Algorytm odśmiecania w locie Ben-Ariego

Grzegorz Wolny

24 kwietnia 2006

Algorytm odśmiecania w locie Ben-Ariego

- Odśmiecanie (*garbage collection*) w locie (*on-the-fly*)
- Własności dobrego algorytmu:
  - bezpieczeństwo – odśmiecona może zostać jedynie pamięć nieużywana
  - żywotność – pamięć nieużywana zostanie kiedyś odśmiecona

Algorytm odśmiecania w locie Ben-Ariego

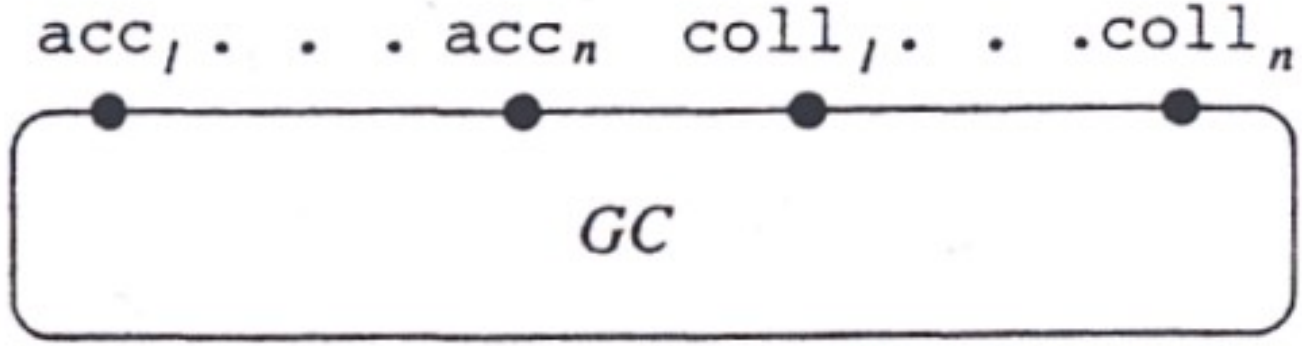
- Węzły (*nodes*) pamięci:
  - każdy ma ustaloną liczbę wskaźników do innych węzłów zwanych dziećmi (*children*)
  - każdy ma kolor czarny lub biały
  - korzenie (*roots*)
  - węzeł pusty (*nil*)
  - węzeł wskazujący na listę węzłów pustych (*free nodes*)
- Dwa współbieżne procesy:
  - modyfikator (*mutator*)
  - zbieracz (*collector*)

Algorytm odśmiecania w locie Ben-Ariego

- Działanie modyfikatora:
  - przekierowanie wskaźnika dostępnego węzła na inny dostępny węzeł
  - pokolorowanie na czarno węzła docelowego
- Działanie zbieracza:
  - pokolorowanie korzeni na czarno
  - przejrzenie węzłów i pokolorowanie synów czarnych węzłów na czarno
  - zliczenie czarnych węzłów; jeśli wynik jest różny od wyniku ostatniego zliczania powrót do punktu pierwszego
  - dodanie białych węzłów do listy wolnych węzłów, przekolorowanie czarnych węzłów na białe

Algorytm odśmiecania w locie Ben-Ariego

- Model w CWB
  - wystąpienie akcji *acc(i)* – dostępność węzła *(i)*
  - wystąpienie akcji *coll(i)* – zebranie węzła *(i)*
  - zakładamy, że jest *N* węzłów, z czego pierwsze *r* to korzenie
  - węzeł *nil* ma numer *1*
  - węzeł wskazujący listę węzłów pustych (*fl*) ma numer *2*
  - zakładamy, że każdy węzeł ma jednego syna



Algorytm odśmiecania w locie Ben-Ariego

- Model pamięci:

$$
\begin{aligned}
&Mem(Colors, Sons) \stackrel{\text{def}}{=} \\
&\quad \sum_{i \in N} \mathtt{setcolor}_i(c).Mem(Colors[i \rightarrow c], Sons) + \\
&\quad \sum_{i \in N} \overline{\mathtt{color}}_i(Colors_i).Mem(Colors, Sons) + \\
&\quad \sum_{i \in N} \overline{\mathtt{son}}_i(Sons_i).Mem(Colors, Sons) + \\
&\quad \sum_{(i,j) \in Ac' \times Ac''} \mathtt{mutate}_{i,j}.Mem(Colors, Sons[i \rightarrow j]) + \\
&\quad \mathtt{free}(i).Mem(Colors, (Sons[i \rightarrow Sons_{fl}])[fl \rightarrow i]) + \\
&\quad \sum_{i \in Ac} \overline{\mathtt{acc}}_i.Mem(Colors, Sons)
\end{aligned}
$$

Algorytm odśmiecania w locie Ben-Ariego

- Model modyfikatora:

$$Mutator \stackrel{\text{def}}{=} \sum_{(i,j)\in N\times N} \overline{\texttt{mutate}}_{i,j} \,.\, \overline{\texttt{setcolor}}_{j}(black).Mutator$$

