμένες περιοχές, όπως η παλαιά γενεά και πιθανόν μια εφεδρική περιοχή αντιγραφής). Οι πιο εξεζητημένοι συλλέκτες απορριμμάτων πάντως μπορούν να μεταβάλλουν το μέγεθος της νεότερης γενεάς προς ικανοποίηση συγκεκριμένων προδιαγραφών σχετικά με τους χρόνους παύσης, λαμβάνοντας αποφάσεις για την αλλαγή του μεγέθους με βάση την παρακολούθηση του προφίλ της συμπεριφοράς του συλλέκτη.

Δεύτερον, η προαγωγή μπορεί να περιορισθεί ελέγχοντας την ηλικία κατά την οποία τα αντικείμενα προάγονται. Η τεχνική της μαζικής προαγωγής μεταφέρει όλους τους επιζώντες μιας συλλογής της νεότερης γενεάς σε μια παλαιότερη γενεά. Αυτή η πολιτική προαγωγής έχει μάλιστα την απλούστερη υλοποίηση, καθώς τα σύνολα ανάμνησης για τη νεότερη γενεά μπορούν να παραμερισθούν μετά το πέρας της συλλογής. Εναλλακτικά, ένας συλλέκτης μπορεί να απαιτεί από ένα αντικείμενο να επιβιώσει από περισσότερους του ενός κύκλους συλλογής πριν το προαγάγει. Στην περίπτωση αυτή, είναι απαραίτητος ένας μηχανισμός καταγραφής της ηλικίας των αντικειμένων. Αυτό μπορεί να επιτευχθεί είτε με τη χρήση κάποιων bits από την επικεφαλίδα των αντικειμένων, με τη διαίρεση μιας γενεάς σε σε υποχώρους ο καθένας από τους οποίους φιλοξενεί αντικείμενα διαφορετικής ηλικίας, είτε με συνδυασμό των δύο. Οι συνηθισμένες διαμορφώσεις περιλαμβάνουν σχήματα οργανωμένα σε βήματα και σχήματα που περιλαμβάνουν μια περιοχή εδέμ και ημιχώρους επιζώντων.

Τέλος, είναι πολλές φορές πιθανή η αποφυγή της υποχρέωσης προαγωγής ορισμένων αντικειμένων. Πολλοί συλλέκτες απορριμμάτων δεσμεύουν μια αθάνατη περιοχή για αντικείμενα που θα επιβιώσουν μέχρι το τέλος του προγράμματος. Τα αντικείμενα αυτά συχνά μπορούν να αναγνωρισθούν είτε κατά την υλοποίηση του συλλέκτη είτε από το μεταγλωττιστή. Τέτοια αντικείμενα συνήθως περιλαμβάνουν τις δομές δεδομένων του συλλέκτη καθώς και αντικείμενα που αναπαριστούν τον υπό εκτέλεση κώδικα.

Οι ρυθμοί προαγωγής μπορούν επίσης να επηρεάσουν το κόστος του φράγματος εγγραφής καθώς επίσης και το μέγεθος των ενθυμούμενων συνόλων. Υψηλότεροι ρυθμοί προαγωγής προκαλούν τη δημιουργία περισσότερων διαγενεαλογικών δεικτών που πρέπει να καταγραφούν. Εάν αυτό επηρεάζει ή όχι την απόδοση των φραγμάτων εγγραφής εξαρτάται από την υλοποίηση.

Η συχνότητα κλήσης των φραγμάτων εγγραφής εξαρτάται επίσης από το αν οι διαφορετικές γενεές μπορούν να συλλεχθούν ανεξάρτητα. Η ανεξάρτητη συλλογή των γενεών απαιτεί την καταγραφή όλων των διαγενεαλογικών δεικτών. Η εγκατάλειψη αυτής της ευελιξίας έχει ως αποτέλεσμα την ταυτόχρονη συλλογή όλων των νεότερων γενεών οποτεδήποτε συλλέγεται η παλαιότερη γενεά και επιτρέπει την καταγραφή μόνο των δεικτών με προέλευση κάποια παλαιότερη γενεά και προορισμό κάποια νεότερη γενεά (οι οποίοι δείκτες αναμένεται να είναι σημαντικά λιγότεροι σε πλήθος). Σημειώνοντας το αντικείμενο και όχι το συγκεκριμένο πεδίο αυτού ως πιθανή προέλευση ενός διαγενεαλογικού δείκτη, το κόστος σε χώρο για την αποθήκευση των συνόλων ανάμνησης μπορεί να μειωθεί.

Οι διαφορετικοί μηχανισμοί που χρησιμοποιεί ο τροποποιητής για την καταγραφή των πιθανών προελεύσεων διαγενεαλογικών δεικτών επηρεάζουν το κόστος της συλλογής. Παρότι μηχανισμοί λιγότερο ακριβείς μπορεί να μειώσουν το κόστος του φράγματος εγγραφής, αυτοί μπορεί να αυξήσουν το φορτίο εργασίας του συλλέκτη. Η καταγραφή του πεδίου προέλευσης με χρήση σειριακών απομονωτών αποθήκευσης είναι ο πιο ακριβής μηχανισμός παρότι ένας απομονωτής ενδέχεται να περιλαμβάνει κάποιες καταγραφές εις διπλούν. Η καταγραφή του αντικειμένου προέλευσης από την άλλη πλευρά απαιτεί τη σάρωση του αντικειμένου για την εύρεση διαγενεαλογικών δεικτών.

Κεφάλαιο 8

Παράλληλη συλλογή απορριμμάτων

Μέχρι τώρα έχουμε υποθέσει πώς, ενώ ο τροποποιητής μπορεί να είναι πολυνηματικός, ο συλλέκτης είναι μονονηματικός. Η υπόθεση αυτή οδηγεί σε φτωχή εκμετάλλευση των πόρων ενός σύγχρονου πολυπύρηνου μηχανήματος. Στο παρόν κεφάλαιο εξετάζουμε πώς μπορεί να παραλληλοποιηθεί ο συλλέκτης, υποθέτοντας ακόμα πως κανένα από τα νήματα-τροποποιητές δεν εκτελείται ενώ πραγματοποιείται συλλογή απορριμμάτων και πως αυτή ολοκληρώνεται πριν αυτά συνεχίσουν την εκτέλεσή τους. Ένας προφανής τρόπος προκειμένου να μειωθεί ο χρόνος παύσης των νημάτων-τροποποιητών είναι οι πυρήνες να συνεργαστούν κατά τη συλλογή απορριμμάτων (και ενώ τα νήματα-τροποποιητές είναι σταματημένα). Η **παράλληλη συλλογή απορριμμάτων** είναι το αντικείμενο αυτού του κεφαλαίου. Πιο συγκεκριμένα θα εξετάσουμε πώς μπορούν να παραλληλοποιηθούν οι 4 βασικές διαδικασίες της συλλογής με εξιχνίαση: σήμανση, εκκαθάριση, αντιγραφή και συμπύκνωση.

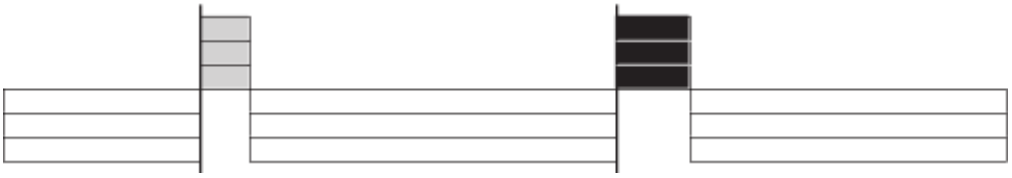
8.1 Υπάρχει επαρκής δουλειά προς παραλληλοποίηση;

Όπως και κατά την παραλληλοποίηση του αλγορίθμου οποιουδήποτε προβλήματος, η πρώτη απαίτηση αφορά στην εξασφάλιση του ότι το φορτίο εργασίας είναι αρκετά μεγάλο ώστε η παραλληλοποίηση να αξίζει τον κόπο. Αναπόφευκτα, η παράλληλη συλλογή απορριμμάτων απαιτεί κάποια μορφή συγχρονισμού των νημάτων-συλλεκτών με το συγχρονισμό αυτό να κοστίζει. Ο συγχρονισμός αυτός μπορεί να επιτευχθεί με τη χρήση είτε κλειδωμάτων είτε ατομικών πρωταρχικών εντολών υλικού όπως για παράδειγμα της CompareAndSwap και την προσεκτική σχεδίαση βοηθητικών δομών δεδομένων. Προκύπτει λοιπόν το ερώτημα αν το φορτίο εργασίας της συλλογής απορριμμάτων επαρκεί ώστε το κέρδος από την παραλληλοποίηση να υπερκεράζει το κόστος του συγχρονισμού.

Ας υποθέσουμε για παράδειγμα, πως ένας συλλέκτης σήμανσης και εκκαθάρισης θα πρέπει να σημάνει μία απλή συνδεδεμένη λίστα: σε κάθε βήμα, η στοίβα σήμανσης θα περιέχει μόνο ένα αντικείμενο, τον επόμενο κόμβο προς εξέταση. Στην περίπτωση αυτή, μόνο ένα από τα νήματα-συλλέκτες θα δουλεύει, ενώ τα υπόλοιπα θα είναι αδρανή, περιμένοντας για δουλειά. Ο Siebert [106] αποδεικνύει πως ο αριθμός φορών n που ένας επεξεργαστής μένει αδρανής κατά τη διάρκεια μιας παράλληλης σήμανσης σε ένα σύστημα με p επεξεργαστές φράσσεται από το μέγιστο βάθος των προσβάσιμων αντικειμένων o:

(α′) Συλλογή με παύση του κόσμου, μονονηματικός τροποποιητής, μονονηματικός συλλέκτης.

(β′) Συλλογή με παύση του κόσμου, πολυνηματικός τροποποιητής, μονονηματικός συλλέκτης.



(γ′) Συλλογή με παύση του κόσμου, πολυνηματικός τροποποιητής, πολυνηματικός συλλέκτης.

Σχήμα 8.1: Συλλογή απορριμμάτων με παύση του κόσμου: κάθε μπάρα αναπαριστά την εκτέλεση σε έναν επεξεργαστή. Οι χρωματισμένες περιοχές αναπαριστούν διαφορετικούς κύκλους συλλογής απορριμμάτων.

$$n \leq (p-1) \cdot \max_{o \in reachable} depth(o) \tag{8.1}$$

Η ανίσωση αυτή πάντως βασίζεται στην όχι ρεαλιστική υπόθεση πώς όλα τα βήματα σήμανσης διαρκούν το ίδιο, κάτι που δεν ισχύει καθώς οι χρόνοι διαφέρουν ανάλογα με το μέγεθος του αντικειμένου. Στις περισσότερες γλώσσες προγραμματισμού πάντως τα περισσότερα αντικείμενα είναι μικρά (περιέχουν λίγους δείκτες).

Μελέτες πάντως έχουν δείξει πώς οι τυπικές εφαρμογές περιλαμβάνουν μια ευρεία ποικιλία από πλούσιες σε δείκτες δομές δεδομένων. Για παράδειγμα, η εξιχνίαση μιας δενδροειδούς δομής δεδομένων θα παράγει περισσότερες μονάδες εργασίας από όσες θα καταναλώνει μέχρις ότου η εξερεύνηση φθάσει στα φύλλα.

8.2 Εξισορρόπηση φορτίου

Μια άλλη απαίτηση από την παράλληλη συλλογή απορριμμάτων είναι το φορτίο της συλλογής να κατανεμηθεί με τέτοιο τρόπο ώστε να ελαχιστοποιηθεί η ανάγκη για συντονισμό και όλοι οι επεξεργαστές να είναι απασχολημένοι ταυτόχρονα. Οι Endo κ.ά. [50] διαπιστώνουν πως χωρίς κάποιο μηχανισμό για **εξισορρόπηση φορτίου**, μια απλοϊκή παραλληλοποίηση μπορεί να οδηγήσει σε πολύ μικρή επιτάχυνση σε πολυεπεξεργαστικά συστήματα.

Δυστυχώς, ο στόχος της ομοιόμορφης κατανομής υπολογιστικού φορτίου και της ελαχιστοποίησης του απαιτούμενου συντονισμού αλληλοσυγκρούονται. Ο στατικός διαμοιρασμός του υπολογιστικού φόρτου δεν είναι πάντα ο καλύτερος. Για παράδειγμα, έστω ένας παράλληλος συλλέκτης σήμανσης και συμπύκνωσης σε σύστημα N επεξεργαστών. Αν ο σωρός διαιρεθεί σε N τμήματα, και ο κάθε επεξεργαστής αναλάβει την επιδιόρθωση των δεικτών στο δικό του τμήμα, τότε, αν ο αριθμός των αντικειμένων και των δεικτών που αυτά περιέχουν ποικίλλει πολύ από τμήμα σε τμήμα, κάποιοι επεξεργαστές θα δουλεύουν περισσότερο από άλλους.

Εξίσου σημαντική με την εξισορρόπηση του υπολογιστικού φορτίου ανάμεσα στους επεξεργαστές είναι και η εξισορρόπηση άλλων πόρων. Σε μία παράλληλη υλοποίηση του αλγόριθμου αντιγραφής του Baker [70], ο Halstead [71], [72] ανέθεσε σε κάθε επεξεργαστή το δικό του χώρο-προς και χώρο-από. Δυστυχώς, αυτός ο στατικός διαμοιρασμός συχνά οδηγούσε σε μία κατάσταση όπου ένας επεξεργαστής εξαντλούσε το δικό του χώρο-από και ενώ υπήρχε χώρος στους χώρους-από των υπολοίπων επεξεργαστών.

Ένας **δυναμικός διαμοιρασμός** είναι περισσότερο κατάλληλος προκειμένου η δουλειά να μοιρασθεί σχεδόν ομοιόμορφα. Ας θεωρήσουμε για παράδειγμα ένα συλλέκτη σήμανσης και συμπύκνωσης: αφού έχει ολοκληρωθεί η σήμανση των ζωντανών αντικειμένων, ο σωρός διαχωρίζεται σε τμήματα, τα οποία έχουν παρόμοιο πλήθος ζωντανών αντικειμένων και δεικτών. Στη συνέχεια, οι επεξεργαστές συμπυκνώνουν παράλληλα ο καθένας το δικό του τμήμα του σωρού. Αυτή είναι ακριβώς η προσέγγιση των Flood κ.ά. [53].

Συχνά η εξαρχής εκτίμηση εργασίας για την καλύτερη διαίρεσή της δεν είναι εφικτή. Σε αυτήν την περίπτωση η συνηθισμένη λύση είναι να **υπερ-διαμερίζεται** η συνολική εργασία ούτως ώστε να υπάρχουν περισσότερα τμήματα εργασίας από ότι επεξεργαστές ή νήματα και οι επεξεργαστές (ή τα νήματα) να τίθενται σε ανταγωνισμό για την απόκτησή τους. Βασικό πλεονέκτημα της τεχνικής αυτής αποτελεί το γεγονός πως είναι ανθεκτική σε αλλαγές του διαθέσιμου πλήθους επεξεργαστών στο συλλέκτη (π.χ. λόγω της επιβάρυνσης από άλλες διεργασίες που εκτελούνται παράλληλα), καθώς οι μικρότερες μονάδες εργασίες ανταλλάσσονται πιο εύκολα.

Απλοποιούμε τους αλγορίθμους που θα παρουσιάσουμε αργότερα εστιάζοντας την προσοχή μας στις τρεις βασικές υπο-εργασίες της απόκτησης, εκτέλεσης και δημιουργίας δουλειάς συλλογής. Θεωρούμε λοιπόν πως, στις περισσότερες περιπτώσεις, κάθε νήμα-συλλέκτης εκτελεί τον ακόλουθο βρόχο:

Αλγόριθμος 8.1 Παράλληλη συλλογή: αφηρημένη λειτουργία νήματος-συλλέκτη

```
1: procedure RUN()
2:     while not TERMINATED() do
3:         ACQUIREWORK()
4:         PERFORMWORK()
5:         GENERATEWORK()
```

8.3 Συγχρονισμός

Εκ πρώτης όψεως ίσως φαίνεται πως η βέλτιστη στρατηγική εξισορρόπησης του φορτίου εργασίας υπαγορεύει τη διαίρεση του στις ελάχιστες δυνατές υποεργασίες, όπως για παράδειγμα η σήμανση ή η αντιγραφή ενός μόνο αντικειμένου. Παρότι η λύση αυτή πετυχαίνει τέλειο διαμοιρασμό εργασίας ανάμεσα στους επεξεργαστές, το κόστος που επιβάλλει ο συντονισμός των τελευταίων την καθιστά απαγορευτική. Ο συγχρονισμός απαιτείται τόσο για την εξασφάλιση της ορθότητας όσο και για την αποφυγή της επανάληψης, ή τουλάχιστον ελαχιστοποίησης της εκτέλεσης ορισμένων μονάδων εργασίας περισσότερες από μια φορές.

Η επίτευξη συγχρονισμού μεταξύ των νημάτων-συλλεκτών κοστίζει σε χρόνο και σε χώρο. Οι μηχανισμοί που εξασφαλίζουν αποκλειστικότητα πρόσβασης χρησιμοποιούν μεταξύ άλλων κλειδώματα ή δομές δεδομένων χωρίς αναμονή. Οι καλώς σχεδιασμένοι αλγόριθμοι ελαχιστοποιούν τις περιπτώσεις όπου απαιτούνται ενέργειες συγχρονισμού, παρέχοντας για παράδειγμα

σε κάθε νήμα-συλλέκτη ιδιωτικές τοπικές δομές δεδομένων. Σε μερικές περιπτώσεις πάντως η μη αποκλειστική πρόσβαση δεν είναι απαραίτητη για την εξασφάλιση της ορθότητας και έτσι κάποιες ενέργειες συγχρονισμού μπορούν να παραλειφθούν.

Οι μοντέρνες υλοποιήσεις παράλληλης συλλογής απορριμμάτων συνήθως θέτουν να νήματα εργασίας σε ανταγωνισμό για την απόκτηση μονάδων εργασίας, τις οποίες στη συνέχεια εκτελούν χωρίς περαιτέρω συγχρονισμό. Οι μονάδες εργασίας οργανώνονται είτε ως τοπικές στοίβες σήμανσης, είτε ως ξεχωριστές περιοχές του σωρού προς σάρωση είτε ως καθολικές δεξαμενές. Οι βοηθητικές δομές δεδομένων για την οργάνωση των μονάδων εργασίας επιβάλλουν ένα κόστος σε χώρο για την αποθήκευση των μεταδεδομένων τους, το οποίο όμως τείνει να είναι μικρό.

8.4 Ταξινόμηση

Στο υπόλοιπο του κεφαλαίου παρουσιάζουμε ειδικές λύσεις στο πρόβλημα της παραλληλοποίησης της σήμανσης, της εκκαθάρισης, της αντιγραφής και της συμπύκνωσης. Σε όλες τις περιπτώσεις επικεντρώνουμε το ενδιαφέρον μας στο πώς οι αλγόριθμοι αποκτούν, εκτελούν και παράγουν εργασίες. Η σχεδίαση και η υλοποίηση αυτών των τριών δραστηριοτήτων καθορίζει τον απαιτούμενο συγχρονισμό, τη διακριτότητα των φορτίων εργασίας των νημάτων καθώς επίσης και πώς αυτά τα φορτία διαμοιράζονται ομοιόμορφα μεταξύ επεξεργαστών.

Οι αλγόριθμοι παράλληλης συλλογής απορριμμάτων μπορούν να κατηγοριοποιηθούν με βάση το αν στο επίκεντρο της σχεδίασης είναι ο επεξεργαστής ή η μνήμη. Οι αλγόριθμοι της πρώτης κατηγορίας τείνουν να επιτρέπουν στα νήματα να αποκτούν κβάντα εργασίας μεταβλητού μεγέθους, συνήθως κλέβοντας εργασία από άλλα νήματα. Λίγη έως και καθόλου σημασία δίνεται στην τοποθεσία των αντικειμένων στη μνήμη, παρότι η τοπικότητα επηρεάζει την επίδοση ακόμη και σε μονοπύρηνες αρχιτεκτονικές. Οι αλγόριθμοι της δεύτερης κατηγορίας από την άλλη πλευρά λαμβάνουν περισσότερο υπόψη τους τη θέση των αντικειμένων στη μνήμη. Συνήθως λειτουργούν σε συνεχόμενα μπλοκ μνήμης σωρού και αποκτούν/καταθέτουν εργασία από/προς διαμοιραζόμενες δεξαμενές.

Τέλος, εξετάζουμε τον τερματισμό της παράλληλης συλλογής απορριμμάτων. Συνήθως, απλώς η εξέταση του κατά πόσο μια δεξαμενή εργασίας είναι κενή δεν επαρκεί, καθώς ένα ενεργό νήμα-συλλέκτης μπορεί να ετοιμάζεται να καταθέσει καινούριες μονάδες εργασίας σε αυτή.

8.5 Παράλληλη Σήμανση

Η φάση της σήμανσης, σε αντίθεση με τις φάσεις της εκκαθάρισης και της συμπύκνωσης δεν είναι εγγενώς παραλληλοποιήσιμη.

8.5.1 Κλοπή εργασιών

Οι Endo κ.ά. [50], οι Flood κ.ά. [53] καθώς και ο Siebert [107] χρησιμοποιούν **κλοπή εργασίας** για την εξισορρόπηση του υπολογιστικού φορτίου. Οποτεδήποτε ένα νήμα-σημαντής ξεμένει από δουλειά σήμανσης, κλέβει δουλειά που ανήκει σε ένα άλλο νήμα-σημαντή. Σε μία παράλληλη υλοποίηση του συντηρητικού συλλέκτη των Boehm και Weiser [28], οι Endo κ.ά. παρέχουν σε κάθε νήμα-σημαντή την δική του τοπική στοίβα σήμανσης καθώς επίσης και μία

κλεπτόμενη ουρά εργασιών (αλγόριθμος 8.2). Περιοδικά, κάθε νήμα-σημαντής ελέγχει την κλεπτόμενη ουρά εργασιών του και αν αυτή είναι άδεια, μεταφέρει όλη την ιδιωτική στοίβα σήμανσης του (εκτός των τοπικών ριζών) στην ουρά. Ένα αδρανές νήμα-σημαντής αποκτά έργο σήμανσης ελέγχοντας αρχικά την δική του κλεπτόμενη ουρά και στη συνέχεια τις κλεπτόμενες ουρές των άλλων νημάτων. Όταν ένα νήμα-σημαντής βρει μια μη κενή κλεπτόμενη ουρά, κλέβει τις μισές καταχωρίσεις αυτής και τις μεταφέρει στην ιδιωτική του στοίβα σήμανσης. Καθώς πολλά νήματα προσπαθούν να κλέψουν δουλειά ταυτόχρονα, οι κλεπτόμενες ουρές εργασιών προστατεύονται με κλειδώματα.

Κάθε παράλληλος συλλέκτης οφείλει να είναι ιδιαίτερα προσεκτικός όσον αφορά την ενημέρωση bitmap σήμανσης και την επεξεργασία μεγάλων πινάκων. Τα bits μιας λέξης σε ένα bitmap σήμανσης πρέπει να ενημερώνονται ατομικά. Αντί να κλειδώνουν ολόκληρη τη λέξη όταν ελέγχουν ένα bit, οι Endo κ.ά. χρησιμοποιούν μια απλή φόρτωση για τον έλεγχο του bit και μόνον αν αυτό έχει την τιμή 0, επιχειρούν ατομικά να εγγράψουν την τιμή 1 σε αυτό, προσπαθώντας εκ νέου, αν η εγγραφή αποτύχει (αλγόριθμος 8.3). Αντίθετα, συλλέκτες σαν αυτόν των Flood κ.ά. [53], οι οποίοι αποθηκεύουν το bit σήμανσης στην επικεφαλίδα του αντικειμένου μπορούν να πραγματοποιούν σημάνσεις χωρίς τη χρήση ατομικών λειτουργιών.

Η επεξεργασία μεγάλων πινάκων αποτελεί πηγή πολλών προβλημάτων. Για παράδειγμα, οι Boehm και Weiser [28] επιχειρούν να αποτρέψουν την υπερχείλιση της στοίβας σήμανσης ωθώντας μεγάλα αντικείμενα σε τμήματα των 128 λέξεων. Παρόμοια, και προκειμένου να βελτιώσουν την εξισορρόπηση φορτίου, ο Endo κ.ά. διαιρούν ένα μεγάλο αντικείμενο σε κομμάτια των 512 λέξεων πριν το τελευταίο εισαχθεί σε κάποια στοίβα σήμανσης ή κλεπτόμενη ουρά.

Ο παράλληλος γενεαλογικός συλλέκτης των Flood κ.ά. [53] διαχειρίζεται τη νέα γενιά με αντιγραφή και την παλαιά με σήμανση και συμπύκνωση. Σε αυτήν την ενότητα εξετάζουμε μόνο την παράλληλη σήμανση (αλγόριθμος 8.4). Ενώ οι Endo κ.ά. χρησιμοποιούν μία στοίβα και μια ουρά ανά επεξεργαστή, οι Flood κ.ά. χρησιμοποιούν μία **διπλά τερματισμένη ουρά** ανά νήμα-σημαντή. Ο αλγόριθμος τους για κλοπή εργασίας δε χρησιμοποιεί κλειδώματα, επιτρέπει το διαμοιρασμό εργασίας σε επίπεδο αντικειμένων και βασίζεται στην ιδέα των Dimpsey κ.ά. [46].

Ένα νήμα-σημαντής αντιμετωπίζει το κάτω μέρος της διπλά συνδεδεμένης ουράς του ως στοίβα σήμανσης: η λειτουργία ώθησης δεν απαιτεί συγχρονισμό ενώ η λειτουργία εξώθησης απαιτεί συγχρονισμό μόνο όταν η διπλά συνδεδεμένη ουρά αποτελείται από ένα μόνο στοιχείο. Νήματα χωρίς εργασία κλέβουν ένα αντικείμενο από το πάνω μέρος των διπλά συνδεδεμένων ουρών των άλλων νημάτων με χρήση της συγχρονισμένης λειτουργίας REMOVE. Ένα βασικό πλεονέκτημα αυτής της σχεδίασης για κλοπή εργασιών είναι πως ο ακριβός μηχανισμός συγχρονισμού ενεργοποιείται μόνο όταν υπάρχει ανάγκη για εξισορρόπηση εργασιών.

Οι διπλά τερματισμένες ουρές έχουν σταθερό μέγεθος ώστε να αποφεύγεται η ανάγκη για εκχώρηση μνήμης στη διάρκεια μιας συλλογής. Καθώς η προσέγγιση αυτή διακινδυνεύει την εμφάνιση υπερχείλισης η Flood κ.ά. χρησιμοποιούν ένα καθολικό σύνολο υπερχείλισης το οποίο επιφέρει μία μικρή επιβάρυνση ανά κλάση. Η δομή κάθε κλάσης Java κρατάει μία λίστα όλων των αντικειμένων υπερχείλισης της κλάσης, τα οποία συνδέονται μεταξύ τους μέσω του πεδίου τύπου τους. Η υπερχείλιση χειρίζεται ως εξής: οποτεδήποτε η ώθηση ενός αντικειμένου στο κάτω μέρος της διπλά-τερματισμένης ουράς θα προκαλέσει υπερχείλιση, τα μισά αντικείμενα του κάτω μέρους μετακινούνται στα αντίστοιχα σύνολα υπερχείλισης των κλάσεων τους. Αντίστροφα, νήματα-σημαντές που βρίσκονται σε αναζήτηση εργασίας επιχειρούν να γεμίσουν τη μισή διπλά τερματισμένη ουρά τους εξετάζοντας πρώτα το σύνολο υπερχείλισης και έπειτα τις διπλά τερματισμένες ουρές των άλλων νημάτων-σημαντών.

Αλγόριθμος 8.2 Παράλληλη συλλογή: σήμανση με κλοπή εργασίας (Endo κ.ά.)

```
1: shared stealableWorkQueue[N]                                   ▷ one per thread
2: me ← myThreadId

3: procedure acquireWork()
4:     if not isEmpty(myMarkStack) then
5:         return
6:     lock(stealableWorkQueue[me])
7:     n ← size(stealableWorkQueue[me])/2      ▷ grab half of my stealable work queue
8:     transfer(stealableWorkQueue[me], n, myMarkStack)
9:     unlock(stealableWorkQueue[me])
10:    if isEmpty(myMarkStack) then
11:        for all j in Threads do
12:            if not locked(stealableWorkQueue[j]) then
13:                if lock(stealableWorkQueue[j]) then
14:                    n ← size(stealableWorkQueue[me])/2    ▷ grab half of his stealable
   work queue
15:                    transfer(stealableWorkQueue[j], n, myMarkStack)
16:                    unlock(stealableWorkQueue[j])
17:                    return

18: procedure performWork()
19:    while pop(myMarkStack, ref) do
20:        for all fld in Pointers(ref) do
21:            child ← ∗fld
22:            if child ≠ null and not isMarked(child) then
23:                setMarked(child)
24:                push(myMarkStack, child)

25: procedure generateWork()    ▷ transfer all my stack to my stealable work queue
26:    if isEmpty(stealableWorkQueue[me]) then
27:        n ← size(myMarkStack)
28:        lock(myStealableWorkQueue[me])
29:        transfer(myMarkStack, n, stealableWorkQueue[me])
30:        unlock(myStealableWorkQueue[me])
```

Αλγόριθμος 8.3 Παράλληλη συλλογή: σήμανση με χρήση bitmap (Endo κ.ά.)

```
1: procedure setMarked(ref)
2:     oldByte ← markByte(ref)
3:     bitPosition ← markBit(ref)
4:     loop
5:     loop
6:         if isMarked(oldByte, bitPosition) then
7:             return
8:         newByte ← mark(oldByte, bitPosition)
9:         if compareAndSet(&markByte(ref), oldByte, newByte) then
10:            return
```

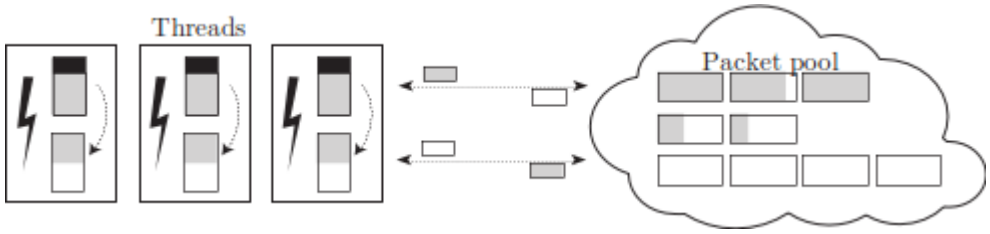
Αλγόριθμος 8.4 Παράλληλη συλλογή: σήμανση με κλοπή εργασίας (Flood κ.ά.)

```
1:  shared overflowSet
2:  shared deque[N]                                          ▷ one per thread
3:  me ← myThreadId

4:  procedure acquireWork()
5:      if not isEmpty(deque[me]) then
6:          return
7:      n ← size(overflowSet) /2
8:      if transfer(overflowSet, n, deque[me]) then
9:          return
10:     for all j in Threads do
11:         ref ← remove(deque[j])                           ▷ try to steal from j
12:         if ref ≠ null then
13:             push(deque[me], ref)
14:             return

15: procedure performWork()
16:     loop
17:         ref ← pop(deque[j])
18:         if ref = null then return
19:         for all fld in Pointers(ref) do
20:             child ← *fld
21:             if child ≠ null and not isMarked(child) then
22:                 setMarked(child)
23:                 if not push(deque[me], child) then
24:                     n ← size(overflowSet) /2
25:                     transfer(deque[me], n, overflowSet)

26: procedure generateWork()
27:     /* nop */
```

Ο Siebert [107] επίσης χρησιμοποιεί κλοπή εργασιών για μια παράλληλη και ταυτόχρονη υλοποίηση της εικονικής μηχανής Jamaica για τη γλώσσα Java. Τα αντικείμενα διασπώνται σε συνδεδεμένα μπλοκ για να φραγεί ο χρόνος κάθε βήματος σήμανσης και συνεπώς ο συλλέκτης δουλεύει με μπλοκ και όχι με αντικείμενα. Μια συνέπεια του παραπάνω γεγονότος είναι πως τα χρώματα συσχετίζονται με μπλοκ. Όπως θα δούμε στο κεφάλαιο 9, νήματα-τροποποιητές και νήματα-συλλέκτες που εκτελούνται ταυτόχρονα ενδέχεται να χρειαστεί να προσπελάσουν λίστες από γκρι μπλοκ. Για να αποφύγει την ανάγκη συγχρονισμού τέτοιων προσπελάσεων, η εικονική μηχανή Jamaica χρησιμοποιεί τοπικές λίστες γκρι μπλοκ ανά επεξεργαστή. Το κόστος αυτής της σχεδίασης είναι πώς το χρώμα ενός μπλοκ αναπαρίσταται από μία ολόκληρη λέξη και όχι ορισμένα bits. Ένα νήμα-σημαντής σημαίνει ένα μπλοκ με γκρι χρώμα χρησιμοποιώντας μια λειτουργία CompareAndSwap για να το συνδέσει μέσω του χρώματος του στην **τοπική** γκρι λίστα του επεξεργαστή στον οποίο εκτελείται. Για την εξισορρόπηση φορτίου, ένα νήμα-σημαντής χωρίς εργασία επιχειρεί να κλέψει όλη τη γκρι λίστα μπλοκ ενός άλλου νήματος-σημαντή. Για να αποτραπεί η επεξεργασία ενός μπλοκ από δύο ή περισσότερα νήματα-σημαντές ταυτόχρονα, ένα νήμα-σημαντής σημαίνεται ως **ανθρακί** κατά τη διάρκεια της επεξεργασίας του σε κάποιο στάδιο σήμανσης. Τα νήματα-κλέφτες επίσης κλέβουν επιχειρώντας να αλλάξουν το χρώμα της κεφαλής μιας γκρι λίστας ενός άλλου επεξεργαστή σε ανθρακί. Αυτός ο μηχανισμός είναι ιδανικός στην περίπτωση όπου το νήμα- θύμα δεν πραγματοποιεί εργασίες συλλογής εκτός ίσως από την πρόσθεση μπλοκ στη γκρι λίστα του ενώ εκτελεί φράγματα εγγραφής. Αυτό είναι ρεαλιστικό σενάριο για έναν ταυτόχρονο, πραγματικού χρόνου συλλέκτη.

