Wait-free προσομοιώσεις αυθαίρετων αντικειμένων

Έχουμε δει ότι το πρόβλημα της ομοφωνίας δεν μπορεί να επιλυθεί με χρήση μόνο read/write καταχωρητών.

Πολλοί μοντέρνοι επεξεργαστές παρέχουν επιπρόσθετα λίγους καταχωρητές που είναι πιο ισχυροί, π.χ., LL/SC ή Compare&Swap.

Θα μελετήσουμε το ακόλουθο ερώτημα:

Δεδομένων δύο τύπων ατομικών αντικειμένων X και Y, υπάρχει wait-free προσομοίωση του τύπου X χρησιμοποιώντας μόνο ατομικά αντικείμενα του τύπου Y και read/write καταχωρητές;

Παράδειγμα: Προσομοίωση μιας ουράς FIFO

Οι λειτουργίες που υποστηρίζονται από μια ουρά FIFO είναι:

- ◊ [enq(Q,x), ack(Q)]
- ◊ [deq(Q), return(Q,x)]

όπου x είναι μια οποιαδήποτε τιμή που έχει αποθηκευτεί στην ουρά (η deq μπορεί να επιστρέψει nill)

Θεώρημα 1

Υπάρχει consensus αλγόριθμος για 2 διεργασίες p_0 και p_1 που χρησιμοποιεί μια ουρά FIFO και read/write καταχωρητές.

Πόρισμα 1

Δεν υπάρχει wait-free προσομοίωση μιας ουράς FIFO από read/write καταχωρητές (για οποιοδήποτε αριθμό διεργασιών).

Παράδειγμα: Προσομοίωση μιας ουράς FIFO

Πρόταση 1

Ο παρακάτω αλγόριθμος επιλύει το πρόβλημα της ομοφωνίας για 2 διεργασίες.

Algorithm 46 Consensus algorithm for two processors, using a FIFO queue: code for processor p_i, $i = 0, 1$.

Initially $Q = \langle 0 \rangle$ and $Prefer[i] = \perp$, $i = 0, 1$

```
1:  Prefer[i] := x                                   // write your input
2:  val := deq(Q)
3:  if val = 0 then y := x      // dequeued the first element, decide on your input
4:  else y := Prefer[1 − i]                          // decide on other's input
```

Η ισχυρή λειτουργία Compare&Swap

value **Compare&Swap**(X: *memory address*; old, new: *value*) {
    previous = X
    if (previous == old) then X = new;
    return previous;
}

Algorithm 47 Consensus algorithm for any number of processors, using compare&swap: code for processor p_i, $0 \leq i \leq n-1$.

Initially $First = \perp$

```
1:  v := compare&swap(First,⊥,x)
2:  if v = ⊥ then                    // this is the first compare&swap
3:      y := x                       // decide on your own input
4:  else y := v                      // decide on someone else's input
```

Πρόταση 2: Ο παραπάνω αλγόριθμος επιλύει το πρόβλημα της ομοφωνίας για οποιοδήποτε αριθμό διεργασιών.

Παράδειγμα: Προσομοίωση μιας ουράς FIFO

Θεώρημα 2

Δεν υπάρχει αλγόριθμος επίλυσης του προβλήματος της ομοφωνίας για $n > 2$ διεργασίες χρησιμοποιώντας μόνο ουρές FIFO και read/write καταχωρητές.

Απόδειξη

Χρησιμοποιούνται valency arguments.

Η wait-free ιεραρχία

Τα ατομικά αντικείμενα μπορούν να κατηγοριοποιηθούν βάσει ενός κριτηρίου που βασίζεται στην ικανότητά τους να επιλύουν το πρόβλημα της ομοφωνίας για συγκεκριμένο αριθμό διεργασιών.