Algorytm odśmiecania w locie Ben-Ariego

- Model zbieracza:

$$Collector \stackrel{\text{def}}{=} \texttt{cl}.BlackenRoots(1)$$

$$BlackenRoots(i) \stackrel{\text{def}}{=} \text{if } i > r \text{ then } Propagate(0,1) \text{ else } \overline{\texttt{setcolor}_i}(black).BlackenRoots(i+1)$$

$$Propagate(old,i) \stackrel{\text{def}}{=} \text{if } i > n \text{ then } Count(old,0,1) \text{ else } \texttt{color}_i(c).\text{if } c = black \text{ then } \texttt{son}_i(j).\overline{\texttt{setcolor}_j}(black).Propagate(old,i+1) \text{ else } Propagate(old,i+1)$$

Algorytm odśmiecania w locie Ben-Ariego

- Model zbieracza, cd.:

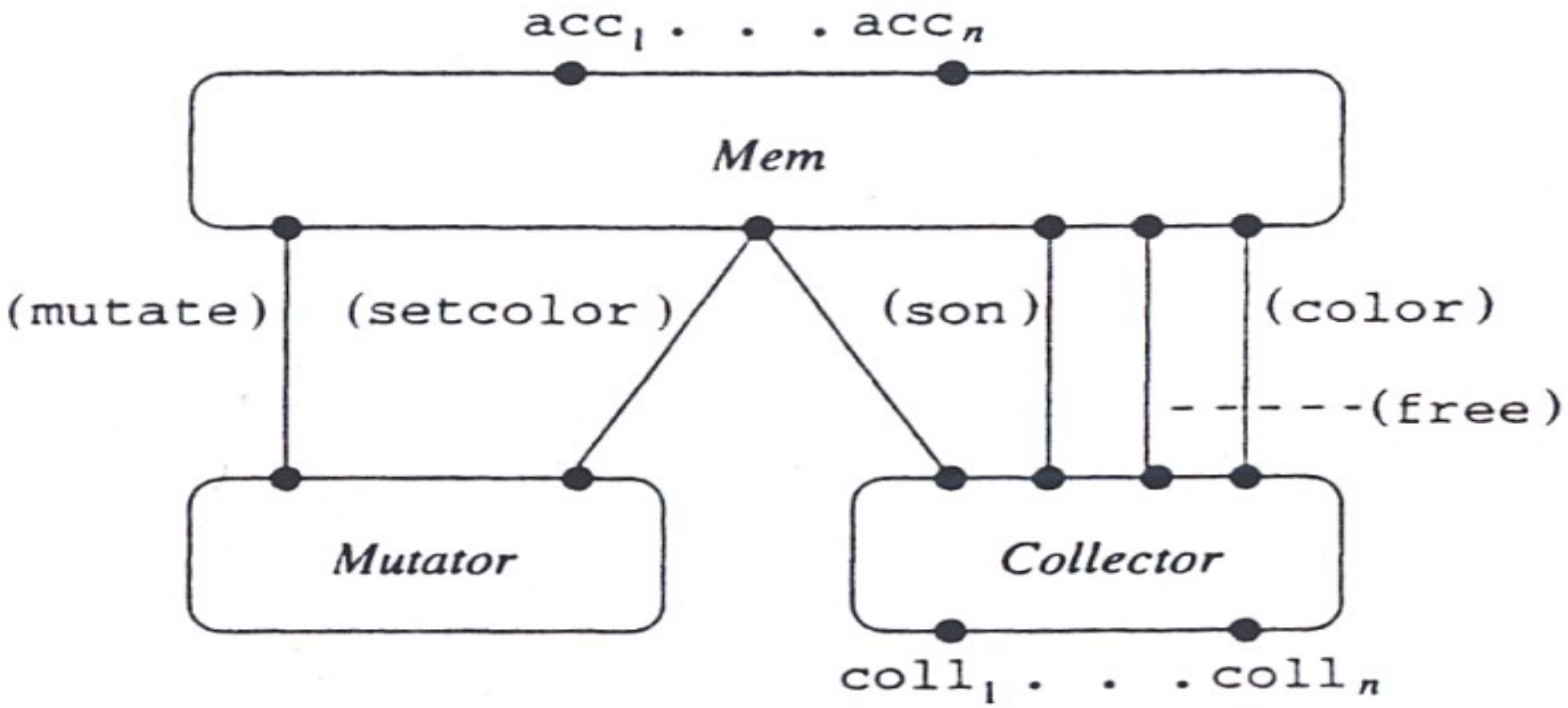
$$
\begin{aligned}
&Count(old, new, i) \stackrel{\text{def}}{=} \\
&\qquad \text{if } i > n \text{ then } (\text{if } old = new \text{ then } Collect(1) \\
&\qquad\qquad \text{else } Propagate(new, 1)) \\
&\qquad \text{else } \mathtt{color}_i(c).\text{if } c = black \\
&\qquad\qquad \text{then } Count(old, new + 1, i + 1) \\
&\qquad\qquad \text{else } Count(old, new, i + 1) \\
&Collect(i) \stackrel{\text{def}}{=} \\
&\qquad \text{if } i > n \text{ then } Collector \\
&\qquad \text{else } \mathtt{color}_i(c).\text{if } c = black \\
&\qquad\qquad \text{then } \overline{\mathtt{setcolor}_i}(white).Collect(i + 1) \\
&\qquad\qquad \text{else } \overline{\mathtt{coll}_i}.\overline{\mathtt{free}}(i).Collect(i + 1)
\end{aligned}
$$

