

Konsensus und die Herlihy-Hierarchie

Prof. Dr. David Sabel

LFE Theoretische Informatik



Letzte Änderung der Folien: 2. November 2020

Fragestellung

- Jede Menge Nebenl. Objekte: Atomares Register, RMW-Objekt, Test-and-set-Objekt, usw.
- Welche Objekte sind „besser“ als die anderen?
- Was bedeutet „besser“?

Intuitiv klar:

- Atomares Register ist eher schwach (z.B. da Mutual-Exklusion schwer)
- RMW-Objekt ziemlich stark, kann viele andere Objekte implementieren

Übersicht

- 1 Einleitung
- 2 Motivationsbeispiel: Zwei Generäle-Problem
- 3 Das Konsensusproblem
- 4 Die Konsensus-Zahl
- 5 Universalität

TCS | 07 Konsensus und die Herlihy-Hierarchie | WS 2020/21 | 2/25 | Einleitung Zwei Generäle Konsensusproblem Konsensus-Zahl Universalität

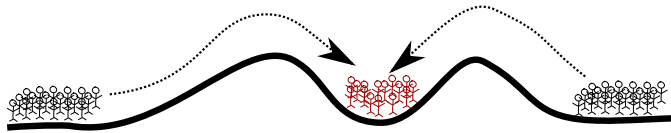
Herlihy's Modell

- Maurice Herlihy hat dies 1991 untersucht / formalisiert
- aber: Anderes Modell!
- Modell: Prozesse können jederzeit abstürzen
- Modellierung: Prozess bleibt vor einem Kommando hängen

Beachte: Alle gesehenen Mutual-Exklusion Algorithmen vertragen keine Abstürze!

Ein Beispiel für abstürzende Prozesse

Das Zwei-Generäle-Problem (auch Coordinated-Attack)

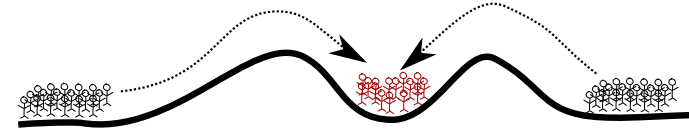


Fakten:

- Angriff nur erfolgreich, wenn beide Divisionen **zeitgleich** angreifen
- Kommunikation nur über **Boten**
- **Boten** müssen durch das Feindesland und kommen daher **nicht immer** an

Problem: Wie sprechen sich die Generäle ab?

Zwei Generäle-Problem

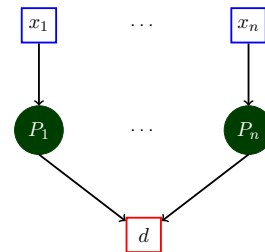


- Rechter General schickt Boten los: "Angriff um 12:00 mittags"
- Rechter General kann nicht sicher sein, dass der Bote ankommt
⇒ wartet auf Bestätigung vom linken General
- Linker General schickt Boten los: "Nachricht erhalten"
- Linker General kann nicht sicher sein, dass der Bote ankommt
⇒ wartet auf Bestätigung vom rechten General
- Rechter General schickt Boten los: "Nachricht erhalten"
- usw.

Tatsächlich: Problem ist unlösbar in diesem Modell!

Das Konsensusproblem

- n Prozesse, die auch abstürzen können
- Prozess i erhält einen **Eingabewert** $x_i \in \{0, 1\}$
- Programmieren die Prozesse, so dass alle (nicht-abstürzenden) Prozesse sich für einen gemeinsamen **Entscheidungswert** $d \in \{0, 1\}$ entscheiden
- **Übereinstimmung:** Alle nicht-abgestürzten Prozesse entscheiden sich für den gleichen Wert d .
- **Gültigkeit:** $d \in \{x_1, \dots, x_n\}$, d.h. d ist einer der Eingabewerte.