Λέμε ότι ένα ατομικό αντικείμενο τύπου X επιλύει το πρόβλημα της ομοφωνίας με wait-free τρόπο για n διεργασίες αν υπάρχει wait-free αλγόριθμος επίλυσης του προβλήματος της ομοφωνίας για n διεργασίες (n-1 από τις οποίες μπορούν να αποτύχουν), ο οποίος χρησιμοποιεί μόνο ατομικά αντικείμενα τύπου X και read/write καταχωρητές.

Ο αριθμός ομοφωνίας ενός τύπου ατομικού αντικειμένου X είναι n (CN(X) = n), αν n είναι ο μεγαλύτερος αριθμός διεργασιών για τον οποίο ο X επιλύει το πρόβλημα της ομοφωνίας με wait-free τρόπο. Ο αριθμός ομοφωνίας είναι ∞ αν ο X επιλύει το πρόβλημα της ομοφωνίας για κάθε n.

Η wait-free ιεραρχία

Για κάθε τύπο αντικειμένου X ποια είναι η μικρότερη τιμή που μπορεί να έχει το CN(X)?

Ο CN των read/write καταχωρητών είναι 1.

Ο CN των ακόλουθων ατομικών αντικειμένων είναι 2: test&set, swap, fetch&add, stacks.

Ο CN του Compare&Swap είναι ∞.

Υπάρχει μια ιεραρχία των τύπων των ατομικών αντικειμένων βάσει του CN τους. Έχει αποδειχθεί ότι υπάρχουν ατομικά αντικείμενα με CN = m, για κάθε m.

Η wait-free ιεραρχία

Θεώρημα 3: Αν CN(X) = m και CN(Y) = n > m, τότε δεν υπάρχει wait-free προσομοίωση του Υ από ατομικά αντικείμενα τύπου Χ και read/write καταχωρητές σε σύστημα με περισσότερες από m διεργασίες.

Απόδειξη: Με εις άτοπο απαγωγή. Έστω ότι υπάρχει μια προσομοίωση του Υ από το Χ και από read/write καταχωρητές σε σύστημα με k > m διεργασίες. Έστω l = min{k,n} και έστω ότι l > m.

Θα δείξουμε ότι υπάρχει wait-free αλγόριθμος για το πρόβλημα της ομοφωνίας για l διεργασίες που χρησιμοποιεί μόνο ατομικά αντικείμενα τύπου Χ και read/write καταχωρητές. Αυτό είναι άτοπο αφού CN(X) = m < l.

Αφού l ≤ n, ∃ wait-free προσομοίωση του Υ από τον Χ για l διεργασίες.

Επίσης, υπάρχει wait-free αλγόριθμος Α επίλυσης του προβλήματος της ομοφωνίας για l διεργασίες χρησιμοποιώντας ατομικά αντικείμενα τύπου Υ και read/write καταχωρητές. Αντικαθιστούμε κάθε αντικείμενο τύπου Υ του Α από ένα στιγμιότυπο της wait-free προσομοίωσής του από τον Χ!

⇒ ∃ αλγόριθμος ομοφωνίας για l διεργασίες από αντικείμενα τύπου Χ και read/write καταχωρητές!

Η wait-free ιεραρχία

Πόρισμα 2

Δεν υπάρχει wait-free προσομοίωση οποιουδήποτε αντικειμένου με αριθμό ομοφωνίας μεγαλύτερο από 1 χρησιμοποιώντας μόνο read/write καταχωρητές.

Πόρισμα 3

Δεν υπάρχει wait-free προσομοίωση οποιουδήποτε αντικειμένου με αριθμό ομοφωνίας μεγαλύτερου από 2 χρησιμοποιώντας ουρές FIFO και read/write καταχωρητές.

Καθολικότητα

Ένας τύπος ατομικού αντικειμένου X είναι **καθολικός** σε συστήματα n διεργασιών αν υπάρχει wait-free προσομοίωση οποιουδήποτε άλλου ατομικού αντικειμένου από ατομικά αντικείμενα τύπου X και read/write καταχωρητές σε τέτοια συστήματα.