Algorytm odśmiecania w locie Ben-Ariego

- Model całego systemu:

$$L \stackrel{\text{def}}{=} \bigcup_{(i,j)\in N\times N}\{\texttt{setcolor}_i, \texttt{color}_i, \texttt{son}_i, \texttt{mutate}_{i,j}, \texttt{free}\}$$

$$GC \stackrel{\text{def}}{=} (Mem(InitColors, InitSons) \mid Mutator \mid Collector)\backslash L$$



Algorytm odśmiecania w locie Ben-Ariego

- Bezpieczeństwo
  - nie może być możliwości wykonania jednocześnie *acc(i)* i *coll(i)*

$$Safe_i = Always([\mathtt{coll_i}]F \vee [\mathtt{acc_i}]F)$$

  - nasz model spełnia tę własność dla każdego *i*

- Żywotność:
  - każdy węzeł musi w końcu być dostępny

$$Live_i = Always(Ev(<\mathtt{acc_i}>T))$$

  - własność spełniona jedynie dla węzłów 1 i 2

Algorytm odśmiecania w locie Ben-Ariego

- Osłabiona własność żywotności
  - acc(i) zajdzie o ile zbieracz działa

$$Live_i = Always(\mu X.<\mathtt{acc_i}>T \vee [-]X)$$

$$Live2_i = Always(\mu X.\nu Y <\mathtt{acc_i}>T \vee ([-\mathtt{cl}]Y \wedge [\mathtt{cl}]X))$$

w każdej chwili albo daje się osiągnąć *acc(i)* albo akcja *cl* więcej nie zajdzie

  - ta własność jest spełniona dla wszystkich *i*

Algorytm odśmiecania w locie Ben-Ariego

- Jeszcze inne podejście do problemu żywotności:

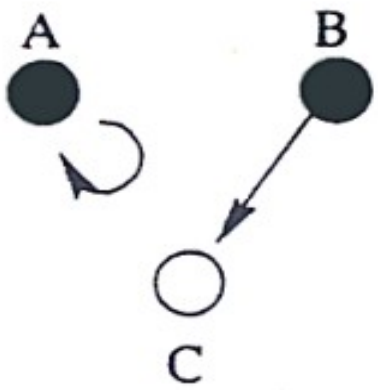
$$Live3_i = Always(<\texttt{acc}_\texttt{i}>T \vee eventually(<\texttt{coll}_\texttt{i}>T))$$

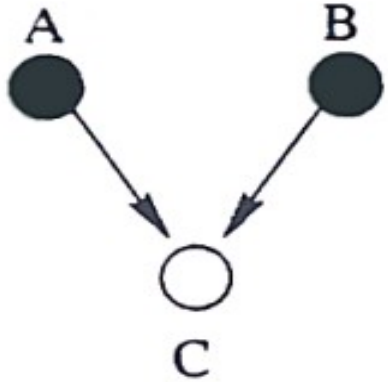
w każdej chwili jeśli węzeł jest niedostępny to zostanie w końcu zebrany

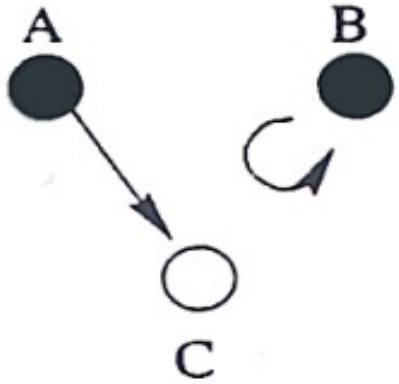
  - ta własność ponownie nie jest spełniona dla węzłów 3 i 4 (problemem jest nieatomowość zbierania - *coll(i).free(i)*)

Algorytm odśmiecania w locie Ben-Ariego

- Modyfikator bez zaczerniania (plik *gc2.vp*)
  - tracimy własność bezpieczeństwa
  - przy naszych założeniach jest ona zachowana







Algorytm odśmiecania w locie Ben-Ariego

- Modyfikator z zamienionymi kolejnością akcjami (*gc3.vp*)
  - zostało wykazane, że taka zmiana powoduje utratę własności bezpieczeństwa
  - znaczne modyfikacje w modelu
  - przy naszych założeniach własność ta jest zachowana
- Sprawdzanie własności bezpieczeństwo poprzez blokady (*gc4.vp*)
  - modyfikujemy proces *Mem*, tak by się blokował przy próbie zebrania dostępnego węzła