1 Treść zadania

(Konserwatny MM) Zaproponuj modyfikację algorytmu Maximal Matching przedstawionego w notatkach do wykładu, która umożliwi jego działanie przy dodatkowym założeniu, że każdy proces jest typu A albo B i dwa dwa procesy tego samego typu nie mogą tworzyć pary. Ponadto możesz przyjąć, że typ danego procesu jest z góry zadany i nigdy nie może zostać zmieniony. Uzasadnij poprawność i zbieżność algorytmu. UWAGA: Jeśli nie zaznaczono inaczej, wszystkie definicje i oznaczenia są tożsame z wykładem

2 Sprostowanie

Po konsultacji z dr Lemieszem- Aby zadanie w jego oczach było zupełnie poprawne należy dodać nowy stan, który nazwiemy *mistake*. Opisuje on sytuacje, gdy w wyniku jakiegoś błędu początkowa preferencja procesu p, była ustawiona na proces q, który nie należy do $N(p)$. Warto zauważyć, że pierwotny algorytm, na którym wzorowana jest ów modyfikacja nie jest odporny na tego typu błędy, stąd moja pierwotna pomyłka. Dalsza część pracy dodaje wskazany stan do algorytmu oraz rozpatruje o ile zmienia się jego funkcjonowanie (a zmienia się niewiele).

3 Rozwiązanie

W celu przedstawienia typu procesu wprowadzimy jednoargumentowy predykat *typ*.

$$typ(p) = A \oplus typ(p) = B \tag{1}$$

Dla każdego procesu p. Dodatkowo dodajemy stan *mistake* zgodnie z intuicją przedstawioną w sprostowaniu.

$$mistake(p) \iff pref_p = q \wedge q \notin N(p) \tag{2}$$

Ponadto modyfikujemy definicję sąsiedztwa (równoważnie możemy dla wszystkich instrukcji, w których sprawdzamy czy q należy do N(p) dodać warunek $typ(p) \neq typ(q)$)

$$N(p) = \{q : \{p, q\} \in E \wedge typ(p) \neq typ(q)\} \tag{3}$$

Następnie modyfikujemy algorytm. Dodajemy instrukcję warunkową, która sprawdza przynależność preferencji procesu p do zbioru jego sąsiadów. Jeśli q do wspomnianego zbioru nie należy, ustawiamy preferencję procesu p na NULL

$$if\ pref_p = q \wedge q \notin N(p)\ then\ pref_p \leftarrow NULL \tag{4}$$

co sprowadza się do zapisu

$$if\ mistake(p)\ then\ pref_p \leftarrow NULL \tag{5}$$

4 Uzsadnienie poprawności

Zgodnie z Lemat 3 z wykładu: Zachodzą warunki specyfikacji S $\iff$ konfiguracja jest ostateczna (tzn. żaden krok algorytmu nie może zmienić konfiguracji)
Dowód: ($\Rightarrow$) Pokażemy, że jeśli zachodzą warunku S to żadna akcja algorytmu nie jest możliwa.
Przypomnienie: Specyfikację S definiujemy następująco: w każdej poprawnej konfiguracji prawdziwe jest zdanie

$$(\forall p)(married(p) \ \vee \ single(p)) \tag{6}$$

Rozpatrzmy po kolei wszystkie możliwe kroki jakie może wykonać algorytm dla danego procesu:

- (Mistake) Proces p musiałby znajdować się w stanie *mistake*, co wykluczamy przez założenia o specyfikacji S
- (Accept proposal) Aby warunek został spełniony, proces p musi nie mieć dopasowanej żadnej pary (być w stanie *free*) oraz musi istnieć stan q o innym typie, który może przyjąć p jako parę (czyli musi być w stanie *wait*). Zgodnie z założeniem, aktualnie posiadamy procesy tylko w dwóch stanach: *single* lub *married*, więc warunek nie zostanie nigdy spełniony
- (Propose) Aby warunek został spełniony, proces p musi nie mieć dopasowanej żadnej pary (być w stanie *free*), żaden z procesów q o innym typie nie może zadeklarować p jako swoją parę (wtedy zaakceptowalibysmy propozycję) oraz musi istnieć proces q o innym typie bez preferencji, czyli q również musiałby być w stanie *free*. Procesy mogą być tylko w stanie *single* lub *married*, wykluczamy
- (Unchain) - Proces p musi byc w stanie *chain*, wykluczamy

Wniosek - gdy konfiguracja S spełniona, wykonanie ruchu jest niemożliwe
($\Leftarrow$) Dokonamy sprawdzenia co się stanie z procesem, który nie jest w stanie *married* bądź *single*. Bez straty ogólności wybieramy proces p zgodnie z powyższym i dokonujemy sprawdzeń:

- Jeśli proces p jest w stanie *mistake* to dochodzi do usunięcia jego preferencji i w zależności od sąsiedztwa przechodzi do stanu *single* bądź *free* - ruch możliwy
- Jeśli proces p jest w stanie *free* to może:
  - Przyjąć propozycję od procesu q o innym typie w stanie *wait*
  - Dokonać *unchain* jeśli q o innym typie jest w stanie *chain*
  - Złożyć propozycję, jeśli q jest w stanie *free*
- Jeśli proces p jest w stanie *chain* to dokona *unchain*

- Jeśli proces p jest w stanie *wait* to musi istniec proces o stanie *free* i przyjmuje propozycję

Jeśli istnieją stany poza *single* oraz *married* w konfiguracji to zawsze można wykonać ruch, czyli nie jest ona ostateczna.