Θα αποδείξουμε ότι: Κάθε αντικείμενο X με αριθμό ομοφωνίας n είναι καθολικό σε σύστημα με το πολύ n διεργασίες.

Βασική Ιδέα: Παρουσιάζουμε καθολικό non-blocking αλγόριθμο (που προσομοιώνει οποιοδήποτε αντικείμενο) σε σύστημα με n διεργασίες χρησιμοποιώντας μόνο ατομικά αντικείμενα ομοφωνίας για n διεργασίες και read/write καταχωρητές.

Η *non-blocking ιδιότητα τερματισμού* εγγυάται ότι στο σύστημα υπάρχει κάποια πρόοδος παρότι ο τερματισμός των λειτουργιών συγκεκριμένων διεργασιών μπορεί να αποτρέπεται/καθυστερεί διαρκώς.

Μια non-blocking προσομοίωση από ατομικά αντικείμενα τύπου Compare&Swap

Βασική Ιδέα

Αναπαράσταση αντικειμένου σαν μια διαμοιραζόμενη συνδεδεμένη λίστα που περιέχει την διατεταγμένη ακολουθία λειτουργιών του αντικειμένου.

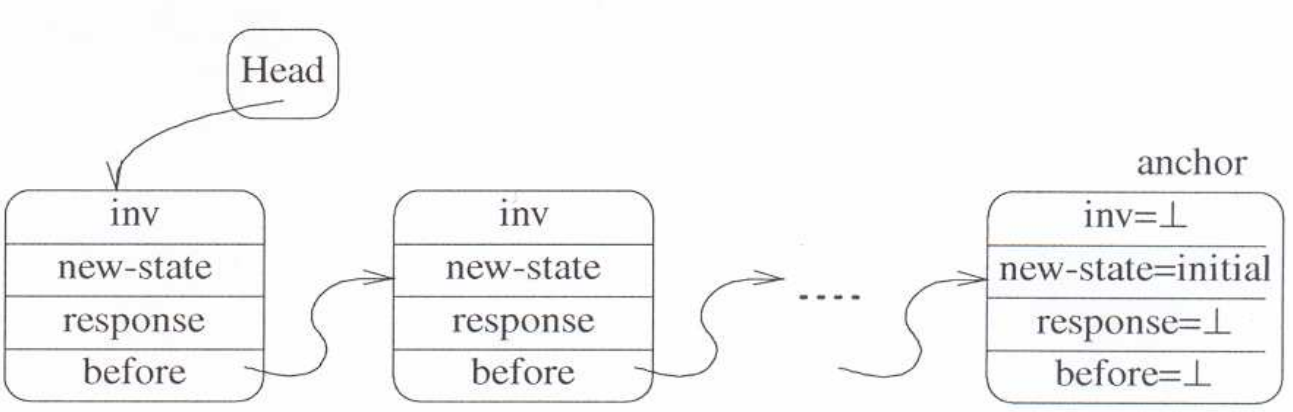
Η εκτέλεση μιας λειτουργίας προσομοιώνεται από την εισαγωγή μιας νέας εγγραφής στην κορυφή της λίστας.

Η διαχείριση της κορυφής της λίστας head γίνεται με τη χρήση ενός Compare&Swap αντικειμένου.

Κάθε λειτουργία αναπαρίσταται από μια διαμοιραζόμενη εγγραφή με πεδία:

- inv: κλήση λειτουργίας & παράμετροι
- new-state: η κατάσταση του αντικειμένου που προκύπτει μετά την εκτέλεση της λειτουργίας
- response: η απόκριση στη λειτουργία & η τιμή απόκρισης
- before: δείκτης στην εγγραφή που αντιστοιχεί στη προηγούμενη λειτουργία στο αντικείμενο

Μια non-blocking προσομοίωση από ατομικά αντικείμενα τύπου Compare&Swap



Algorithm 48 A nonblocking universal algorithm using compare&swap: code for processor p_i, $0 \leq i \leq n-1$.