8.5.2 Τερματισμός με κλοπή εργασιών

Τελικώς, ο συλλέκτης θα πρέπει να είναι σε θέση να καθορίσει το τέλος μιας φάσης, δηλαδή πότε όλα τα συνεργαζόμενα νήματα έχουν ολοκληρώσει τις εργασίες τους. Οι Endo κ.ά. [50] προσπάθησαν αρχικώς να ανιχνεύσουν τον τερματισμό με τη χρήση ενός απλού καθολικού μετρητή των άδειων στοιβών σήμανσης και των άδειων κλεπτόμενων ουρών σήμανσης. Ωστόσο ο ανταγωνισμός για την ατομική ενημέρωση του μετρητή σειριοποιούσε τον τερματισμό και μεγάλα συστήματα (με 32 ή και περισσότερους επεξεργαστές) δαπανούσαν πολύ χρόνο για την απόκτηση του κλειδώματος. Οι Endo κ.ά. αντιμετώπισαν το πρόβλημα παρέχοντας σε κάθε επεξεργαστή δύο τοπικές σημαίες που υποδείκνυαν αν η τοπική στοίβα σήμανσης και η τοπική κλεπτόμενη ουρά ήταν άδεια. Οι σημαίες αυτές μπορούν να τεθούν σε λογικό 0 ή 1 χωρίς κλειδώματα. Για να εντοπίσει τον τερματισμό ένας επεξεργαστής θέτει σε 0 μια καθολική σημαία **διακοπτόμενου εντοπισμού** και στη συνέχεια ελέγχει όλες τις σημαίες των υπολοίπων επεξεργαστών. Τέλος, ελέγχει εκ νέου τη σημαία διακοπτόμενου εντοπισμού διότι αυτή να έχει τεθεί σε λογικό 1 στο ενδιάμεσο από κάποιον άλλο επεξεργαστή ο οποίος συνεχίζει να δουλεύει. Αν αυτό δε συμβαίνει, τότε η φάση της σήμανσης έχει ολοκληρωθεί. Η τεχνική αυτή απαιτεί την τήρηση ενός αυστηρού πρωτοκόλλου κατά την κλοπή όλης τη δουλειάς ενός επεξεργαστή B από έναν επεξεργαστή A. Ο επεξεργαστής A πρέπει να θέσει σε 0 την τοπική σημαία που αντιστοιχεί στην τοπική στοίβα, στη συνέχεια να θέσει σε 1 την καθολική σημαία διακοπτόμενου εντοπισμού και τέλος να θέσει σε 1 την τοπική σημαία του επεξεργαστή B που αντιστοιχεί στην τοπική κλεπτόμενη ουρά του τελευταίου. Δυστυχώς όμως, όπως αποδεικνύουν οι Petrank και Colodner [91], το πρωτόκολλο είναι ελαττωματικό στην περίπτωση που περισσότερα του ενός νήματα προσπαθούν να εντοπίσουν τον τερματισμό της φάσης σήμανσης. Έστω ότι οι επεξεργαστές A και B ανιχνεύουν ταυτόχρονα τον τερματισμό της φάσης σήμανσης και πως ο επεξεργαστής C κλέβει όλη τη δουλειά από τον επεξεργαστή D. Η ακόλουθη αλληλουχία γεγονότων είναι πιθανή: ο A θέτει την καθολική



Σχήμα 8.2: Γκρι πακέτα. Κάθε νήμα ανταλλάσσει ένα άδειο πακέτο με ένα πακέτο αναφορών προς αντικείμενα προς εξιχνίαση. Η σήμανση γεμίζει ένα άδειο πακέτο με νέες αναφορές προς αντικείμενα προς εξιχνίαση: όταν αυτό γεμίσει, το νήμα το ανταλλάσσει με ένα καινούριο άδειο πακέτο από την καθολική δεξαμενή.

σημαία σε 0, ο επεξεργαστής C κλέβει τη δουλειά από τον επεξεργαστή D και τη θέτει σε 1, ο επεξεργαστής B θέτει ξανά σε 0 με αποτέλεσμα ο επεξεργαστής A να μην αντιληφθεί ποτέ την ενδιάμεση τιμή 1 της σημαίας.

Οι Petrank και Colodner [91] εξασφαλίζουν πως μόνο ένα νήμα προσπαθεί να εντοπίσει τον τερματισμό κάθε φορά εισάγοντας ένα κλείδωμα: μια συγχρονισμένη, καθολική λέξη **ταυτότητας ανιχνευτή**. Πριν επιχειρήσει να ανιχνεύσει τον τερματισμό, ένα νήμα υποχρεώνεται να ελέγξει πως η ταυτότητα ανιχνευτή έχει την τιμή -1, (που σημαίνει πώς κανένα άλλο νήμα δεν προσπαθεί να ανιχνεύσει τον τερματισμό παράλληλα) και αν ναι, την ενημερώνει ατομικά με τη δική του ταυτότητα, διαφορετικά περιμένει.

Οι Flood κ.ά. [53] εντοπίζουν τον τερματισμό με τη χρήση μιας λέξης κατάστασης με ένα bit για κάθε συμμετέχον νήμα το οποίο πρέπει να ενημερώνεται ατομικά. Αρχικά, όλα τα νήματα βρίσκονται σε ενεργή κατάσταση (τιμή 1). Όταν ένα νήμα δεν έχει δουλειά και επίσης δεν έχει καταφέρει να κλέψει, θέτει το bit κατάστασής του σε 0 και μπαίνει σε ένα βρόχο ελέγχου τού κατά πόσο όλα τα bits της λέξης κατάστασης είναι 0. Αν ναι, όλα τα νήματα έχουν εκτελέσει τις εργασίες τους και η φάση σήμανσης έχει ολοκληρωθεί. Αν πάλι όχι, το νήμα ψάχνει για δουλειά στις διπλά-τερματισμένες ουρές των άλλων νημάτων. Αν καταφέρει να βρει δουλειά, θέτει το bit κατάστασής του σε 1 και προσπαθεί να την κλέψει. Αν δεν τα καταφέρει, θέτει εκ νέου το bit κατάστασής του σε 0 και ξαναμπαίνει στον ίδιο βρόχο. Η τεχνική αυτή δεν κλιμακώνει όταν ο αριθμός των νημάτων είναι μεγαλύτερος από το μέγεθος της λέξης κατάστασης σε bits.

8.5.3 Γκρι πακέτα

Οι Ossia κ.ά. [85], καθώς και οι Barabash κ.ά. [13] παρατηρούν πώς η χρήση στοιβών σήμανσης και κλοπής εργασίας είναι βέλτιστη όταν ο αριθμός των νημάτων που συμμετέχουν στη συλλογή είναι γνωστός εξαρχής. Αυτό δε συμβαίνει εάν κάθε νήμα-τροποποιητής συμμετέχει πραγματοποιώντας μια μικρή εργασία σε κάθε εκχώρηση. Επίσης παρατηρούν πως είναι πιθανόν δύσκολο για ένα νήμα-σημαντή να επιλέξει την καλύτερη ουρά από την οποία θα κλέψει και ταυτόχρονα να εντοπίσει τον τερματισμό της σήμανσης. Αντ' αυτού, εξισορροπούν το υπολογιστικό φορτίο βάζοντας κάθε νήμα-σημαντή να ανταγωνίζεται για την απόκτηση **πακέτων** αποτελούμενων από εργασίες σήμανσης. Το σύστημα τους χρησιμοποιεί ένα σταθερό αριθμό από 1000 πακέτα των 512 εγγραφών.

Κάθε νήμα-σημαντής χρησιμοποιεί δύο πακέτα: επεξεργάζεται καταχωρίσεις από το πακέτο εισόδου και προσθέτει εργασίες σήμανσης στο πακέτο εξόδου. Ο όρος **γκρι πακέτα** οφείλεται στο γεγονός ότι τόσο το πακέτο εισόδου όσο και το πακέτο εξόδου περιλαμβάνουν γκρι

εγγραφές (αναφορές προς γκρι αντικείμενα) σύμφωνα με την τριχρωματική αφαίρεση. Κάθε νήμα-σημαντής ανταγωνίζεται με στόχο να αποκτήσει ένα πακέτο από μια καθολική δεξαμενή. Αφού επεξεργασθεί όλες τις καταχωρίσεις ενός πακέτου, το επιστρέφει στη δεξαμενή. Όταν το πακέτο εξόδου του είναι πλήρες, το επιστρέφει στη δεξαμενή και παίρνει πίσω ένα φρέσκο πακέτο. Όπως φαίνεται και στο σχήμα 8.2, η κεντρική δεξαμενή αποτελείται από τρεις συνδεδεμένες λίστες εκ των οποίων η πρώτη αποτελείται από τελείως άδεια πακέτα, η δεύτερη από πακέτα σχεδόν μισογεμάτα και η τρίτη από πακέτα σχεδόν γεμάτα. Τα νήματα προτιμούν την τρίτη για την εισαγωγή ενός πακέτου εισόδου (διαδικασία getInPacket, αλγόριθμος 8.5) και την πρώτη για την εξαγωγή ενός πακέτου εξόδου (διαδικασία getOutPacket, αλγόριθμος 8.5).

Η τεχνική της παραλληλοποίησης της σήμανσης με γκρι πακέτα έχει διάφορα πλεονεκτήματα. Ξεχωρίζοντας την είσοδο από την έξοδο, οι Ossia κ.ά. αποφεύγουν την εναλλαγή των ρόλων για τα πακέτα ενός νήματος-σημαντή: το φορτίο εργασίας κατανέμεται ομοιόμορφα στους επεξεργαστές καθώς ένας επεξεργαστής έχει την τάση να μην καταναλώνει την έξοδό του. Καθώς ένα γκρι πακέτο περιλαμβάνει μια ουρά από (αναφορές σε) αντικείμενα που θα επεξεργασθούν με τη σειρά, προσφέρεται η δυνατότητα προφόρτωσης των επόμενων αντικειμένων προς σήμανση.

Η τεχνική απαιτεί συγχρονισμό μόνο κατά τη μεταφορά των πακέτων από και προς την καθολική δεξαμενή. Αυτό μειώνει τον αριθμό των εντολών μνήμης **fence** που πρέπει να εισαχθούν από το μεταγλωττιστή σε αρχιτεκτονικές με χαλαρό μοντέλο συνέπειας μνήμης. Ένα νήμα χρειάζεται να εκτελέσει μία εντολή **fence** μόνο όταν αποκτά πακέτα από την καθολική δεξαμενή ή επιστρέφει πακέτα προς αυτή, και όχι μετά τη σήμανση και ώθηση ενός αντικειμένου σε ένα πακέτο. Ο Ossia κ.ά. χρησιμοποιούν ένα διάνυσμα από **bits σήμανσης** όταν σαρώνουν συντηρητικά τις στοίβες των νημάτων-τροποποιητών προκειμένου να προσδιορίσουν αν μία υποτιθέμενη αναφορά πραγματικά δείχνει προς κάποιο εκχωρηθέν αντικείμενο. Τα bits σήμανσης χρησιμοποιούνται επίσης για το συγχρονισμό μεταξύ των νημάτων-τροποποιητών και των νημάτων- συλλεκτών. Οι εκχωρητές μνήμης χρησιμοποιούν τοπικούς απομονωτές εκχώρησης. Κατά την υπερχείλιση ενός τοπικού απομονωτή εκχώρησης, ο εκχωρητής εκτελεί μία εντολή **fence** και στη συνέχεια θέτει σε λογικό 1 τα bits όλων των αντικείμενων που έχουν εκχωρηθεί στον τοπικό απομονωτή, εξασφαλίζοντας με τον τρόπο αυτό πώς οι αποθηκεύσεις που αφορούν την εκχώρηση και αρχικοποίηση νέων αντικειμένων δεν προηγούνται των αποθηκεύσεων που αφορούν την ενημέρωση των bits σήμανσης τους (αλγόριθμος 8.6). Δύο ακόμη εντολές **fence** είναι απαραίτητες. Πρώτον, όταν ένα νήμα-εξιχνιαστής αποκτά ένα καινούριο πακέτο εισόδου, ελέγχει τα bits εκχώρησης όλων των αντικειμένων του πακέτου και καταγράφει σε μία ιδιωτική δομή δεδομένων για κάθε τέτοιο αντικείμενο αν είναι ασφαλής η εξιχνίαση με ρίζα αυτό (αν το αντίστοιχο bit είναι σε λογικό 1). Στη συνέχεια το νήμα εκτελεί μία εντολή **fence** πριν προχωρήσει στην εξιχνίαση με ρίζες τα ασφαλή αντικείμενα. Η εξιχνίαση με ρίζες τα μη ασφαλή αντικείμενα αναβάλλεται: αυτά προστίθενται σε ένα πακέτο αναβολής, το οποίο κάποια στιγμή αργότερα θα μεταφερθεί σε μία καθολική δεξαμενή πακέτων αναβολής. Δεύτερον, ένα νήμα-εξιχνιαστής εκτελεί μια εντολή **fence** όταν επιστρέφει το πακέτο εξόδου του στην καθολική δεξαμενή (για να αποτρέψει την αναδιάταξη αποθηκεύσεων στο πακέτο με την προσθήκη του πακέτου στην καθολική δεξαμενή). Κατά την απόκτηση ενός πακέτου εισόδου από την καθολική δεξαμενή υπάρχει μια εξάρτηση δεδομένων μεταξύ της φόρτωσης του δείκτη προς το πακέτο και την πρόσβαση στα περιεχόμενα και οι σύγχρονες αρχιτεκτονικές δεν αναδιατάσσουν τις παραπάνω ενέργειες. Συνεπώς η εκτέλεση μιας εντολής **fence** δεν είναι απαραίτητη σε αυτήν την περίπτωση.

Η τεχνική των γκρι πακέτων καθιστά εξαιρετικά απλή την παρακολούθηση της κατάστασης. Κάθε καθολική δεξαμενή συσχετίζεται με έναν καθολικό μετρητή του πλήθους των πακέτων

της. Ο μετρητής ενημερώνεται ατομικά κατά την προσθήκη ή αφαίρεση ενός πακέτου από τη δεξαμενή. Η μέτρηση του πλήθους των πακέτων είναι προσεγγιστική καθώς η μέτρηση μπορεί να διαβασθεί αμέσως μετά την προσθήκη ενός πακέτου αλλά πριν την αύξηση του μετρητή. Ωστόσο η συνθήκη τερματισμού απλώς ορίζει ο μετρητής της καθολικής δεξαμενής άδειων πακέτων να ισούται με το πλήθος των διαθέσιμων πακέτων. Η απόκτηση/επιστροφή ενός πακέτου και η ενημέρωση του μετρητή της αντίστοιχης καθολικής δεξαμενής δε χρειάζεται να είναι μία ενιαία αδιαίρετη λειτουργία. Για να εξασφαλισθεί πώς ο μετρητής κενών πακέτων δε θα μηδενισθεί προσωρινά, κάθε νήμα-σημαντής πρέπει να αποκτήσει ένα καινούριο πακέτο εισόδου πριν επιστρέψει το πακέτο εξόδου του στην καθολική δεξαμενή.

Αλγόριθμος 8.5 Παράλληλη συλλογή: διαχείριση γκρι πακέτων (Ossia κ.ά.)

```
1: shared fullPool                                   ▷ global pool of full packets
2: shared halfFullPool                               ▷ global pool of half full packets
3: shared emptyPool                                  ▷ global pool of empty packets

4: function getInPacket()
5:     atomic
6:     inPacket ← remove(fullPool)
7:     if isEmpty(inPacket) then
8:         atomic
9:         inPacket ← remove(halfFullPool)
10:    if isEmpty(inPacket) then
11:        inPacket, outPacket ← outPacket, inPacket
12:    return not isEmpty(inPacket)

13: procedure testAndMarkSafe(packet)
14:    for all ref in packet do
15:        safe(ref) ← allocBit(ref) = true          ▷ private data structure

16: procedure getOutPacket()
17:    if isFull(outPacket) then
18:        generateWork()
19:    if outPacket = null then
20:        atomic
21:        outPacket ← remove(emptyPool)
22:    if outPacket = null then
23:        atomic
24:        remove(halfFullPool)
25:    if outPacket = null then
26:        if not isFull(inPacket) then
27:            inPacket, outPacket ← outPacket, inPacket
28:            return

29: procedure addOutPacket(ref)
30:    getOutPacket()
31:    if outPacket = null or isFull(outPacket) then
32:        dirtyCard(ref)
33:    else
34:        add(outPacket, ref)
```

Αλγόριθμος 8.6 Παράλληλη συλλογή: εκχώρηση με χρήση γκρι πακέτων (Ossia κ.ά.)

```
1:  function SEQUENTIALALLOCATE(n)
2:      result ← free
3:      newFree ← result + n
4:      if newFree ≤ labLimit then
5:          free ← newFree
6:          return result

7:      /* local allocation buffer overflow */
8:      fence
9:      for all obj in lab do
10:         ALLOCBIT(obj) ← true
11:     lab, labLimit ← NEWLAB()
12:     if lab = null then
13:         return null                          ▷ signal "Memory exhausted"
14:     SEQUENTIALALLOCATE(n)
```

Αλγόριθμος 8.7 Παράλληλη συλλογή: σήμανση με χρήση γκρι πακέτων (Ossia κ.ά.)

```
1:  shared fullPool                              ▷ global pool of full packets

2:  procedure ACQUIREWORK()
3:      if ISEMPTY(inPacket) then
4:          if GETINPACKET() then
5:              TESTANDMARKSAFE(inPacket)
6:              fence

7:  procedure PERFORMWORK()
8:      for all ref in inPacket do
9:          if SAFE(ref) then
10:             for all fld in Pointers(ref) do
11:                 child ← *fld
12:                 if child ≠ null and not ISMARKED(child) then
13:                     SETMARKED(child)
14:                     ADDOUTPACKET(child)
15:         else
16:             ADDDEFERREDPACKET(ref)           ▷ defer tracing of unsafe objects

17: procedure GENERATEWORK()
18:     fence
19:     ADD(fullPool, outPacket)
20:     outPacket ← null
```

8.6 Παράλληλη αντιγραφή

Η παραλληλοποίηση των αλγορίθμων συλλογής με αντιγραφή αντιμετωπίζει λίγο πολύ τα ίδια ζητήματα με την παραλληλοποίηση των αλγορίθμων συλλογής με σήμανση. Ωστόσο, παρότι η σήμανση ενός αντικειμένου δύο φορές είναι αβλαβής, αυτό πρέπει να αντιγραφεί μόνο μία φορά.

8.6.1 Διαμοιρασμός εργασίας ανάμεσα στους επεξεργαστές

Οι Blelloch και Cheng [23], [35] παραλληλοποιούν την αντιγραφή στο πλαίσιο της **επαναληπτικής συλλογής**.

Οι επαναληπτικοί συλλέκτες είναι αυξητικοί ή ταυτόχρονοι συλλέκτες που αντιγράφουν αντικείμενα την ώρα εκτέλεσης του τροποποιητή, λαμβάνοντας ειδική φροντίδα για την επιδιόρθωση πεδίων που εγγράφησαν από τον τροποποιητή κατά τη διάρκεια ενός κύκλου συλλογής.

Κάθε νήμα-αντιγραφέας έχει τη δική του στοίβα εργασίας. Οι Blelloch και Cheng ισχυρίζονται πώς οι στοίβες προσφέρουν ευκολότερο συγχρονισμό μεταξύ των νημάτων-αντιγραφέων όπως επίσης και μικρότερο ποσοστό κατακερματισμού από ότι οι ουρές Cheney. Το φορτίο εργασίας εξισορροπείται βάζοντας τα νήματα-αντιγραφείς να μεταφέρουν περιοδικά μονάδες εργασίας μεταξύ των τοπικών στοιβών και μιας καθολικής στοίβας (αλγόριθμος 8.8). Μια απλή μοιραζόμενη στοίβα απαιτεί το συγχρονισμό μεταξύ των νημάτων που εκτελούν λειτουργίες ώθησης και εξώθησης καταχωρίσεων. Δυστυχώς, δεν υπάρχει τρόπος ατομικής αύξησης ή μείωσης του δείκτη στοίβας για την εισαγωγή και εξαγωγή ενός στοιχείου με τη χρήση πρωταρχικών εντολών υλικού όπως η FetchAndAdd και η χρήση ενός κλειδώματος θα σειριοποιούσε την πρόσβαση στη στοίβα. Η εντολή FetchAndAdd μπορεί ωστόσο να χρησιμοποιηθεί ώστε να επιτραπεί σε πολλαπλά νήματα είτε να εισάγουν στοιχεία στη στοίβα είτε να εξάγουν στοιχεία από τη στοίβα, καθώς οι λειτουργίες αυτές έχουν ως τελικό αποτέλεσμα είτε την αύξηση είτε τη μείωση του δείκτη στοίβας κατά περισσότερες από μία θέσεις. Αφού ένα νήμα-αντιγραφέας έχει πετύχει να μετακινήσει το δείκτη στοίβας, πιθανώς κατά αρκετές θέσεις, μπορεί να διαβάσει από ή να γράψει στις θέσεις αυτές χωρίς τον κίνδυνο εμφάνισης καταστάσεων συναγωνισμού.

Οι Cheng και Blelloch επιβάλλουν αυτού του είδους την πρόσβαση στη μοιραζόμενη στοίβα χρησιμοποιώντας ένα μηχανισμό που ονομάζουν **δωμάτια**. Υπάρχουν δύο δωμάτια: ένα ώθησης και ένα εξώθησης και ανά πάσα χρονική στιγμή ένα εκ των δύο πρέπει να είναι άδειο. Όπως φαίνεται και στον αλγόριθμο 8.9, σε κάθε επανάληψη του βρόχου συλλογής, ένα νήμα-αντιγραφέας εισέρχεται αρχικά στο δωμάτιο εξώθησης και εκτελεί ένα προκαθορισμένο αριθμό μονάδων εργασίας. Αποκτά αντικείμενα προς σάρωση είτε από την τοπική του στοίβα είτε από την καθολική στοίβα με χρήση της εντολής FetchAndAdd. Οι παραγόμενες μονάδες εργασίας εισάγονται στην τοπική στοίβα. Στη συνέχεια το νήμα-αντιγραφέας εγκαταλείπει το δωμάτιο εξώθησης και περιμένει μέχρις ότου και τα υπόλοιπα νήματα το έχουν εγκαταλείψει πριν προσπαθήσει να εισέλθει στο δωμάτιο ώθησης. Το πρώτο νήμα-αντιγραφέας που καταφέρνει να εισέλθει κλείνει την πόρτα (μεταβλητή *gate*) ούτως ώστε να απαγορευθεί η είσοδος στα υπόλοιπα νήματα-αντιγραφείς. Ενώ βρίσκεται μέσα στο δωμάτιο ώθησης, το νήμα-αντιγραφέας αδειάζει πλήρως την τοπική του στοίβα, μεταφέροντας όλα τα στοιχεία της στην καθολική στοίβα, αφού πρώτα δεσμεύσει τον απαραίτητο χώρο με την εντολή FetchAndAdd. Το τελευταίο νήμα-αντιγραφέας που αποχωρεί από το δωμάτιο ώθησης κλείνει την πύλη.

Το πρόβλημα του μηχανισμού είναι πως ένας επεξεργαστής που περιμένει για να εισέλθει στο

δωμάτιο ώθησης υποχρεούται να περιμένει μέχρις ότου όλοι οι επεξεργαστές που βρίσκονται ήδη μέσα τελειώσουν τη σήμανση των αντικειμένων τους με γκρι χρώμα. Ο χρόνος σήμανσης των αντικειμένων με γκρι χρώμα είναι συγκρίσιμος με το χρόνο απόκτησης ή κατάθεσης νέων μονάδων εργασίας και ένας επεξεργαστής που προσπαθεί να εισέλθει στη φάση ώθησης πρέπει να περιμένει μέχρις ότου όλοι οι υπόλοιποι επεξεργαστές που βρίσκονται στη φάση εξώθησης τελειώσουν τη σήμανση των αντικειμένων τους με γκρι χρώμα. Μεγάλες διακυμάνσεις στο χρόνο που χρειάζονται οι επεξεργαστές για τη σήμανση των αντικειμένων τους με γκρι χρώμα καθιστούν αυτό το χρόνο αδράνειας σημαντικό. Μια πιο χαλαρή αφαίρεση θα επέτρεπε στους επεξεργαστές να εγκαταλείψουν το δωμάτιο εξώθησης χωρίς να εισέλθουν στο δωμάτιο ώθησης. Καθώς η σήμανση των αντικειμένων με γκρι χρώμα δε σχετίζεται με τη μοιραζόμενη στοίβα, αυτή η εργασία μπορεί να πραγματοποιηθεί εκτός των δωματίων. Αυτό αυξάνει σημαντικά την πιθανότητα το δωμάτιο εξώθησης να είναι άδειο και συνεπώς ένα νήμα-αντιγραφέας να μπορέσει να εισέλθει στο δωμάτιο ώθησης.

Ο αρχικός μηχανισμός δωματίων των Cheng και Blelloch επιτρέπει απλή ανίχνευση τερματισμού. Η τοπική στοίβα κάθε νήματος-αντιγραφέα είναι κενή όταν αυτό εγκαταλείπει το δωμάτιο ώθησης και επομένως μένει στο τελευταίο νήμα-αντιγραφέα που αποχωρεί να ελέγξει αν και η καθολική μοιραζόμενη στοίβα είναι επίσης άδεια. Ωστόσο, ο χαλαρός ορισμός του μηχανισμού των δωματίων επιτρέπει στα νήματα να εργάζονται και εκτός δωματίων. Η υιοθέτηση αυτού συνεπάγεται πως η μοιραζόμενη στοίβα πρέπει να διατηρεί μία καθολική μεταβλητή που μετράει πόσα νήματα έχουν δανεισθεί αντικείμενα από αυτή. Το τελευταίο νήμα-αντιγραφέας που αποχωρεί από το δωμάτιο ώθησης ελέγχει ταυτόχρονα αν η μοιραζόμενη στοίβα είναι κενή και ο καθολικός αυτός μετρητής μηδενικός.

8.6.2 Αντιγράφοντας αντικείμενα παράλληλα

Για να εξασφαλισθεί πως μόνο ένα νήμα-αντιγραφέας αντιγράφει ένα αντικείμενο, τα νήματα πρέπει να συναγωνισθούν ώστε να αντιγράψουν ένα αντικείμενο και να εγκαταστήσουν τη διεύθυνση προώθησης στην επικεφαλίδα του παλαιού του αντιγράφου. Ο τρόπος αντιγραφής ενός αντικειμένου από τα νήματα εξαρτάται από το εάν αυτά μοιράζονται μια μοναδική περιοχή εκχώρησης μνήμης. Στην περίπτωση αυτή, τα νήματα αποφεύγουν ορισμένη σπατάλη πληρώνοντας όμως το κόστος της χρήσης μιας ατομικής λειτουργίας για εκχώρηση. Σε αυτήν την περίπτωση, οι Blelloch και Cheng [23] θέτουν να νήματα σε ανταγωνισμό για την εγγραφή μιας ειδικής τιμής 'busy' στην λέξη της επικεφαλίδας του αντικειμένου όπου θα αποθηκευθεί η διεύθυνση προώθησης αυτού. Το νήμα-νικητής αντιγράφει το αντικείμενο πριν αποθηκεύσει τη διεύθυνση προώθησης, ενώ τα νήματα-ηττημένοι πρέπει να σπινάρουν μέχρις ότου παρατηρήσουν μία έγκυρη διεύθυνση. Εναλλακτικά, αν κάθε νήμα-αντιγραφέας γνωρίζει εξαρχής σε ποια θέση θα αντιγράψει ένα αντικείμενο (για παράδειγμα επειδή θα το αντιγράψει σε έναν τοπικό απομονωτή εκχώρησης), τα νήματα ανταγωνίζονται για να εγγράψουν ατομικά τη διεύθυνση προώθησης προτού αντιγράψουν το αντικείμενο.

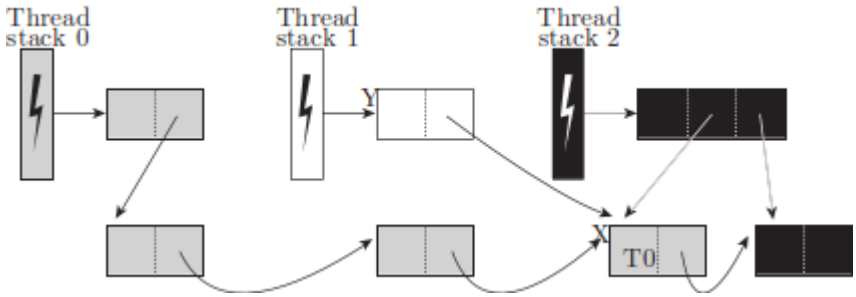
Ο συλλέκτης των Flood κ.ά. [53] στον οποίο αναφερθήκαμε προηγουμένως είναι γενεαλογικός. Η παλαιά γενεά διαχειρίζεται από συλλογή με σήμανση και συμπύκνωση και η νέα γενεά από συλλογή με αντιγραφή. Είδαμε παραπάνω τον τρόπο με τον οποίο παραλληλοποιούν τη φάση της σήμανσης. Εξετάζουμε τώρα πώς παραλληλοποιούν την αντιγραφή. Διπλά-τερματισμένες κλεπτόμενες ουρές χρησιμοποιούνται και εδώ για τη διαχείριση των αντικειμένων που πρόκειται να σαρωθούν. Ωστόσο, η παράλληλη αντιγραφή αντιμετωπίζει δύο προβλήματα άγνωστα στην παράλληλη σήμανση: πρώτον, είναι επιθυμητή η ελαχιστοποίηση του ανταγωνισμού για την εκχώρηση μνήμης όπου θα αντιγραφεί ένα αντικείμενο

Αλγόριθμος 8.8 Παράλληλη συλλογή: αντιγραφή (Cheng & Blelloch)

```
1: shared sharedStack                                   ▷ the shared stack of work
2: myCopyStack[k]                                       ▷ local stack has k slots max
3: sp ← 0

4: procedure RUN()
5:     while not TERMINATED() do
6:         ENTERROOM()                                  ▷ enter pop room
7:         for i ← 1 to k do
8:             if ISLOCALSTACKEMPTY() then
9:                 ACQUIREWORK()
10:                if ISLOCALSTACKEMPTY() then
11:                    break
12:            PERFORMWORK()
13:        GENERATEWORK()
14:        if EXITROOM() then                           ▷ leave push room
15:            TERMINATE()

16: procedure ACQUIREWORK()                             ▷ move work from shared stack
17:    SHAREDPOP()

18: procedure PERFORMWORK()
19:    ref ← LOCALPOP()
20:    SCAN(ref)                                        ▷ see algorithm 4.2

21: procedure GENERATEWORK()                            ▷ move work to shared stack
22:    SHAREDPUSH()

23: function ISLOCALSTACKEMPTY()
24:    return sp = 0

25: procedure LOCALPUSH(ref)
26:    myCopyStack[sp++] ← ref

27: function LOCALPOP() return myCopyStack[--sp]

28: procedure SHAREDPOP()                               ▷ move work from shared stack
29:    cursor ← FETCHANDADD(&sharedStack, 1)            ▷ try to grab from shared stack
30:    if cursor > stackLimit then                      ▷ shared stack empty
31:        FETCHANDADD(&sharedStack, −1)                ▷ readjust stack
32:    else
33:        myCopyStack[sp++] ← cursor[0]                ▷ move work to local stack

34: procedure SHAREDPUSH()                              ▷ move work to shared stack
35:    cursor ← FETCHANDADD(&sharedStack, −sp) −sp
36:    for i ← 0 to sp − 1 do
37:        cursor[i] ← myCopyStack[i]
38:    sp ← 0
```

Αλγόριθμος 8.9 Παράλληλη συλλογή: συγχρονισμός λειτουργιών ώθησης/εξώθησης με δωμάτια (Cheng & Blelloch)

```
1:  shared gate ← OPEN
2:  shared popClients                    ▷ number of clients currently in the pop room
3:  shared pushClients                   ▷ number of clients currently in the push room

4:  procedure enterRoom()
5:      while gate ≠ OPEN do                                   ▷ do nothing:wait
6:      FetchAndAdd(&popClients, 1)                            ▷ try to start popping
7:      while gate ≠ OPEN do
8:          FetchAndAdd(&popClients, −1)                       ▷ back out since did not succeed
9:          while gate ≠ OPEN do                               ▷ do nothing:wait
10:         FetchAndAdd(&popClients, 1)                        ▷ try again

11: procedure transitionRooms()
12:     gate ← CLOSED
13:     FetchAndAdd(&pushClients, 1)                           ▷ move from popping to pushing
14:     FetchAndAdd(&popClients, −1)
15:     while popClients > 0 do          ▷ can't start pushing until none other popping

16: procedure exitRoom()
17:     pushers ← FetchAndAdd(&pushClients, −1) −1             ▷ stop pushing
18:     if pushers = 0 then                  ▷ I was last in room: check termination
19:         if isEmpty(sharedStack) then
20:             gate ← OPEN
21:             return true
22:         else
23:             gate ← OPEN
24:             return false
```



Σχήμα 8.3: Εξιχνίαση κυρίαρχου νήματος. Τα νήματα 0 έως 2 με χρώμα μαύρο γκρι και άσπρο αντίστοιχα, έχουν εξιχνιάσει ένα γράφο αντικειμένων. Το χρώμα κάθε αντικειμένου υποδεικνύει τον επεξεργαστή στη μνήμη του οποίου θα αντιγραφεί. Το πρώτο πεδίο κάθε αντικειμένου είναι η επικεφαλίδα του. Το νήμα $T0$ είναι αυτό που τελευταίο κλείδωσε το αντικείμενο X.

και δεύτερον ένα ζωντανό αντικείμενο πρέπει να αντιγραφεί μόνο μία φορά. Ο ανταγωνισμός για την εκχώρηση μνήμης ελαχιστοποιείται με τη χρήση τοπικών απομονωτών εκχώρησης για κάθε νήμα-αντιγραφέα, τόσο για την αντιγραφή στους χώρους επιζώντων στη νέα γενεά όσο και για την προαγωγή στην παλαιά γενεά. Για να αντιγράψει ένα αντικείμενο, ένα νήμα-αντιγραφέας πραγματοποιεί μια υποθετική εκχώρηση στον τοπικό του απομονωτή εκχώρησης και στη συνέχεια επιχειρεί μια εντολή CompareAndSwap στον δείκτη προώθησης. Αν η τελευταία πετύχει, το νήμα-αντιγραφέας αντιγράφει το αντικείμενο. Αν όχι, επιστρέφει την τιμή του δείκτη προώθησης που εγκατέστησε το νήμα-νικητής.

Ο διαχειριστής μνήμης του Ogasawara [84] λαμβάνει υπόψη τη μη ομοιόμορφη προσπέλαση μνήμης και διαιρεί το σωρό σε τμήματα με μία ή περισσότερες σελίδες και κάθε τμήμα απεικονίζεται σε έναν επεξεργαστή. Ο εκχωρητής μνήμης, ο οποίος χρησιμοποιείται τόσο από τα νήματα-τροποποιητές όσο και από τα νήματα-συλλέκτες προτιμά να εκχωρεί μπλοκ από τη μνήμη του προτιμώμενου επεξεργαστή. Για ένα νήμα-τροποποιητή, αυτός ταυτίζεται με τον επεξεργαστή στον οποίο εκτελείται το νήμα. Τα νήματα-συλλέκτες προσπαθούν πάντοτε να αντιγράψουν ζωντανά αντικείμενα σε σελίδες μνήμης που σχετίζονται με τον προτιμώμενο επεξεργαστή των τελευταίων. Καθώς το νήμα-τροποποιητής στο οποίο εκχωρήθηκε ένα αντικείμενο δεν είναι απαραίτητα και το νήμα που το χρησιμοποιεί περισσότερο συχνά, ο συλλέκτης χρησιμοποιεί πληροφορία **κυρίαρχου νήματος** για να προσδιορίζει τον προτιμώμενο επεξεργαστή κάθε αντικειμένου. Πρώτον, ο προτιμώμενος επεξεργαστής αντικειμένων προς τα οποία υπάρχουν άμεσες αναφορές από τη στοίβα ενός νήματος-τροποποιητή θα είναι ο επεξεργαστής στον οποίο το νήμα εκτελέσθηκε για τελευταία φορά. Αυτό πιθανόν να δημιουργεί την απαίτηση τα νήματα να ενημερώνουν περιοδικά την ταυτότητα του προτιμώμενου επεξεργαστή τους. Δεύτερον, ο συλλέκτης μπορεί να χρησιμοποιήσει πληροφορίες σχετικές με το κλείδωμα αντικειμένων για να ταυτοποιήσει το **κυρίαρχο νήμα**. Οι μηχανισμοί κλειδωμάτων συχνά αποθηκεύουν την ταυτότητα του νήματος-τροποποιητή που έχει κλειδώσει ένα αντικείμενο σε μια λέξη στην επικεφαλίδα του αντικειμένου. Παρότι με αυτόν τον τρόπο ταυτοποιείται το νήμα-τροποποιητής (και συνεπώς και ο αντίστοιχος επεξεργαστής) που τελευταίο κλείδωσε το αντικείμενο, η προσέγγιση αυτή φαντάζει αρκετή καθώς τα περισσότερα αντικείμενα δεν ξεφεύγουν ποτέ από το νήμα-τροποποιητή στο οποίο εκχωρήθηκαν. Τέλος, ο συλλέκτης μπορεί να διαδώσει την ταυτότητα του προτιμώμενου επεξεργαστή από αντικείμενα γονείς σε αντικείμενα παιδιά.