5 Dowód zbieżności

Rozważmy funkcję potencjału $F(t) = (mis, c, f, w, m + s)$ gdzie:

- mis - liczba procesów w stanie *mistake*
- c - liczba procesów w stanie *chain*
- f - liczba procesów w stanie *free*
- w - liczba procesów w stanie *wait*
- m+s - liczba procesów w stanie *married* lub *single*

Sprawdźmy zmianę każdego z argumentów przy każdej możliwej egzekucji algorytmu dla jednego procesu:

- Mistake: $mistake(p)$ zmienia się na $free(p)$ bądź $single(p)$ w zależności od otoczenia, co trochę komplikuje zapis, jednakże istotny jest fakt, iż za każdym razem, gdy ów procedura jest spełniona, liczba procesów w stanie mistake, która jest na pierwszej pozycji krotki, maleje.

  $$(mis - 1, c, f, w, m + s + 1) \prec (mis, c, f, w, m + s) \tag{7}$$

  lub

  $$(mis - 1, c, f + 1, w, m + s) \prec (mis, c, f, w, m + s) \tag{8}$$

  Co sprowadza się do

  $$F(t + 1) \prec F(t) \tag{9}$$

- Accept proposal: $free(p)$ i $wait(q)$ zmienia się na $married(p)$ i $married(q)$ (lub single, jeśli nie może w married, ale nas to zbytnio nie interesuje, bo grupujemy married i single razem)

  $$(mis, c, f - 1, w - 1, m + s + 2) \prec (mis, c, f, w, m + s) \tag{10}$$

  Element na trzecim miejscu zmniejsza się przy zachowaniu wartości pierwszego i drugiego. Stąd wnioskujemy, iż:

  $$F(t + 1) \prec F(t) \tag{11}$$

- Proposal: $free(p)$ i $free(q)$ zamieniają się w $wait(p)$ i $free(q)$

$$(mis, c, f-1, w+1, m+s) \prec (mis, c, f, w, m+s) \tag{12}$$

Element na trzecim miejscu zmniejsza się przy zachowaniu wartości pierwszego i drugiego. Stąd wnioskujemy, iż:

$$F(t+1) \prec F(t) \tag{13}$$

- Unchain: $chain(p)$ zmienia się na $free(p)$

$$(mis, c-1, f+1, w, m+s) \prec (mis, c, f, w, m+s) \tag{14}$$

Element na drugim miejscu zmniejsza się przy zachowaniu pierwszego. Stąd wnioskujemy, iż:

$$F(t+1) \prec F(t) \tag{15}$$

Zauważamy, że porządek leksykograficzny jest dobrym porządkiem, czyli ma wartość najmniejszą, która w naszym przypadku wynosi $(0, 0, 0, 0, n)$, czyli wszystkie procesy są *married* lub *single*. Jak sprawdzić zbieżność? Można to zrobić korzystając z reguły **stars and bars** (będziemy szacować mocno z góry). W opisywanym modelu istnieje 5 zmiennych, każda z nich jest zmienną całkowitą nieujemną $mis, c, f, w, m+s \geq 0$. Wiedząc, iż liczba procesów jest równa n wnioskujemy, iż w dowolnym kroku t algorytmu suma wszystkich zmiennych musi być równa n (Każdy z procesów może być tylko w jednym staniem w jednym momencie oraz każdy z procesów musi mieć jakiś stan). Stąd:

$$mis + c + f + w + (m+s) = n \tag{16}$$

Zgodnie z regułą **stars and bars** rozwiązanie powyższego równania można otrzymać na:

$$\binom{n+k-1}{k-1} \tag{17}$$

sposobów. Biorąc pod uwagę, iż mamy 5 zmiennych, dokonujemy podstawienia $k = 5$ i dokonujemy odpowiednich obliczeń:

$$\binom{n+5-1}{5-1} = \frac{(n+4)!}{4! * (n+4-4)!} = \frac{(n+1)(n+2)(n+3)(n+4)}{24} \tag{18}$$

na podstawie których wnioskujemy, iż mając n procesów osiągniemy konfigurację legalną w nie więcej niż $O(n^4)$ krokach.