Initially *Head* points to the anchor record

```
1:  when inv occurs:                          // operation invocation, including parameters
2:      allocate a new opr record pointed to by point with point.inv := inv
3:      repeat
4:          h := Head
5:          point.new-state, point.response := apply(inv,h.new-state)
6:          point.before := h
7:      until compare&swap(Head, h, point) = h
8:      enable the output indicated by point.response        // operation response
```

Μια non-blocking προσομοίωση από ατομικά αντικείμενα τύπου Compare&Swap

- Τα σημεία σειριοποίησης των λειτουργιών του αντικειμένου προς προσομοίωση καθορίζονται από την διάταξη των λειτουργιών στη λίστα. Έτσι, η σειριοποιησιμότητα ισχύει προφανώς.
- Ο αλγόριθμος είναι non-blocking: Αν κάποια διεργασία δεν επιτύχει να εισάγει τη λειτουργία του στην λίστα, θα πρέπει η compare&swap κάποιας άλλης διεργασίας να επιτύχει.
- Ο αλγόριθμος δεν είναι wait-free αφού μπορεί να επιτυγχάνει πάντα η compare&swap του ίδιου επεξεργαστή και όλοι οι άλλοι επεξεργαστές να μην καταφέρνουν να εκτελέσουν τη δική τους λειτουργία.

Αρνητικά

- χρησιμοποιεί Compare&Swap και όχι consensus αντικείμενα
- είναι non-blocking και όχι wait-free
- χρησιμοποιεί απεριόριστη μνήμη

Μια non-blocking προσομοίωση από ατομικά αντικείμενα ομοφωνίας (consensus objects)

Ένα ατομικό **αντικείμενο ομοφωνίας** Ο υποστηρίζει τη μοναδική λειτουργία, [decide(O,in), return(O,out)], όπου in και out είναι τιμές ενός πεδίου τιμών.

Για τις ακολουθίες λειτουργιών του αντικειμένου ισχύει ότι όλες οι out τιμές είναι ίδιες και ίσες με κάποια in τιμή.

1η Προσπάθεια

Αντικατάσταση του αντικειμένου compare&Swap με ένα consensus object.

Πρόβλημα

Μετά την πρώτη λειτουργία που θα εκτελεστεί σε ένα αντικείμενο consensus, αυτό θα επιστρέφει πάντα την ίδια τιμή. Τα αντικείμενα consensus είναι 1-shot. Έτσι, δεν μπορούμε να χρησιμοποιούμε το αντικείμενο επαναληπτικά.

Μια non-blocking προσομοίωση από ατομικά αντικείμενα ομοφωνίας

Λύση

Κάθε εγγραφή χρησιμοποιεί το δικό της αντικείμενο consensus.

Αντικαθιστούμε το πεδίο before με ένα πεδίο after. Το πεδίο αυτό είναι ένα αντικείμενο ομοφωνίας που η τιμή του v είναι η επόμενη λειτουργία που εφαρμόζεται στο προς προσομοίωση ατομικό αντικείμενο.