8.6.3 Ξεχωριστοί ημιχώροι-από και ημιχώροι-προς

Ο Halstead [71] παραλληλοποιεί τη συλλογή απορριμμάτων με αντιγραφή παραχωρώντας σε κάθε συλλέκτη κατά Cheney το δικό του ξεχωριστό χώρο-από και χώρο-προς. Παρότι κάθε νήμα αντιγραφής διαθέτει το δικό του συνεχόμενο κομμάτι μνήμης προς σάρωση, αυτό βρίσκεται σε ανταγωνισμό με τα υπόλοιπα νήματα για την αντιγραφή αντικειμένων και την εγκατάσταση δεικτών προώθησης. Η απλή αυτή σχεδίαση ωστόσο μπορεί να οδηγήσει σε χαμηλή ποιότητα εξισορρόπησης φορτίου εργασίας, καθώς ένας επεξεργαστής ενδέχεται να ξεμείνει από δουλειά και ενώ οι υπόλοιποι είναι απασχολημένοι. Επιπλέον, απαιτεί κάποιο μηχανισμό χειρισμού της περίπτωσης όπου υπερχειλίζει ο χώρος-προς ενός νήματος-αντιγραφέα παρότι υπάρχει χώρος σε άλλους χώρους-προς.

8.6.4 Σωροί οργανωμένοι κατά μπλοκ

Μία προσέγγιση είναι να υπερ-διαμερισθεί ο χώρος-προς και στη συνέχεια τα νήματα να τεθούν σε ανταγωνισμό για την απόκτηση μπλοκ σάρωσης και αντιγραφής. Οι Imai κ.ά. [65] διαιρούν το σωρό σε **κομμάτια** σταθερού μεγέθους και παρέχουν σε κάθε νήμα-αντιγραφέα τα δικά του κομμάτια σάρωσης και αντιγραφής επιζώντων αντικειμένων. Όταν το κομμάτι αντιγραφής ενός νήματος-αντιγραφέα γεμίσει, αυτό μεταφέρεται σε μια καθολική δεξαμενή και τα ανενεργά νήματα ανταγωνίζονται για την απόκτηση και σάρωση του και το αρχικό νήμα-αντιγραφέας αποκτά ένα καινούριο κομμάτι από το διαχειριστή ελεύθερων κομματιών.

Δύο μηχανισμοί χρησιμοποιούνται για την εξισορρόπηση του φορτίου εργασίας. Αρχικά, τα κομμάτια αντιγραφής (τα οποία ονομάζουν μονάδες επέκτασης σωρού) είναι σχετικά μικρά (256 λέξεις). Δεύτερον, κάθε κομμάτι διαιρείται σε μικρότερα **μπλοκ** (τα οποία ονομάζουν μονάδες κατανομής φορτίου), με τυπικό μέγεθος 32 λέξεις. Κάθε νήμα-αντιγραφέας προσφέρεται να παραχωρήσει μερικά από τα μη σαρωμένα μπλοκ του οποτεδήποτε χρειάζεται ένα νέο μπλοκ σάρωσης.

Μετά τη σάρωση ενός πεδίου και την αύξηση του δείκτη σάρωσης, ένα νήμα-αντιγραφέας ελέγχει αν ο τελευταίος έχει φθάσει το τέλος του μπλοκ. Αν ναι και το επόμενο αντικείμενο είναι μικρότερο από ένα μπλοκ, το νήμα-αντιγραφέας εγγράφει στο δείκτη σάρωσης τη διεύθυνση του τρέχοντος μπλοκ αντιγραφής. Με τον τρόπο αυτό μειώνεται η συμφόρηση στην καθολική δεξαμενή καθώς το νήμα-αντιγραφέας δε χρειάζεται να ανταγωνισθεί για την απόκτηση ενός μπλοκ σάρωσης.

Αν πάλι το αντικείμενο είναι μεγαλύτερο από ένα μπλοκ και μικρότερο από ένα κομμάτι, το νήμα-αντιγραφέας εγγράφει στο δείκτη σάρωσης τη διεύθυνση του τρέχοντος κομματιού αντιγραφής. Αν το αντικείμενο είναι μεγάλο, το νήμα-αντιγραφέας συνεχίζει τη σάρωση του. Τέλος, τα μεγάλα αντικείμενα μετακινούνται στην καθολική δεξαμενή αμέσως μετά την αντιγραφή τους.

8.7 Παράλληλη εκκαθάριση

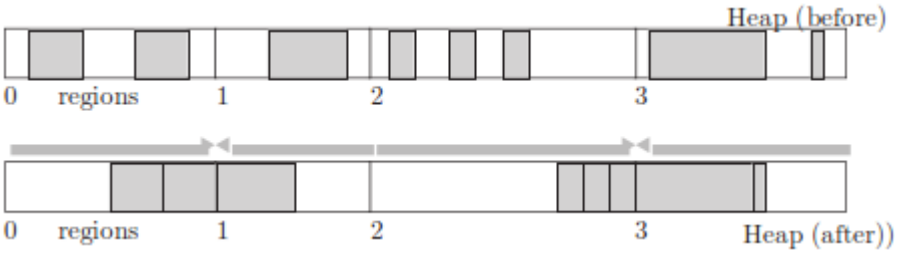
Εξετάζουμε τώρα πώς μπορούν να παραλληλοποιηθούν οι φάσεις της εκκαθάρισης και της συμπύκνωσης. Και οι δύο φάσεις έχουν την ιδιότητα πως η εξιχνίαση έχει ολοκληρωθεί και τα ζωντανά αντικείμενα του σωρού εντοπισθεί. Επίσης και οι δύο είναι εγγενώς παραλληλοποιήσιμες.

Η παραλληλοποίηση της εκκαθάρισης μπορεί να επιτευχθεί είτε διαμερίζοντας στατικά το σωρό σε συνεχόμενες περιοχές, είτε υπερ-διαμερίζοντας τον σε μπλοκ και αφήνοντας να νήματα να ανταγωνίζονται για την εκκαθάριση ενός μπλοκ σε μία καθολική ελεύθερη λίστα μπλοκ. Ωστόσο, με την υιοθέτηση αυτής της απλής στρατηγικής είναι πιθανόν η ελεύθερη λίστα να καταστεί σημείο συμφόρησης και η συλλογή να σειριοποιηθεί. Ευτυχώς όμως σε ένα τέτοιου είδους παράλληλο σύστημα, κάθε επεξεργαστής διατηρεί συνήθως τοπικές ξεχωριστές ελεύθερες λίστες με μπλοκ διαφορετικών μεγεθών για την εκχώρηση μνήμης και έτσι το πρόβλημα του ανταγωνισμού ανάγεται στο χειρισμό της επιστροφής ολόκληρων ελεύθερων μπλοκ σε έναν καθολικό εκχωρητή μπλοκ. Επιπλέον, η οκνηρή εκκαθάριση αποτελεί από τη φύση της μιας παράλληλη λύση στο πρόβλημα της εκκαθάρισης μερικώς γεμάτων μπλοκ η οποία εξισορροπεί τις εργασίες εκκαθάρισης σύμφωνα με τους ρυθμούς εκχώρησης μνήμης στα νήματα-τροποποιητές.

Το πρώτο και μοναδικό βήμα της φάσης εκκαθάρισης όταν η τελευταία πραγματοποιείται οκνηρώς είναι η ταυτοποίηση πλήρως άδειων μπλοκ και η επιστροφή τους στον εκχωρητή μπλοκ. Για να μειώσουν τον ανταγωνισμό μεταξύ των νημάτων-εκκαθαριστών, οι Endo κ.ά. [50] παρέχουν σε κάθε ένα από αυτά έναν αριθμό από συνεχόμενα μπλοκ προς τοπική επεξεργασία. Ο συλλέκτης τους χρησιμοποιεί bitmap σήμανσης, τα οποία αποθηκεύονται στις επικεφαλίδες των μπλοκ, ξεχωριστά από αυτά. Αυτή η προσέγγιση καθιστά εύκολο τον προσδιορισμό του κατά πόσο ένα μπλοκ είναι τελείως κενό από ζωντανά αντικείμενα. Τα άδεια μπλοκ ταξινομούνται και συνενώνονται πριν επιστραφούν σε μια τοπική ελεύθερη λίστα. Τα μερικώς γεμάτα από ζωντανά αντικείμενα μπλοκ εισάγονται σε τοπικές ξεχωριστές λίστες ανάκτησης για μπλοκ ξεχωριστών μεγεθών προς μελλοντική οκνηρή εκκαθάριση από τα νήματα-τροποποιητές. Μόλις ένας επεξεργαστής ολοκληρώσει την εκκαθάριση του τοπικού του συνόλου εκκαθάρισης, συγχωνεύει την ελεύθερη λίστα μπλοκ του με την καθολική ελεύθερη λίστα μπλοκ. Η τελευταία ερώτηση που μένει να απαντηθεί αφορά στο τι κάνει ένα νήμα-τροποποιητής στην περίπτωση που τόσο η τοπική λίστα ανάκτησης όσο και η καθολική δεξαμενή μπλοκ είναι άδειες. Μια λύση είναι να κλέψει ένα μπλοκ από ένα άλλο νήμα-τροποποιητή, κάτι που απαιτεί το συγχρονισμό της απόκτησης του επόμενου μπλοκ προς εκκαθάριση. Το κόστος αυτό δεν είναι μεγάλο αν αναλογισθεί κανείς πως αφενός η απόκτηση ενός μπλοκ προς εκκαθάριση συμβαίνει πιο σπάνια από ότι η αίτηση για εκχώρηση μνήμης στο εσωτερικό ενός μπλοκ και αφετέρου ο ανταγωνισμός πολλών νημάτων για την απόκτηση ενός μπλοκ προς εκκαθάριση δεν αναμένεται να συμβαίνει συχνά στην πράξη.

8.8 Παράλληλη συμπύκνωση

Η παραλληλοποίηση αλγορίθμων συλλογής απορριμμάτων με σήμανση και συμπύκνωση αφορά στην παράλληλη σήμανση των ζωντανών αντικειμένων και στη συνέχεια στην παράλληλη μετακίνηση αυτών. Ωστόσο η παράλληλη ολισθαίνουσα συμπύκνωση είναι απλούστερη από την παράλληλη αντιγραφή, τουλάχιστον σε συνεχόμενους σωρούς. Για παράδειγμα, όταν τα ζωντανά αντικείμενα έχουν σημανθεί ο προορισμός των αντικειμένων που θα μετακινηθούν δεν αλλάζει: οι καταστάσεις συναγωνισμού επηρεάζουν την επίδοση και όχι τόσο την ορθότητα. Μετά τη φάση της σήμανσης, οι περισσότεροι συλλέκτες με σήμανση και συμπύκνωση απαιτούν δύο ή και περισσότερα περάσματα στο σωρό προκειμένου να προσδιορίσουν τη διεύθυνση προώθησης κάθε αντικειμένου, να ενημερώσουν τις αναφορές προς κάθε αντικείμενο και να μετακινήσουν τα αντικείμενα. Όπως είδαμε στο κεφάλαιο 3 διαφορετικοί αλγόριθμοι μπορεί να εκτελούν αυτές τις εργασίες με διαφορετική σειρά ή να συνδυάσουν την εκτέλεση δύο εργασιών σε ένα μόνο πέρασμα.

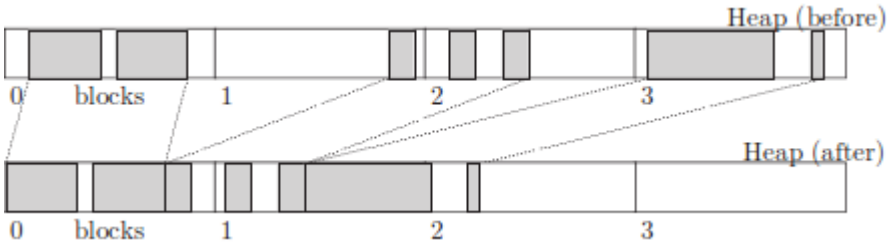


Σχήμα 8.4: Διαίρεση του σωρού σε μία περιοχή ανά νήμα-συμπυκνωτή και εναλλαγή της κατεύθυνσης ολίσθησης αντικειμένων μεταξύ δύο διαδοχικών νημάτων.

Οι Flood κ.ά. [53] χρησιμοποιούν παράλληλη σήμανση και παράλληλη συμπύκνωση για τη διαχείριση της παλαιάς γενεάς στον παράλληλο γενεαλογικό συλλέκτη της εικονικής τους μηχανής για τη γλώσσα Java. Ο συλλέκτης χρησιμοποιεί τρεις επιπλέον φάσεις μετά την ολοκλήρωση της παράλληλης σήμανσης για να (i) υπολογίσει διευθύνσεις προώθησης (ii) ενημερώσει αναφορές και (iii) μετακινήσει αντικείμενα. Το ενδιαφέρον χαρακτηριστικό της σχεδίασης τους είναι πως χρησιμοποιούν διαφορετική στρατηγική εξισορρόπησης φορτίου σε κάθε διαφορετική φάση συμπύκνωσης. Οι αλγόριθμοι συμπύκνωσης για μονοεπεξεργαστικά συστήματα πραγματοποιούν την ολίσθηση των ζωντανών αντικειμένων στο ένα άκρο του σωρού. Αν όμως πολλαπλά νήματα-συμπυκνωτές μετακινούν αντικείμενα παράλληλα, τότε απαιτείται προσοχή ούτως ώστε να αποτραπεί η εγγραφή δεδομένων ενός ζωντανού αντικειμένου από ένα νήμα-συμπυκνωτή πριν τη μετακίνηση του αντικειμένου από ένα άλλο νήμα-συμπυκνωτή. Για το λόγο αυτό, οι Flood κ.ά. δε συμπυκνώνουν όλα τα αντικείμενα σε ένα πυκνό άκρο του σωρού, αλλά αντίθετα διαιρούν το σωρό σε περιοχές και αναθέτουν σε κάθε νήμα συμπύκνωσης τη δική του περιοχή. Κάθε νήμα-συμπυκνωτής είναι υπεύθυνο για την ολίσθηση ζωντανών αντικειμένων μόνο στη δική του περιοχή. Επιπλέον, προκειμένου να μειωθεί ο (περιορισμένος) κατακερματισμός που μπορεί να προκύψει από τη χρήση αυτής της στρατηγικής διαμέρισης του σωρού, η κατεύθυνση ολίσθησης αντικειμένων εναλλάσσεται μεταξύ δύο διαδοχικών περιοχών (σχήμα 8.4).

Το πρώτο βήμα είναι η εγκατάσταση ενός δείκτη προώθησης στην επικεφαλίδα κάθε ζωντανού αντικειμένου. Ο δείκτης αυτός θα αποθηκεύει τη διεύθυνση στην οποία το αντικείμενο πρόκειται να μετακινηθεί. Σε αυτή τη φάση, ο σωρός υπερ-διαμερίζεται για να βελτιωθεί η εξισορρόπηση φορτίου. Ο χώρος διαιρείται σε M **μονάδες** ευθυγραμμισμένες με βάση τα αντικείμενα και η κάθε μονάδα έχει σχεδόν το ίδιο μέγεθος. Οι Flood κ.ά. υπολόγισαν πως μία καλή επιλογή για τον UltraSPARC διακομιστή είναι ο αριθμός των χρησιμοποιούμενων μονάδων να είναι τετραπλάσιος του αριθμού των νημάτων-συμπυκνωτών ($M = 4N$). Τα νήματα-συμπυκνωτές ανταγωνίζονται μεταξύ τους για την απόκτηση μονάδων και στη συνέχεια υπολογίζουν τον όγκο των ζωντανών δεδομένων σε κάθε μονάδα. Μάλιστα, για να διευκολυνθούν οι επόμενες φάσεις, συνενώνουν γειτονικά απορρίμματα. Μόλις ο όγκος των ζωντανών δεδομένων σε κάθε μονάδα γίνει γνωστός, ο σωρός μπορεί να διαιρεθεί σε N **περιοχές** που καθεμία απ' τις οποίες περιλαμβάνει σχεδόν τον ίδιο όγκο ζωντανών δεδομένων. Αυτές οι περιοχές είναι ευθυγραμμισμένες με τις μονάδες της προηγούμενης φάσης. Επίσης υπολογίζεται η διεύθυνση προορισμού για κάθε ζωντανό αντικείμενο, λαμβάνοντας υπόψη την κατεύθυνση ολίσθησης σε κάθε περιοχή. Τα νήματα-συμπυκνωτές πλέον ανταγωνίζονται μεταξύ τους για την απόκτηση μονάδων και εγκαθιστούν τους δείκτες προώθησης σε κάθε ζωντανό αντικείμενο των μονάδων τους.

Η επόμενη φάση πραγματοποιεί την ενημέρωση των κατάλληλων αναφορών με τις νέες διευθύνσεις των μετακινηθέντων αντικειμένων. Ως συνήθως αυτό απαιτεί τη σάρωση στοιβών των



Σχήμα 8.5: Συμπύκνωση με ολίσθηση μπλοκ. Ο Abuaiadh κ.ά. ολισθαίνουν ολισθαίνουν ολόκληρα μπλοκ και όχι ξεχωριστά αντικείμενα.

νημάτων-τροποποιητών, των αναφορών σε αντικείμενα ενός υποχώρου του σωρού που είναι αποθηκευμένες σε πεδία αντικειμένων εκτός της μονάδας καθώς επίσης και των ζωντανών αντικειμένων αυτού του υποχώρου του σωρού (για παράδειγμα στην παλαιά γενεά). Οποιαδήποτε κατάλληλη στρατηγική διαμοιρασμού φορτίου μπορεί να χρησιμοποιηθεί. Οι Flood κ.ά. χρησιμοποιούν εκ νέου τη διαμέριση σε μονάδες για τη σάρωση του προς συμπύκνωση υποχώρου του σωρού (της παλαιάς γενεάς) παρότι η σάρωση της νέας γενεάς πραγματοποιείται ως μια αδιαίρετη εργασία, δηλαδή σειριακά. Η τελευταία φάση μετακινεί τα αντικείμενα. Οι Flood κ.ά. αναθέτουν σε κάθε νήμα-συμπυκνωτή τη μετακίνηση αντικειμένων μιας περιοχής. Αυτή η προσέγγιση εξισορροπεί το φόρτο εργασίας ομοιόμορφα μεταξύ των νημάτων- συμπυκνωτών, αφού οι περιοχές ορίσθηκαν ώστε να έχουν σχεδόν τον ίδιο όγκο ζωντανών δεδομένων.

Υπάρχουν δύο μειονεκτήματα όσον αφορά τον τρόπο που ο αλγόριθμος συμπυκνώνει αντικείμενα. Πρώτον, πραγματοποιεί τρία περάσματα στο σωρό. Δεύτερον, αντί να συμπυκνώσουν όλα τα ζωντανά αντικείμενα στο ένα άκρο της παλαιάς γενεάς του σωρού, οι Flood κ.ά. συμπυκνώνουν την παλαιά γενεά σε N πυκνά τμήματα, αφήνοντας $\lceil \frac{N+1}{2} \rceil$ κενά για εκχώρηση. Δε σπαταλάται χώρος σε κάθε συμπυκνωμένο τμήμα εκτός ίσως για λόγους ευθυγράμμισης των αντικειμένων. Ωστόσο, αν ο αριθμός των περιοχών ή νημάτων-συμπυκνωτών είναι πολύ μεγάλος, η εκχώρηση μεγάλων αντικειμένων στα νήματα-τροποποιητές μπορεί να καταστεί αδύνατη.

Οι Abuaiadh κ.ά. [2] για να αντιμετωπίσουν το πρώτο πρόβλημα υπολογίζουν μόνο και δεν αποθηκεύουν τις διευθύνσεις προώθησης, χρησιμοποιώντας το bitmap σήμανσης και ένα διάνυσμα μετατόπισης που αποθηκεύει τη νέα διεύθυνση του πρώτου ζωντανού αντικειμένου σε κάθε μικρό μπλοκ του σωρού. Λύνουν το δεύτερο πρόβλημα υπερ-διαμερίζοντας το σωρό σε έναν αριθμό σχετικά μεγάλων περιοχών. Για παράδειγμα, προτείνουν το πλήθος των περιοχών να είναι τετραπλάσιο του αριθμού των επεξεργαστών και η κάθε περιοχή να έχει μέγεθος τουλάχιστον 4 MB. Οι περιοχές του σωρού συμπυκνώνονται με τη σειρά. Τα νήματα- συμπυκνωτές ανταγωνίζονται για την απόκτηση μιας περιοχής χρησιμοποιώντας μια ατομική λειτουργία για να αυξήσουν έναν καθολικό μετρητή (ή δείκτη). Αν η αύξηση είναι επιτυχής, το νήμα-συμπυκνωτής έχει αποκτήσει την περιοχή. Αν πάλι όχι, η συγκεκριμένη περιοχή βρίσκεται στην κατοχή κάποιου άλλου νήματος-συμπυκνωτή και το νήμα προσπαθεί να αποκτήσει την επόμενη περιοχή. Ένας πίνακας αποθηκεύει δείκτες προς την αρχή του ελεύθερου χώρου κάθε περιοχής. Αφού αποκτήσει μια περιοχή για συμπύκνωση, ένα νήμα-συμπυκνωτής διεκδικεί μία περιοχή στην οποία μπορεί να μετακινήσει αντικείμενα. Ένα νήμα-συμπυκνωτής αποκτά μια περιοχή προσπαθώντας να γράψει την ειδική τιμή **null** στο αντίστοιχο πεδίο του πίνακα. Τα νήματα-συμπυκνωτές ποτέ δε συμπυκνώνουν από ή προς μια περιοχή της οποίας ο αντίστοιχος δείκτης στον πίνακα έχει την τιμή **null**, ενώ δε μεταφέρονται αντικείμενα από κάποια περιοχή με μικρότερο αριθμό προς κάποια περιοχή με μεγαλύτερο αριθμό. Η πρόοδος εξασφαλίζεται αφού ένα νήμα-συμπυκνωτής μπορεί πάντοτε να συμπυκνώσει μια περιοχή στον

εαυτό της. Όταν ένα νήμα-συμπυκνωτής ολοκληρώσει τη συμπύκνωση μιας περιοχής, ενημερώνει τον κατάλληλο δείκτη του πίνακα και στην περίπτωση που μια περιοχή είναι γεμάτη, ο τελευταίος εξακολουθεί να έχει την τιμή **null**.

Οι Abuaiadh κ.ά. εξερεύνησαν δύο τρόπους μετακίνησης αντικειμένων. Η βέλτιστη συμπύκνωση με τον ελάχιστο κατακερματισμό προκύπτει από τη μετακίνηση ζωντανών αντικειμένων ξεχωριστά, όπως ακριβώς περιγράφηκε πιο πάνω. Επειδή κάθε αντικείμενο ενός μπλοκ μετακινείται σε μία θέση που εν μέρει προσδιορίζεται από το διάνυσμα μετατόπισης για το εν λόγω μπλοκ, τα αντικείμενα ενός μπλοκ δε διασκορπίζονται μεταξύ δύο διαφορετικών περιοχών. Οι Abuaiadh κ.ά. δοκίμασαν επίσης να μειώσουν το χρόνο συμπύκνωσης θυσιάζοντας την ποιότητα της δια μέσου της μετακίνησης ολόκληρων μπλοκ (256 bytes). Καθώς τα αντικείμενα ενός χώρου μνήμης που εκχωρείται σειριακά έχουν την τάση να ζουν και να πεθαίνουν σε συστάδες, οι Abuaiadh κ.ά. υπολόγισαν πως η τεχνική αυτή μπορεί να μειώσει το συνολικό χρόνο συμπύκνωσης κατά 20% πληρώνοντας το κόστος της αύξησης του μεγέθους της περιοχής συμπύκνωσης κατά ένα πολύ μικρό ποσοστό. Από την άλλη πλευρά όμως, δεν είναι δύσκολη η επινόηση μιας περίπτωσης όπου η τεχνική αυτή οδηγεί σε μηδενική συμπύκνωση.

Ο μηχανισμός υπολογισμού μόνο και όχι αποθήκευσης της διεύθυνσης προώθησης ενός αντικειμένου υιοθετήθηκε αργότερα από τον αλγόριθμο Compressor των Kermany και Petrank [73]. Ο αλγόριθμος Compressor ωστόσο εισάγει κάποιες διαφορές. Πρώτον, καθώς η δεύτερη φάση της συλλογής σαρώνει το bitmap σήμανσης, υπολογίζει εκτός από το διάνυσμα μετατόπισης και ένα επιπλέον βοηθητικό διάνυσμα **πρώτου αντικειμένου**, το οποίο δεικτοδοτείται από τις σελίδες προορισμού των αντικειμένων και ονομάζεται διάνυσμα πρώτου αντικειμένου. Σε κάθε θέση του βοηθητικού διανύσματος αποθηκεύεται η αρχική διεύθυνση του πρώτου αντικειμένου που θα μετακινηθεί στην αντίστοιχη σελίδα. Η συμπύκνωση καθεαυτή εκκινεί με την ενημέρωση των ριζών.

Η δεύτερη διαφορά είναι πως στη συνέχεια κάθε νήμα-συμπυκνωτής ανταγωνίζεται για την απόκτηση μιας σελίδας-προς από το βοηθητικό διάνυσμα πρώτου αντικειμένου. Μετά την επιτυχή προσπάθεια απόκτησης μιας σελίδας-προς, ένα νήμα-συμπυκνωτής αντιστοιχίζει την εικονική αυτή σελίδα σε μία καινούρια φυσική σελίδα και ξεκινά να μεταφέρει αντικείμενα με χρήση των διανυσμάτων σήμανσης και μετατόπισης και εκκινώντας από τη διεύθυνση που υποδεικνύει το διάνυσμα πρώτου αντικειμένου. Η απόκτηση μιας νέας σελίδας-προς, στην οποία μπορεί να μεταφέρει αντικείμενα επιτρέπει στον αλγόριθμο Compressor τη χρήση παράλληλων νημάτων- συμπυκνωτών. Εκ πρώτης όψεως, φαίνεται πώς ο Compressor είναι αλγόριθμος συλλογής με αντιγραφή και όχι με σήμανση και εκκαθάριση. Στην πραγματικότητα όμως πρόκειται για ένα συλλέκτη με σήμανση και ολισθαίνουσα συμπύκνωση. Κάθε νήμα-συμπυκνωτής διαχειρίζεται κάθε ζεύγος από μια εικονική σελίδα-από και μια εικονική σελίδα-προς με χρήση μίας φυσικής σελίδας μνήμης: όπως ακριβώς προσθέτει μια αντιστοίχιση μεταξύ της εικονικής σελίδας-προς και μιας καινούριας φυσικής σελίδας τη στιγμή της απόκτησης της σελίδας-προς, με τον ίδιο τρόπο αφαιρεί μια αντιστοίχιση μεταξύ μιας εικονικής σελίδας-από και της αντίστοιχης φυσικής σελίδας τη χρονική στιγμή της ολοκλήρωσης της μεταφοράς των ζωντανών αντικειμένων από τη σελίδα-προς.

Αυτή η σχεδίαση ελαχιστοποιεί την επιβάρυνση εξαιτίας του συγχρονισμού των νημάτων-συμπυκνωτών. Μόνο η προσπάθεια απόκτησης μιας σελίδας-προς από το διάνυσμα πρώτου-αντικειμένου εκτελείται συγχρονισμένα από ένα νήμα-συμπυκνωτή. Αν η προσπάθεια είναι ανεπιτυχής, το νήμα-συμπυκνωτής προσπαθεί να αποκτήσει τη σελίδα-προς στην οποία αντιστοιχεί η επόμενη θέση του διανύσματος πρώτου αντικειμένου. Ο τερματισμός της συμπύκνωσης είναι εξίσου απλός: η εκτέλεση ενός νήματος-συμπύκνωσης ολοκληρώνεται όταν αυτό έχει εξετάσει και την τελευταία θέση του διανύσματος πρώτου αντικειμένου.

8.9 Θέματα προς εξέταση

8.9.1 Ορολογία

Οι δημοσιεύσεις του 20ού αιώνα χρησιμοποιούν αυθαίρετα και αδιακρίτως τους όρους παράλληλος, ταυτόχρονος και πραγματικού-χρόνου. Από το 2000 ωστόσο, οι ερευνητές έχουν υιοθετήσει μία συνεπή χρήση των όρων. Ένας παράλληλος συλλέκτης απορριμμάτων είναι ένας συλλέκτης που χρησιμοποιεί πολλά νήματα συλλογής, τα οποία εκτελούνται παράλληλα. Ο κόσμος μπορεί να διακόπτεται αλλά και να μην διακόπτεται κατά τη διάρκεια εκτέλεσης των παράλληλων νημάτων-συλλεκτών. Με τον ίδιο τρόπο που είναι επιθυμητό να επιτρέπεται σε πολλαπλά νήματα-τροποποιητές να κάνουν χρήση όλων των παράλληλων πόρων, η επίτρεψη της παράλληλης συλλογής όταν η αρχιτεκτονική το επιτρέπει αποτελεί συνετή πράξη.

8.9.2 Αξίζει η παραλληλοποίηση;

Λαμβάνοντας υπόψη το νόμο του Amdahl [1] προκύπτει το εύλογο ερώτημα αν υπάρχει επαρκής δουλειά προς παραλληλοποίηση. Είναι εύκολο να φανταστεί κανείς σενάρια που δεν προσφέρουν ευκαιρίες για παραλληλοποίηση: ένα συνηθισμένο παράδειγμα τέτοιου σεναρίου αποτελεί η εξιχνίαση μιας απλής συνδεδεμένης λίστας. Όπως παρατηρεί ο Siebert [106], υπάρχουν πολλές ενδείξεις πώς οι πραγματικές εφαρμογές χρησιμοποιούν ένα πολύ πιο πλούσιο ρεπερτόριο από δομές δεδομένων προσφέροντας δυνατότητες παραλληλοποίησης υψηλού βαθμού. Εκτός από τη σήμανση, οι υπόλοιπες δραστηριότητες της συλλογής απορριμμάτων προσφέρουν εμφανώς περισσότερες δυνατότητες εκμετάλλευσης του παράλληλου υλικού. Για παράδειγμα, η εκκαθάριση και η συμπύκνωση είναι εγγενώς παραλληλοποιήσιμες διαδικασίες (παρότι η δεύτερη απαιτεί λίγη παραπάνω προσοχή). Ακόμη και στη φάση της σήμανσης ωστόσο, οι στοίβες και τα σύνολα ανάμνησης των νημάτων-τροποποιητών δύνανται να σαρώνονται παράλληλα και με μία μικρή επιβάρυνση συγχρονισμού. Η παράλληλη εξιχνίαση του γράφου αντικειμένων απαιτεί προσεκτικό σχεδιασμό όσον αφορά το χειρισμό των λιστών εργασιών ούτως ώστε να περιορισθεί το κόστος συγχρονισμού και ταυτόχρονα οι παράλληλοι πόροι του συστήματος να χρησιμοποιούνται όσο το δυνατόν αποδοτικότερα.

8.9.3 Στρατηγικές εξισορρόπησης φορτίου εργασίας

Η αποδοτική παραλληλοποίηση της συλλογής απαιτεί προσεκτικό συμβιβασμό μεταξύ της εξισορρόπησης του φορτίου εργασίας και του απαραίτητου συγχρονισμού μεταξύ των επεξεργαστών. Η εξισορρόπηση του φορτίου εργασίας εξασφαλίζει την αποτροπή μιας κατάστασης όπου ορισμένοι επεξεργαστές εκτελούν όλες τις εργασίες και κάποιοι άλλοι είναι αδρανείς. Ο συγχρονισμός είναι απαραίτητος καθώς εξασφαλίζει την ακεραιότητα τόσο των ιδιωτικών δομών δεδομένων των νημάτων όσο και των καθολικών δομών δεδομένων του σωρού.

Η γενική λύση ορίζει την ανάθεση στα νήματα-συλλέκτες κβάντων εργασίας τα οποία μπορούν να εκτελέσουν χωρίς περαιτέρω συγχρονισμό. Η φθηνότερη λύση όσον αφορά το κόστος συγχρονισμού αφορά τη στατική διαίρεση της εργασίας είτε κατά την εκκίνηση του προγράμματος

[1] Ο νόμος του Amdahl δηλώνει πώς η επιτάχυνση που προκύπτει από την παραλληλοποίηση ενός προγράμματος εξαρτάται από το μέρος του προγράμματος που μπορεί να παραλληλοποιηθεί. Συγκεκριμένα, αν s είναι ο χρόνος που δαπανάται (από ένα σειριακό επεξεργαστή) στα σειριακά μέρη ενός προγράμματος και p είναι ο χρόνος που δαπανάται (από ένα σειριακό επεξεργαστή) στα μέρη που μπορούν να εκτελεστούν παράλληλα από n επεξεργαστές, η επιτάχυνση είναι $\frac{1}{s+\frac{p}{n}}$.

είτε ακριβώς πριν από κάθε κύκλο συλλογής. Στην περίπτωση αυτή απαιτείται συντονισμός μεταξύ των παράλληλων νημάτων μόνο για τον εντοπισμό του τερματισμού μιας φάσης συλλογής. Ωστόσο η στατική διαμέριση μπορεί να μην οδηγήσει σε ικανοποιητική εξισορρόπηση του φορτίου εργασίας. Το φορτίο εργασίας μπορεί να εξισορροπηθεί επίσης υπερ-διαμερίζοντας τη διαθέσιμη εργασία σε υποεργασίες και θέτοντας τα παράλληλα νήματα-συλλέκτες να ανταγωνίζονται μεταξύ τους για την απόκτηση κβάντων εργασίας από μια καθολική δεξαμενή καθώς και να επιστρέφουν καινούρια κβάντα εργασίας σε αυτή. Η δεύτερη προσέγγιση έχει εμφανώς μεγαλύτερο κόστος συγχρονισμού.

Συχνά βέβαια είναι δυνατή η εφαρμογή διαφορετικών στρατηγικών εξισορρόπησης του φορτίου εργασιών σε διαφορετικές φάσεις ενός κύκλου συλλογής. Η πληροφορία που γίνεται διαθέσιμη μετά το πέρας μιας φάσης (συνήθως της φάσης σήμανσης) μπορεί να χρησιμοποιηθεί προκειμένου να εκτιμηθεί μια δίκαιη διαίρεση της εργασίας των επόμενων φάσεων ανάμεσα στα παράλληλα νήματα-συλλέκτες.

