WS 2020 / 2021 08.02.2021

Übungen zur Vorlesung
Modern Concurrency Theory
Blatt 10

Prof. Dr. Roland Meyer
Sebastian Wolff

Abgabe bis 15.02.2021 um 10:00 Uhr

Aufgabe 10.1 (Linearisierungspunkte)

Geben Sie die Linearisierungspunkte des folgenden Programms an:

```
struct Node { int data; Node* next; }
class MichaelScottQueue {
  Node* Head, Tail;

  MichaelScottQueue() { // Konstruktor
    Head = Tail = new Node(); Head->data = 0; Head->next = NULL;
  }

  void enqueue(int x) {
    Node* node = new Node();
    node->data = x;
    node->next = NULL;
    while (true) {
      Node* tail = Tail;
      Node* next = tail->next;
      if (tail != Tail) continue;
      if (next != NULL) {
        CAS(Tail, tail, next);
        continue;
      }
      if (CAS(tail->next, next, node)) {
        CAS(Tail, tail, node);
        break;
  } } }

  (bool,int) dequeue() {
    while (true) {
      Node* head = Head;
      Node* tail = Tail;
      Node* next = head->next;
      if (head != Head) continue;
      if (next == NULL)
        return (false, 0);
      if (head == tail) {
        CAS(Tail, tail, next);
        continue;
      }
      int x = next->data;
      if (CAS(Head, head, next)) {
        return (true, x);
  } } }
}
```

Begründen Sie Ihre Wahl. Transformieren Sie das Programm, falls nötig. Erläutern Sie ihre Transformation kurz und erklären Sie, warum diese das Programmverhalten nicht ändert.

Aufgabe 10.2 (Fences)

Neben den in der Vorlesung eingeführten Befehlen gibt es noch sogenannte *Fences*, die in der Praxis eine wichtige Rolle spielen, um SC-Verhalten unter TSO zu erzwingen. Dazu können TSO-Programme den Befehl `mfence` verwenden, welcher nur ausgeführt werden kann, falls der Buffer des ausführenden Threads leer ist. (Intuitiv entspricht ein `mfence` einem `assume` $\mathit{buffer} = \epsilon$.)

Erweitern Sie die TSO-Transitionsrelation um Fences.

Aufgabe 10.3 (Kreise)

Betrachten Sie $\tau.act \in [\![P]\!]_{\mathrm{TSO}}$. Zeigen Sie: falls $\mathit{HBTrace}(\tau)$ kreisfrei ist aber $\mathit{HBTrace}(\tau.act)$ zyklisch, so ist act ein Store der Form $act = (t, st, adr, val)$.

Definition (Data Race)

Sei $\tau \in [\![P]\!]_{\mathrm{MM}}$ mit $\mathit{HBTrace}(\tau) = (N, \lambda, \rightarrow_{\mathrm{po}}, \rightarrow_{\mathrm{co}}, \rightarrow_{\mathrm{rf}})$. Dann enthält $\mathit{HBTrace}(\tau)$ ein *Data-Race*, falls zwei Knoten $n_1, n_2 \in N$ mit Beschriftung $\lambda(n_i) = (t_i, lab_i, adr_i, val_i)$ existieren für die gilt:

$$\begin{array}{rl} & adr_1 = adr_2 \quad \text{und} \quad n_1 \neq n_2 \\ \text{und} & \neg(n_1 \rightarrow_{\mathrm{po}}^{*} n_2) \quad \text{und} \quad \neg(n_2 \rightarrow_{\mathrm{po}}^{*} n_1) \\ \text{und} & (lab_1, lab_2) \in \{(ld, st), (st, ld), (st, st)\}\ . \end{array}$$

Vereinfachende Annahme

Sei $\tau.act \in [\![P]\!]_{\mathrm{TSO}}$ mit $act = (t, st, adr, val)$. Sei $\mathit{HBTrace}(\tau) = (N, \lambda, \rightarrow_{\mathrm{po}}, \dots)$ kreisfrei. Dann existieren $n \in N$ und $\sigma.act \in [\![P]\!]_{\mathrm{SC}}$ mit $\mathit{HBTrace}(\sigma) = (N', \lambda', \rightarrow_{\mathrm{po'}}, \dots)$ für die gilt:

$$\begin{array}{rl} & n = max(\rightarrow_{\mathrm{po}}^{t}) \quad \text{und} \quad \lambda(n) \neq act \\ \text{und} & N' = N \setminus \{n\} \quad \text{und} \quad \lambda' = \lambda|_{N \setminus \{n\}} \quad \text{und} \quad \rightarrow_{\mathrm{po'}} = \rightarrow_{\mathrm{po}}|_{N \setminus \{n\}}\ . \end{array}$$

Aufgabe 10.4 (DRF Theorem)

Zeigen Sie das *Data-Race-Freedom (DRF) Theorem*:

$$\text{Falls } \mathit{Traces}_{\mathrm{SC}}(P) \text{ frei von Data-Races ist,} \\ \text{so gilt } \mathit{Traces}_{\mathrm{TSO}}(P) \subseteq \mathit{Traces}_{\mathrm{SC}}(P).$$

Hinweis: Benutzen Sie das Lemma von Shasha&Snir und Aufgabe 10.3 um mit der vereinfachenden Annahme eine SC-Berechnung zu konstruieren. Weisen Sie ein Data-Race in dieser Berechnung nach.

Abgabe bis 15.02.2021 um 10:00 Uhr per Mail an sebastian.wolff@tu-bs.de.