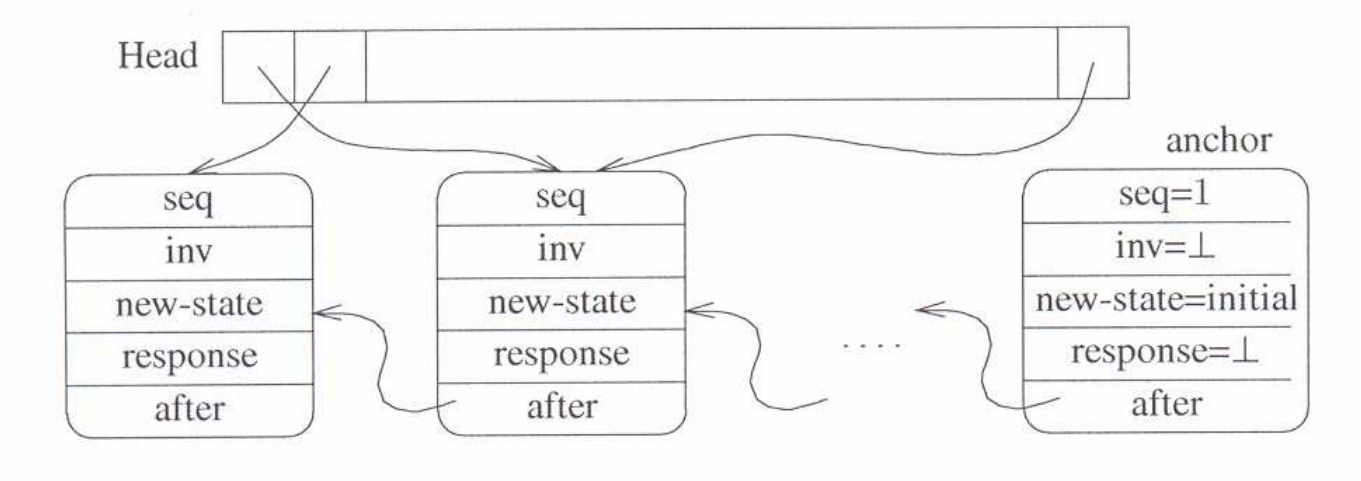
Πρόβλημα

Πως γνωρίζει κάθε διεργασία ποια εγγραφή βρίσκεται στη θέση head της λίστας;

Χρησιμοποιείται ένας πίνακας Head από δείκτες, έναν για κάθε διεργασία, κάθε ένας από τους οποίους αναπαριστά την τελευταία εγγραφή που η αντίστοιχη διεργασία έχει δει στη θέση head της λίστας. Αυτοί οι δείκτες δεν δείχνουν απαραίτητα τη σωστή εγγραφή.

Χρησιμοποιούνται επίσης sequence numbers.

Μια non-blocking προσομοίωση από ατομικά αντικείμενα ομοφωνίας



Algorithm 49 A nonblocking universal algorithm using consensus objects: code for processor p_i, $0 \leq i \leq n-1$.

Initially $Head[j]$ points to anchor, for all j, $0 \leq j \leq n-1$

```
1:  when inv occurs:                          // operation invocation, including parameters
2:      allocate a new opr record pointed to by point with point.inv := inv
3:      for j := 0 to n − 1 do                // find record with highest sequence number
4:          if Head[j].seq > Head[i].seq then Head[i] := Head[j]
5:      repeat
6:          win := decide(Head[i].after,point)          // try to thread your record
7:          win.seq := Head[i].seq + 1
8:          win.new-state, win.response := apply(win.inv,Head[i].new-state)
9:          Head[i] := win                  // point to the record at the head of the list
10:     until win = point
11:     enable the output indicated by point.response       // operation response
```

Μια non-blocking προσομοίωση από ατομικά αντικείμενα ομοφωνίας

Η σειριοποιησιμότητα είναι εγγυημένη με τον ίδιο τρόπο όπως και στον προηγούμενο αλγόριθμο.

Για κάθε καθολική κατάσταση C, έστω ότι

$$\text{max-head}(C) = \max\{\text{Head}[i].\text{seq} \mid 0 \leq i \leq n-1\}$$

Για κάθε i, η Head[i].seq είναι μονοτονικά μη-φθίνουσα κατά τη διάρκεια κάθε εκτέλεσης.

Λήμμα: Αν μια διεργασία εκτελεί l επαναλήψεις της ανακύκλωσης repeat, τότε το max-head αυξάνει κατά τουλάχιστον l.

Ο αλγόριθμος είναι non-blocking.

Αν κάποια διεργασία εκτελεί έναν μη-πεπερασμένο αριθμό βημάτων, το max-head αυξάνει μη-πεπερασμένα. Άρα άλλες διεργασίες επιτυγχάνουν να εκτελέσουν τις λειτουργίες τους.