8.9.4 Χειρισμός εξιχνίασης

Η εξιχνίαση του σωρού περιλαμβάνει την κατανάλωση εργασίας και την παραγωγή περαιτέρω εργασίας. Μια δομή δεδομένων, όπως μια στοίβα ή μια ουρά είναι απαραίτητη για την καταγραφή της εργασίας. Μια μοναδική, διαμοιραζόμενη δομή θα οδηγούσε σε υψηλό κόστος συγχρονισμού και συνεπώς θα πρέπει τα νήματα-συλλέκτες να διατηρούν τις δικές τους ιδιωτικές δομές δεδομένων. Ωστόσο η εξισορρόπηση του φορτίου εργασίας απαιτεί την ύπαρξη ενός μηχανισμού μεταφοράς μονάδων εργασίας μεταξύ των νημάτων. Η πρώτη απόφαση αφορά στην επιλογή του μηχανισμού. Δομές δεδομένων που υποστηρίζουν κλοπή εργασίας μπορούν να χρησιμοποιηθούν ώστε να επιτραπεί η μεταφορά μονάδων εργασίας μεταξύ των νημάτων. Η ιδέα είναι η κοινή λειτουργία (ώθηση και εξώθηση στοιχείων κατά την εξιχνίαση) να καταστεί όσο φθηνότερη (δηλαδή ασυγχρόνιστη) γίνεται, ενώ ταυτόχρονα επιτρέπονται λιγότερο συχνές λειτουργίες (ασφαλής μεταφορά εργασίας μεταξύ των νημάτων). Οι Endo κ.ά. [50] παρέχουν σε κάθε νήμα-σημαντή μία ιδιωτική στοίβα και μία κλεπτόμενη ουρά εργασίας, ενώ οι Flood κ.ά. [53] χρησιμοποιούν απλώς μια διπλά τερματισμένη ουρά τόσο για τη σήμανση όσο και για την κλοπή. Η τεχνική των γκρι πακέτων [85] διατηρεί μια καθολική δεξαμενή πακέτων εργασιών. Εδώ κάθε νήμα-σημαντής ανταγωνίζεται για την απόκτηση ενός πακέτου εργασίας από την καθολική δεξαμενή και επιστρέφει καινούρια εργασία σε αυτή μέσω ενός φρέσκου πακέτου. Οι Cheng και Blelloch [35] λύνουν το πρόβλημα του συγχρονισμού των λειτουργιών ώθησης και εξώθησης στοίβας χωρίζοντας την εξιχνίαση σε δύο φάσεις που ονομάζουν δωμάτια. Η απλούστερη εκδοχή του αλγορίθμου τους ορίζει πώς όλα τα νήματα-αντιγραφείς βρίσκονται είτε στο δωμάτιο ώθησης είτε στο δωμάτιο εξώθησης. Σε κάθε περίπτωση όλα τα νήματα-αντιγραφείς επιθυμούν να μετακινήσουν το δείκτη στοίβας προς την ίδια κατεύθυνση και αυτό έχει ως αποτέλεσμα τη δυνατότητα χρήσης μιας ατομικής εντολής όπως η FetchAndAdd.

Η δεύτερη απόφαση αφορά το μέγεθος και τον τρόπο της μεταφερόμενης εργασίας. Η ελάχιστη μεταφερόμενη μονάδα εργασίας είναι ένα στοιχείο της στοίβας. Ωστόσο, η χρήση μικρών δομών δεδομένων μπορεί να οδηγήσει στη διακίνηση μεγαλύτερου όγκου δεδομένων μεταξύ των νημάτων. Στον παράλληλο, ταυτόχρονο και πραγματικού χρόνου συλλέκτη του, ο Siebert [107] επιτρέπει σε έναν αδρανή επεξεργαστή να κλέψει ολόκληρη τη δουλειά από έναν άλλον επεξεργαστή. Αυτή η απόφαση είναι συνετή μόνο αν η περίπτωση να ξεμείνουν και οι υπόλοιποι επεξεργαστές από δουλειά περίπου την ίδια χρονική στιγμή είναι σχεδόν απίθανη. Μια συνήθης πρακτική είναι να μεταφέρεται ένα ενδιάμεσο μέγεθος εργασίας ανάμεσα στα νήματα. Με τη

χρήση γκρι πακέτων σταθερού μεγέθους αυτό γίνεται αυτόματα. Μια άλλη επιλογή είναι η μεταφορά της μισής ιδιωτικής στοίβας σήμανσης ενός νήματος-σημαντή. Αν οι ιδιωτικές στοίβες σήμανσης έχουν σταθερό μέγεθος, τότε απαιτείται και ένας μηχανισμός χειρισμού της υπερχείλισης. Και αυτή η περίπτωση αντιμετωπίζεται αυτόματα από την τεχνική των γκρι πακέτων: όταν ένα γεμάτο πακέτο εξόδου γεμίζει μεταφέρεται στην καθολική και δεξαμενή και λαμβάνεται ένα άδειο πακέτο από αυτή. Οι Flood κ.ά. [53] χειρίζονται την υπερχείλιση μέσω αντικειμένων κλάσεων στη γλώσσα Java, πληρώνοντας ένα μικρό κόστος σε χώρο για κάθε κλάση.

Στο επίκεντρο της σχεδίασης των παραπάνω αλγορίθμων είναι ο επεξεργαστής. Στρατηγικές όπου στο επίκεντρο της σχεδίασης είναι η μνήμη και οι οποίες λαμβάνουν υπόψη τις θέσεις των αντικειμένων στο σωρό είναι πιο συνήθεις σε αλγορίθμους για συλλογή με παράλληλη αντιγραφή, όπου οι λίστες εργασίες οργανώνονται ως ουρές Cheney. Τα ζητήματα σχεδίασης αφορούν: (i) το μέγεθος των μπλοκ (κβάντων εργασίας) (ii) τη σειρά επεξεργασίας των μπλοκ καθώς και τον καθορισμό του ποια μπλοκ θα επιστραφούν στην καθολική δεξαμενή και (iii) στην ιδιοκτησία ποιου νήματος-τροποποιητή βρίσκεται ένα αντικείμενο. Υπάρχουν δύο πτυχές όσον αφορά την επιλογή του μεγέθους των μπλοκ. Πρώτον, κάθε μετακινών συλλέκτης πρέπει να διαθέτει μια ιδιωτική περιοχή στο σωρό για την αντιγραφή αντικειμένων. Το κομμάτι μνήμης που αντιστοιχεί στην περιοχή πρέπει να είναι αρκετά μεγάλο ώστε να μειωθεί η συμφόρηση στο διαχειριστή κομματιών που προκύπτει από τον ανταγωνισμό των νημάτων. Μεγάλο μέγεθος κομματιών ωστόσο οδηγεί σε χαμηλή ποιότητα εξισορρόπησης του φορτίου εργασίας και έτσι τα κομμάτια συνήθως διαιρούνται περαιτέρω σε μπλοκ, τα οποία λειτουργούν ως κβάντα εργασίας σε ένα συλλέκτη κατά Cheney. Δεύτερον, η απόφαση σχετικά με το ποιο θα είναι το επόμενο αντικείμενο που θα επεξεργασθεί επηρεάζει την τοπικότητα τόσο του συλλέκτη όσο και του τροποποιητή. Και στις δύο περιπτώσεις φαίνεται προτιμότερο να επιλεγεί το επόμενο μη σαρωθέν αντικείμενο του μπλοκ εκχώρησης, επιστρέφοντας ενδιάμεσα, μη σαρωθέντα ή και πλήρως σαρωθέντα μπλοκ στην καθολική δεξαμενή. Η απόφαση μπορεί επίσης να βασισθεί στο ποιος είναι ο επεξεργαστής που έχει τη μεγαλύτερη πιθανότητα χρησιμοποίησης του αντικειμένου. Ο Ogasawara [84] εισάγει την έννοια ενός κυρίαρχου νήματος για να κατευθύνει την επιλογή του ποιος επεξεργαστής πρέπει να αντιγράψει ένα αντικείμενο (και άρα σε ποια θέση αυτό θα αντιγραφεί).

8.9.5 Συγχρονισμός χαμηλού επιπέδου

Πολλές φορές εκτός από το συγχρονισμό λειτουργιών που αφορούν τις δομές δεδομένων του συλλέκτη απαιτείται και ο συγχρονισμός των λειτουργιών που δρουν σε κάθε ξεχωριστό αντικείμενο του σωρού. Για παράδειγμα, δεν έχει σημασία αν ένα αντικείμενο σημανθεί παραπάνω από μία φορές. Αν ωστόσο ο συλλέκτης χρησιμοποιεί ένα ξεχωριστό διάνυσμα για την αποθήκευση των bits σήμανσης, πρέπει να εξασφαλισθεί η ατομικότητα της τροποποίησης των τελευταίων. Καθώς τα σύνολα εντολών των περισσότερων σύγχρονων επεξεργαστών δεν παρέχουν εντολές για την ενημέρωση ενός συγκεκριμένου bit σε μία λέξη μνήμης, η σήμανση ενός bit ενδέχεται να προκαλέσει την αναμονή σε κάποιο βρόχο για την ατομική ενημέρωση ολόκληρου του byte. Από την άλλη πλευρά, αν το bit σήμανσης αποθηκεύεται στην επικεφαλίδα ενός αντικειμένου ή το διάνυσμα σήμανσης χρησιμοποιεί ένα ξεχωριστό byte σήμανσης για κάθε αντικείμενο δεν απαιτείται κανένας συγχρονισμός.

Ένας συλλέκτης αντιγραφής δεν πρέπει να "σημάνει" (δηλαδή αντιγράψει) ένα αντικείμενο περισσότερες από μία φορές καθώς αυτό αλλάζει την τοπολογία του γράφου αντικειμένων και πιθανώς έχει καταστροφικές συνέπειες για τον τροποποιητή. Η αντιγραφή ενός αντικειμένου

και η αποθήκευση της διεύθυνσης προώθησης του από ένα νήμα-αντιγραφέας πρέπει να γίνεται αντιληπτή ως μια μοναδική αδιαίρετη λειτουργία από τα υπόλοιπα νήματα αντιγραφής. Ένα πλήθος από διαφορετικές τεχνικές έχει υιοθετηθεί όσον αφορά το χειρισμό της διεύθυνσης προώθησης. Ένα νήμα- αντιγραφέας μπορεί να επιχειρήσει να εγγράψει ατομικά μια ειδική τιμή 'busy' στο πεδίο διεύθυνσης προώθησης ενός αντικειμένου και αν τα καταφέρει, να αντιγράψει στη συνέχεια το αντικείμενο και τέλος να ενημερώσει το πεδίο διεύθυνσης προώθησης με τη διεύθυνση του αντιγράφου. Κάθε άλλο νήμα-αντιγραφέας που θα διαβάσει την τιμή 'busy' οφείλει να περιμένει μέχρις ότου διαβάσει τη διεύθυνση προώθησης. Το κόστος συγχρονισμού μπορεί να ελαττωθεί με τον έλεγχο του πεδίου διεύθυνσης προώθησης πριν την προσπάθεια ατομικής εγγραφής της τιμής 'busy' σε αυτό. Ένα νήμα-αντιγραφέας μπορεί ακόμη να αντιγράψει ένα αντικείμενο αν το πεδίο διεύθυνσης προώθησης είναι κενό, και στη συνέχεια να επιχειρήσει να εγγράψει ατομικά στο πεδίο αυτό τη διεύθυνση του αντιγράφου, καταστρέφοντας το αντικείμενο αντίγραφο αν η εγγραφή αποτύχει. Η αποδοτικότητα της τελευταίας προσέγγισης θα εξαρτηθεί από τη συχνότητα συγκρούσεων κατά την εγκατάσταση των διευθύνσεων προώθησης.

8.9.6 Εκκαθάριση και συμπύκνωση

Οι φάσεις της εκκαθάρισης και της συμπύκνωσης σαρώνουν γραμμικά το σωρό (η συμπύκνωση μάλιστα περισσότερες της μίας φοράς). Και οι δύο φάσεις είναι κατάλληλες για παραλληλοποίηση. Η απλούστερη πολιτική εξισορρόπησης φορτίου διαμερίζει το σωρό σε τόσα τμήματα όσα είναι οι επεξεργαστές. Η υιοθέτηση αυτής της προσέγγισης ωστόσο μπορεί να οδηγήσει σε μη ομοιόμορφη εξισορρόπηση υπολογιστικού φορτίου αν οι ποσότητες εργασίας των διαμερίσεων είναι άνισες. Σε πρώτη προσέγγιση, η ποσότητα εργασίας είναι ανάλογη με τον αριθμό των αντικειμένων σε μια διαμέριση. Η πληροφορία αυτή είναι διαθέσιμη μετά το πέρας της φάσης σήμανσης και μπορεί να χρησιμοποιηθεί για να διαμερισθεί ο σωρός σε όχι ισομεγέθη τμήματα ώστε το καθένα από αυτά περιλαμβάνει περίπου την ίδια ποσότητα εργασίας.

Ωστόσο, αυτή η στρατηγική προϋποθέτει πώς κάθε διαμέριση μπορεί να επεξεργάζεται ανεξάρτητα από τις υπόλοιπες. Αυτό δεν είναι αληθές αν η επεξεργασία μιας διαμέρισης μπορεί να καταστρέψει πληροφορίες από τις οποίες εξαρτάται κάποια άλλη διαμέριση. Για παράδειγμα, ένα νήμα-συμπυκνωτής δεν μπορεί να μετακινήσει αντικείμενα με τυχαία σειρά στον προορισμό τους καθώς έτσι διακινδυνεύει να καταστρέψει ζωντανά αλλά όχι ακόμη μετακινηθέντα αντικείμενα. Η λύση στο πρόβλημα να υπερ-διαμερισθεί ο σωρός και τα νήματα ανταγωνίζονται για τις επόμενες διαμερίσεις που θα χρησιμοποιήσουν (μία διαμέριση από την οποία θα μετακινήσουν αντικείμενα και μία διαμέριση στην οποία θα τα μεταφέρουν).

8.9.7 Τερματισμός

Τέλος, ο τερματισμός οιασδήποτε φάσης συλλογής πρέπει να καθορίζεται ορθά. Η χρήση παράλληλων νημάτων ασφαλώς και καθιστά τον εντοπισμό του τερματισμού πιο πολύπλοκο. Το βασικό πρόβλημα είναι πώς ενώ ένα νήμα-συλλέκτης επιχειρεί να προσδιορίσει εάν η φάση έχει ολοκληρωθεί, ένα άλλο νήμα-συλλέκτης μπορεί να παράγει εργασία. Μια σωστή λύση στο πρόβλημα είναι η ανάθεση του εντοπισμού του τερματισμού σε ένα μόνο νήμα-συλλέκτη ενώ τα υπόλοιπα υποδεικνύουν ατομικά αν είναι απασχολημένα ή όχι. Απαιτείται ιδιαίτερη προσοχή όσον αφορά το σχεδιασμό του πρωτοκόλλου που ορίζει τη μετάβαση των διαφόρων μεταβλητών σημαιών από την λογική τιμή 0 στη λογική τιμή 1 και αντίστροφα. Συστήματα

με μια καθολική δεξαμενή εργασιών από την άλλη πλευρά επιτρέπουν σε περισσότερα του ενός νήματα να εντοπίζουν τον τερματισμό μιας φάσης της συλλογής.

Κεφάλαιο 9

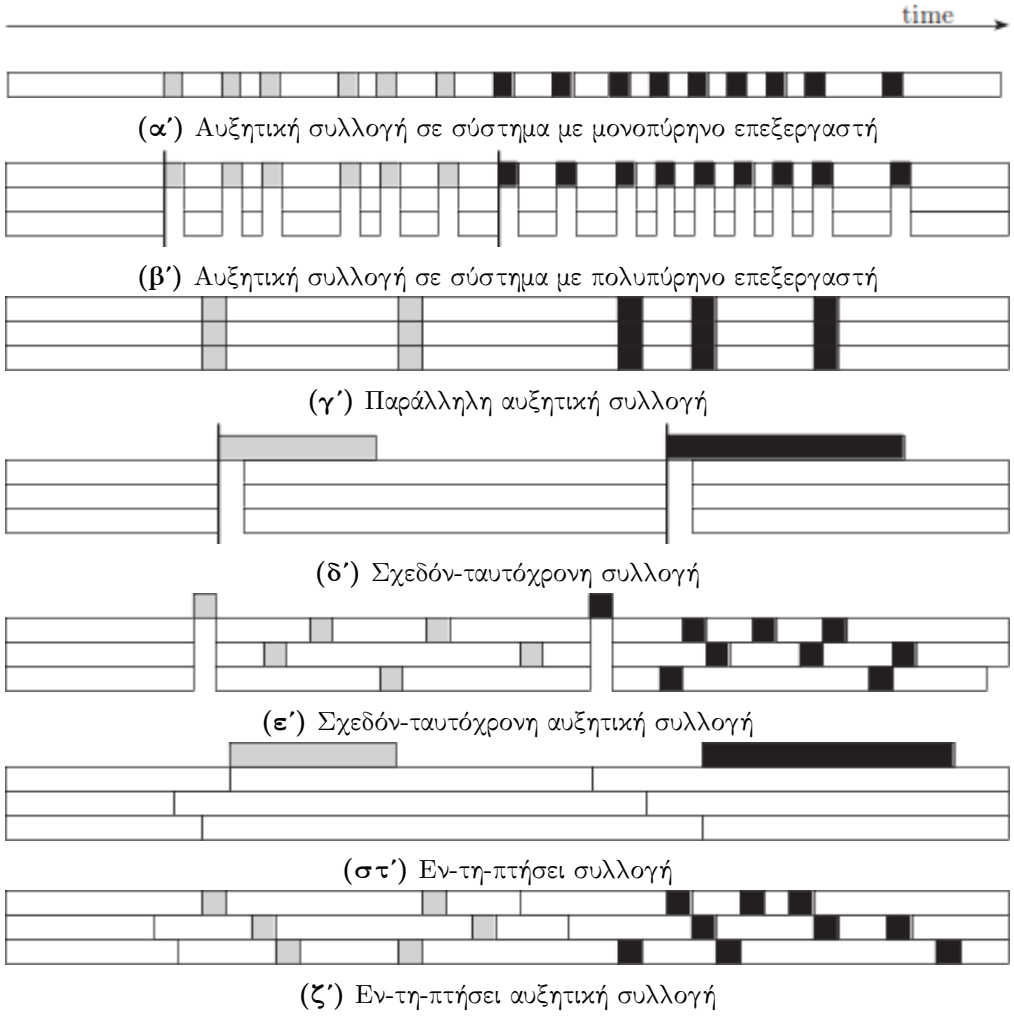
Ταυτόχρονη συλλογή απορριμμάτων

Οι βασικές αρχές της ταυτόχρονης συλλογής απορριμμάτων επινοήθηκαν αρχικώς ως ένα μέσο για την μείωση των χρόνων παύσης για συλλογή απορριμμάτων σε μονοεπεξεργαστικά συστήματα. Μέχρι στιγμής έχουμε υποθέσει πως τα νήματα- τροποποιητές παραμένουν αδρανοποιημένα όσο εκτελείται συλλογή απορριμμάτων και πως η συλλογή αυτή ολοκληρώνεται στο σύνολό της πριν τα νήματα-τροποποιητές συνεχίσουν την εκτέλεσή τους.

Εκτός από την παράλληλη συλλογή απορριμμάτων που εξετάσαμε στο προηγούμενο κεφάλαιο, ο χρόνος παύσης σε ένα μονοεπεξεργαστικό μηχάνημα δύναται να μειωθεί με το να "σπάσει" ο κύκλος της συλλογής απορριμμάτων σε μικρότερα κβάντα και η εκτέλεση του συλλέκτη να παρεμβάλλεται για κάθε ένα από αυτά τα κβάντα χρόνου στην εκτέλεση του τροποποιητή. Η μέθοδος αυτή που αναμιγνύει την εκτέλεση τροποποιητή και συλλέκτη είναι γνωστή ως **αυξητική συλλογή απορριμμάτων**. Η τεχνική αυτή είναι σημαντικά πολυπλοκότερη από ότι φαίνεται εκ πρώτης όψεως: ο συλλέκτης παύει να εκτελείται πλέον ατομικά ως προς τον τροποποιητή, με αποτέλεσμα ο γράφος προσβασιμότητας των αντικειμένων πιθανώς να αλλάζει μεταξύ δύο διαδοχικών δραστηριοποιήσεων αυτού. Επομένως, οι αυξητικοί συλλέκτες θα πρέπει να έχουν έναν τρόπο να παρακολουθούν τις αλλαγές στο γράφο των προσβάσιμων αντικειμένων ούτως ώστε αν χρειαστεί, να επανεξετάσουν αντικείμενα ή πεδία αυτών.

Παρότι αυτή η ανάμιξη δημιουργεί την ψευδαίσθηση πως συλλέκτης και τροποποιητής εκτελούνται ταυτόχρονα, αυτό δε συμβαίνει: τα νήματα-τροποποιητές αδρανοποιούνται κάθε φορά που εκτελείται ο συλλέκτης. Στην περίπτωση που ο συλλέκτης αποτελείται από περισσότερα του ενός νήματα, έχουμε να κάνουμε με **παράλληλη αυξητική συλλογή απορριμμάτων**. Η σημαντικότερη δυσκολία που προκύπτει αν επιτρέψουμε στα νήματα-συλλέκτες να εκτελεστούν ταυτόχρονα με τα νήματα-τροποποιητές αφορά στο να εξασφαλιστεί πώς ο συλλέκτης και ο τροποποιητής διατηρούν σε κάθε χρονική στιγμή μία συνεπή εικόνα του σωρού. Μία περίπτωση όπου μπορεί να προκύψει ασυνέπεια είναι εάν ο τροποποιητής επιχειρήσει να πειράξει την τιμή ενός μερικώς εξετασθέντος ή αντιγεγραμμένου αντικειμένου ή να αποκτήσει πρόσβαση σε διάφορα μεταδεδομένα ταυτόχρονα με το συλλέκτη. Ο απαιτούμενος συγχρονισμός μεταξύ συλλέκτη και τροποποιητή δεν έρχεται χωρίς κόστος.

Υποθέτοντας πως ο συλλέκτης είναι μονονηματικός και ο τροποποιητής πολυνηματικός, και πως ο τροποποιητής αδρανοποιείται για ένα πολύ σύντομο χρονικό διάστημα στην αρχή του κύκλου εκτέλεσης του συλλέκτη, ώστε ο τελευταίος να εξετάσει τις μεταβλητές-ρίζες των πρώτου, τότε έχουμε να κάνουμε με **σχεδόν-ταυτόχρονη συλλογή απορριμμάτων**.



Σχήμα 9.1: Αυξητική και ταυτόχρονη συλλογή απορριμμάτων. Κάθε μπάρα αναπαριστά μία εκτέλεση σε έναν επεξεργαστή. Οι έγχρωμες περιοχές αναπαριστούν διαφορετικούς κύκλους συλλογής απορριμμάτων.

Εάν πάλι ο μονονηματικός συλλέκτης εκτελεστεί αυξητικά, όπως πριν, έχουμε **σχεδόν-ταυτόχρονη αυξητική συλλογή απορριμμάτων**. Αξίζει να προσέξουμε πώς στην τελευταία περίπτωση ο μονονηματικός συλλέκτης μπορεί να εκτελείται σε διαφορετικό επεξεργαστή κάθε φορά.

Τέλος, αν θέλουμε να καταργήσουμε τελείως το χρόνο παύσης του τροποποιητή και να επιτρέψουμε στο μονονηματικό συλλέκτη να εκτελεστεί πραγματικά ταυτόχρονα με αυτόν, τότε ανάλογα με το αν αυτός εκτελείται συνεχόμενα ή αυξητικά, έχουμε **εν-τη-πτήσει συλλογή απορριμμάτων** είτε **εν-τη-πτήσει αυξητική συλλογή απορριμμάτων**. Δίνουμε ιδιαίτερη έμφαση στην ορολογία καθώς μέχρι πρόσφατα οι όροι παράλληλη, ταυτόχρονη, εν-τη-πτήσει και πραγματικού-χρόνου χρησιμοποιούνταν αυθαίρετα στη βιβλιογραφία.

9.1 Ορθότητα ταυτόχρονης συλλογής

Ένας ορθός ταυτόχρονος συλλέκτης πρέπει να έχει τις ακόλουθες δύο ιδιότητες:

- η **ασφάλεια** απαιτεί τη διατήρηση τουλάχιστον όλων των ζωντανών αντικειμένων και
- η **ζωντάνια** απαιτεί τον τερματισμό του κύκλου συλλογής.

9.1.1 Η τριχρωματική αφαίρεση

Η ορθότητα ταυτόχρονων συλλεκτών συνήθως αποδεικνύεται μέσω αναλλοίωτων που βασίζονται στην τριχρωματική αφαίρεση και πρέπει να διατηρούνται τόσο από το συλλέκτη όσο και από τον τροποποιητή. Υπενθυμίζουμε πως:

Λευκά αντικείμενα είναι εκείνα τα οποία δεν έχουν ακόμη ανακαλυφθεί από το συλλέκτη από την αρχή του τρέχοντος κύκλου συλλογής. Τα αντικείμενα που παραμένουν λευκά στο τέλος του κύκλου θεωρούνται μη προσβάσιμα απορρίμματα.

Γκρι αντικείμενα είναι εκείνα τα οποία έχουν ανακαλυφθεί από το συλλέκτη, αλλά ένα ή περισσότερα πεδία τους δεν έχουν εξετασθεί ακόμα (μπορεί να αναφέρονται σε λευκά αντικείμενα).

Μαύρα αντικείμενα είναι εκείνα τα οποία έχουν ανακαλυφθεί από το συλλέκτη και των οποίων όλα τα πεδία έχουν εξετασθεί (και δεν περιέχουν δείκτες προς λευκά αντικείμενα). Τα μαύρα αντικείμενα δεν επανεξετάζονται εκτός και αν αλλάξει το χρώμα τους.

Ο συλλέκτης απορριμμάτων θεωρείται πως μετακινεί ένα γκρι μέτωπο κύματος, που είναι το σύνορο μεταξύ των μαύρων και των λευκών αντικειμένων. Το πρόβλημα με την ταυτόχρονη εκτέλεση τροποποιητή και συλλέκτη είναι πώς οι τελευταίοι ενδέχεται να μην μοιράζονται μια συνεπή εικόνα του σωρού, καθώς το γκρι μέτωπο κύματος δεν είναι πλέον αυστηρό σύνορο μεταξύ μαύρων και λευκών αντικειμένων.

Αν ο τροποποιητής εκτελείται ταυτόχρονα με το συλλέκτη και τροποποιήσει πεδία αντικειμένων που βρίσκονται μπροστά από το μέτωπο κύματος - γκρι αντικείμενα (των οποίων τα πεδία θα πρέπει να σαρωθούν) ή λευκά αντικείμενα (τα οποία δεν έχουν ανακαλυφθεί ακόμη)-τότε δεν υπάρχει πρόβλημα καθώς ο συλλέκτης θα επανεξετάσει τα αντικείμενα αυτά στο μέλλον

(αν είναι ακόμη προσβάσιμα). Επίσης δεν υπάρχει πρόβλημα αν τροποποιεί αντικείμενα πίσω από το μέτωπο κύματος-μαύρα αντικείμενα (των οποίων τα πεδία έχουν ήδη σαρωθεί)-αρκεί να εισάγονται η διαγράφονται δείκτες μόνο προς μαύρα ή γκρι αντικείμενα (για τα οποία ο συλλέκτης έχει ήδη αποφανθεί πώς είναι προσβάσιμα). Όλες οι υπόλοιπες ενημερώσεις δεικτών μπορεί να οδηγήσουν σε μία κατάσταση όπου τροποποιητής και συλλέκτης έχουν διαφορετική εικόνα του σωρού.

Αυτό που σε κάθε περίπτωση πρέπει να αποφευχθεί είναι η εισαγωγή ενός δείκτη προς κάποιο λευκό αντικείμενο στο πεδίο κάποιου μαύρου αντικειμένου, καθώς στην περίπτωση αυτή το πρώτο αντικείμενο, παρότι προσβάσιμο, θα χαθεί (και έτσι θα συλλεχθεί εσφαλμένα).

Ο Wilson [119] αποδεικνύει πως αντικείμενα χάνονται αν κάποια στιγμή κατά την εξιχνίαση του γράφου αντικειμένων ισχύουν ταυτόχρονα οι ακόλουθες δύο συνθήκες:

Συνθήκη 1: ο τροποποιητής αποθηκεύει σε ένα μαύρο αντικείμενο ένα δείκτη προς ένα λευκό αντικείμενο και

Συνθήκη 2: όλα τα μονοπάτια δια μέσου γκρι αντικειμένων προς το λευκό αντικείμενο καταστρέφονται.

9.1.2 Η ασθενής και η ισχυρή τριχρωματική αναλλοίωτη

Για να αποφευχθεί η εσφαλμένη συλλογή ζωντανών αντικειμένων, πρέπει να εξασφαλισθεί πως οι δύο συνθήκες δεν μπορούν να επικρατήσουν ταυτόχρονα. Για να μπορεί να εγγυηθεί ο συλλέκτης πως δε χάνει προσβάσιμα αντικείμενα, πρέπει να βεβαιωθεί πως βρίσκει όλα τα λευκά αντικείμενα προς τα οποία υπάρχει δείκτης σε μαύρα αντικείμενα. Είναι αρκετό για ένα λευκό αντικείμενο να είναι προσβάσιμο από κάποιο γκρι αντικείμενο είτε άμεσα είτε έμμεσα μέσω μιας αλυσίδας λευκών αντικειμένων. Σε αυτήν την περίπτωση η δεύτερη συνθήκη δεν επικρατεί ποτέ. Η ασθενής τριχρωματική αναλλοίωτη που πρέπει να διατηρεί ο συλλέκτης διατυπώνεται ως εξής:

Ασθενής τριχρωματική αναλλοίωτη: Όλα τα λευκά αντικείμενα προς τα οποία υπάρχει δείκτης σε κάποιο μαύρο αντικείμενο είναι προσβάσιμα από κάποιο γκρι αντικείμενο, είτε άμεσα, είτε μέσω μιας αλυσίδας λευκών αντικειμένων.

Οι ταυτόχρονοι συλλέκτες μη-αντιγραφής έχουν το πλεονέκτημα πως όλοι οι λευκοί δείκτες αυτόματα μετατρέπονται σε γκρι/μαύρους όταν το αντικείμενο προς το οποίο δείχνουν σκιάζεται γκρι ή μαύρο αντίστοιχα. Επομένως η παρουσία λευκών δεικτών στο εσωτερικό μαύρων αντικειμένων δεν αποτελεί πρόβλημα, καθώς τα αντικείμενα προς τα οποία δείχνουν, εφόσον τηρείται η ασθενής τριχρωματική αναλλοίωτη θα σκιαστούν τελικώς γκρι ή μαύρα πριν το τέλος του τρέχοντος κύκλου συλλογής.

Αντίθετα, οι ταυτόχρονοι συλλέκτες αντιγραφής είναι πιο περιορισμένοι καθώς διαθέτουν ρητά δύο αντίγραφα για κάθε ζωντανό αντικείμενο στο τέλος κάθε κύκλου συλλογής (ένα λευκό αντίγραφο στο χώρο-από και ένα μαύρο αντίγραφο στο χώρο-προς), όπου τα λευκά αντικείμενα απορρίπτονται μαζί με τα αντικείμενα- απορρίμματα. Εξ ορισμού, ο συλλέκτης δεν επισκέπτεται ξανά μαύρα αντικείμενα. Επομένως ένας ταυτόχρονος συλλέκτης αντιγραφής δεν πρέπει ποτέ να επιτρέψει ένα πεδίο δείκτης ενός μαύρου αντικειμένου του χώρου-προς να αναφέρεται προς ένα λευκό αντικείμενο του χώρου-από. Η ισχυρή τριχρωματική αναλλοίωτη που πρέπει να διατηρεί ένας τέτοιος συλλέκτης διατυπώνεται ως εξής:

Ισχυρή τριχρωματική αναλλοίωτη: Δεν υπάρχουν δείκτες από μαύρα αντικείμενα προς λευκά αντικείμενα.

9.1.3 Χρώμα τροποποιητή

Κατά την κατηγοριοποίηση των αλγορίθμων ταυτόχρονης συλλογής απορριμμάτων συχνά χρειάζεται να αναφερθούμε στο χρώμα των αντικειμένων-ριζών του τροποποιητή αντιμετωπίζοντας τον τροποποιητή σαν αντικείμενο. Ένας τροποποιητής είναι γκρι είτε αν δεν έχει πραγματοποιηθεί ακόμη εξιχνίαση από τις ρίζες του είτε αν οι ρίζες του έχουν σαρωθεί και πρέπει να σαρωθούν εκ νέου. Αυτό σημαίνει πως οι ρίζες ενός γκρι τροποποιητή μπορούν να αναφέρονται σε αντικείμενα όλων των χρωμάτων. Ένας τροποποιητής είναι μαύρος αν έχει πραγματοποιηθεί η εξιχνίαση από τις ρίζες του και αυτές δε θα σαρωθούν ξανά. Υπό την ισχυρή τριχρωματική αναλλοίωτη αυτό σημαίνει πως οι ρίζες ενός μαύρου τροποποιητή μπορούν να αναφέρονται μόνο σε μαύρα ή γκρι αντικείμενα. Υπό την ασθενή τριχρωματική αναλλοίωτη, οι ρίζες ενός μαύρου τροποποιητή μπορούν να αναφέρονται σε λευκά αντικείμενα, αρκεί τα τελευταία να είναι προσβάσιμα από κάποιο γκρι αντικείμενο είτε έμμεσα είτε μέσω μιας αλυσίδας λευκών αντικειμένων.

Το χρώμα του τροποποιητή έχει επιπτώσεις στον τερματισμό ενός κύκλου συλλογής. Εξ ορισμού, αλγόριθμοι ταυτόχρονης συλλογής απορριμμάτων πρέπει να σαρώσουν ξανά τις ρίζες ενός γκρι τροποποιητή. Αυτό ενδέχεται να οδηγήσει σε καινούρια εργασία εξιχνίασης αν βρεθεί ένας όχι μαύρος δείκτης. Όταν η εξιχνίαση αυτή ολοκληρωθεί, οι ρίζες πρέπει να σαρωθούν ξανά καθώς ο τροποποιητής μπορεί και πάλι στο ενδιάμεσο να έχει αποθηκεύσει ένα μη μαύρο δείκτη κ.ό.κ. Στη χειρότερη περίπτωση μπορεί να χρειαστεί να ανασταλεί η εκτέλεση όλων των νημάτων-τροποποιητών ώστε να σαρωθούν για μια τελευταία φορά οι ρίζες τους.

Οι εν-τη-πτήσει συλλέκτες ξεχωρίζουν τα νήματα-τροποποιητές καθώς δεν αναστέλλουν την εκτέλεση όλων ταυτόχρονα για να εξετάσουν τις ρίζες τους. Οι συλλέκτες αυτής της κατηγορίας πρέπει να λειτουργήσουν με νήματα-τροποποιητές διαφορετικών χρωμάτων.

9.1.4 Χρώμα εκχώρησης

Το χρώμα του τροποποιητή επηρεάζει επίσης το χρώμα που λαμβάνει ένα αντικείμενο κατά την εκχώρησή του, καθώς ο ο δείκτης προς αυτό αποθηκεύεται στον τροποποιητή, ο οποίος ανάλογα με το χρώμα του πρέπει να διατηρεί την ασθενή ή ισχυρή τριχρωματική αναλλοίωτη. Το χρώμα εκχώρησης επηρεάζει επίσης το πόσο γρήγορα θα απελευθερωθεί ένα αντικείμενο από τη στιγμή που αυτό καθίσταται μη- προσβάσιμο. Εάν ένα αντικείμενο εκχωρείται ως μαύρο ή γκρι τότε δε θα ελευθερωθεί στη διάρκεια του τρέχοντος κύκλου συλλογής (διότι τα μαύρα και γκρι αντικείμενα θεωρούνται ζωντανά) ακόμη και εάν καταστεί μη-προσβάσιμο. Σε ένα γκρι τροποποιητή τα αντικείμενα μπορούν να εκχωρούνται ως λευκά και έτσι να αποφεύγεται η χωρίς λόγο διατήρηση νέων αντικειμένων. Αντίθετα, σε ένα μαύρο τροποποιητή νέα αντικείμενα δεν μπορούν να εκχωρούνται ως λευκά εκτός και αν διατηρείται η ασθενής τριχρωματική αναλλοίωτη και υπάρχει κάποια εγγύηση πως ο λευκός δείκτης θα αποθηκευθεί σε ένα ζωντανό αντικείμενο που βρίσκεται μπροστά από το μέτωπο κύματος και δε θα συλλεγεί εσφαλμένα. Τέλος, ένα καινούριο αντικείμενο δεν περιέχει (ακόμη) δείκτες και συνεπώς είναι πάντα ασφαλές να εκχωρείται ως μαύρο.

