APUVS, Blatt 9

Jan Fajerski and Kai Warncke and Magnus Müller

17. Januar 2011

Aufgabe 9.1

Die Variable `participant` wird benutzt um für den Fall, dass die Wahl von zwei Knoten gestartet wurde, einen Ausfall des Prozesses mit der höchsten Prozessnummer (also dem zukünftigen Leader) kompensieren zu können. Im Beispiel in Abbildung 1 starten 2 Knoten unabhängig voneinander eine Wahl. Jeder Knoten reagiert entsprechend dem Algorithmus von Seite 11 im Vorlesungsskript. Alle Knoten bis auf 6, die eine Nachricht verschickt haben, haben `participant` auf `true` gesetzt.
Erhält nun der Knoten 5 die Nachricht `<election, 4>`, also eine Nachricht aus dem nicht von ihm gestarteten Wahlprozess, so ernennt er sich selbst zum Leader und schickt dies als Nachricht an den folgenden Prozess (siehe Abbildung 2). Alle Prozesse aktzeptieren zunächst 5 als Leader.
Irgendwann erreicht auch die Wahlnachricht `<election, 6>` (als Teil der ursprünglich von 5 initiierten Wahl) Knoten 5. Dann gibt 5 seinen Leaderstatus wieder ab und 6 wird zum Leader gewählt (siehe Abbildung 3). Die Variable ist also nicht essentiell für den Algorithmus. Ohne `participant` müsste man einen Ausfall erkennen (beispielsweise durch einen Timeout) und eine neue Wahl veranlassen.
Mit `participant` hingegen wird auch bei Ausfall eines Prozesses ein Leader gewählt. Es dauert nur länger bis tatsächlich der Prozess mit der höchsten Prozessnummer als Leader gewählt ist.

Aufgabe 9.2

Die minimale Zeit T' ist in Abhängigkeit von i bei N Prozessen wie folgt definiert:

$$\begin{aligned} T'(N) &= T \\ T'(i) &= T'(i+1) + T_{PROCESS} + T_{TRANS} \end{aligned}$$

Der höchste Prozess (also derjenige mit der höchsten Prozessnummer) wird nie die den Ausführungszweig in dem die Zeit T' gewartet wird abarbeiten, da er nie eine Antwort auf seine e-Nachricht bekommt. Darum kann man T' des höchsten Prozesses einfach auf T setzen.