Ο αλγόριθμος δεν είναι wait-free.

Μια wait-free προσομοίωση από ατομικά αντικείμενα ομοφωνίας

- Χρησιμοποιούμε τη μέθοδο helping, η οποία βασίζεται στην παροχή βοήθειας για τη διεκπεραίωση μιας λειτουργίας μιας διεργασίας από τις άλλες διεργασίες.
  - Πρέπει να είναι γνωστό ποιες διεργασίες προσπαθούν να εκτελέσουν λειτουργίες. Η πληροφορία αυτή καταγράφεται σε έναν πίνακα Announce, όπου Announce[i] είναι ένας δείκτης στην εγγραφή που στο τρέχον βήμα η p_i προσπαθεί να εισάγει στη λίστα.
  - Για να επιλεγεί η διεργασία που θα βοηθηθεί σε κάθε βήμα, χρησιμοποιούμε προτεραιότητες. Η προτεραιότητα αποδίδεται σε κάθε διεργασία εκ περιτροπής (round robin) χρησιμοποιώντας τα ids.
  - Αν η p_i θέλει να εκτελέσει κάποια λειτουργία, έχει προτεραιότητα να εκτελέσει την k-οστή λειτουργία αν $k = i \bmod n$.

Μια wait-free προσομοίωση από ατομικά αντικείμενα ομοφωνίας

Algorithm 50 A wait-free universal algorithm using consensus objects: code for processor p_i, $0 \leq i \leq n-1$.

Initially $Head[j]$ and $Announce[j]$ point to the anchor record, for all j, $0 \leq j \leq n-1$

```
1:  when inv occurs:                                   // operation invocation, with parameters
2:      allocate a new opr record pointed to by Announce[i]
            with Announce[i].inv := inv and Announce[i].seq := 0
3:      for j := 0 to n − 1 do                         // find highest sequence number
4:          if Head[j].seq > Head[i].seq then Head[i] := Head[j]
5:      while Announce[i].seq = 0 do
6:          priority := Head[i].seq+1 mod n            // id of processor with priority
7:          if Announce[priority].seq = 0              // check whether help is needed
8:              then point := Announce[priority]       // choose other record
9:              else point := Announce[i]              // choose your own record
10:         win := decide(Head[i].after,point)         // try to thread chosen record
11:         win.new-state, win.response := apply(win.inv,Head[i].new-state)
12:         win.seq := Head[i].seq + 1
13:         Head[i] := win                             // point to the record at the head of the list

14:     enable the output indicated by win.response    // operation response
```

Μια wait-free προσομοίωση από ατομικά αντικείμενα ομοφωνίας (consensus objects)

Θεώρημα: Υπάρχει wait-free προσομοίωση οποιουδήποτε αντικειμένου σε σύστημα n διεργασιών η οποία χρησιμοποιεί μόνο αντικείμενα ομοφωνίας n διεργασιών και καταχωρητές ανάγνωσης/εγγραφής. Κάθε επεξεργαστής ολοκληρώνει κάθε λειτουργία που θέλει να εκτελέσει μέσα σε O(n) βήματα ανεξάρτητα από τη συμπεριφορά των υπολοίπων διεργασιών.

Απόδειξη: Έστω C_1 η $1^η$ καθολική κατάσταση στην οποία ο p_i έχει εκδηλώσει τη διάθεσή του να εκτελέσει μια λειτουργία.

Για κάθε καθολική κατάσταση C, max-head(C) είναι ο μέγιστος sequence number σε κάθε εγγραφή του Head. Το max-head(C) αυξάνει χωρίς όριο.

Έστω C_2 η $1^η$ καθολική κατάσταση στην οποία ισχύει ότι max-head(C_2) mod n = i. Η λειτουργία της p_i έχει εισαχθεί στην ουρά μέχρι την C_2.

Θεώρημα: Κάθε αντικείμενο X με CN(X) = n είναι καθολικό για κάθε σύστημα με το πολύ n διεργασίες.