9.2 Τεχνικές φράγματος για ταυτόχρονη συλλογή

Σύμφωνα με τον Pirinen, [93] οι τεχνικές φράγματος που διατηρούν μία εκ των δύο τριχρωματικών αναλλοίωτων βασίζονται σε έναν αριθμό από ενέργειες για να διαχειρισθούν την εισαγωγή και διαγραφή δεικτών. Οι τεχνικές αυτές μπορούν:

- Να αυξήσουν το μέγεθος του μετώπου κύματος, **σκιάζοντας** γκρι ένα αντικείμενο αν αυτό ήταν λευκό. Η σκίαση ενός γκρι ή μαύρου αντικειμένου δεν έχει κανένα αποτέλεσμα.
- Να μετακινήσουν προς τα εμπρός το μέτωπο κύματος **σαρώνοντας** ένα αντικείμενο για να το χρωματίσουν μαύρο.
- Να μετακινήσουν προς τα πίσω το μέτωπο κύματος **αντιστρέφοντας** το χρώμα ενός αντικειμένου από μαύρο σε γκρι.

9.2.1 Τεχνικές γκρι τροποποιητή

Αρχικά παρουσιάζουμε προσεγγίσεις που λειτουργούν με γκρι τροποποιητή. Όλες αυτές οι τεχνικές διατηρούν την ισχυρή αναλλοίωτη χρησιμοποιώντας ένα **φράγμα εισαγωγής** κατά την εγγραφή αναφορών στο σωρό για να αποτρέψουν την αποθήκευση λευκών δεικτών σε μαύρα αντικείμενα. Καθώς ο τροποποιητής είναι γκρι, δεν απαιτείται φράγμα εγγραφής.

- Το φράγμα εγγραφής του Steele, [69] το οποίο φαίνεται στον αλγόριθμο 9.1 παρουσιάζει τη μεγαλύτερη ακρίβεια ανάμεσα σε όλες τις τεχνικές απλώς επειδή σημειώνει το αντικείμενο προέλευσης υπό τροποποίηση. Δε μεταβάλλει καμία απόφαση όσον αφορά την προσβασιμότητα ενός αντικειμένου, αλλά προκαλεί τη μετακίνηση του μετώπου κύματος προς τα πίσω αλλάζοντας το χρώμα του τροποποιημένου μαύρου αντικειμένου από μαύρο σε γκρι. Αναβάλλει την απόφαση σχετικά με την προσβασιμότητα του λευκού αντικειμένου προορισμού μέχρις ότου το μαύρο αντικείμενο προέλευσης σαρωθεί εκ νέου (ο εισαχθείς δείκτης ενδέχεται να έχει διαγραφεί στο ενδιάμεσο). Η ακρίβεια αυτή έρχεται εις βάρος της προόδου, καθώς το μέτωπο κύματος μετακινείται προς τα πίσω.
- Οι Boehm κ.ά. [27] υλοποίησαν μια παραλλαγή του φράγματος εγγραφής του Steele η οποία αγνοεί το χρώμα του εισαγόμενου δείκτη, όπως φαίνεται στον αλγόριθμο 9.1. Αρχικά υλοποίησαν αυτό το φράγμα χρησιμοποιώντας τα βρώμικα bits της εικονικής μνήμης για να καταγράφουν τις τροποποιημένες σελίδες χωρίς να χρειάζεται έτσι οι λειτουργίες εγγραφής πεδίων αντικειμένων του σωρού να παρακολουθούνται σε επίπεδο λογισμικού. Η αρχική υλοποίηση του φράγματος μάλιστα δεν ήλεγχε αν το αντικείμενο προέλευσης είναι μαύρο, καθιστώντας το φράγμα λιγότερο ακριβές. Ο τερματισμός της συλλογής γινόταν με παύση του κόσμου οπότε και οι βρώμικες σελίδες σαρώνονταν εκ νέου.
- Οι Dijkstra κ.ά. [44], [45] σχεδίασαν ένα φράγμα εγγραφής (αλγόριθμος 9.1) λιγότερο ακριβές από το αντίστοιχο του Steele: το αντικείμενο προορισμού του εισαγόμενου δείκτη σκιάζεται ως προσβάσιμο (μη λευκό) παρότι ο δείκτης μπορεί στη συνέχεια να διαγραφεί. Αυτή η απώλεια ακρίβειας συμβάλλει στην πρόοδο καθώς το μέτωπο κύματος μετακινείται προς τα εμπρός. Η αρχική μάλιστα υλοποίηση του φράγματος ήταν ακόμη

λιγότερο ακριβής καθώς το αντικείμενο προορισμού του εισαγόμενου δείκτη σκιαζόταν χωρίς να ελέγχεται αν το χρώμα του αντικειμένου προέλευσης είναι μαύρο. Η παράλειψη αυτού του ελέγχου επιτρέπει τη χαλάρωση της απαίτησης για ατομική εκτέλεση όλου του φράγματος εγγραφής, υπό την προϋπόθεση βέβαια πως οι ξεχωριστές λειτουργίες της αποθήκευσης και της σκίασης εκτελούνται ατομικά.

9.2.2 Τεχνικές μαύρου τροποποιητή

Οι πρώτες δύο προσεγγίσεις εφαρμόζουν αυξητική ενημέρωση για τη διατήρηση της ισχυρής αναλλοίωτης χρησιμοποιώντας ένα φράγμα ανάγνωσης για να αποτρέψουν τον τροποποιητή από την απόκτηση λευκών δεικτών (δηλαδή να αποτρέψουν την εισαγωγή ενός λευκού δείκτη σε ένα μαύρο τροποποιητή). Η τρίτη προσέγγιση χρησιμοποιεί ένα **φράγμα διαγραφής** στις λειτουργίες εγγραφής δεικτών στο σωρό για τη διατήρηση της ασθενούς αναλλοίωτης. Υπό την ασθενή τριχρωματική αναλλοίωτη ένας μαύρος τροποποιητής μπορεί να διατηρεί λευκούς δείκτες: είναι μαύρος καθώς δεν απαιτείται η εκ νέου σάρωση των ριζών του και έτσι μπορεί να φορτώνει τιμές δεικτών προς λευκά αντικείμενα, αφού το φράγμα εγγραφής προστατεύει τα τελευταία από μια πιθανή διαγραφή.

- Ο Baker [70] χρησιμοποίησε το φράγμα ανάγνωσης που φαίνεται στον αλγόριθμο 9.2. Η προσέγγισή του είναι λιγότερο ακριβής από την προσέγγιση των Dijkstra κ.ά., καθώς εσφαλμένα διατηρεί αντικείμενα (που θα ήταν λευκά) απλώς επειδή κάποιο νήμα- τροποποιητής φορτώνει τις τιμές δεικτών προς αυτά κατά τη διάρκεια ενός κύκλου συλλογής (σε αντίθεση με λευκά αντικείμενα που εισάγονται πίσω από το μέτωπο κύματος και τα οποία πρέπει να διατηρηθούν). Το φράγμα ανάγνωσης του Baker σχεδιάσθηκε αρχικώς για ένα συλλέκτη αντιγραφής, όπου η ενέργεια της σκίασης αντιγράφει ένα αντικείμενο από το χώρο-από στο χώρο-προς και έτσι η ρουτίνα SHADE επιστρέφει τη διεύθυνση του αντιγράφου του αντικειμένου στο χώρο-προς.
- Οι Appel κ.ά. [5] υλοποίησαν μία λιγότερο ακριβή παραλλαγή του φράγματος ανάγνωσης του Baker, (αλγόριθμος 9.2) χρησιμοποιώντας μηχανισμούς προστασίας σελίδων εικονικής μνήμης για την παγίδευση των προσβάσεων νημάτων-τροποποιητών σε γκρι σελίδες με αποτέλεσμα να μη χρειάζεται η παρακολούθηση των λειτουργιών ανάγνωσης σε επίπεδο λογισμικού. Μια παγιδευμένη πρόσβαση μπορεί να ολοκληρωθεί μετά τη σάρωση της αντίστοιχης σελίδας (και την άρση του αποκλεισμού της). Αυτό το φράγμα ανάγνωσης μπορεί επίσης να χρησιμοποιηθεί και από ένα ταυτόχρονο συλλέκτη αντιγραφής αφού η σάρωση θα προωθήσει τυχόν πεδία-δείκτες προς αντικείμενα του χώρου-προς του αντικειμένου προέλευσης.
- Οι Abraham και Patel [1], και ο Yuasa [121] σχεδίασαν ανεξάρτητα το φράγμα διαγραφής που φαίνεται στον αλγόριθμο 9.2. Αυτό το φράγμα διαγραφής παρουσιάζει τη μικρότερη ακρίβεια από όλες τις τεχνικές καθώς διατηρεί κάθε μη προσβάσιμο αντικείμενο προς το οποίο ο τελευταίος δείκτης διαγράφηκε κατά τη διάρκεια του κύκλου συλλογής.

9.3 Ταυτόχρονη σήμανση και εκκαθάριση

Στην ενότητα αυτή εξετάζουμε την **ταυτόχρονη συλλογή με σήμανση και εκκαθάριση**. Όπως είδαμε και πριν, το πιο σημαντικό θέμα όσον αφορά έναν ταυτόχρονο συλλέκτη

Αλγόριθμος 9.1 Ταυτόχρονη συλλογή: φράγματα γκρι τροποποιητή

(9.1.α΄) Φράγμα εγγραφής του Steele

```
1: procedure WRITE(src, i, ref)
2:     atomic
3:     src[i] ← ref
4:     if ISBLACK(src) then
5:         if ISWHITE(ref) then
6:             REVERT(src)
```

(9.1.β΄) Φράγμα εγγραφής του Boehm κ.ά

```
1: procedure WRITE(src, i, ref)
2:     atomic
3:     src[i] ← ref
4:     if ISBLACK(src) then
5:         REVERT(src)
```

(9.1.γ΄) Φράγμα εγγραφής του Dijkstra κ.ά

```
1: procedure WRITE(src, i, ref)
2:     atomic
3:     src[i] ← ref
4:     if ISBLACK(src) then
5:         SHADE(src)
```

Αλγόριθμος 9.2 Ταυτόχρονη συλλογή: φράγματα μαύρου τροποποιητή

(9.2.α′) Φράγμα ανάγνωσης του Baker

```
1: function Read(src, i)
2:     atomic
3:     ref ← src[i]
4:     if isGrey(src) then
5:         ref ← shade(src)
6:     return ref
```

(9.2.β′) Φράγμα ανάγνωσης του Appel κ.ά

```
1: function Read(src, i)
2:     atomic
3:     src[i] ← ref
4:     if isGrey(src) then
5:         scan(src)
6:     return src[i]
```

(9.2.γ′) Φράγμα εγγραφής των Abraham κ.ά / Yuasa

```
1: procedure Write(src, i, ref)
2:     atomic
3:     if isGrey(src) or isWhite(src) then
4:         shade(src[i])
5:     src[i] ← ref
```

είναι η εξασφάλιση της ορθότητας. Συλλέκτης και τροποποιητής οφείλουν να συνεργάζονται προκειμένου να διαβεβαιώσουν πως μοιράζονται μια συνεπή εικόνα της μνήμης του σωρού. Ο τροποποιητής από την πλευρά του οφείλει να αποτρέπει τη μη ορατότητα ζωντανών αντικειμένων στο συλλέκτη, ενώ ένας συλλέκτης που μετακινεί αντικείμενα οφείλει να εξασφαλίζει πως ο τροποποιητής χρησιμοποιεί τις σωστές διευθύνσεις των μετακινηθέντων αντικειμένων.

Οι ταυτόχρονοι συλλέκτες σήμανσης και εκκαθάρισης είναι οι απλούστεροι από όλους τους ταυτόχρονους συλλέκτες. Καθώς δε μεταβάλλουν πεδία-δείκτες, ο τροποποιητής μπορεί ελεύθερα να διαβάζει τις τιμές δεικτών χωρίς να χρειάζεται να προστατευθεί από το συλλέκτη. Δεν υπάρχει επομένως εγγενής ανάγκη για φράγμα ανάγνωσης όταν χρησιμοποιείται μη-μετακινών συλλέκτης. Ειδάλλως, η χρήση φράγματος ανάγνωσης για τη διατήρηση της ισχυρής αναλλοίωτης σε ένα σύστημα με συλλέκτη που δεν μετακινεί αντικείμενα θεωρείται πολύ ακριβή, καθώς οι αναγνώσεις δεδομένων του σωρού από τον τροποποιητή είναι πολύ συχνότερες από ότι οι εγγραφές. Για παράδειγμα, ο Zorn [124] μέτρησε και υπολόγισε το ποσοστό των φορτώσεων και αποθηκεύσεων δεικτών στο σύστημα SPUR LISP από 13% έως 15% και 4% αντίστοιχα. Η εξαίρεση στον παραπάνω γενικό κανόνα προκύπτει όταν τεχνικές βελτιστοποιήσεων του μεταγλωττιστή χρησιμοποιούνται για να εξαλείψουν τα περιττά φράγματα, όπως από τους Hosking κ.ά. [61] και τους Zee και Rinard [122], ή για να ενσωματώσουν ένα τμήμα του έργου των φραγμάτων στο ήδη υπάρχον κόστος που αφορά τον έλεγχο των δεικτών έναντι της ειδικής τιμής **null**, όπως από τους Bacon κ.ά. [9]. Γι αυτό το λόγο οι ταυτόχρονοι συλλέκτες σήμανσης και εκκαθάρισης συνήθως χρησιμοποιούν το φράγμα αυξητικής ενημέρωσης των Dijkstra κ.ά. [44], [45], είτε το φράγμα εγγραφής εισαγωγής του Steele [69], είτε το φράγμα εγγραφής εισαγωγής των Boehm κ.ά. [27], είτε το φράγμα εγγραφής διαγραφής του Yuasa [121].

9.3.1 Αρχικοποίηση

Αντί να επιτρέπεται η εκτέλεση του τροποποιητή μέχρις ότου η μνήμη εξαντληθεί, οι ταυτόχρονοι συλλέκτες μπορούν να εκτελούνται ακόμη και όταν ο τροποποιητής μπορεί να δεσμεύει μνήμη για τη δημιουργία αντικειμένων. Αν μία συλλογή ενεργοποιηθεί πολύ αργά, ενδέχεται να μην υπάρχει επαρκής μνήμη για την ικανοποίηση ενός αιτήματος εκχώρησης, στην οποία περίπτωση θα καθυστερήσει η εκτέλεση του τροποποιητή μέχρις ότου τελειώσει ο κύκλος συλλογής. Τη στιγμή που εκκινεί ο κύκλος συλλογής, η ρυθμαπόδοση σταθερής κατάστασης του συλλέκτη πρέπει να επαρκεί για την ολοκλήρωση του κύκλου πριν εξαντληθεί η μνήμη από τον τροποποιητή και να επιδρά στη ρυθμαπόδοση του τελευταίου το ελάχιστο δυνατό. Το πότε και πώς θα ενεργοποιηθεί ένας κύκλος συλλογής εξασφαλίζοντας τη διαθεσιμότητα επαρκούς μνήμης για την ικανοποίηση των αιτημάτων του τροποποιητή και ενώ εκτελείται ο συλλέκτης, καθώς και το πότε ο κύκλος θα τελειώσει, ούτως ώστε η μνήμη από τα απορρίμματα να ανακτηθεί, εξαρτώνται από τη δρομολόγηση εργασιών συλλογής παράλληλα με την εκτέλεση του τροποποιητή.

Ο αλγόριθμος 9.3 απεικονίζει την αλληλουχία ενεργειών κατά την εκχώρηση μνήμης στον τροποποιητή για έναν ταυτόχρονο συλλέκτη σήμανσης και εκκαθάρισης ο οποίος δρομολογεί ένα ποσό έργου συλλογής αυξητικά σε κάθε εκχώρηση μνήμης μέσω της διαδικασίας COLLECTENOUGH. Η συνάρτηση BEHIND αποφασίζει σχετικά με το πότε και πόσο έργο συλλογής πρέπει να εκτελεστεί, εξασφαλίζοντας πως ο τροποποιητής δεν βρίσκεται πολύ μπροστά από το συλλέκτη, ώστε η συνάρτηση ALLOCATE να μην μπορεί να ικανοποιήσει αιτήματα μνήμης.

Ο αλγόριθμος 9.4 δείχνει τι ακριβώς συμβαίνει όταν δρομολογείται η εκτέλεση έργου συλ-

λογής. Η αρχικά κενή λίστα εργασιών γεμίζει με αναφορές προς αντικείμενα προς τα οποία δείχνουν οι ρίζες. Υποθέτοντας πως η εξέταση των ριζών σημαίνει την αναστολή λειτουργίας και εξέταση όλων των νημάτων-τροποποιητών, σε αυτό το σημείο κανένα νήμα-τροποποιητής δεν περιλαμβάνει κάποια λευκή αναφορά. Επομένως έχουμε ένα σχεδόν-ταυτόχρονο τρόπο λειτουργίας, με μία μικρή φάση παύσης του κόσμου για την αρχικοποίηση της συλλογής. Τα γκρι πλέον αντικείμενα- ρίζες αναπαριστούν το αρχικό μέτωπο κύματος από το οποίο θα συνεχίσει η εξερεύνηση του γράφου των αντικειμένων. Εφόσον οι ρίζες τους έχουν εξετασθεί, τα νήματα-τροποποιητές μπορούν πλέον να συνεχίσουν είτε ως μαύρα (εφόσον δεν έχουν λευκές αναφορές) είτε ως γκρι, ανάλογα με τα φράγματα τροποποίησης που είναι σε χρήση.

Αλγόριθμος 9.3 Ταυτόχρονη συλλογή: εκχώρηση για σχεδόν-ταυτόχρονη σήμανση και εκκαθάριση

```
 1: function New()
 2:     collectEnough()
 3:     ref ← allocate()                ▷ must initialize black if collector is black
 4:     if ref = null then
 5:         error "Out of memory"
 6:     return ref

 7: function collectEnough()
 8:     atomic
 9:     while behind() do
10:         if not markSome() then
11:             return null
```

Η αναστολή της λειτουργίας των νημάτων-τροποποιητών ενδέχεται να οδηγήσει σε απαράδεκτες παύσεις. Η χρήση γκρι φραγμάτων τροποποίησης καθιστά εφικτή την ενεργοποίηση των φραγμάτων και την αναβολή της εξέτασης όλων των ριζών, η οποία πραγματοποιείται αργότερα και ενώ ταυτόχρονα εκτελούνται τα νήματα-τροποποιητές.

9.3.2 Τερματισμός σήμανσης

Ο τερματισμός της φάσης σήμανσης για ένα μαύρο τροποποιητή είναι μία σχετικά απλή διαδικασία και συμβαίνει όταν δεν υπάρχουν πλέον γκρι αντικείμενα στη λίστα εργασιών. Σε αυτό το σημείο, θεωρώντας ακόμη και την ασθενή τριχρωματική αναλλοίωτη ο τροποποιητής μπορεί να περιλαμβάνει μόνο μαύρες αναφορές, καθώς δεν υπάρχουν λευκά αντικείμενα προσβάσιμα από γκρι αντικείμενα (δεν υπάρχουν καθόλου γκρι αντικείμενα). Καθώς ο τροποποιητής είναι μαύρος, δεν απαιτείται η επανεξέταση των ριζών του.

Ο τερματισμός της φάσης σήμανσης για έναν γκρι τροποποιητή είναι λίγο πιο περίπλοκος, καθώς ο τροποποιητής μπορεί να έχει αποκτήσει λευκούς δείκτες από τη στιγμή που εξετάστηκαν οι ρίζες του για την αρχικοποίηση της συλλογής. Επομένως οι ρίζες ενός γκρι τροποποιητή πρέπει να επανεξετασθούν πριν η φάση σήμανσης τερματισθεί. Εάν η επανεξέταση δεν ανακαλύψει φρέσκα γκρι αντικείμενα, τότε η φάση της σήμανσης τερματίζεται και εκκινεί η φάση της εκκαθάρισης. Η τελευταία μπορεί να είναι πρόθυμη ή οκνηρή.

Αλγόριθμος 9.4 Ταυτόχρονη συλλογή: σχεδόν-ταυτόχρονη σήμανση

```
1:  shared worklist ← empty

2:  function markSome()
3:      if isEmpty(worklist) then                          ▷ initiate collection
4:          scan(Roots)                ▷ invariant: collector holds no white references
5:          if isEmpty(worklist) then          ▷ invariant: no more grey references
6:              /* marking terminates */
7:              sweep()                                    ▷ lazy or eager sweep
8:              return false                               ▷ terminate marking
9:      /* collection continues */
10:     ref ← remove(worklist)
11:     scan(ref)                                ▷ continue marking, if still behind
12:     return true

13: procedure shade(ref)
14:     if not isMarked() then
15:         setMarked(ref)
16:         add(worklist, ref)

17: procedure scan(ref)
18:     for all fld in Pointers(ref) do
19:         child ← *fld
20:         if child ≠ null then
21:             shade(child)

22: procedure revert(ref)
23:     add(worklist, ref)

24: function isWhite(ref)
25:     return not isMarked(ref)

26: function isGrey(ref)
27:     return refinworklist

28: function isBlack(ref)
29:     return isMarked(ref) and not isGrey(ref)
```

9.3.3 Ταυτόχρονη σήμανση και ταυτόχρονη εκκαθάριση

Μέχρι στιγμής έχουμε θεωρήσει την εκτέλεση μόνο της φάσης της σήμανσης ταυτόχρονα με τον τροποποιητή και πως η φάση της εκκαθάρισης έπεται αυτής σειριακά. Εάν η εκκαθάριση είναι οκνηρή, αιτήματα εκχώρησης μνήμης από τον τροποποιητή ενδέχεται να ενεργοποιούν την ταυτόχρονη εκκαθάριση που αντιστοιχεί στη σήμανση του προηγούμενου κύκλου συλλογής και ενώ ήδη ένας νέος κύκλος συλλογής βρίσκεται στην επόμενη φάση σήμανσης. Το γεγονός αυτό μπορεί να προκαλέσει σύγχυση όσον αφορά τα χρώματα των αντικειμένων. Θα πρέπει με κάποιο τρόπο να μπορούν να διαχωρισθούν τα πραγματικά λευκά απορρίμματα από την προηγούμενη φάση σήμανσης (τα οποία πρέπει να εκκαθαρισθούν) από τα μέχρι μη σημασμένα λευκά αντικείμενα της τρέχουσας φάσης σήμανσης. Ο Lamport [75] ξεχωρίζει τα δύο είδη αντικειμένων χρησιμοποιώντας ένα καινούριο χρώμα, το μωβ. Όταν ολοκληρώνεται η φάση σήμανσης, όλα τα λευκά αντικείμενα-απορρίμματα χρωματίζονται μωβ. Η εκκαθάριση θα συλλέξει τα μωβ αντικείμενα, προσθέτοντάς τα στην ελεύθερη λίστα και επαναχρωματίζοντάς σε μαύρα ή λευκά, ανάλογα με το χρώμα του εκχωρητή.

Ο Lamport χρησιμοποιεί πολλαπλά ταυτόχρονα νήματα-σημαντές και νήματα- εκκαθαριστές και ορίζει έναν κύκλο συλλογής ως εξής:

1. Περίμενε μέχρις ότου όλα τα νήματα-σημαντές και όλα τα νήματα- εκκαθαριστές ολοκληρώσουν τις εργασίες τους.
2. Άλλαξε το χρώμα των λευκών αντικειμένων σε μωβ και των μαύρων αντικειμένων σε λευκό (για την αποφυγή αιωρούμενων απορριμμάτων) ή γκρι (στην περίπτωση που το αντικείμενο έχει σκιασθεί ταυτόχρονα από το φράγμα εγγραφής ενός νήματος-τροποποιητή).
3. Σκίασε όλες τις ρίζες.
4. Εκκίνησε όλα τα νήματα-σημαντές και όλα τα νήματα-εκκαθαριστές.

Η σήμανση αγνοεί όλα τα μωβ αντικείμενα: ένα νήμα-τροποποιητής δεν μπορεί ποτέ να αποκτήσει μια αναφορά προς ένα μωβ αντικείμενο και συνεπώς τα γκρι αντικείμενα δε δείχνουν ποτέ σε μωβ αντικείμενα και τα μωβ αντικείμενα δε σκιάζονται ποτέ. Το πρόβλημα με αυτή την προσέγγιση είναι πώς για να εκκινήσει η εκκαθάριση πρέπει να έχει αλλαχθεί το χρώμα από λευκό σε μωβ όλων των αντικειμένων-απορριμμάτων του προηγούμενου κύκλου. Παρόμοια, κατά την έναρξη της σήμανσης, πρέπει το χρώμα όλων των μαύρων αντικειμένων του προηγούμενου κύκλου να αλλαχθεί σε λευκό ή γκρι.

9.3.4 Εν-τη-πτήσει σήμανση

Μέχρι στιγμής έχουμε υποθέσει πως, είτε πρόκειται για την εκκίνηση είτε για τον τερματισμό της φάσης σήμανσης, αναστέλλεται η εκτέλεση όλων των νημάτων- τροποποιητών ταυτόχρονα για να σαρωθούν οι ρίζες τους. Στη συνέχεια, κάθε νήμα- τροποποιητής συνεχίζει να εκτελείται είτε ως μαύρο ή γκρι ανάλογα με το φράγμα εγγραφής που χρησιμοποιεί. Αυτές οι παύσεις του κόσμου μειώνουν τον ταυτοχρονισμό. Μια εναλλακτική προσέγγιση είναι οι ρίζες κάθε νήματος- τροποποιητή να σαρώνονται ξεχωριστά και ταυτόχρονα με την εκτέλεση των υπόλοιπων νημάτων-τροποποιητών. Η διαφορετική ταυτόχρονη εκτέλεση ορισμένων νημάτων-τροποποιητών ως γκρι και ορισμένων άλλων ως μαύρων εισάγει επιπλέον πολυπλοκότητα και επηρεάζει επίσης τον εντοπισμό του τερματισμού της φάσης σήμανσης.

Η εν-τη-πτήσει συλλογή ποτέ δεν αναστέλλει την εκτέλεση όλων των νημάτων- τροποποιητών ταυτόχρονα, αλλά εμπλέκει κάθε ένα από αυτά σε μία σειρά **χειραψιών** που δεν απαιτούν καθολικό συγχρονισμό. Ο συλλέκτης παρακινεί ασύγχρονα όλα τα νήματα-τροποποιητές, ένα προς ένα, να σταματήσουν την εκτέλεσή τους σε κάποιο βολικό χρονικό σημείο. Ο συλλέκτης μπορεί στη συνέχεια να εξετάσει (και πιθανώς τροποποιήσει) τις ρίζες κάθε νήματος-τροποποιητή πριν του επιτρέψει τη συνέχιση της εκτέλεσής του. Η προσέγγιση αυτή επιτρέπει την ταυτόχρονη εκτέλεση των υπόλοιπων νημάτων-τροποποιητών κατά την παύση της εκτέλεσης ενός νήματος-τροποποιητή.

Η χρήση χειραψιών εισήχθη για πρώτη φορά από τους Doligez και Leroy [48] και Doligez και Gonthier [47] σε ένα συλλέκτη σήμανσης και εκκαθάρισης για τη γλώσσα προγραμματισμού ML.

9.4 Ταυτόχρονη Αντιγραφή

Σε αυτήν και την επόμενη ενότητα εξετάζουμε πώς μπορεί να ελαχιστοποιηθεί ο κατακερματισμός του σωρού αντιγράφοντας ζωντανά αντικείμενα από το χώρο-από στο χώρο-προς ταυτόχρονα με την εκτέλεση του τροποποιητή.

Η ορθότητα της συλλογής απορριμμάτων με ταυτόχρονη αντιγραφή απαιτεί να προστατευθούν τόσο τα νήματα-συλλέκτες από τις ενέργειες του τροποποιητή, όσο και ο τροποποιητής από τις ενέργειες των νημάτων-συλλεκτών. Επιπλέον, οι ενημερωμένες τιμές δεικτών που γράφει ο τροποποιητής θα πρέπει να διαδοθούν στα αντικείμενα-αντίγραφα που κατασκευάζουν ταυτόχρονα στον χώρο-προς τα νήματα- συλλέκτες.

Κατά την ταυτόχρονη συλλογή με αντιγραφή, ένας μαύρος τροποποιητής πρέπει εξ ορισμού να διαθέτει μόνο δείκτες προς αντικείμενα του χώρου-προς (αν διέθετε δείκτες προς αντικείμενα του χώρου-από, τότε ο συλλέκτης δε θα τα επισκεπτόταν ούτε θα τα προωθούσε, με αποτέλεσμα την παραβίαση της ορθότητας). Το γεγονός αυτό είναι γνωστό και ως **αναλλοίωτη του χώρου-προς** του μαύρου τροποποιητή: ο τροποποιητής λειτουργεί πάντοτε μπροστά από το μέτωπο κύματος στο χώρο-προς. Όμοια, ένας γκρι τροποποιητής πρέπει εξ ορισμού να διαθέτει μόνο δείκτες προς αντικείμενα του χώρου-από στην αρχή ενός κύκλου συλλογής. Αν απουσιάζει ένα φράγμα εγγραφής που θα προωθήσει ένα δείκτη προς ένα αντικείμενο του χώρου-από στο αντίγραφο του στο χώρο-προς, ο γκρι τροποποιητής δεν μπορεί να αποκτήσει άμεσα δείκτες προς αντικείμενα του χώρου-προς από πεδία αντικειμένων του χώρου-από (καθώς ο συλλέκτης αντιγραφής δεν προωθεί δείκτες στο εσωτερικό αντικειμένων του χώρου-από). Το γεγονός αυτό είναι γνωστό και ως **αναλλοίωτη του χώρου-από** του γκρι τροποποιητή.

Φυσικά για να τερματισθεί η εκτέλεση ενός ταυτόχρονου συλλέκτη αντιγραφής πρέπει όλοι οι δείκτες των νημάτων-τροποποιητών να αναφέρονται μόνο σε αντικείμενα του χώρου-προς. Συνεπώς ένας ταυτόχρονος συλλέκτης αντιγραφής που επιτρέπει σε γκρι νήματα-τροποποιητές να επενεργούν στο χώρο-από πρέπει τελικώς να προωθήσει τις ρίζες τους. Επιπλέον, ενημερώσεις στο χώρο-από από τα νήματα-τροποποιητές πρέπει να αντικατοπρισθούν στο χώρο-προς, ειδάλλως θα χαθούν.

9.4.1 Ο αλγόριθμος του Baker

Η διατήρηση της αναλλοίωτης του χώρου-προς από όλα τα νήματα-τροποποιητές αποτελεί ίσως την απλούστερη προσέγγιση για την επίτευξη ταυτόχρονης συλλογής με αντιγραφή καθώς με τον τρόπο αυτό εξασφαλίζεται πως τα νήματα-τροποποιητές ποτέ δε βλέπουν αντικείμενα τα οποία ο συλλέκτης δεν έχει αντιγράψει ακόμα ή βρίσκεται στο μέσο της διαδικασίας αντιγραφής τους. Η διατήρηση της αναλλοίωτης του χώρου-προς σε ένα σχεδόν-ταυτόχρονο κόσμο απαιτεί την (ατομική) αναστολή της εκτέλεσης όλων των νημάτων-τροποποιητών στην αρχή του κύκλου συλλογής ώστε να ληφθούν και προωθηθούν οι ρίζες τους (αντιγράφοντας τα αντικείμενα προς τα οποία αναφέρονται). Τα πλέον μαύρα νήματα-τροποποιητές περιέχουν μόνο γκρι δείκτες προς αντικείμενα του χώρου-προς, αλλά τα τελευταία δεν έχουν σαρωθεί ακόμη και περιέχουν ακόμη δείκτες προς αντικείμενα του χώρου-από. Το φράγμα ανάγνωσης μαύρου τροποποιητή του Baker [70] σχεδιάσθηκε αρχικά ώστε η αυξητική συλλογή να προφυλαχθεί έναντι στην πιθανή απόκτηση ενός τέτοιου δείκτη από τον τροποποιητή και στη συνέχεια επεκτάθηκε από τον Halstead [72] για την υποστήριξη ταυτόχρονης συλλογής με αντιγραφή. Το φράγμα ανάγνωσης δημιουργεί στα νήματα-τροποποιητές την ψευδαίσθηση πως ο κύκλος συλλογής έχει ολοκληρωθεί, αποτρέποντάς τα από το να διασχίσουν το μέτωπο κύματος του συλλέκτη μεταξύ του χώρου-προς και του χώρου- από. Η σχεδόν ταυτόχρονη αντιγραφή κατά Baker απεικονίζεται στον αλγόριθμο 9.5 ως μία διασκευή του αλγορίθμου 4.2. Παρατηρούμε πως το φράγμα ανάγνωσης πυροδοτείται μόνο κατά τη φόρτωση της τιμής ενός δείκτη που περιέχεται σε ένα γκρι αντικείμενο του χώρου-προς. Μόνο τότε απαιτείται η κλήση της συνάρτησης FORWARD ώστε να εξασφαλισθεί πώς η τιμή του δείκτη που φορτώνεται αντιστοιχεί σε αντικείμενο του χώρου-προς. Ο συγχρονισμός μεταξύ συλλέκτη και τροποποιητή πραγματοποιείται σε επίπεδο μπλοκ: ο συλλέκτης σαρώνει ατομικά το επόμενο γκρι αντικείμενο, ενώ ο τροποποιητής ατομικά προωθεί μια αναφορά που φορτώθηκε από ένα γκρι αντικείμενο. Η ατομικότητα εκτέλεσης των παραπάνω ενεργειών εξασφαλίζει πώς ένα νήμα-τροποποιητής δεν μπορεί ποτέ να φορτώσει μια αναφορά από ένα αντικείμενο που εκείνη την ώρα σαρώνεται από το συλλέκτη (και να το χρωματίσει από γκρι σε μαύρο).

Η ατομικότητα του φράγματος εγγραφής READ εξασφαλίζει πως ο τροποποιητής βλέπει τη σωστή κατάσταση τόσο του αντικειμένου src (είτε αυτό είναι γκρι είτε όχι) όσο και του αντικειμένου αναφοράς ($src[i]$) και ταυτόχρονα επιτρέπει στον τροποποιητή να αντιγράψει το αντικείμενο αναφοράς αν αυτό είναι στο χώρο-από χωρίς να παρεμβαίνει στη δραστηριότητα σάρωσης αντιγραφής αντικειμένων του συλλέκτη.