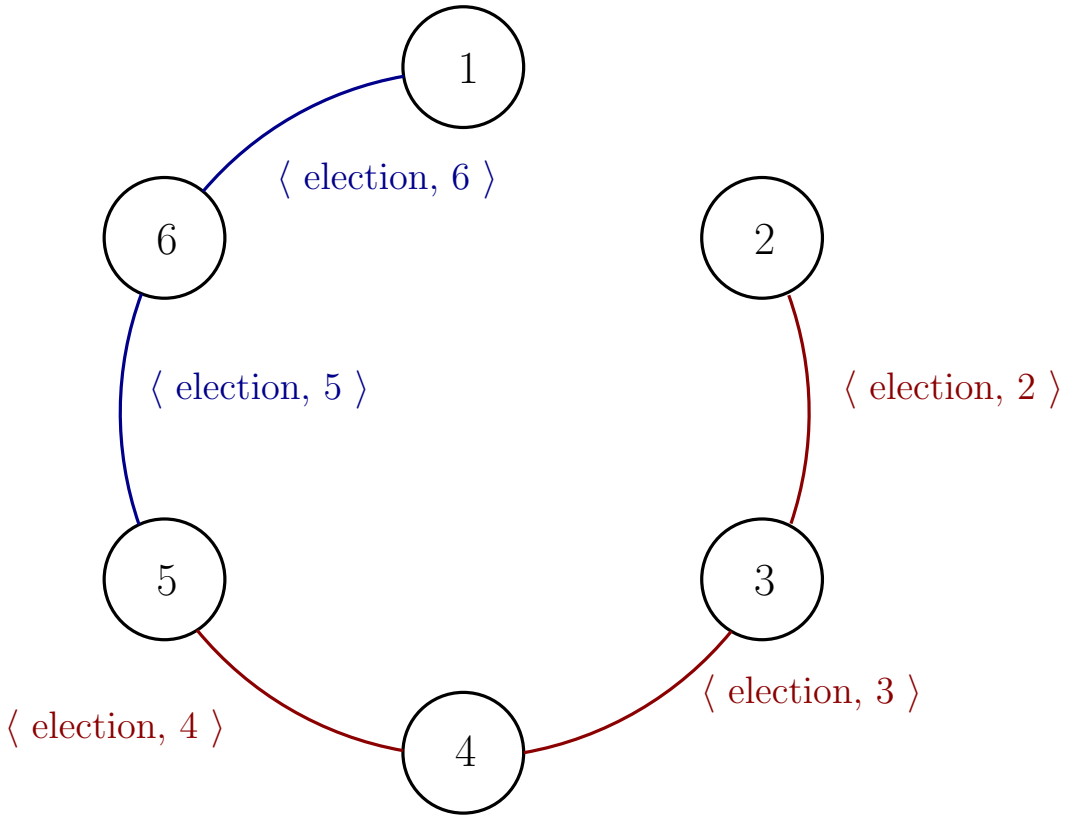

Abbildung 1: Knoten 2 und 5 starten eine Wahl

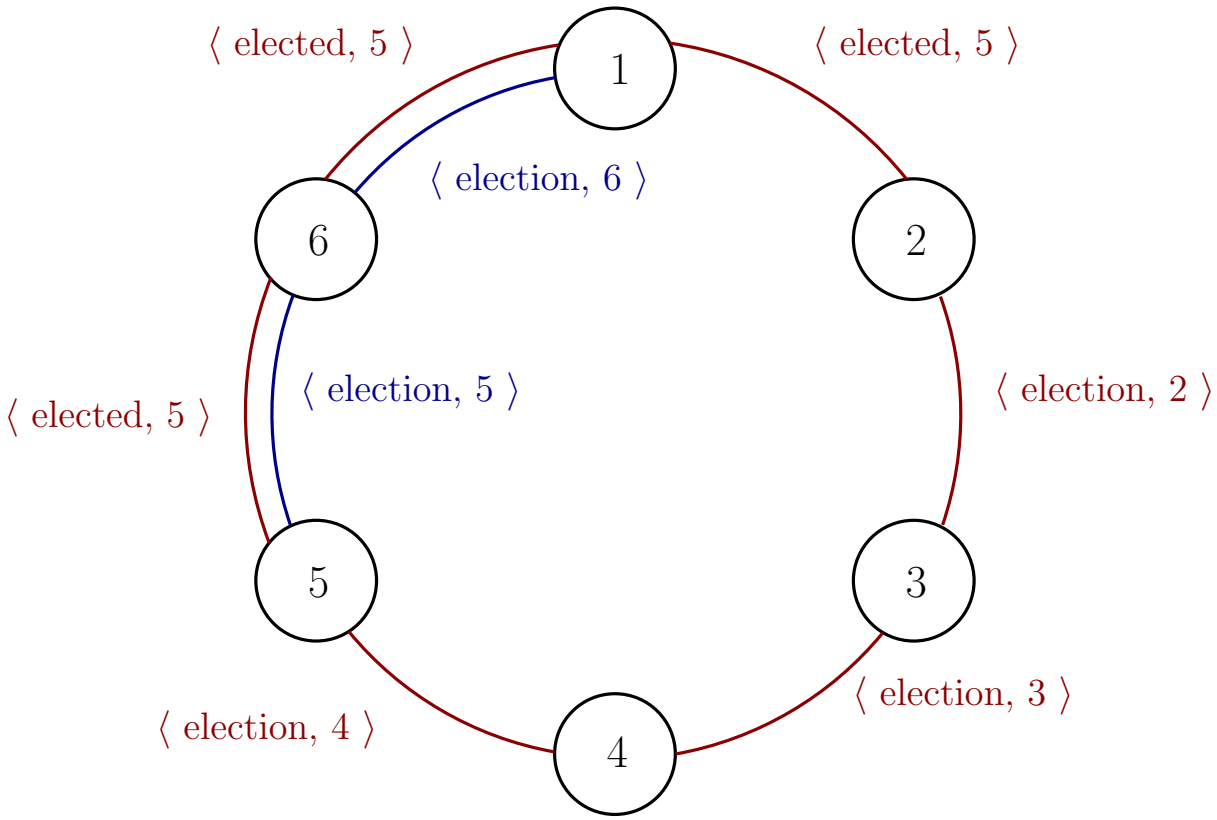

Abbildung 2: Knoten 5 ernennt sich selbst zum Leader

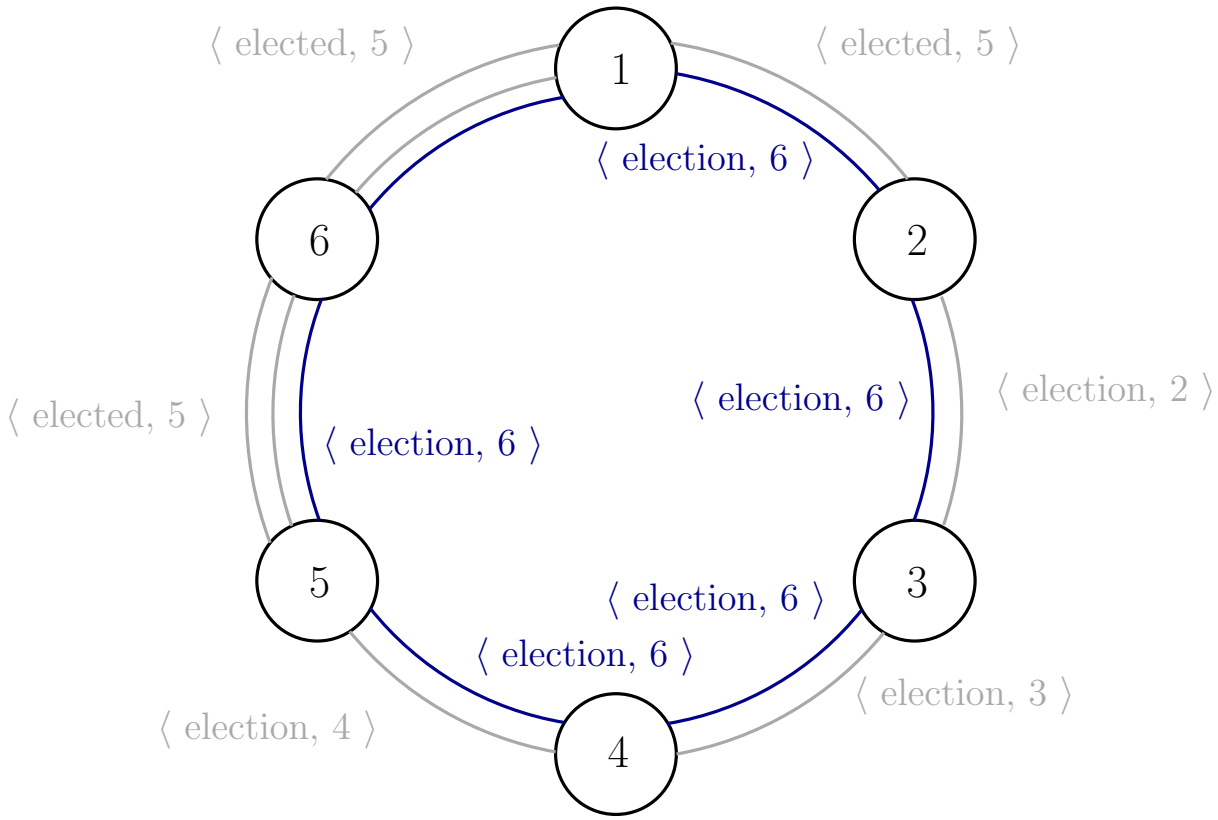

Abbildung 3: Knoten 6 wird letztendlich zum Leader gewählt

Der darunterliegende Prozess muss mindestens diese Zeit, plus die Zeit die Nächsthöhere zum Versenden der eigenen e-Nachricht benötigt ($T_{PROCESS}$) und der Nachrichtenlaufzeit der c-Nachricht benötigt, warten.

Aufgabe 9.3

Der unter `http://de.wikipedia.org/w/index.php?title=Echo-Algorithmus&oldid=50781161` beschriebene Algorithmus implementiert nicht den `echo`-Algorithmus. Beispielsweise zählen die Knoten, dir rot sind nicht die eingehenden `explore`-Nachrichten mit und der `Initiator` beendet sich sobald er *einen* `echo` Marker erhalten hat. Da wir die Funktion des Algorithmus nicht ergründen können, können wir nur davon ausgehen, dass die Farben nötig sind, da sie den Programmablauf beeinflussen.

Aufgabe 9.4

Jeder Knoten (ausser der `Initiator`) darf ausfallen *nachdem* er ein Token an seinen Vaterknoten geschickt hat. Fällt einer dieser Knoten vorher aus, terminiert der Algorithmus nicht, da jeder Knoten von jedem Nachbarn ein `Token` erwartet. Fällt der `Initiator` aus wird nie ein `Leader` gewählt. Dieser darf also nicht ausfallen.

Aufgabe 9.5

```
  received = 0;
  father = null;
  children = []; // liste der kinder

Initiator:
  forall (q in neighbours) do
    send <token> to q;
  while (received < #neighbours) {
    receive msg from q;
    if (msg == <child>) {
      children = [q | children];
    }
    received = received + 1;
  }
  decide

Participant:
  receive <token> from neighbour q;
  father = q;
  received = received + 1;
  forall (q in neighbours, q is not father) do
```

```
  send <token> to q;
while (received < #neighbours) do {
  receive msg from c;
  if (msg == <child>){
    children = [c | children];
  }
  received = received + 1;
}
send <child> to father;
```