Programowanie współbieżne

Lista zadań nr 10

Na ćwiczenia 5. stycznia 2022

Zadanie 1. Rozważmy klasę **CoarseList** (implementacja listowa zbioru zabezpieczona globalnym zamkiem, TAoMP2e rozdział 9.4).

1. Przypomnij, jakie punkty linearyzacji należy wybrać w metodach **add()**, **remove()** i **contains()** by następujący niezmiennik (mapa abstrakcji) był zachowany: "element należy do zbioru ⇔ węzeł na liście, w którym znajduje się element jest osiągalny z węzła **head**".
2. Pokaż, że powyższy warunek przestaje być niezmiennikiem, jeśli metody będą linearyzowane w momencie zajęcia zamka.
3. Zmodyfikuj ten warunek tak, by dla metod linearyzowanych w momencie zajęcia zamka, **CoarseList** nadal była poprawną implementacją zbioru

Zadanie 2. Wyjaśnij, dlaczego metody **add()** i **remove()** klasy **FineList** (implementacja listowa zbioru zabezpieczona drobnoziarnistymi zamkami, TAoMP2e r. 9.5) są linearyzowalne. Dla każdej z metod rozważ osobno przypadek wywołania zakończonego sukcesem i porażką.

Zadanie 3. Podaj implementację metody **contains()** dla klasy **FineList**. Uzasadnij jej poprawność.

Zadanie 4. Zaprezentuj wykonanie metody **remove()** klasy **OptimisticList** (optymistyczna implementacja listowa zbioru, TAoMP2e r. 9.6) które pętli się w nieskończoność. Zakładamy, że każdy z zamków występujących w elementach listy jest niegłodzący, głodzenie w metodzie **remove()** musi zatem wynikać z nieograniczonej liczby obrotów pętli **while(true) {}**. W jaki sposób mogą to wymusić inne wątki operujące współbieżnie na liście?

Zadanie 5. Klasa **OptimisticList** wykorzystuje ponowne przejście przez listę w celu sprawdzenia, czy zablokowane zamkami elementy są nadal osiągalne z początku listy (metoda **validate()**). Zamiast tego warunku można sprawdzać warunek silniejszy, czy lista nie uległa modyfikacji. Fakt

zmodyfikowania listy można wyrazić za pomocą zwiększenia znacznika czasu (w praktyce może to być licznik zabezpieczony zamkiem). Wzorując się na klasie **OptimisticList** podaj implementację listową zbioru według tego pomysłu.

Zadanie 6. Dla każdej z poniższych modyfikacji listowych implementacji zbioru wyjaśnij, że otrzymany algorytm jest nadal linearyzowalny, lub podaj kontrprzykład wskazujący, że nie jest.

1. W klasie **OptimisticList** metoda **contains()** zajmuje zamki w dwóch węzłach przed stwierdzeniem, czy klucz jest tam obecny. Niech zmodyfikowana metoda **contains()** nie zajmuje żadnych zamków.
2. W klasie **LazyList** (TAoMP2e, r. 9.7) metoda **contains()** wykonuje się bez użycia zamków, ale bada wartość bitu marked. Niech zmodyfikowana metoda **contains()** pomija badanie tego bitu.

Zadanie 7. Czy można, bez straty poprawności, zmodyfikować metodę **remove()** klasy **LazyList** tak, by zajmowała tylko jeden zamek?