Αλγόριθμος 9.5 Ταυτόχρονη συλλογή: σχεδόν-ταυτόχρονη αντιγραφή (Baker)

```
1: shared worklist ← empty
2: procedure COLLECT()
3:     atomic
4:     FLIP()
5:     for all fld in Roots do PROCESS(fld)
6:     loop
7:         atomic
8:         if ISEMPTY(worklist) then
9:             break
10:        ref ← REMOVE(worklist)
11:        SCAN(ref)

12: procedure FLIP()
13:    fromspace, tospace ← tospace, fromspace
14:    free, top ← tospace, tospace + extent

15: procedure SCAN(ref)
16:    for all fld in Pointers(ref) do
17:        PROCESS(fld)

18: procedure FORWARD(fromRef)
19:    toRef ← FORWARDINGADDRESS(fromRef)
20:    if toRef = null then
21:        toRef ← COPY(fromRef)
22:    return toRef

23: function COPY(fromRef)
24:    toRef ← free
25:    free ← free+ SIZE(fromRef)
26:    if free > top then
27:        error "Out of memory!"
28:    MOVE(fromRef, toRef)
29:    FORWARDINGADDRESS(fromRef) ← toRef
30:    ADD(worklist, toRef)
31:    return toRef

32: function READ(src, i)
33:    atomic
34:    ref ← src[i]
35:    if ISGREY(src) then
36:        ref ← FORWARD(ref)
37:    return ref
```

9.4.2 Τα έμμεσα φράγματα του Brooks

Μία εναλλακτική τεχνική που δεν απαιτεί τη διατήρηση της αναλλοίωτης του χώρου-προς επιτρέπει την πρόοδο του τροποποιητή χωρίς ο τελευταίος να ανησυχεί για το μέτωπο κύματος. Ο Brooks [29] παρατηρεί πως, αν κάθε αντικείμενο (είτε του χώρου-από είτε του χώρου-προς) περιλαμβάνει ένα μη μηδενικό δείκτη προώθησης (είτε προς το αντίγραφό στο χώρο-από είτε προς το αντίγραφο στο χώρο-προς), τότε ο έλεγχος του αντικειμένου προέλευσης *src* στο φράγμα ανάγνωσης μπορεί να παραλειφθεί. Ένα αντικείμενο του χώρου-από το οποίο δεν έχει ακόμη αντιγραφεί θα περιέχει ένα έμμεσο πεδίο που δείχνει προς τον εαυτό του. Κατά την αντιγραφή ενός αντικειμένου, το έμμεσο πεδίο του αντιγράφου του στο χώρο-προς ενημερώνεται ατομικά ώστε να αναφέρεται στο αντίγραφο του αντικειμένου στο χώρο-προς. Το έμμεσο πεδίο του αντιγράφου του αντικειμένου στο χώρο-προς περιλαμβάνει μια αναφορά προς τον εαυτό του. Όλες οι προσβάσεις στο σωρό, είτε αφορούν την ανάγνωση ή εγγραφή πεδίων δεικτών ή μη απαιτούν πλέον μία άνευ όρων λειτουργία αποδεικτοδότησης για να ακολουθήσουν έναν έμμεσο δείκτη προς το αντίγραφο του ενός αντικειμένου στο χώρο-προς (αν αυτό υπάρχει). Το φράγμα Read του τροποποιητή κατά τον Brooks παίρνει τη μορφή που φαίνεται στον αλγόριθμο 9.6.

Το μόνο πρόβλημα είναι πως πλέον ο τροποποποιητής μέσω του φράγματος ανάγνωσης μπορεί να διαβάσει την τιμή ενός δείκτη που αναφέρεται σε ένα αντικείμενο του χώρου-από που δεν έχει ακόμη αντιγραφεί και βρίσκεται μπροστά από το μέτωπο κύματος. Η παρουσία του έμμεσου δείκτη προώθησης επιτρέπει στον τροποποιητή να εκτελείται ως γκρι και συνεπώς να μπορεί να διατηρεί αναφορές προς αντικείμενα του χώρου-από. Για να αποτρέψει την εισαγωγή δεικτών προς αντικείμενα του χώρου- από πίσω από το μέτωπο κύματος, ο Brooks χρησιμοποιεί ένα φράγμα εγγραφής κατά Dijkstra, όπως φαίνεται στον αλγόριθμο 9.6.

Επειδή πλέον τα νήματα-τροποποιητές εκτελούνται ως γκρι, κατά τον τερματισμό της αντιγραφής απαιτείται η εκ νέου σάρωση των στοιβών τους προκειμένου να αντικαταστηθούν τυχόν μη προωθημένες αναφορές.

Αλγόριθμος 9.6 Ταυτόχρονη συλλογή: τα έμμεσα φράγματα του Brooks

```
1: function Read(src, i)
2:     src ← forwardingAddress(src)
3:     return src[i]

4: procedure Write(src, i, ref)
5:     src ← forwardingAddress(src)
6:     if isBlack(src) then                ▷ src is behind wavefront in tospace
7:         ref ← forward(ref)
8:     src[i] ← ref
```

9.5 Ταυτόχρονη Συμπύκνωση

Στο κεφάλαιο 3 εξετάσαμε τεχνικές συλλογής απορριμμάτων με σήμανση και συμπύκνωση. Κοινό χαρακτηριστικό τους αποτελεί η αποσύζευξη των φάσεων σήμανσης και συμπύκνωσης. Η αποσύζευξη αυτή έχει ως αποτέλεσμα ένας συλλέκτης ταυτόχρονης σήμανσης και συμπύκνωσης να έχει μεγαλύτερη ελευθερία από ότι ένας συλλέκτης ταυτόχρονης αντιγραφής ως

προς τη σειρά μετακίνησης των ζωντανών αντικειμένων. Η μετακίνηση μπορεί για παράδειγμα να πραγματοποιηθεί λαμβάνοντας υπόψη τις διευθύνσεις των αντικειμένων και όχι απλώς με τη σειρά που αυτά ανακαλύπτονται από την εξιχνίαση του γράφου.

9.5.1 Ο αλγόριθμος Compressor

Ο αλγόριθμος Compressor εκμεταλλεύεται την ελευθερία που προκύπτει από την αποσύζευξη των φάσεων της σήμανσης και της συμπύκνωσης και επιτρέπει στα νήματα-συμπυκνωτές να εκτελεσθούν ταυτόχρονα με τα νήματα-τροποποιητές.

Υπενθυμίζουμε πως ο αλγόριθμος Compressor πρώτα υπολογίζει ένα βοηθητικό διάνυσμα πρώτου-αντικειμένου που απεικονίζει μία σελίδα του χώρου-προς στο πρώτο αντικείμενο του χώρου-από που θα μετακινηθεί σε αυτή. Τα παράλληλα νήματα- συμπυκνωτές τίθενται σε ανταγωνισμό για την απόκτηση μιας μη απεικονισμένης εικονικής σελίδας του χώρου-προς, την απεικόνιση της σε μία φυσική σελίδα, την εγκατάσταση σε αυτήν αντιγράφων αντικειμένων του χώρου-από και την ανακατεύθυνση των πεδίων δεικτών των αντικειμένων αντιγράφων προς το χώρο-προς. Όταν όλα τα αντικείμενα μιας εικονικής σελίδας του χώρου-από έχουν αντιγραφεί, η απεικόνιση της τελευταίας σε κάποια φυσική σελίδα αφαιρείται αμέσως.

Για να επιτύχει ταυτόχρονη συμπύκνωση, ο αλγόριθμος Compressor εκμεταλλεύεται τους μηχανισμούς προστασίας των εικονικών σελίδων μνήμης όπως και στην προσέγγιση των Appel κ.ά. [5], όπου η προστασία υπηρετούσε ως το φράγμα ανάγνωσης για ταυτόχρονη συλλογή με αντιγραφή, που αποτρέπει τον τροποποιητή από την προσπέλαση σελίδων του χώρου-προς, των οποίων τα αντικείμενα δεν έχουν ακόμη αντιγραφεί ή περιέχουν μη προωθημένους δείκτες. Οι Ossia κ.ά. [86] επίσης χρησιμοποίησαν τους μηχανισμούς προστασίας για να επιτρέψουν απλώς την ταυτόχρονη προώθηση δεικτών σε σελίδες που περιείχαν ζωντανά αντικείμενα μεταφερθέντα από τη φάση συμπύκνωσης. Ο αλγόριθμος Compressor χρησιμοποιεί τους μηχανισμούς προστασίας για να καθοδηγήσει τόσο τη συμπύκνωση όσο και την προώθηση δεικτών. Ο αλγόριθμος Compressor προστατεύει τις σελίδες του χώρου-προς από προσπελάσεις εγγραφής και ανάγνωσης καθυστερώντας την απεικόνισή τους σε φυσικές σελίδες. Ο υπολογισμός του βοηθητικού διανύσματος πρώτου-αντικειμένου και η προστασία του χώρου-προς πραγματοποιείται ταυτόχρονα με την εκτέλεση των νημάτων-τροποποιητών στα οποία επιτρέπεται η πρόσβαση μόνο στο χώρο-από. Στη συνέχεια ο αλγόριθμος Compressor αναστέλλει για ένα πολύ μικρό χρονικό διάστημα την εκτέλεση **όλων** των νημάτων-τροποποιητών προκειμένου να ενημερώσει τις ρίζες τους με τις διευθύνσεις προώθησης των αντικειμένων αναφοράς τους στο χώρο-προς. Η εκτέλεση των νημάτων- τροποποιητών συνεχίζεται αμέσως μετά από αυτή τη μικρή παύση και κανένα ζωντανό αντικείμενο δεν έχει ακόμη μετεγκατασταθεί στις προστατευμένες σελίδες του χώρου-προς. Μια προσπάθεια προσπέλασης μιας προστατευμένης σελίδας του χώρου- προς από κάποιο νήμα-τροποποιητή θα οδηγήσει στην **παγίδευση** του. Η συνέχιση της εκτέλεσης του απαιτεί τον χειρισμό της παγίδευσης, ο οποίος με τη σειρά του περιλαμβάνει την εκτέλεση των εργασιών συμπύκνωσης, δηλαδή την απεικόνιση της εικονικής σελίδας του χώρου-προς σε κάποια φυσική σελίδα, τη μεταφορά σε αυτήν των αντιγράφων των ζωντανών αντικειμένων που της αναλογούν (το νήμα-τροποποιητής επί της ουσίας πραγματοποιεί αυξητικά εργασίες συμπύκνωσης) και την προώθηση των απαραίτητων δεικτών προς τις διευθύνσεις των παραπάνω αντιγράφων. Αξίζει να τονίσουμε πως αντιγράφονται μόνο τα αντικείμενα που χωρούν ολόκληρα στη συγκεκριμένα σελίδα του χώρου-προς. Η ταυτόχρονη συμπύκνωση ακόμη απαιτεί να επιτρέπεται η πρόσβαση στη σελίδα σε ένα νήμα-συμπυκνωτή κατά τη διάρκεια που αυτή απαγορεύεται στα νήματα-τροποποιητές. Για αυτόν το λόγο και όταν τα αντικείμενα μιας φυσικής σελίδας είναι έτοιμα προς αντιγραφή, ο

αλγόριθμος Compressor απεικονίζει δύο εικονικές σελίδες προς αυτή: τη συνηθισμένη (και ακόμη προστατευμένη) εικονική σελίδα του χώρου-προς και μία ιδιωτική, μη προστατευμένη εικονική σελίδα που ανήκει στο νήμα-συμπυκνωτή. Όταν οι εργασίες συμπύκνωσης για τη σελίδα ολοκληρωθούν, αίρεται η προστασία της εικονικής σελίδας του χώρου-προς ώστε να συνεχισθεί η εκτέλεση των νημάτων-τροποποιητών και η ιδιωτική απεικόνιση ακυρώνεται.

Ο αλγόριθμος Compressor κάνει χρήση της κλασικής τριχρωματικής αφαίρεσης. Οι σελίδες του χώρου-από είναι λευκές, οι προστατευμένες σελίδες του χώρου-προς είναι γκρι και οι μη προστατευμένες σελίδες του χώρου-προς είναι μαύρες. Αρχικά, κατά τη διάρκεια υπολογισμού των διευθύνσεων προώθησης στο χώρο-προς και του διανύσματος πρώτου-αντικειμένου τα νήματα-τροποποιητές λειτουργούν ως γκρι. Από τη στιγμή κατά την οποία θα τους επιτραπεί η πρόσβαση σε σελίδες του χώρου-προς, λειτουργούν ως μαύρα. Το φράγμα ανάγνωσης πραγματοποιεί την προαναφερθείσα διπλή απεικόνιση, ώστε να αποτρέψει τα μαύρα νήματα-τροποποιητές από το να διαβάσουν παλαιές τιμές αναφορών γκρι σελίδων, προς τις οποίες εκείνη τη στιγμή μεταφέρονται αντίγραφα αντικειμένων του χώρου-από. Ο αλγόριθμος Compressor πρέπει να χειριστεί και άλλες πτυχές της τριχρωματικής αναλλοίωτης. Πιο συγκεκριμένα, αμέσως μετά την ολοκλήρωση της φάσης σήμανσης και πριν ξεκινήσει ο προσδιορισμός του διανύσματος πρώτου-αντικειμένου, τα καινούρια αντικείμενα των νημάτων-τροποποιητών πρέπει να εκχωρούνται στο χώρο-προς, ώστε οι εκχωρήσεις αυτές να μην παρέμβουν στον υπολογισμό των διευθύνσεων προώθησης. Επιπλέον, όταν τα νήματα-τροποποιητές αρχίσουν να λειτουργούν ως μαύρα, τα πεδία δείκτες αυτών των νέων αντικειμένων πρέπει να σαρωθούν, ούτως ώστε τυχόν ληγμένες αναφορές προς αντικείμενα του χώρου-από να επιδιορθωθούν με τις κατάλληλες διευθύνσεις προώθησης στο χώρο-προς. Το ίδιο ισχύει και για τις καθολικές ρίζες.

Επειδή η επίδοση του αλγορίθμου Compressor εξαρτάται άμεσα από το κόστος των μηχανισμών απεικόνισης και προστασίας της εικονικής μνήμης, το οποίο σύμφωνα με τους Hosking και Moss [62] μπορεί να αυξηθεί πολύ, είναι σημαντικό οι ενέργειες των μηχανισμών αυτών να εκτελούνται όσο πιο μαζικά γίνεται. Για παράδειγμα, ο αλγόριθμος Compressor μετακινεί οκτώ εικονικές σελίδες κατά το χειρισμό της παγίδευσης ενός νήματος-τροποποιητή.

Ένα μειονέκτημα του αλγορίθμου Compressor είναι πώς εάν ένα νήμα-τροποποιητής παγιδευθεί κατά την προσπάθεια προσπέλασης μιας προστατευμένης σελίδας του χώρου-προς, τότε εκτός από το να αντιγράψει όλα τα αντικείμενα εκείνης της σελίδας πρέπει και να προωθήσει **όλους** τους δείκτες των αντικειμένων αυτών ώστε αυτοί να αναφέρονται στις νέες διευθύνσεις των στόχων τους. Το γεγονός αυτό μπορεί να προκαλέσει μεγάλη παύση στην εκτέλεση του τροποποιητή.

Τέλος, το σχήμα 9.2 απεικονίζει τον τρόπο με τον οποίο ο αλγόριθμος Compressor οδηγεί τη συμπύκνωση χρησιμοποιώντας το μηχανισμό προστασίας εικονικών σελίδων. Όπως φαίνεται και στο σχήμα, οι εικονικές σελίδες ομαδοποιούνται λογικά στις ακόλουθες κατηγορίες:

Ζωντανή: η σελίδα περιέχει κυρίως ζωντανά αντικείμενα.

Καταδικασμένη: η σελίδα περιέχει μερικά ζωντανά αντικείμενα αλλά κυρίως νεκρά αντικείμενα και είναι υποψήφια προς συμπύκνωση.

Ελεύθερη: η σελίδα δεν περιλαμβάνει καθόλου αντικείμενα και χρησιμοποιείται για εκχώρηση.

Καινούρια Ζωντανή: στη σελίδα έχει εκχωρηθεί μνήμη για αντίγραφα ζωντανών αντικειμένων (τα οποία ωστόσο δεν έχουν ακόμη αντιγραφεί).

Νεκρή: η σελίδα δεν είναι απεικονισμένη προς κάποια φυσική σελίδα και μπορεί να ανακυκλωθεί.

9.6 Ταυτόχρονη Καταμέτρηση Αναφορών

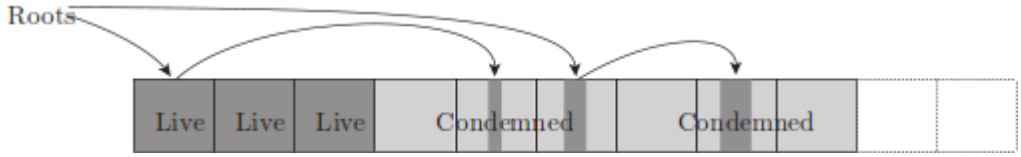
Εξετάσαμε τη συλλογή με καταμέτρηση αναφορών στο κεφάλαιο 5. Είδαμε πως τα βασικά μειονεκτήματα της απλοϊκής καταμέτρησης αναφορών αφορούν την αδυναμία συλλογής κύκλων απορριμμάτων και το υψηλό κόστος της διαχείρισης των μετρητών αναφορών, ειδικά σε καταστάσεις συναγωνισμού μεταξύ πολλαπλών νημάτων-τροποποιητών. Ο αλγόριθμος Recycler (αλγόριθμος 5.5) χρησιμοποιώντας την τεχνική της δοκιμαστικής διαγραφής λύνει το πρόβλημα της αδυναμίας συλλογής κύκλων απορριμμάτων. Η καταμέτρηση αναφορών με αναβολή αποτρέπει τα νήματα-τροποποιητές από το να μεταβάλλουν τους μετρητές αναφορών τοπικών μεταβλητών, ενώ τέλος η καταμέτρηση αναφορών με συγκέντρωση αποφεύγει τις περιττές ενημερώσεις των αναφοράς που ακυρώνονται από αργότερες λειτουργίες εγγραφής δεικτών από τα νήματα-τροποποιητές. Και οι τρεις λύσεις απαιτούν την παύση του κόσμου την ώρα που ο συλλέκτης ενημερώνει τους μετρητές αναφορών και ανακτά τη μνήμη από αντικείμενα απορρίμματα. Εξετάζουμε τώρα πώς η απαίτηση για παύση του κόσμου μπορεί να αποφευχθεί καθώς και ποιες αλλαγές απαιτούνται ώστε ένας συλλέκτης με καταμέτρηση αναφορών να εκτελεστεί ταυτόχρονα με τα νήματα-τροποποιητές.

9.6.1 Απλοϊκή καταμέτρηση αναφορών

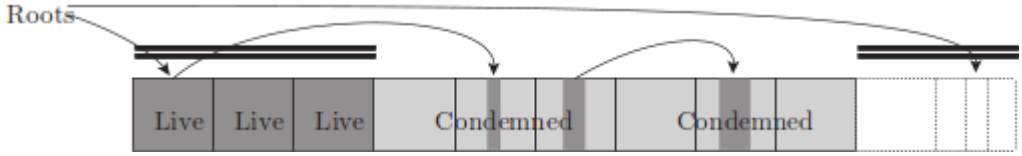
Η ορθότητα ενός συλλέκτη απορριμμάτων καταμέτρησης αναφορών απαιτεί την εξασφάλιση της ισχύος της αναλλοίωτης ότι ο μετρητής αναφορών κάθε αντικειμένου ισούται με το πλήθος των αναφορών προς αυτό. Η εξασφάλιση της αναλλοίωτης είναι ιδιαίτερα περίπλοκη όταν εμπλέκονται πολλαπλά νήματα-τροποποιητές. Εκ πρώτης όψεως, φαίνεται πως η ασφαλής εκτέλεση της λειτουργίας Write είναι πιο δύσκολη από την ασφαλή εκτέλεση της λειτουργίας Read. Η ενημέρωση ενός δείκτη περιλαμβάνει τρεις ενέργειες: την αύξηση του μετρητή αναφορών του νέου αντικειμένου αναφοράς, τη μείωση του μετρητή αναφορών του παλαιού αντικειμένου αναφοράς και την εγγραφή του δείκτη. Ο συντονισμός των ενεργειών αυτών είναι απαραίτητος ούτως ώστε να αποφεύγεται η πρόωρη αποδέσμευση αντικειμένων (λόγω για παράδειγμα προσωρινής μηδενικής τιμής μετρητών αναφορών) όσο και η επ' άπειρον αιώρηση απορριμμάτων στο σωρό.

Η δυσκολία της ταυτόχρονης συλλογής απορριμμάτων με καταμέτρηση αναφορών δεν έγκειται μόνο στην εξασφάλιση της ατομικής αύξησης και μείωσης μετρητών αναφορών. Το δυσκολότερο πρόβλημα αφορά το συγχρονισμό των τροποποιήσεων των μετρητών αναφορών με τις φορτώσεις και αποθηκεύσεις δεικτών. Ένας τρόπος επίτευξης της ατομικότητας των λειτουργιών Read και Write του τροποποιητή είναι το κλείδωμα του αντικειμένου στο οποίο ανήκει το πεδίο που διαβάζεται η γράφεται.

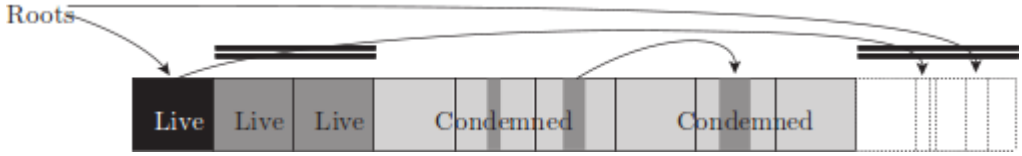
Είναι επιθυμητή η εύρεση μιας λύσης που δε χρησιμοποιεί κλειδώματα αλλά πρωταρχικές εντολές των οποίων η ατομικότητα εξασφαλίζεται από το υλικό. Δυστυχώς, οι πρωταρχικές εντολές που αφορούν μόνο μία θέση μνήμης δεν επαρκούν για την εξασφάλιση της ασφάλειας. Οι Detlefs κ.ά. [40] δείχνουν πως η εντολή CompareAndSwap2, η οποία ατομικά ενημερώνει δύο διαφορετικές θέσεις μνήμης, επαρκεί. Παρότι δεν είναι επαρκής για τη διατήρηση ακριβών τιμών των μετρητών αναφορών για κάθε χρονική στιγμή, η χρήση της εξασφαλίζει την ασθενέστερη αναλλοίωτη πως:



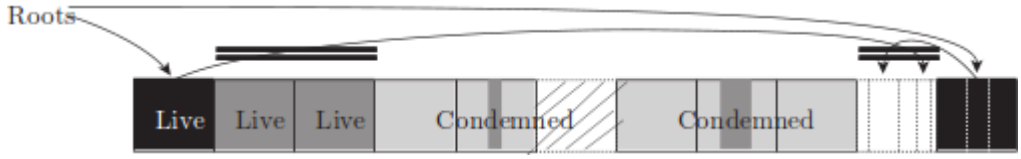
(α′) Αρχική διαμόρφωση αλγορίθμου Compressor. Όλες οι σελίδες ανήκουν στο χώρο-από.



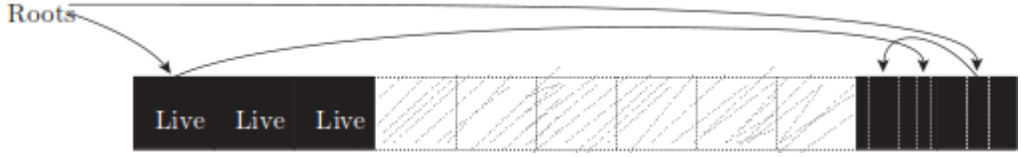
(β′) Υπολογισμός πληροφοριών προώθησης και προστασία όλων των σελίδων του χώρου-προς. Μεταξύ αυτών είναι οι δεσμευμένες σελίδες προς τις οποίες θα μεταφερθούν ζωντανά αντικείμενα και οι ζωντανές σελίδες που δεν έχουν καταδικασθεί για εκκένωση. Στη συνέχεια οι ρίζες των τροποποιητών δείχνουν προς το χώρο-προς. Τα νήματα- τροποποιητές που θα προσπαθήσουν να προσπελάσουν μια προστατευμένη σελίδα του χώρου-προς θα παγιδευθούν.



(γ′) Η παγίδευση σε μία ζωντανή σελίδα προωθεί τους δείκτες που περιλαμβάνονται σε αυτή, ώστε να αναφέρονται προς τα αντίγραφα στο χώρο-προς των προορισμών τους. Η προστασία της ζωντανής σελίδας αίρεται, όταν έχουν πλέον προωθηθεί όλοι οι δείκτες αυτής.



(δ′) Η παγίδευση σε μία δεσμευμένη σελίδα του χώρου-προς γεμίζει τη σελίδα εκκενώνοντας σελίδες του χώρου-από. Τα πεδία δείκτες των αντικειμένων αυτών ενημερώνονται ώστε να δείχνουν προς το χώρο-προς. Στη συνέχεια αίρεται η προστασία της σελίδας του χώρου-προς και καταργείται η απεικόνιση προς πλήρως εκκενωμένες σελίδες του χώρου-από.



(ε′) Η συμπύκνωση ολοκληρώνεται όταν όλες οι ζωντανές σελίδες έχουν σαρωθεί, όλοι οι δείκτες που αυτές περιλαμβάνουν έχουν προωθηθεί, όλα τα ζωντανά αντικείμενα καταδικασμένων σελίδων έχουν αντιγραφεί στο χώρο-προς και όλες οι αναφορές των τελευταίων έχουν προωθηθεί.

Σχήμα 9.2: Ταυτόχρονη εκτέλεση αλγορίθμου Compressor.

- όσο υπάρχει τουλάχιστον μια αναφορά προς ένα αντικείμενο, ο μετρητής αναφορών αυτού θα είναι μη μηδενικός και
- αν δεν υπάρχουν αναφορές προς ένα αντικείμενο, ο μετρητής αναφορών του τελικώς θα μηδενισθεί.

9.6.2 Καταμέτρηση αναφορών με απομόνωση

Οι αλγόριθμοι συλλογής απορριμμάτων με πρόθυμη καταμέτρηση αναφορών απαιτούν τη χρήση είτε κλειδωμάτων είτε πρωταρχικών ατομικών εντολών που τροποποιούν πολλές λέξεις μνήμης ταυτόχρονα. Οι μηχανισμοί κλειδωμάτων δεν είναι ιδιαίτερα φθηνοί, ενώ οι ατομικές πρωταρχικές εντολές πολλαπλών θέσεων μνήμης δεν προσφέρονται από όλες τις αρχιτεκτονικές συνόλου εντολών. Η καταμέτρηση αναφορών με αναβολή περιορίζει κάπως το πρόβλημα με τη μη εφαρμογή λειτουργιών καταμέτρησης αναφορών σε τοπικές μεταβλητές και την αναβολή της ανάκτησης μνήμης αντικειμένων με μηδενικό μετρητή αναφορών. Η **καταμέτρηση αναφορών με απομόνωση** υποστηρίζει την εκτέλεση πολυνηματικών εφαρμογών, χρησιμοποιώντας απλές εντολές φόρτωσης και αποθήκευσης στο φράγμα εγγραφής του τροποποιητή.

Ο DeTreville [41] σε έναν υβριδικό συλλέκτη για τη γλώσσα Modula-2+, που χρησιμοποιεί έναν εφεδρικό συλλέκτη σήμανσης και εκκαθάρισης για την αντιμετώπιση κύκλων απορριμμάτων, προκειμένου να αποφύγει το κόστος του συγχρονισμού των λειτουργιών καταμέτρησης αναφορών από διαφορετικά νήματα-τροποποιητές, ανέθετε στα τελευταία να καταγράφουν τις διευθύνσεις του παλαιού και του νέου αντικειμένου αναφοράς ενός δείκτη σε έναν απομονωτή. Στη συνέχεια, ένα ξεχωριστό νήμα-καταμετρητής αναφορών επεξεργάζεται το χώρο καταγραφής και προσαρμόζει τους μετρητές αναφορών εξασφαλίζοντας με τον τρόπο αυτό την ατομικότητα των ενημερώσεων. Δυστυχώς όμως η καταμέτρηση αναφορών με απομόνωση δεν εξαφανίζει το πρόβλημα του συντονισμού των λειτουργιών καταμέτρησης αναφορών και της εγγραφής του δείκτη. Ο DeTreville πρότεινε δύο λύσεις, από τις οποίες όμως καμία δεν είναι εξ ολοκλήρου ικανοποιητική. Η πρώτη του προσέγγιση αφορά στην προστασία με κλείδωμα ολόκληρης της λειτουργίας WRITE. Για να αποφύγει το κόστος της ατομικής εκτέλεσης κάθε λειτουργίας WRITE, η δεύτερη λύση του παρέχει σε κάθε νήμα-τροποποιητή το δικό του τοπικό απομονωτή, ο οποίος περιοδικά περνά στον έλεγχο του νήματος-καταμετρητή αναφορών. Η προσέγγιση αυτή ωστόσο μεταφέρει την ευθύνη της εξασφάλισης της ατομικής εκτέλεσης των λειτουργιών WRITE στον προγραμματιστή, ο οποίος επιβαρύνεται με το ιδιαίτερα επίπονο έργο του χειρισμού κλειδωμάτων.

Οι Bacon και Rajan [11] επίσης παρέχουν σε κάθε νήμα- τροποποιητή το δικό του τοπικό απομονωτή, απαιτούν όμως η ενημέρωση ενός δείκτη να είναι ατομική. Το φράγμα εγγραφής του τροποποιητή σε έναν επεξεργαστή προσθέτει την παλαιά και την καινούρια τιμή του πεδίου i ενός αντικειμένου στον τοπικό απομονωτή $localUpdates$. Η καταμέτρηση αναφορών των τοπικών μεταβλητών αναβάλλεται και πάλι, ενώ για να αποτραπεί η πρόωρη διαγραφή αντικειμένων, ο χρόνος διαιρείται σε **εποχές** χρησιμοποιώντας έναν καθολικό αριθμό εποχής και τοπικούς αριθμούς εποχής για κάθε νήμα-τροποποιητή. Περιοδικά, όπως και στην καταμέτρηση αναφορών με αναβολή, ένας επεξεργαστής θα διακόψει την εκτέλεση ενός νήματος-τροποποιητή και θα εξετάσει όλες τις τοπικές στοίβες, καταχωρώντας τις αναφορές που βρίσκει σε έναν τοπικό απομονωτή $myStackBuffer$. Στη συνέχεια ο επεξεργαστής μεταφέρει τους απομονωτές $myStackBuffer$ και $myUpdates$ στο συλλέκτη και ενημερώνει τον τοπικό αριθμό εποχής e. Τέλος, δρομολογεί το νήμα-συλλέκτη στον επόμενο επεξεργαστή πριν επαναφέρει σε λειτουργία το σταματημένο νήμα-τροποποιητή.

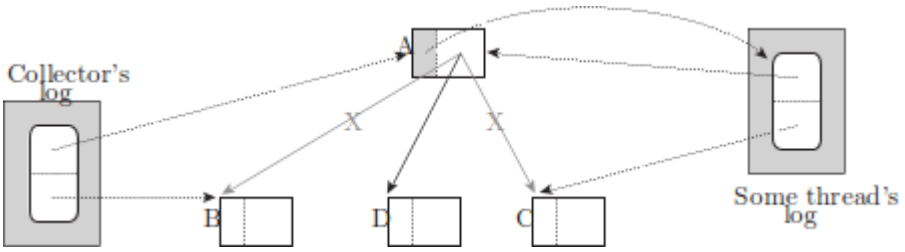
Το νήμα-συλλέκτης εκτελείται στον επόμενο επεξεργαστή. Σε κάθε κύκλο συλλογής k, ο συλλέκτης εφαρμόζει τις αυξήσεις μετρητών αναφορών της εποχής k και τις μειώσεις μετρητών αναφορών της εποχής $k-1$. Τέλος, ενημερώνει τον καθολικό αριθμό εποχής. Το πλεονέκτημα της μεθόδου είναι πως ποτέ δεν απαιτείται η αναστολή λειτουργίας όλων των νημάτων-τροποποιητών: ο συλλέκτης λειτουργεί εν- τη-πτήσει.

Αλγόριθμος 9.7 Ταυτόχρονη συλλογή: ταυτόχρονη καταμέτρηση αναφορών με χρήση απομονωτή

```
1:  shared epoch
2:  shared updatesBuffer[]                                  ▷ one buffer per epoch

3:  procedure WRITE(src, i, ref)
4:      if src = Roots then
5:          src[i] ← ref
6:      else
7:          old ← ATOMICEXCHANGE(&src[i], ref)
8:          LOG(old, ref)

9:  procedure LOG(old, new)
10:     myUpdates ← myUpdates + [< old, new >]

11: procedure COLLECT()
12:     myStackBuffer ← []
13:     for all local ref in myStacks do                    ▷ deferred reference counting
14:         myStackBuffer ← myStackBuffer + [< ref, ref >]
15:         atomic
16:         updatesBuffer[e] ← updatesBuffer[e] + myStackBuffer
17:         atomic
18:         updatesBuffer[e] ← updatesBuffer[e] + myUpdates
19:     myUpdates ← []
20:     e ← e + 1

21:     me ← myProcessorId
22:     if me < MAX_PROCESSORS then
23:         SCHEDULE(collect, me + 1)          ▷ schedule collect() on the next processor
24:     else                                   ▷ the last processor updates the reference counts
25:         for all < old, new > in updatesBuffer[epoch] do
26:             ADDREFERENCE(old)
27:         for all < old, new > in updatesBuffer[epoch] do
28:             ADDREFERENCE(old)
29:         RELEASE(updatesBuffer[epoch − 1])                ▷ free the old buffer
30:         epoch ← epoch + 1
```

9.6.3 Ταυτόχρονη κυκλική καταμέτρηση αναφορών

Ο αλγόριθμος Recycler [8], [11] ανακτά τη μνήμη από κύκλους απορριμμάτων εξερευνώντας υποψήφιους υπογράφους και εφαρμόζοντας δοκιμαστική διαγραφή στους μετρητές αναφορών.



Σχήμα 9.3: Ταυτόχρονη καταμέτρηση αναφορών με συγκέντρωση. Στη διάρκεια της προηγούμενης εποχής το αντικείμενο A τροποποιήθηκε ώστε να δείχνει στο αντικείμενο C και οι τιμές των πεδίων του αποθηκεύθηκαν σε κάποιον τοπικό απομονωτή καταγραφής. Ωστόσο, το αντικείμενο A έχει τροποποιηθεί ξανά σε αυτήν την εποχή (δείχνει πλέον στο αντικείμενο D) και συνεπώς έχει σημανθεί ως βρώμικο και καταγραφεί ξανά. Το αρχικό αντικείμενο αναφοράς B μπορεί να βρεθεί στον καθολικό χώρο καταγραφής, όπως και στο σχήμα 5.2. Η αναφορά προς το αντικείμενο C μπορεί να βρεθεί στον τοπικό απομονωτή καταγραφής ενός νήματος-τροποποιητή, προς τον οποίο υπάρχει δείκτης από το αντικείμενο A.

Παρότι η χρήση απομονωτή επιτυχώς μεταβιβάζει την καταμέτρηση αναφορών σε έναν εφεδρικό επεξεργαστή, ο αλγόριθμος Recycler αντιμετωπίζει τρία προβλήματα στην προσπάθεια συλλογής κύκλων απορριμμάτων σε ένα περιβάλλον ταυτοχρονισμού:

1. Δεν μπορεί να εγγυηθεί πως θα εξερευνήσει εκ νέου τον ίδιο υπογράφο, καθώς ο γράφος μπορεί να έχει τροποποιηθεί από τα νήματα-τροποποιητές ενόσω αυτός εντοπίζει κύκλους απορριμμάτων.

2. Διαγραφές δεικτών ενδέχεται να αποσυνδέσουν το γράφο.

3. Οι μετρητές αναφορών μπορεί να είναι ανενημέρωτοι.

Η ασύγχρονη έκδοση του αλγορίθμου λειτουργεί σε δύο φάσεις για να επιλύσει τα παραπάνω προβλήματα. Η πρώτη φάση είναι λίγο πολύ η σύγχρονη εκδοχή του αλγορίθμου που παρουσιάσθηκε στο κεφάλαιο 5. Ωστόσο ο ασύγχρονος Recycler αναβάλλει την αποδέσμευση της μνήμης αντικειμένων που εντοπίζει η διαδικασία COLLECTWHITE (αλγόριθμος 5.5) μέχρι την επόμενη φάση, η οποία ελέγχει ότι αυτά είναι ακόμη απορρίμματα. Η προσέγγιση αυτή παρουσιάζει τα ακόλουθα μειονεκτήματα. Θεωρητικά και σπάνια στην πράξη είναι πιθανόν ορισμένοι κύκλοι απορριμμάτων να μη συλλεγούν: δεν υπάρχει εγγύηση πληρότητας του ασύγχρονου συλλέκτη. Επιπλέον, η δοκιμαστική διαγραφή δε μπορεί να χρησιμοποιήσει τον αυθεντικό μετρητή αναφορών αλλά έναν ειδικό κυκλικό μετρητή αναφορών που επίσης αποθηκεύεται στην επικεφαλίδα των αντικειμένων. Τρίτον, ο αλγόριθμος πρέπει να εξερευνήσει εκ νέου τους υποψήφιους κύκλους απορριμμάτων στη δεύτερη φάση, για να αποφύγει τη λανθασμένη αποδέσμευση μνήμης ζωντανών αντικειμένων.

Το βασικό πρόβλημα είναι πως ο συλλέκτης Recycler προσπαθεί να εφαρμόσει έναν αλγόριθμο που έχει σχεδιασθεί για σύγχρονη συλλογή σε έναν ταυτόχρονο κόσμο όπου η τοπολογία του γράφου αντικειμένων διαρκώς μεταβάλλεται.

9.6.4 Λήψη ενός στιγμιοτύπου του σωρού

Στο κεφάλαιο 5 είδαμε πώς η καταμέτρηση αναφορών με συγκέντρωση παρέχει στο συλλέκτη ένα στιγμιότυπο του σωρού. Σε έναν τοπικό σε κάθε νήμα τροποποιητή απομονωτή, ο οποίος μεταφέρεται σύγχρονα στο συλλέκτη, αποθηκεύονται αντίγραφα αντικειμένων, των οποίων πεδία δείκτες έχουν τροποποιηθεί. Η λειτουργία όλων των νημάτων-τροποποιητών αναστέλλεται στην αρχή ενός κύκλου συλλογής, οι απομονωτές διαβιβάζονται στο συλλέκτη και δεσμεύεται μνήμη για νέους απομονωτές. Ο συλλέκτης απλώς χρησιμοποιεί το αντίγραφο ενός αντικειμένου για να βρει τα παλαιά αντικείμενα αναφοράς των δεικτών αυτού και να μειώσει τους μετρητές αναφορών αυτών, ενώ χρησιμοποιεί την τρέχουσα κατάσταση του αντικειμένου για να αυξήσει τους μετρητές αναφορών των νέων αντικειμένων αναφορών των δεικτών αυτού. Το νήμα-καταμετρητής αναφορών μπορεί να εκτελείται ταυτόχρονα με τα νήματα-τροποποιητές. Στην περίπτωση αυτή, η εκτέλεση των νημάτων-τροποποιητών αναστέλλεται προσωρινά, μέχρις ότου οι απομονωτές αυτών μεταβιβασθούν στο συλλέκτη. Μόλις ολοκληρωθεί η μεταφορά, επανεκκινείται η εκτέλεση των νημάτων-τροποποιητών. Ο συλλέκτης είναι επιφορτισμένος με τη διόρθωση των μετρητών αναφορών των παλαιών και των νέων παιδιών κάθε τροποποιημένου αντικειμένου. Η μείωση των μετρητών αναφορών των παλαιών παιδιών γίνεται εύκολα, όπως και πριν, χρησιμοποιώντας το αντίγραφο ενός αντικειμένου. Η αύξηση όμως των μετρητών αναφορών των παιδιών ενός αντικειμένου τη στιγμή μεταφοράς των απομονωτών είναι πιο περίπλοκη, καθώς το αντικείμενο ενδέχεται να έχει τροποποιηθεί εκ νέου.

Αν το αντικείμενο παραμένει καθαρό, τότε η κατάσταση του δεν έχει αλλάξει και οι μετρητές αναφορών των τρεχόντων παιδιών του αυξάνονται. Αν πάλι το αντικείμενο έχει τροποποιηθεί από τη στιγμή μεταφοράς των απομονωτών, τότε έχει ξανασημανθεί ως βρώμικο και η κατάσταση του τη στιγμή της μεταφοράς μπορεί να βρεθεί σε ένα φρέσκο (καινούριο) απομονωτή κάποιου νήματος-τροποποιητή. Ο βρώμικος δείκτης του αντικειμένου αναφέρεται σε αυτόν τον απομονωτή καταγραφής, ο οποίος μπορεί να αναγνωσθεί χωρίς να απαιτείται ο συγχρονισμός του νήματος-συλλέκτη με το νήμα-τροποποιητή στο οποίο ο τελευταίος ανήκει.

9.6.5 Καταμέτρηση αναφορών με ολισθαίνουσες όψεις

Για τη λήψη ενός στιγμιοτύπου του σωρού, αναστέλλεται η εκτέλεση όλων των νημάτων-τροποποιητών ταυτόχρονα προκειμένου να πραγματοποιηθεί η μεταφορά των απομονωτών. Η τεχνική των **ολισθαινουσών όψεων** χαλαρώνει τον παραπάνω περιορισμό και προσφέρει τη δυνατότητα να διακόπτεται η εκτέλεση ενός μόνο νήματος-τροποποιητή κάθε φορά. Είναι προφανές πως αυτή η προσέγγιση έχει ως αποτέλεσμα να παρέχεται στο συλλέκτη μια παραμορφωμένη εικόνα του σωρού. Οι τιμές διαφορετικών αντικειμένων καταγράφονται και μεταβιβάζονται στο νήμα-συλλέκτη σε διαφορετικές χρονικές στιγμές. Η τεχνική δεν απαιτεί τη χρήση κλειδωμάτων ούτε και ατομικών πρωταρχικών εντολών υλικού, αλλά συντονίζει κάθε νήμα-τροποποιητή με το νήμα- συλλέκτη με τη χρήση 4 χειραψιών όμοιων με εκείνες που χρησιμοποιούν οι Doligez και Gonthier [47]. Η τεχνική των ολισθαινουσών όψεων χρησιμοποιείται ευρέως για απλή, καθαρή καταμέτρηση αναφορών από τους Levanoni και Petrank [76], [77], [78], για τη διαχείριση της παλαιάς γενεάς γενεαλογικών συλλεκτών από τους Azatchi και Petrank [7], καθώς και σε συνδυασμό με κυκλική καταμέτρηση αναφορών από τους Paz κ.ά. [89], [88].

9.7 Θέματα προς εξέταση

Πρωταρχικός στόχος τόσο των αυξητικών όσο και των ταυτόχρονων συλλεκτών απορριμμάτων είνα η ελαχιστοποίηση των παύσεων συλλογής που αντιλαμβάνεται ο τροποποιητής. Είτε η παύση οφείλεται στην εκτέλεση ενός μικρού κβάντου εργασιών συλλογής από τον τροποποιητή, είτε προκαλείται από το γεγονός πως ο τροποποιητής πρέπει να συγχρονισθεί με (και πιθανώς να περιμένει) το συλλέκτη μέχρις ότου ο τελευταίος ολοκληρώσει κάποιες εργασίες, οι αυξητικοί/ταυτόχρονοι συλλέκτες θυσιάζουν έως ένα βαθμό τη ρυθμαπόδοση του τροποποιητή με σκοπό τη μείωση της διατάραξης της εκτέλεσής του από το συλλέκτη. Σε έναν ιδανικό κόσμο, τα ταυτόχρονα νήματα-συλλέκτες θα μπορούσαν να εκτελεσθούν τελείως παράλληλα με τα νήματα-τροποποιητές. Στον πραγματικό κόσμο όμως, όπως είδαμε, η ταυτόχρονη συλλογή απορριμμάτων απαιτεί την επικοινωνία και το συγχρονισμό μεταξύ συλλέκτη και τροποποιητή έως ένα βαθμό, κάτι που επιτυγχάνεται δια μέσου φραγμάτων (ανάγνωσης και εγγραφής) τροποποιητή. Επιπλέον, ο συναγωνισμός μεταξύ συλλέκτη και τροποποιητή για χρόνο εκτέλεσης και μνήμη μπορεί επίσης να μειώσει τη ρυθμαπόδοση του τροποποιητή.

Αντίστροφα, η αυξητική ή η ταυτόχρονη συλλογή απορριμμάτων μπορεί να βελτιώσει τη ρυθμαπόδοση ορισμένων εφαρμογών. Οι αυξητικοί/ταυτόχρονοι συλλέκτες επιβάλλουν μια επιβάρυνση στις λειτουργίες φόρτωσης/αποθήκευσης του τροποποιητή προκειμένου να μειώσουν τις αντιλαμβανόμενες από τους χρήστες των εφαρμογών παύσεις. Ωστόσο, ο χρήστης μιας εφαρμογής μπορεί να είναι ένα άλλο πρόγραμμα, το οποίο μάλιστα είναι ιδιαίτερα ευαίσθητο σε καθυστερήσεις. Ο Ossia κ.ά. [86] φέρνουν ως παράδειγμα τα τριών επιπέδων συστήματα επεξεργασίας δοσοληψιών. Επισημαίνουν πως καθυστερήσεις που προκύπτουν από συλλογή με παύση του κόσμου μπορεί να έχουν ως αποτέλεσμα την αποτυχία δοσοληψιών και την ανάγκη επανεκτέλεσής τους. Εκτελώντας λίγη παραπάνω εργασία σε ένα φράγμα εγγραφής, όλη η δουλειά της επανεκτέλεσης των δοσοληψιών μπορεί να αποφευχθεί.

Κάθε διαφορετική τεχνική ταυτόχρονης συλλογής απορριμμάτων που εξετάσαμε σε αυτό το κεφάλαιο επηρεάζει με το δικό της τρόπο τα παραπάνω κόστη. Ταυτόχρονοι συλλέκτες καταμέτρησης αναφορών επιβάλλουν ένα ιδιαίτερα υψηλό κόστος στις λειτουργίες φόρτωσης και αποθήκευσης δεικτών του τροποποιητή. Οι ταυτόχρονοι συλλέκτες σήμανσης και εκκαθάρισης, οι οποίοι δε μετακινούν αντικείμενα, επιβάλλουν ένα μικρό σχετικά κόστος στις προσπελάσεις δεικτών (ανάλογα με το φράγμα εγγραφής), ενδέχεται όμως να προκαλέσουν τον κατακερματισμό της μνήμης. Οι ταυτόχρονοι συλλέκτες αντιγραφής και οι ταυτόχρονοι συλλέκτες συμπύκνωσης, οι οποίοι μετακινούν αντικείμενα, χρειάζονται επιπλέον συγχρονισμό ώστε να προστατεύουν τον τροποποιητή από, ή να τον ενημερώνουν για αντικείμενα που μετακινούνται. Οι ταυτόχρονοι συλλέκτες αντιγραφής επίσης επιβάλλουν επιπλέον κόστος σε χώρο. Σε όλους πάντως τους ταυτόχρονους συλλέκτες, το αν χρησιμοποιείται ένα φράγμα εγγραφής ή ένα φράγμα ανάγνωσης επηρεάζει σε κάθε περίπτωση διαφορετικά τη ρυθμαπόδοση του τροποποιητή και εξαρτάται μεταξύ άλλων, από τη σχετική συχνότητα των λειτουργιών εγγραφής και ανάγνωσης και την ποσότητα εργασίας που εκτελεί το φράγμα.

Οι ταυτόχρονοι συλλέκτες σήμανσης και εκκαθάρισης συνήθως απαιτούν από τον τροποποιητή να ενημερώσει το νήμα-σημαντή για ένα αντικείμενο προς σήμανση μέσω ενός φράγματος εγγραφής. Οι ταυτόχρονοι συλλέκτες αντιγραφής και οι ταυτόχρονοι συλλέκτες συμπύκνωσης συνήθως προστατεύουν τον τροποποιητή από την πρόσβαση σε αντικείμενα που έχουν μετακινηθεί αλλού, απαιτώντας από τον τελευταίο εκτελέσει ένα φράγμα ανάγνωσης. Απαιτείται ένας συμβιβασμός ανάμεσα στη συχνότητα εκτέλεσης φραγμάτων και το μέγεθος εργασιών που εκτελούνται σε αυτά. Ένα φράγμα που πυροδοτεί την αντιγραφή και τη σάρωση ενός αντικειμένου είναι σίγουρα πιο ακριβό από ένα φράγμα που απλώς αντιγράφει, το οποίο με τη

σειρά του θα είναι πιο ακριβό από ένα φράγμα που απλώς ανακατευθύνει ένα δείκτη προέλευσης. Παρόμοια, η εκτέλεση περισσότερων εργασιών νωρίτερα, μπορεί να έχει ως αποτέλεσμα την εκτέλεση λιγότερων εργασιών από τα φράγματα αργότερα.

Ένα ακόμη κόστος των ταυτόχρονων συλλεκτών απορριμμάτων αφορά το πλήθος των αιωρούμενων στο σωρό αντικειμένων-απορριμμάτων. Η έλλειψη της ανάγκης συλλογής αιωρούμενων αντικειμένων-απορριμμάτων επιτρέπει τον ταχύτερο τερματισμό ενός κύκλου ταυτόχρονης συλλογής.

Επιπλέον, το αν η εκτέλεση των νημάτων-τροποποιητών πρέπει να διακοπεί στην αρχή ενός κύκλου ταυτόχρονης συλλογής (ώστε να εξασφαλισθεί πώς ο συλλέκτης έχει εντοπίσει όλες τις ρίζες) ή στο τέλος (ώστε να εξασφαλισθεί ο τερματισμός του κύκλου) επηρεάζει τη ρυθμαπόδοση. Τα κριτήρια τερματισμού ενός κύκλου ταυτόχρονης συλλογής μπορούν τέλος να επηρεάσουν την ποσότητα των αιωρούμενων στο σωρό αντικειμένων απορριμμάτων.

Η αυξητική και η ταυτόχρονη συλλογή απορριμμάτων είναι ιδιαίτερα κατάλληλη όταν ο όγκος των ζωντανών αντικειμένων αναμένεται να είναι πολύ μεγάλος. Σε αυτή την περίπτωση, ακόμη και η παράλληλη συλλογή με διακοπή του κόσμου, η οποία κάνει χρήση όλων των επεξεργαστών θα οδηγούσε σε μη αποδεκτούς χρόνους παύσης. Ωστόσο, ένα μειονέκτημα των αυξητικών και των ταυτόχρονων συλλεκτών είναι πως αυτοί δε συλλέγουν όλα τα αντικείμενα απορρίμματα πριν τελειώσει ο κύκλος συλλογής. Για να εξασφαλισθεί πως ο τροποποιητής δε θα ξεμείνει από μνήμη πριν ολοκληρωθεί ένας κύκλος ταυτόχρονης συλλογής, πρέπει να παρέχεται στο συλλέκτη αρκετός χώρος στο σωρό ή ένα γεναιόδωρο μερίδιο των πόρων του επεξεργαστή (εις βάρος πάντα του τροποποιητή).

9.7.1 Ταυτόχρονη σήμανση και εκκαθάριση

Πολλά από τα ζητήματα που αφορούν την ταυτόχρονη συλλογή απορριμμάτων με σήμανση και εκκαθάριση είναι κοινά για τους περισσότερους ταυτόχρονους συλλέκτες. Η σχεδίαση ενός ταυτόχρονου συλλέκτη είναι σαφώς πιο πολύπλοκη από τη σχεδίαση ενός συλλέκτη που λειτουργεί με παύση του κόσμου. Προκύπτει το ερώτημα κατά πόσον οι απαιτήσεις από το συλλέκτη δικαιολογούν αυτήν την επιπλέον πολυπλοκότητα, καθώς μια απλούστερη λύση, όπως η χρήση ενός γενεαλογικού συλλέκτη, είναι αρκετή.

Οι γενεαλογικοί συλλέκτες απορριμμάτων προσφέρουν αναμενόμενους χρόνους παύσης της τάξης των χιλιοστών του δευτερολέπτου για τις περισσότερες εφαρμογές. Ωστόσο στη χειρότερη περίπτωση, όπου και πρέπει να συλλεγεί ολόκληρος ο σωρός, ο χρόνος παύσης μιας εφαρμογής μπορεί να αυξηθεί υπερβολικά ανάλογα με το μέγεθος του σωρού, τον όγκο των ζωντανών αντικειμένων κ.ο.κ. Οι ταυτόχρονοι συλλέκτες από την άλλη πλευρά προσφέρουν συντομότερους και πιο προβλέψιμους χρόνους παύσης.

Η ταυτόχρονη συλλογή με σήμανση και εκκαθάριση μπορεί, όπως και η μη ταυτόχρονη συλλογή με σήμανση και εκκαθάριση, να οδηγήσει σε κατακερματισμό του σωρού. Η συλλογή με αντιγραφή και η συλλογή με σήμανση και συμπύκνωση εκτός από το ότι αντιμετωπίζουν τον κατακερματισμό της μνήμης επιτρέπουν επίσης την σειριακή εκχώρηση μνήμης, η οποία μπορεί να είναι ταχύτερη από την εκχώρηση με χρήση ελεύθερων λιστών και μπορεί επίσης να παρέχει καλύτερη τοπικότητα αναφορών στα νήματα-τροποποιητές. Από την άλλη πλευρά βέβαια οι συλλέκτες σήμανσης και εκκαθάρισης χρησιμοποιούν αποδοτικότερα το διαθέσιμο χώρο μνήμης σε σύγκριση με τους συλλέκτες αντιγραφής, καθώς δεν απαιτούν την ύπαρξη ενός εφεδρικού χώρου αντιγραφής. Επιπλέον, οι μη-μετακινούντες ταυτόχρονοι συλλέκτες έχουν το εξής πλεονέκτημα σε σχέση με τους μετακινούντες ταυτόχρονους συλλέκτες: η

εξασφάλιση της συνέπειας του σωρού είναι σχετικά απλή. Όλοι οι ταυτόχρονοι συλλέκτες απαιτούν από τα νήματα-τροποποιητές να τους ενημερώνουν σχετικά με τη μεταβολή της τοπολογίας του σωρού ούτως ώστε να αποτραπεί η περίπτωση κάποιο νήμα-τροποποιητής να κρύβει αντικείμενα από το συλλέκτη. Ωστόσο, οι ταυτόχρονοι μετακινούντες συλλέκτες πρέπει επιπλέον να εξασφαλίσουν τόσο πως μόνο ένα νήμα-συλλέκτης μετακινεί ένα αντικείμενο όσο και πως η ενημέρωση όλων των δεικτών προς ένα μετακινηθέν αντικείμενο εμφανίζεται να πραγματοποιείται ατομικά στα νήματα-τροποποιητές. Τέλος η ταυτόχρονη συλλογή με σήμανση και εκκαθάριση προσφέρει έναν αριθμό από σχεδιαστικές επιλογές υλοποίησης. Τα καινούρια αντικείμενα μπορούν να εκχωρούνται είτε ως μαύρα, γκρι ή λευκά. Ένα νέο αντικείμενο εκχωρείται πάντοτε ως μαύρο σε ένα μαύρο τροποποιητή, ενώ ένας γκρι τροποποιητής προσφέρει περισσότερες επιλογές. Τα νέα αντικείμενα μπορούν να εκχωρούνται ως μαύρα, γκρι, ή λευκά και επιπλέον η απόφαση μπορεί να διαφέρει ανάλογα με τη φάση του κύκλου συλλογής, τις αρχικές τιμές των πεδίων του καινούριου αντικειμένου ή την πρόοδο της φάσης της εκκαθάρισης.

9.7.2 Ταυτόχρονη αντιγραφή και συμπύκνωση

Η ταυτόχρονη συλλογή με αντιγραφή και η ταυτόχρονη συλλογή με σήμανση και συμπύκνωση μειώνει τον κατακερματισμό της μνήμης και επίσης αποτρέπει την εμφάνιση μεγάλων χρόνων παύσης. Όπως και στις άλλες τεχνικές ταυτόχρονης συλλογής απορριμμάτων, ο συλλέκτης πρέπει να προστατεύεται από τις ενέργειες του τροποποιητή, ώστε να μην υπάρχουν μη ορατά στο συλλέκτη προσβάσιμα αντικείμενα. Όμως, επειδή ο συλλέκτης μετακινεί αντικείμενα πρέπει και ο τροποποιητής να προστατευθεί από τις ενέργειες του συλλέκτη, ώστε να μην προσπελάσει μη έγκυρα αντίγραφα αντικειμένων. Ο Baker [70] προστατεύει τον τροποποιητή εξασφαλίζοντας την διατήρηση μιας αναλλοίωτης του χώρου-προς, η οποία δεν επιτρέπει στον τροποποιητή τη διατήρηση δεικτών προς αντικείμενα του χώρου-από. Ο Brooks [29] προστατεύει τον τροποποιητή επιτρέποντάς του να συνεχίζει να λειτουργεί στο χώρο-από και αναθέτοντάς του την ενημέρωση των διευθύνσεων προώθησης προς αντικείμενα του χώρου-προς καθώς αυτά δημιουργούνται. Η συμπύκνωση μπορεί να πραγματοποιηθεί με παρόμοιους τρόπους χωρίς την απαίτηση να αντιγράφονται όλα τα αντικείμενα σε έναν κύκλο συλλογής.

Η εφαρμογή μιας εκ των των παραπάνω τεχνικών πάντως μπορεί να έχει ως αποτέλεσμα μεγαλύτερους χρόνους παύσης σε σύγκριση με μη-μετακινούντες ταυτόχρονους συλλέκτες: σε οποιαδήποτε προσπάθεια προσπέλασης του σωρού ο τροποποιητής μπορεί να παγιδευθεί και χρειασθεί να περιμένει μέχρις ότου ένα ή περισσότερα αντικείμενα μετακινηθούν.

9.7.3 Ταυτόχρονη καταμέτρηση αναφορών

Το βασικό πρόβλημα της καταμέτρησης αναφορών σχετίζεται με το πως θα εξασφαλισθεί πώς οι μετρητές αναφορών των αντικειμένων έχουν σωστές τιμές ενώ ο γράφος αντικειμένων τροποποιείται ταυτόχρονα. Η απλούστερη λύση είναι να απαιτείται από τα νήματα-τροποποιητές να κλειδώνουν ένα αντικείμενο πριν το τροποποιήσουν. Συνήθως όμως το κόστος των κλειδωμάτων θεωρείται υψηλό και συνεπώς πρέπει να βρεθούν εναλλακτικές προσεγγίσεις. Οι τρέχουσες λύσεις βασίζονται στην αποφυγή των καταστάσεων συναγωνισμού μεταξύ νημάτων-τροποποιητών που μπορεί να διακινδυνεύσουν την συνέπεια των μετρητών αναφορών. Ο διαχειριστής μνήμης ενδιαφέρεται μόνο να εξασφαλίσει τη συνέπεια του σωρού: δεν είναι δική του ευθύνη η εξασφάλιση της ορθότητας του προγράμματος χρήστη.

Για να διατηρηθεί η συνοχή του σωρού, απαιτείται η σειριοποίηση των εγγραφών δεικτών και των λειτουργιών ενημέρωσης των μετρητών αναφορών. Μια μερική λύση είναι η χρήση της καταμέτρησης αναφορών με αναβολή, καθώς έτσι αναβάλλεται η αποδέσμευση της μνήμης των απορριμμάτων η οποία μάλιστα ανατίθεται σε ένα ξεχωριστό νήμα-συλλέκτη. Ωστόσο η λύση αυτή αφορά μόνο τις φορτώσεις και αποθηκεύσεις δεικτών και όχι τις εγγραφές σε πεδία δείκτες στο εσωτερικό αντικειμένων. Προκύπτει έτσι το ερώτημα του πώς μπορούν οι ενημερώσεις μετρητών αναφοράς που απορρέουν από τις εγγραφές πεδίων δεικτών να ανατεθούν από τα νήματα-τροποποιητές σε ένα νήμα-συλλέκτη. Η απλούστερη λύση είναι κάθε νήμα- τροποποιητής να απομονώνει τις λειτουργίες καταμέτρησης αναφορών του και να τις περνά περιοδικά στο νήμα-συλλέκτη. Η καταμέτρηση αναφορών με συγκέντρωση πηγαίνει την παραπάνω ιδέα ένα βήμα πιο πέρα: λαμβάνοντας ένα στιγμιότυπο των αντικειμένων πριν αυτά τροποποιηθούν επιτρέπει στο νήμα-συλλέκτη να αποφύγει την εφαρμογή περιττών ενημερώσεων των μετρητών αναφορών. Και στις δύο περιπτώσεις πάντως η διαχείριση των μετρητών αναφορών και η αποδέσμευση αντικειμένων διαχωρίζεται πλήρως από τις εγγραφές δεικτών και πραγματοποιείται εξ' ολοκλήρου από ένα ξεχωριστό συλλέκτη ή και πολλαπλά νήματα-συλλέκτες που τρέχουν παράλληλα. Η λήψη ενός στιγμιοτύπου της κατάστασης του σωρού επίσης απλοποιεί την κυκλική καταμέτρηση αναφορών. Οι αλγόριθμοι δοκιμαστικής διαγραφής πρέπει να διασχίσουν έναν υπογράφο του σωρού πολλές φορές. Διασχίζοντας το στιγμιότυπο, ο συλλέκτης μπορεί να εξασφαλίσει πως εξερευνά τον ίδιο υπογράφο κάθε φορά παρά τις ταυτόχρονες δραστηριότητες νημάτων-τροποποιητών.

Κεφάλαιο 10

Συλλογή απορριμμάτων πραγματικού-χρόνου

Οι ταυτόχρονοι και αυξητικοί αλγόριθμοι συλλογής απορριμμάτων που εξετάσαμε στο προηγούμενο κεφάλαιο προσπαθούν να μειώσουν τους χρόνους παύσης που αντιλαμβάνεται ο τροποποιητής, είτε με σπάσιμο του κύκλου συλλογής σε μικρά κβάντα χρόνου και την παρεμβολή της εκτέλεσης του συλλέκτη στην εκτέλεση του τροποποιητή για κάθε ένα από αυτά στον ίδιο επεξεργαστή, είτε με την ταυτόχρονη εκτέλεση του συλλέκτη σε έναν διαφορετικό επεξεργαστή. Πολλοί από τους αλγορίθμους αυτούς σχεδιάσθηκαν με στόχο την υποστήριξη εφαρμογών πραγματικού-χρόνου όπου οι παύσεις μεγάλης διάρκειας έχουν ως αποτέλεσμα τη δραματική μείωση της επίδοσης. Παρότι οι αρχικοί αυξητικοί και ταυτόχρονοι συλλέκτες κατηγοριοποιήθηκαν ως πραγματικού-χρόνου, αυτό είναι αληθές μόνο εάν τηρούνται κάποιες αυστηρές προϋποθέσεις. Ωστόσο, όπως αντιλαμβανόμαστε σήμερα τα συστήματα πραγματικού-χρόνου, κανείς από τους αλγορίθμους αυτούς δεν μπορεί να παρέχει ισχυρές εγγυήσεις προόδου στον τροποποιητή. Η πρόοδος του τροποποιητή δεν μπορεί πλέον να εγγυηθεί όταν ο τελευταίος πρέπει να αποκτήσει κάποιο κλείδωμα (κατά την εκτέλεση κάποιου φράγματος εγγραφής ή ανάγνωσης για παράδειγμα). Ακόμη χειρότερα, ένας διακοπτόμενος αλγόριθμος χρονοδρομολόγησης ενδέχεται τυχαία να δίνει μεγαλύτερη προτεραιότητα στη δρομολόγηση των ταυτόχρονων νημάτων-συλλεκτών. Η **συλλογή απορριμμάτων πραγματικού-χρόνου** οφείλει να δικαιολογεί με ακρίβεια τις παύσεις λειτουργίας των νημάτων-τροποποιητών, ενώ ταυτόχρονα εξασφαλίζει τη μη υπέρβαση των χωρικών ορίων.

10.1 Συστήματα πραγματικού-χρόνου

Τα **συστήματα πραγματικού-χρόνου** επιβάλλουν λειτουργικές προθεσμίες σε συγκεκριμένες εργασίες μιας εφαρμογής. Αυτά τα συστήματα πραγματικού-χρόνου θα πρέπει να αποκρίνονται σε γεγονότα εισόδου εντός ενός καθορισμένου χρονικού παραθύρου. Μια εργασία που αποτυγχάνει στην τήρηση μιας προθεσμίας μπορεί να υποβαθμίσει την υπηρεσία ή ακόμη χειρότερα να προκαλέσει την πλήρη κατάρρευση του συστήματος. Συνεπώς, ένα σύστημα πραγματικού-χρόνου οφείλει να είναι ορθό όχι μόνο όσον αφορά τη λογική της εφαρμογής, αλλά και όσον αφορά την αποκρισιμότητα της σε γεγονότα πραγματικού χρόνου.

Ένα **απαλό σύστημα πραγματικού-χρόνου** (όπως για παράδειγμα η απεικόνιση βίντεο) μπορεί να ανεχτεί χαμένες προθεσμίες εις βάρος της ποιότητας της υπηρεσίας. Ενώ

πολλές χαμένες προθεσμίες θα έχουν ως αποτέλεσμα μη αποδεκτή ποιότητα υπηρεσίας, μια συνηθισμένη χαμένη προθεσμία δεν την επηρεάζει πολύ. Ο Printezis [96] προτείνει ένα απαλό στόχο που προσδιορίζει ταυτόχρονα ένα μέγιστο χρόνο συλλογής, μία χρονική σχισμή συγκεκριμένης διάρκειας και έναν αποδεκτό ρυθμό αποτυχίας. Σε κάθε διάστημα εντός αυτής της χρονικής σχισμής, ο συλλέκτης πρέπει να αποφύγει να χρησιμοποιήσει περισσότερο χρόνο από το μέγιστο και όλες οι παραβιάσεις αυτού του στόχου πρέπει να είναι εντός του αποδεκτού ρυθμού αποτυχίας.

Ένας τέτοιος απαλός στόχος είναι ανεπαρκής για **σκληρά συστήματα πραγματικού-χρόνου** (όπως για παράδειγμα ο έλεγχος ενός κινητήρα), όπου οι χαμένες προθεσμίες σημαίνουν την αποτυχία του συστήματος. Ένα ορθώς σχεδιασμένο σκληρό σύστημα πραγματικού -χρόνου οφείλει να εξασφαλίζει την ικανοποίηση όλων των περιορισμών πραγματικού-χρόνου. Λαμβάνοντας υπόψη την παρουσία τέτοιων χρονικών περιορισμών, είναι σημαντικό η αποκρισιμότητα της συλλογής απορριμμάτων σε συστήματα πραγματικού χρόνου να μπορεί να χαρακτηρισθεί με τέτοιο τρόπο ώστε να απεικονίζει τόσο τις ανάγκες της εφαρμογής όσο και τη συμπεριφορά του συλλέκτη.

Η συνολική επίδοση ή ρυθμαπόδοση των συστημάτων πραγματικού-χρόνου είναι λιγότερο σημαντική από την **προβλεψιμότητα** της επίδοσης. Η χρονική συμπεριφορά μιας εργασίας πραγματικού-χρόνου θα πρέπει να μπορεί να προσδιορισθεί είτε αναλυτικά κατά τη σχεδίαση είτε εμπειρικά κατά την εκτέλεση ούτως ώστε ο χρόνος απόκρισης της να είναι γνωστός εξαρχής (σε κάποιο αποδεκτό βαθμό εμπιστοσύνης). Ο **χρόνος εκτέλεσης χειρότερης περίπτωσης** μιας εργασίας είναι ο μέγιστος χρόνος που μπορεί να χρειαστεί (αγνοώντας τυχόν επαναδρομολόγηση) για την απομονωμένη εκτέλεση της εργασίας σε μια συγκεκριμένη πλατφόρμα υλικού. Συστήματα πραγματικού-χρόνου που υποστηρίζουν την ταυτόχρονη εκτέλεση πολλών εργασιών πρέπει να δρομολογούν τις εργασίες με γνώμονα την ικανοποίηση των χρονικών περιορισμών των τελευταίων. Η γνώση σχετικά με την ικανοποίηση αυτών των περιορισμών απαιτεί την εκ των προτέρων πραγματοποίηση **ανάλυσης δρομολόγησης**, υποθέτοντας ένα συγκεκριμένο αλγόριθμο χρονοδρομολόγησης (συνήθως βασισμένο σε προτεραιότητες).

Οι εφαρμογές πραγματικού-χρόνου συχνά τρέχουν σε εξειδικευμένα ενσωματωμένα συστήματα. Οι αλγόριθμοι αυξητικής συλλογής απορριμμάτων είναι κατάλληλοι για χρήση σε ενσωματωμένα συστήματα με μονοπύρηνο επεξεργαστή, ενώ οι αλγόριθμοι παράλληλης και ταυτόχρονης συλλογής απορριμμάτων είναι πιο κατάλληλοι για χρήση σε ενσωματωμένα συστήματα με πολυπύρηνο επεξεργαστή. Σε κάθε περίπτωση πάντως, τα ενσωματωμένα συστήματα συχνά επιβάλλουν στενότερους χωρικούς περιορισμούς σε σύγκριση με τις πλατφόρμες γενικού σκοπού.

Για όλους τους παραπάνω λόγους, οι αλγόριθμοι συλλογής με παύση του κόσμου, παράλληλης συλλογής ή ακόμη και ταυτόχρονης συλλογής που επιβάλλει όμως μη προβλέψιμους χρόνους παύσης δεν είναι κατάλληλοι για χρήση σε εφαρμογές πραγματικού -χρόνου. Ας θεωρήσουμε για παράδειγμα τη δρομολόγηση συλλογής του σχήματος 10.1, η οποία προκύπτει όταν η προσπάθεια ανάκτησης μνήμης εξαρτάται από το συνολικό πλήθος και μέγεθος των αντικειμένων που χρησιμοποιεί η εφαρμογή, τις διασυνδέσεις μεταξύ των αντικειμένων αυτών καθώς και το βαθμό δυσκολίας ανάκτησης αρκετής μνήμης για την ικανοποίηση μελλοντικών αιτημάτων. Δοθείσης μιας τέτοιας δρομολόγησης, ο τροποποιητής δεν έχει κάποια εγγύηση προβλέψιμης και βιώσιμης χρησιμοποίησης του επεξεργαστή.

Σχήμα 10.1: Μη προβλέψιμη συχνότητα και διάρκεια εκτέλεσης συμβατικών συλλεκτών απορριμμάτων. Οι παύσεις λόγω συλλογής απεικονίζονται με γκρι χρώμα.

10.2 Δρομολόγηση συλλογής πραγματικού-χρόνου

Ο χρόνος και ο τρόπος πυροδότησης της εκτέλεσης εργασιών συλλογής απορριμμάτων είναι οι βασικοί παράγοντες που επηρεάζουν την επίδραση του συλλέκτη στον τροποποιητή. Οι αλγόριθμοι συλλογής με παύση του κόσμου αναβάλλουν όλες τις εργασίες συλλογής μέχρις ότου η μη ικανοποίηση ενός αιτήματος εκχώρησης μνήμης εντοπίσει την έλλειψη χώρου. Οι αλγόριθμοι αυξητικής συλλογής επιβαρύνουν τις λειτουργίες εγγραφής και ανάγνωσης του τροποποιητή καθώς και τις λειτουργίες εκχώρησης με ενέργειες συλλογής. Οι αλγόριθμοι ταυτόχρονης συλλογής τέλος πυροδοτούν την εκτέλεση ενός μικρού φορτίου συλλογής ταυτόχρονα (πιθανώς και παράλληλα) με την εκτέλεση του τροποποιητή, χρησιμοποιώντας φράγματα τροποποίησης για το συγχρονισμό του συλλέκτη με τον τελευταίο. Για να διατηρηθεί μια κατανάλωση χώρου σταθερής κατάστασης, ο συλλέκτης πρέπει να ελευθερώνει και ανακυκλώνει νεκρά αντικείμενα με τον ίδιο ρυθμό που ο τροποποιητής δημιουργεί καινούρια αντικείμενα. Πιθανός κατακερματισμός της μνήμης οδηγεί σε σπατάλη χώρου και έχει ως αποτέλεσμα την αδυναμία ικανοποίησης αιτημάτων εκχώρησης από τον εκχωρητή, εκτός και αν ο συλλέκτης συμπυκνώσει το σωρό. Η μεταφορά αντικειμένων ωστόσο που προκαλεί η συμπύκνωση, είναι ακριβή διαδικασία και μπορεί να επηρεάσει δυσμενώς την προσπάθεια ικανοποίησης περιορισμών πραγματικού-χρόνου.

Οι Henriksson [59], Detlefs [39], Cheng και Blelloch [35] καθώς και οι Pizlo και Vitek [94], προτείνουν εναλλακτικές τεχνικές για τη δρομολόγηση της συλλογής απορριμμάτων σε συστήματα πραγματικού-χρόνου, καθώς και για το χαρακτηρισμό του πώς αυτή επηρεάζει την εκτέλεση του τροποποιητή.

Η **δρομολόγηση με βάση την εργασία** επιβάλλει έργο συλλογής ως φόρο στις μονάδες εργασίας του τροποποιητή.

Η **δρομολόγηση με βάση την αδράνεια** τρέχει έργο συλλογής στη διάρκεια των χρονικών διαστημάτων αδράνειας των νημάτων- τροποποιητών πραγματικού-χρόνου. Αυτό επιτυγχάνεται εύκολα σε ένα σύστημα με χρονοδρομολόγηση προτεραιοτήτων, σημαίνοντας τα νήματα- συλλέκτες με τη χαμηλότερη δυνατή προτεραιότητα.

Η **δρομολόγηση με βάση το χρόνο** δεσμεύει ένα προκαθορισμένο διάστημα του χρόνου εκτέλεσης για την εκτέλεση του εργασιών συλλογής, κατά τη διάρκεια του οποίου αναστέλλεται η εκτέλεση του τροποποιητή. Με τον τρόπο αυτό παρέχεται κάποιας μορφής εγγύηση σχετικά με μια προκαθορισμένη απαίτηση ελάχιστης χρησιμοποίησης τροποποιητή.

Παράρτημα Α΄

Μεταφράσεις Ξένων Όρων

Μετάφραση	Αγγλικός Όρος
συλλογή απορριμμάτων	garbage collection
ανακύκλωση	recycling
σήμανση και εκκαθάριση	mark-sweep
σήμανση και συμπύκνωση	mark-compact
αντιγραφή	copying
καταμέτρηση αναφορών	reference counting
παράλληλη	parallel
ταυτόχρονη	concurrent
πραγματικού-χρόνου	real-time
εγγράφημα δραστηριοποίησης	activation record
ρητή αποδέσμευση	explicit deallocation
σύστημα εκτέλεσης	run-time system
ξεκρέμαστος δείκτης	dangling pointer
διαρροή μνήμης	memory leak
ιδιοκτησία	ownership
ψηφίδα	module
συντηρητική	conservative
ρυθμαπόδοση	throughput
ασφαλής	safe
πλήρης	complete
αιωρούμενα απορρίμματα	floating garbage
παύση του κόσμου	stop-the-world
χρόνος παύσης	pause time
ελάχιστη χρησιμοποίηση τροποποιητή	minimum mutator utilization
φραγμένη χρησιμοποίηση τροποποιητή	bounded mutator utilization
χωρικό κόστος	space overhead
κλιμακωσιμότητα	scalability
μεταφερσιμότητα	portability
σωρός	heap
αντικείμενο	object
πεδίο	field
αναφορά	reference
επικεφαλίδα	header

γράφος αντικειμένων	object graph
ρίζα	root
εκχωρητής	allocator
συλλέκτης	collector
τροποποιητής	mutator
μονονηματικός	single-threaded
πολυνηματικός	multi-threaded
άμεσος	direct
έμμεσος	indirect
ζωντανό	live
προσβασιμότητα μέσω δεικτών	pointer reachability
νεκρό	dead
φράγμα ανάγνωσης	read barrier
φράγμα εγγραφής	write barrier
ατομικός	atomic
εξιχνίαση	tracing
τριχρωματική αφαίρεση	tricolour abstraction
οκνηρή εκκαθάριση	lazy sweeping
τυχαίος	arbitrary
γραμμικοποιών	linearizing
ολισθαίνων	sliding
χώρος-από	fromspace
χώρος-προς	tospace
εφεδρικός	backup
ιεραρχική αποσύνθεση	hierarchical decomposition
επαναδιάταξη αντικειμένων	object reordering
καταμέτρηση αναφορών με αναβολή	deferred reference counting
καταμέτρηση αναφορών με συγκέντρωση	coalesced reference counting
κυκλική καταμέτρηση αναφορών	cyclic reference counting
δοκιμαστική διαγραφή	trial deletion
μαζικά	en masse
μείζων	major
ελάσσων	minor
ημιχώρος γήρανσης	aging semispace
συστοιχία κάδων	bucket brigade
εδέμ	eden
ημιχώρος επιβίωσης	survivor semispace
δημογραφική προαγωγή με ανάδραση	demographic feedback-mediated tenuring
ενδιαφέρων δείκτης	interesting pointer
εξισορρόπηση φορτίου	load balancing
δυναμικός διαμοιρασμός	dynamic balancing
κλοπή εργασίας	work stealing
κλεπτόμενη ουρά εργασιών	stealable work queue
διπλά τερματισμένη ουρά	deque (double-ended queue)
διακοπτόμενος εντοπισμός	interrupted-detection
ταυτότητα ανιχνευτή	detector-identity
επαναληπτική	replicating
εξιχνίαση κυρίαρχου νήματος	dominant thread tracing
κομμάτι	chunk

μπλοκ	block
μονάδα	unit
αυξητική	incremental
σχεδόν-ταυτόχρονη	mostly-concurrent
εν-τη-πτήσει	on-the-fly
φράγμα διαγραφής	deletion barrier
χειραψία	handshake
καταδικασμένη	condemned
καταμέτρηση αναφορών με απομόνωση	buffered reference counting
ολισθαίνουσες όψεις	sliding views
απαλό	soft
σκληρό	hard
προβλεψιμότητα	predictability
ανάλυση δρομολόγησης	schedulability analysis
με βάση την εργασία	work-based
με βάση την αδράνεια	slack-based
με βάση το χρόνο	time-based

Βιβλιογραφία

[1] Santosh G. Abraham and Janak H. Patel. Parallel garbage collection on a virtual memory system. In *International Conference on Parallel Processing, ICPP'87, University Park, PA, USA, August 1987.*, pages 243–246. Pennsylvania State University Press, 1987.

[2] Diab Abuaiadh, Yoav Ossia, Erez Petrank, and Uri Silbershtein. An efficient parallel heap compaction algorithm. In John M. Vlissides and Douglas C. Schmidt, editors, *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2004, October 24-28, 2004, Vancouver, BC, Canada*, pages 224–236. ACM, 2004.

[3] Andrew W. Appel. Garbage collection can be faster than stack allocation. *Inf. Process. Lett.*, 25(4):275–279, 1987.

[4] Andrew W. Appel. Allocation without locking. *Softw., Pract. Exper.*, 19(7):703–705, 1989.

[5] Andrew W. Appel, John R. Ellis, and Kai Li. Real-time concurrent collection on stock multiprocessors. In Richard L. Wexelblat, editor, *Proceedings of the ACM SIGPLAN'88 Conference on Programming Language Design and Implementation (PLDI), Atlanta, Georgia, USA, June 22-24, 1988*, pages 11–20. ACM, 1988.

[6] Tom Axford. Reference counting of cyclic graphs for functional programs. *Comput. J.*, 33(5):466–470, 1990.

[7] Hezi Azatchi, Yossi Levanoni, Harel Paz, and Erez Petrank. An on-the-fly mark and sweep garbage collector based on sliding views. In Ron Crocker and Guy L. Steele Jr., editors, *Proceedings of the 2003 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications, OOPSLA 2003, October 26-30, 2003, Anaheim, CA, USA*, pages 269–281. ACM, 2003.

[8] David F. Bacon, C. Richard Attanasio, Han Bok Lee, V. T. Rajan, and Stephen E. Smith. Java without the coffee breaks: A nonintrusive multiprocessor garbage collector. In Michael Burke and Mary Lou Soffa, editors, *Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Snowbird, Utah, USA, June 20-22, 2001*, pages 92–103. ACM, 2001.

[9] David F. Bacon, Perry Cheng, and V. T. Rajan. Controlling fragmentation and space consumption in the metronome, a real-time garbage collector for java. In Frank Mueller and Ulrich Kremer, editors, *Proceedings of the 2003 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'03). San Diego, California, USA, June 11-13, 2003*, pages 81–92. ACM, 2003.

[10] David F. Bacon, Perry Cheng, and V. T. Rajan. A unified theory of garbage collection. In John M. Vlissides and Douglas C. Schmidt, editors, *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2004, October 24-28, 2004, Vancouver, BC, Canada*, pages 50–68. ACM, 2004.

[11] David F. Bacon and V. T. Rajan. Concurrent cycle collection in reference counted systems. In Jørgen Lindskov Knudsen, editor, *ECOOP 2001 - Object-Oriented Programming, 15th European Conference, Budapest, Hungary, June 18-22, 2001, Proceedings*, volume 2072 of *Lecture Notes in Computer Science*, pages 207–235. Springer, 2001.

[12] Scott B Baden. Low-overhead storage reclamation in the smalltalk-80 virtual machine. *Smalltalk-80: bits of history, words of advice*, 11669:331, 1983.

[13] Katherine Barabash, Ori Ben-Yitzhak, Irit Goft, Elliot K. Kolodner, Victor Leikehman, Yoav Ossia, Avi Owshanko, and Erez Petrank. A parallel, incremental, mostly concurrent garbage collector for servers. *ACM Trans. Program. Lang. Syst.*, 27(6):1097–1146, 2005.

[14] David A. Barrett and Benjamin G. Zorn. Using lifetime predictors to improve memory allocation performance. In Robert Cartwright, editor, *Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation (PLDI), Albuquerque, New Mexico, USA, June 23-25, 1993*, pages 187–196. ACM, 1993.

[15] David A. Barrett and Benjamin G. Zorn. Garbage collection using a dynamic threatening boundary. In David W. Wall, editor, *Proceedings of the ACM SIGPLAN'95 Conference on Programming Language Design and Implementation (PLDI), La Jolla, California, USA, June 18-21, 1995*, pages 301–314. ACM, 1995.

[16] George Belotsky. C++ memory management: From fear to triumph. O'Reilly linuxdevcenter.com, 2003.

[17] Stephen Blackburn and Kathryn S. McKinley. Ulterior reference counting: fast garbage collection without a long wait. In Ron Crocker and Guy L. Steele Jr., editors, *Proceedings of the 2003 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications, OOPSLA 2003, October 26-30, 2003, Anaheim, CA, USA*, pages 344–358. ACM, 2003.

[18] Stephen M. Blackburn, Perry Cheng, and Kathryn S. McKinley. Oil and water? high performance garbage collection in java with mmtk. In Anthony Finkelstein, Jacky Estublier, and David S. Rosenblum, editors, *26th International Conference on Software Engineering (ICSE 2004), 23-28 May 2004, Edinburgh, United Kingdom*, pages 137–146. IEEE Computer Society, 2004.

[19] Stephen M. Blackburn, Robin Garner, Chris Hoffmann, Asjad M. Khan, Kathryn S. McKinley, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z. Guyer, Martin Hirzel, Antony L. Hosking, Maria Jump, Han Bok Lee, J. Eliot B. Moss, Aashish Phansalkar, Darko Stefanovic, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann. The dacapo benchmarks: java benchmarking development and analysis. In Peri L. Tarr and William R. Cook,

editors, *Proceedings of the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22-26, 2006, Portland, Oregon, USA*, pages 169–190. ACM, 2006.

[20] Stephen M. Blackburn and Antony L. Hosking. Barriers: friend or foe? In David F. Bacon and Amer Diwan, editors, *Proceedings of the 4th International Symposium on Memory Management, ISMM 2004, Vancouver, BC, Canada, October 24-25, 2004*, pages 143–151. ACM, 2004.

[21] Stephen M. Blackburn and Kathryn S. McKinley. Immix: a mark-region garbage collector with space efficiency, fast collection, and mutator performance. In Rajiv Gupta and Saman P. Amarasinghe, editors, *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation, Tucson, AZ, USA, June 7-13, 2008*, pages 22–32. ACM, 2008.

[22] Ricki Blau. Paging on an object-oriented personal computer for smalltalk. *Perform. Eval.*, 3(4):313, 1983.

[23] Guy E. Blelloch and Perry Cheng. On bounding time and space for multiprocessor garbage collection. In Barbara G. Ryder and Benjamin G. Zorn, editors, *Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Atlanta, Georgia, USA, May 1-4, 1999*, pages 104–117. ACM, 1999.

[24] Hans-J Boehm, Paul F Dubois, et al. Dynamic memory allocation and garbage collection. *Computers in Physics*, 9(3):297–303, 1995.

[25] Hans-Juergen Boehm. Reducing garbage collector cache misses. In Craig Chambers and Antony L. Hosking, editors, *ISMM 2000, International Symposium on Memory Management, Minneapolis, Minnesota, October 15-16, 2000 (in conjunction with OOPSLA 2000), Conference Proceedings*, pages 59–64. ACM, 2000.

[26] Hans-Juergen Boehm. The space cost of lazy reference counting. In Neil D. Jones and Xavier Leroy, editors, *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*, pages 210–219. ACM, 2004.

[27] Hans-Juergen Boehm, Alan J. Demers, and Scott Shenker. Mostly parallel garbage collection. In David S. Wise, editor, *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation (PLDI), Toronto, Ontario, Canada, June 26-28, 1991*, pages 157–164. ACM, 1991.

[28] Hans-Juergen Boehm and Mark Weiser. Garbage collection in an uncooperative environment. *Softw., Pract. Exper.*, 18(9):807–820, 1988.

[29] Rodney A. Brooks. Trading data space for reduced time and code space in real-time garbage collection on stock hardware. In *LISP and Functional Programming*, pages 256–262, 1984.

[30] David R. Brownbridge. Cyclic reference counting for combinator machines. In *FPCA*, pages 273–288, 1985.

[31] Albin M. Butters. Total cost of ownership: A comparison of C/C++ and Java. Technical report, Evans Data Corporation, June 2007.

[32] Brad Calder, Chandra Krintz, Simmi John, and Todd M. Austin. Cache-conscious data placement. In Dileep Bhandarkar and Anant Agarwal, editors, *ASPLOS-VIII Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, California, USA, October 3-7, 1998.*, pages 139–149. ACM Press, 1998.

[33] Patrick J. Caudill and Allen Wirfs-Brock. A third generation smalltalk-80 implementation. In Norman K. Meyrowitz, editor, *Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'86), Portland, Oregon, Proceedings.*, pages 119–130. ACM, 1986.

[34] C. J. Cheney. A nonrecursive list compacting algorithm. *Commun. ACM*, 13(11):677–678, 1970.

[35] Perry Cheng and Guy E. Blelloch. A parallel, real-time garbage collector. In Michael Burke and Mary Lou Soffa, editors, *Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Snowbird, Utah, USA, June 20-22, 2001*, pages 125–136. ACM, 2001.

[36] Trishul M. Chilimbi, Mark D. Hill, and James R. Larus. Cache-conscious structure layout. In Barbara G. Ryder and Benjamin G. Zorn, editors, *Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Atlanta, Georgia, USA, May 1-4, 1999*, pages 1–12. ACM, 1999.

[37] Jacques Cohen and Alexandru Nicolau. Comparison of compacting algorithms for garbage collection. *ACM Trans. Program. Lang. Syst.*, 5(4):532–553, 1983.

[38] George E. Collins. A method for overlapping and erasure of lists. *Commun. ACM*, 3(12):655–657, 1960.

[39] David Detlefs. A hard look at hard real-time garbage collection. In *7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2004), 12-14 May 2004, Vienna, Austria*, pages 23–32. IEEE Computer Society, 2004.

[40] David Detlefs, Paul Alan Martin, Mark Moir, and Guy L. Steele Jr. Lock-free reference counting. In Ajay D. Kshemkalyani and Nir Shavit, editors, *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, PODC 2001, Newport, Rhode Island, USA, August 26-29, 2001*, pages 190–199. ACM, 2001.

[41] John DeTreville. Experience with garbage collection for modula-2+ in the topaz environment. In *OOPSLA/ECOOP*, volume 90. Citeseer, 1990.

[42] L. Peter Deutsch and Daniel G. Bobrow. An efficient, incremental, automatic garbage collector. *Commun. ACM*, 19(9):522–526, 1976.

[43] Sylvia Dieckmann and Urs Hölzle. A study of the allocation behavior of the specjvm98 java benchmark. In Rachid Guerraoui, editor, *ECOOP'99 - Object-Oriented Programming, 13th European Conference, Lisbon, Portugal, June 14-18, 1999, Proceedings*, volume 1628 of *Lecture Notes in Computer Science*, pages 92–115. Springer, 1999.

[44] Edsger W. Dijkstra, Leslie Lamport, Alain J. Martin, Carel S. Scholten, and Elisabeth F. M. Steffens. On-the-fly garbage collection: an exercise in cooperation. In Friedrich L. Bauer and Klaus Samelson, editors, *Language Hierarchies and Interfaces, International Summer School, Marktoberdorf, Germany, July 23 - August 2, 1975*, volume 46 of *Lecture Notes in Computer Science*, pages 43–56. Springer, 1975.

[45] Edsger W. Dijkstra, Leslie Lamport, Alain J. Martin, Carel S. Scholten, and Elisabeth F. M. Steffens. On-the-fly garbage collection: An exercise in cooperation. *Commun. ACM*, 21(11):966–975, 1978.

[46] Robert T. Dimpsey, Rajiv Arora, and Kean Kuiper. Java server performance: A case study of building efficient, scalable jvms. *IBM Systems Journal*, 39(1):151–174, 2000.

[47] Damien Doligez and Georges Gonthier. Portable, unobtrusive garbage collection for multiprocessor systems. In Hans-Juergen Boehm, Bernard Lang, and Daniel M. Yellin, editors, *Conference Record of POPL'94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon, USA, January 17-21, 1994*, pages 70–83. ACM Press, 1994.

[48] Damien Doligez and Xavier Leroy. A concurrent, generational garbage collector for a multithreaded implementation of ML. In Mary S. Van Deusen and Bernard Lang, editors, *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Charleston, South Carolina, USA, January 1993*, pages 113–123. ACM Press, 1993.

[49] Daniel R. Edelson. Precompiling C++ for garbage collection. In Yves Bekkers and Jacques Cohen, editors, *Memory Management, International Workshop IWMM 92, St. Malo, France, September 17-19, 1992, Proceedings*, volume 637 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 1992.

[50] Toshio Endo, Kenjiro Taura, and Akinori Yonezawa. A scalable mark-sweep garbage collector on large-scale shared-memory machines. In *Proceedings of the ACM/IEEE Conference on Supercomputing, SC 1997, November 15-21, 1997, San Jose, CA, USA*, page 48. IEEE, 1997.

[51] Robert Fenichel and Jerome C. Yochelson. A LISP garbage-collector for virtual-memory computer systems. *Commun. ACM*, 12(11):611–612, 1969.

[52] Robert P. Fitzgerald and David Tarditi. The case for profile-directed selection of garbage collectors. In Craig Chambers and Antony L. Hosking, editors, *ISMM 2000, International Symposium on Memory Management, Minneapolis, Minnesota, October 15-16, 2000 (in conjunction with OOPSLA 2000), Conference Proceedings*, pages 111–120. ACM, 2000.

[53] Christine H. Flood, David Detlefs, Nir Shavit, and Xiolan Zhang. Parallel garbage collection for shared memory multiprocessors. In Saul Wold, editor, *Proceedings of the 1st Java Virtual Machine Research and Technology Symposium, April 23-24, 2001, Monterey, CA, USA*. USENIX, 2001.

[54] John K. Foderaro and Richard J. Fateman. Characterization of VAX macsyma. In Paul S. Wang, editor, *SYMSAC 1981, Proceedings of the Symposium on Symbolic*

and Algebraic Manipulation, Snowbird, Utah, USA, August 5-7, 1981, pages 14–19. ACM, 1981.

[55] Daniel P. Friedman and David S. Wise. Reference counting can manage the circular environments of mutual recursion. *Inf. Process. Lett.*, 8(1):41–45, 1979.

[56] Robin Garner, Stephen M. Blackburn, and Daniel Frampton. Effective prefetch for mark-sweep garbage collection. In Greg Morrisett and Mooly Sagiv, editors, *Proceedings of the 6th International Symposium on Memory Management, ISMM 2007, Montreal, Quebec, Canada, October 21-22, 2007*, pages 43–54. ACM, 2007.

[57] David R. Hanson. Storage management for an implementation of SNOBOL4. *Softw., Pract. Exper.*, 7(2):179–192, 1977.

[58] Barry Hayes. Using key object opportunism to collect old objects. In Andreas Paepcke, editor, *Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'91), Sixth Annual Conference, Phoenix, Arizona, USA, October 6-11, 1991, Proceedings.*, pages 33–46. ACM, 1991.

[59] Roger Henriksson. *Scheduling garbage collection in embedded systems.* PhD thesis, Lund University, 1998.

[60] Matthew Hertz and Emery D. Berger. Quantifying the performance of garbage collection vs. explicit memory management. In Ralph E. Johnson and Richard P. Gabriel, editors, *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2005, October 16-20, 2005, San Diego, CA, USA*, pages 313–326. ACM, 2005.

[61] Antony L. Hosking and Jiawan Chen. Mostly-copying reachability-based orthogonal persistence. In Brent Hailpern, Linda M. Northrop, and A. Michael Berman, editors, *Proceedings of the 1999 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA '99), Denver, Colorado, USA, November 1-5, 1999.*, pages 382–398. ACM, 1999.

[62] Antony L. Hosking and J. Eliot B. Moss. Protection traps and alternatives for memory management of an object-oriented language. In *SOSP*, pages 106–119, 1993.

[63] Xianglong Huang, Stephen M. Blackburn, Kathryn S. McKinley, J. Eliot B. Moss, Zhenlin Wang, and Perry Cheng. The garbage collection advantage: improving program locality. In John M. Vlissides and Douglas C. Schmidt, editors, *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2004, October 24-28, 2004, Vancouver, BC, Canada*, pages 69–80. ACM, 2004.

[64] R John M Hughes. A semi-incremental garbage collection algorithm. *Software: Practice and Experience*, 12(11):1081–1082, 1982.

[65] Akira Imai and Evan Tick. Evaluation of parallel copying garbage collection on a shared-memory multiprocessor. *IEEE Trans. Parallel Distrib. Syst.*, 4(9):1030–1040, 1993.

[66] Richard E. Jones, Antony L. Hosking, and J. Eliot B. Moss. *The Garbage Collection Handbook: The art of automatic memory management.* CRC Press, 2011.

[67] Richard E. Jones and Rafael Dueire Lins. *Garbage collection - algorithms for automatic dynamic memory management*. Wiley, 1996.

[68] Richard E. Jones and Chris Ryder. A study of java object demographics. In Richard E. Jones and Stephen M. Blackburn, editors, *Proceedings of the 7th International Symposium on Memory Management, ISMM 2008, Tucson, AZ, USA, June 7-8, 2008*, pages 121–130. ACM, 2008.

[69] Guy L. Steele Jr. Multiprocessing compactifying garbage collection. *Commun. ACM*, 18(9):495–508, 1975.

[70] Henry G. Baker Jr. List processing in real time on a serial computer. *Commun. ACM*, 21(4):280–294, 1978.

[71] Robert H. Halstead Jr. Implementation of multilisp: Lisp on a multiprocessor. In *LISP and Functional Programming*, pages 9–17, 1984.

[72] Robert H. Halstead Jr. Multilisp: A language for concurrent symbolic computation. *ACM Trans. Program. Lang. Syst.*, 7(4):501–538, 1985.

[73] Haim Kermany and Erez Petrank. The compressor: concurrent, incremental, and parallel compaction. In Michael I. Schwartzbach and Thomas Ball, editors, *Proceedings of the ACM SIGPLAN 2006 Conference on Programming Language Design and Implementation, Ottawa, Ontario, Canada, June 11-14, 2006*, pages 354–363. ACM, 2006.

[74] Michael S. Lam, Paul R. Wilson, and Thomas G. Moher. Object type directed garbage collection to improve locality. In Yves Bekkers and Jacques Cohen, editors, *Memory Management, International Workshop IWMM 92, St. Malo, France, September 17-19, 1992, Proceedings*, volume 637 of *Lecture Notes in Computer Science*, pages 404–425. Springer, 1992.

[75] Leslie Lamport. Garbage collection with multiple processes: an exercise in parallelism. In *Proc. of the 1976 International Conference on Parallel Processing*, pages 50–54, 1976.

[76] Joseph Levanoni and Erez Petrank. A scalable reference counting garbage collector. 1999.

[77] Yossi Levanoni and Erez Petrank. An on-the-fly reference counting garbage collector for java. In Linda M. Northrop and John M. Vlissides, editors, *Proceedings of the 2001 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications, OOPSLA 2001, Tampa, Florida, USA, October 14-18, 2001.*, pages 367–380. ACM, 2001.

[78] Yossi Levanoni and Erez Petrank. An on-the-fly reference-counting garbage collector for java. *ACM Trans. Program. Lang. Syst.*, 28(1):1–69, 2006.

[79] Henry Lieberman and Carl Hewitt. A real-time garbage collector based on the lifetimes of objects. *Commun. ACM*, 26(6):419–429, 1983.

[80] Simon Marlow, Tim Harris, Roshan P. James, and Simon L. Peyton Jones. Parallel generational-copying garbage collection with a block-structured heap. In Richard E. Jones and Stephen M. Blackburn, editors, *Proceedings of the 7th International*

Symposium on Memory Management, ISMM 2008, Tucson, AZ, USA, June 7-8, 2008, pages 11–20. ACM, 2008.

[81] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part I. *Commun. ACM*, 3(4):184–195, 1960.

[82] Phil McGachey and Antony L. Hosking. Reducing generational copy reserve overhead with fallback compaction. In Erez Petrank and J. Eliot B. Moss, editors, *Proceedings of the 5th International Symposium on Memory Management, ISMM 2006, Ottawa, Ontario, Canada, June 10-11, 2006*, pages 17–28. ACM, 2006.

[83] David A. Moon. Garbage collection in a large lisp system. In *LISP and Functional Programming*, pages 235–246, 1984.

[84] Takeshi Ogasawara. Numa-aware memory manager with dominant-thread-based copying GC. In Shail Arora and Gary T. Leavens, editors, *Proceedings of the 24th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2009, October 25-29, 2009, Orlando, Florida, USA*, pages 377–390. ACM, 2009.

[85] Yoav Ossia, Ori Ben-Yitzhak, Irit Goft, Elliot K. Kolodner, Victor Leikehman, and Avi Owshanko. A parallel, incremental and concurrent GC for servers. In Jens Knoop and Laurie J. Hendren, editors, *Proceedings of the 2002 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Berlin, Germany, June 17-19, 2002*, pages 129–140. ACM, 2002.

[86] Yoav Ossia, Ori Ben-Yitzhak, and Marc Segal. Mostly concurrent compaction for mark-sweep GC. In David F. Bacon and Amer Diwan, editors, *Proceedings of the 4th International Symposium on Memory Management, ISMM 2004, Vancouver, BC, Canada, October 24-25, 2004*, pages 25–36. ACM, 2004.

[87] Harel Paz, David F. Bacon, Elliot K. Kolodner, Erez Petrank, and V. T. Rajan. An efficient on-the-fly cycle collection. *ACM Trans. Program. Lang. Syst.*, 29(4), 2007.

[88] Harel Paz and Erez Petrank. Using prefetching to improve reference-counting garbage collectors. In Shriram Krishnamurthi and Martin Odersky, editors, *Compiler Construction, 16th International Conference, CC 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, March 26-30, 2007, Proceedings*, volume 4420 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 2007.

[89] Harel Paz, Erez Petrank, David F. Bacon, Elliot K. Kolodner, and V. T. Rajan. An efficient on-the-fly cycle collection. In Rastislav Bodík, editor, *Compiler Construction, 14th International Conference, CC 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3443 of *Lecture Notes in Computer Science*, pages 156–171. Springer, 2005.

[90] EJH Pepels, MCJD van Eekelen, and Marinus Jacobus Plasmeijer. *A cyclic reference counting algorithm and its proof*. Department of Theoretical Computer Science and Computational Models, Faculty of Science, University of Nijmegen, 1988.

[91] Erez Petrank and Elliot K. Kolodner. Parallel copying garbage collection using delayed allocation. *Parallel Processing Letters*, 14(2):271–286, 2004.

[92] Erez Petrank and Dror Rawitz. The hardness of cache conscious data placement. In John Launchbury and John C. Mitchell, editors, *Conference Record of POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, OR, USA, January 16-18, 2002*, pages 101–112. ACM, 2002.

[93] Pekka P. Pirinen. Barrier techniques for incremental tracing. In Simon L. Peyton Jones and Richard E. Jones, editors, *International Symposium on Memory Management, ISMM '98, Vancouver, British Columbia, Canada, 17-19 October, 1998, Conference Proceedings*, pages 20–25. ACM, 1998.

[94] Filip Pizlo, Erez Petrank, and Bjarne Steensgaard. A study of concurrent real-time garbage collectors. In Rajiv Gupta and Saman P. Amarasinghe, editors, *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation, Tucson, AZ, USA, June 7-13, 2008*, pages 33–44. ACM, 2008.

[95] Tony Printezis. Hot-swapping between a mark&sweep and a mark&compact garbage collector in a generational environment. In Saul Wold, editor, *Proceedings of the 1st Java Virtual Machine Research and Technology Symposium, April 23-24, 2001, Monterey, CA, USA*. USENIX, 2001.

[96] Tony Printezis. On measuring garbage collection responsiveness. *Sci. Comput. Program.*, 62(2):164–183, 2006.

[97] Tony Printezis and David Detlefs. A generational mostly-concurrent garbage collector. In Craig Chambers and Antony L. Hosking, editors, *ISMM 2000, International Symposium on Memory Management, Minneapolis, Minnesota, October 15-16, 2000 (in conjunction with OOPSLA 2000), Conference Proceedings*, pages 143–154. ACM, 2000.

[98] John Michael Robson. An estimate of the store size necessary for dynamic storage allocation. *J. ACM*, 18(2):416–423, 1971.

[99] John Michael Robson. Bounds for some functions concerning dynamic storage allocation. *J. ACM*, 21(3):491–499, 1974.

[100] Helena Rodrigues and Richard E. Jones. A cyclic distributed garbage collector for network objects. In Özalp Babaoglu and Keith Marzullo, editors, *Distributed Algorithms, 10th International Workshop, WDAG '96, Bologna, Italy, October 9-11, 1996, Proceedings*, volume 1151 of *Lecture Notes in Computer Science*, pages 123–140. Springer, 1996.

[101] Narendran Sachindran, J. Eliot B. Moss, and Emery D. Berger. Mc^2: high-performance garbage collection for memory-constrained environments. In John M. Vlissides and Douglas C. Schmidt, editors, *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2004, October 24-28, 2004, Vancouver, BC, Canada*, pages 81–98. ACM, 2004.

[102] Jon D Salkild. Implementation and analysis of two reference counting algorithms. *Master's thesis, University College, London*, 1987.

[103] Patrick M. Sansom and Simon L. Peyton Jones. Generational garbage collection for haskell. In *FPCA*, pages 106–116, 1993.

[104] Robert A Saunders. The lisp system for the q-32 computer. *The Programming Language LISP: Its Operation and Applications*, pages 220–231, 1964.

[105] Robert Allen Shaw. *Empirical Analysis of a LISP System*. PhD thesis, Stanford, CA, USA, 1988. Order No: GAX88-15047.

[106] Fridtjof Siebert. Limits of parallel marking garbage collection. In Richard E. Jones and Stephen M. Blackburn, editors, *Proceedings of the 7th International Symposium on Memory Management, ISMM 2008, Tucson, AZ, USA, June 7-8, 2008*, pages 21–29. ACM, 2008.

[107] Fridtjof Siebert. Concurrent, parallel, real-time garbage-collection. In Jan Vitek and Doug Lea, editors, *Proceedings of the 9th International Symposium on Memory Management, ISMM 2010, Toronto, Ontario, Canada, June 5-6, 2010*, pages 11–20. ACM, 2010.

[108] David Siegwart and Martin Hirzel. Improving locality with parallel hierarchical copying GC. In Erez Petrank and J. Eliot B. Moss, editors, *Proceedings of the 5th International Symposium on Memory Management, ISMM 2006, Ottawa, Ontario, Canada, June 10-11, 2006*, pages 52–63. ACM, 2006.

[109] Jeremy Singer, Gavin Brown, Ian Watson, and John Cavazos. Intelligent selection of application-specific garbage collectors. In Greg Morrisett and Mooly Sagiv, editors, *Proceedings of the 6th International Symposium on Memory Management, ISMM 2007, Montreal, Quebec, Canada, October 21-22, 2007*, pages 91–102. ACM, 2007.

[110] Sunil Soman, Chandra Krintz, and David F. Bacon. Dynamic selection of application-specific garbage collectors. In David F. Bacon and Amer Diwan, editors, *Proceedings of the 4th International Symposium on Memory Management, ISMM 2004, Vancouver, BC, Canada, October 24-25, 2004*, pages 49–60. ACM, 2004.

[111] James W. Stamos. Static grouping of small objects to enhance performance of a paged virtual memory. *ACM Trans. Comput. Syst.*, 2(2):155–180, 1984.

[112] James William Stamos. *A large object-oriented virtual memory: grouping strategies, measurements, and performance*. PhD thesis, Massachusetts Institute of Technology, 1982.

[113] Darko Stefanovic and J. Eliot B. Moss. Characterization of object behaviour in standard ml of new jersey. In *LISP and Functional Programming*, pages 43–54, 1994.

[114] David Ungar. Generation scavenging: A non-disruptive high performance storage reclamation algorithm. In William E. Riddle and Peter B. Henderson, editors, *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Pittsburgh, Pennsylvania, USA, April 23-25, 1984*, pages 157–167. ACM, 1984.

[115] David Ungar and Frank Jackson. Tenuring policies for generation-based storage reclamation. In Norman K. Meyrowitz, editor, *Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'88), San Diego, California, USA, September 25-30, 1988, Proceedings.*, pages 1–17. ACM, 1988.

[116] David Ungar and Frank Jackson. An adaptive tenuring policy for generation scavengers. *ACM Trans. Program. Lang. Syst.*, 14(1):1–27, 1992.

[117] David Michael Ungar. *The Design and Evaluation of A High Performance Smalltalk System*. PhD thesis, EECS Department, University of California, Berkeley, Feb 1986.

[118] J. White. Address/memory management for a gigantic LISP environment. In *LISP Conference*, pages 119–127, 1980.

[119] Paul R. Wilson. Uniprocessor garbage collection techniques. In Yves Bekkers and Jacques Cohen, editors, *Memory Management, International Workshop IWMM 92, St. Malo, France, September 17-19, 1992, Proceedings*, volume 637 of *Lecture Notes in Computer Science*, pages 1–42. Springer, 1992.

[120] Paul R. Wilson, Michael S. Lam, and Thomas G. Moher. Effective ‘‘static-graph" reorganization to improve locality in garbage-collected systems. In David S. Wise, editor, *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation (PLDI), Toronto, Ontario, Canada, June 26-28, 1991*, pages 177–191. ACM, 1991.

[121] Taiichi Yuasa. Real-time garbage collection on general-purpose machines. *Journal of Systems and Software*, 11(3):181–198, 1990.

[122] Karen Zee and Martin C. Rinard. Write barrier removal by static analysis. In Mamdouh Ibrahim and Satoshi Matsuoka, editors, *Proceedings of the 2002 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications, OOPSLA 2002, Seattle, Washington, USA, November 4-8, 2002.*, pages 191–210. ACM, 2002.

[123] Benjamin G. Zorn. *Comparative Performance Evaluation of Garbage Collection Algorithms*. PhD thesis, EECS Department, University of California, Berkeley, Dec 1989.

[124] Benjamin G. Zorn. Comparing mark-and-sweep and stop-and-copy garbage collection. In *LISP and Functional Programming*, pages 87–98, 1990.