Beachte: Übereinstimmung \Rightarrow Algorithmus ist wait-frei:

Jeder nicht-abstürzende Prozess entscheidet sich nach endlich vielen Schritten

Das Konsensusproblem (2)

Bei nicht-abstürzenden Prozessen wäre das Problem einfach:

- Alle Prozesse teilen sich (über gemeinsamen Speicher) untereinander alle ihre Eingabewerte mit
- anschließend berechnet jeder Prozess $d = f(x_1, \dots, x_n) = d$
- wobei f eine feste Funktion ist, die alle Prozesse kennen, z.B. $f(x_1, \dots, x_n) = x_1$.

Lösung mit dreiwertigem RMW-Objekt

Objekte und Initialisierung:

x : RMW-Objekt mit den möglichen Werten $\perp, 0, 1$, initial \perp

x_i : Eingabewert von Prozess i

d_i : Entscheidungswert, den Prozess i trifft.

Programm des i . Prozesses

- (1) $d_i := \text{read-modify-write}(x, f_i)$;
- (2) if $d_i = \perp$ then
 $d_i := x_i$;

Funktion f_i des i . Prozesses

```
function  $f_i(v)$   
  if  $v = \perp$  then return  $x_i$   
  else return  $v$   
end function
```

} erster Prozess setzt sein x_i als neuen Wert,
alle anderen nicht-abstürzenden Prozesse lesen diesen
Wert
⇒ alle d_i -Werte identisch

Lösung des Konsensusproblems

Mit dreiwertigem RMW-Objekt:

- Der Algorithmus ist eine wait-freie Lösung
- für beliebig viele Prozesse

Resultat für **atomare Register mit read und write**:

Selbst für 2 Prozesse und nur einen Absturz nicht lösbar:

Theorem

Es gibt keinen Konsensus-Algorithmus für atomares *read* und *write*, der einen Absturz tolerieren kann.

Folgerung: Atomare Register können kein RMW-Objekt simulieren

Beweisskizze

- Statt direkt mit Abstürzen zu argumentieren zeigen wir:
 Ein wait-freier Konsensusalgorithmus für atomare Register
 und 2 Prozesse existiert nicht
- Algorithmen, die Abstürze tolerieren müssen wait-frei sein, denn:
 nicht wait-frei
 ⇒ Prozess wartet auf Bedingung, die ein anderer Prozess wahr macht
 ⇒ Endlosschleife, wenn anderer Prozess abstürzt
 ⇒ Absturz nicht tolerierbar

Beweisskizze (2)

- Wir verwenden Berechnungsbäume, um die Ausführung zweier Prozesse P_1 und P_2 zu betrachten
- Berechnungsbaum:
 - Knoten: Zustände der Prozesse und des Speichers
 - Kanten: Wesentliche Schritte, d.h. Lese- oder Schreiboperation auf dem gemeinsamen Speicher (interne Schritte sind zusammengefasst)
 - Kante nach links = Schritt von P_1
 - Kante nach rechts = Schritt von P_2 • ...

Beweisskizze (3)

- Berechnungsbaum:
 - ...
 - Blätter sind mit Entscheidungswert $\in \{0, 1\}$ markiert.
 - Innere Knoten mit den noch möglichen Entscheidungswerten.
 - Bivalenter Knoten: Markierung 0 und 1
 - Univalenter Knoten: Markierung 0 (0-valent) oder Markierung 1 (1-valent)
 - Da Algorithmus wait-frei, ist der Baum endlich!

Beweisskizze (4)

Aussage:

Jeder 2-Prozess-Konsensusalgorithmus hat einen bivalenten Anfangszustand.

Beweis:

- Nehme an P_1 hat Eingabewert $x_1 = 0$ und P_2 hat Eingabewert $x_2 = 1$.
- Wenn nur P_1 Schritte macht (linkster Pfad im Baum), dann muss P_1 als Entscheidungswert 0 berechnen
- Wenn nur P_2 Schritte macht, dann muss als Entscheidungswert 1 berechnet werden. □

Beweisskizze (5)

Ein Knoten ist **kritisch**, wenn er bivalent ist und jeder Nachfolger ist univalent.

Aussage:

Jeder Berechnungsbaum mit bivalenten Anfangszustand zu einem Konsensusalgorithmus hat einen kritischen Knoten.

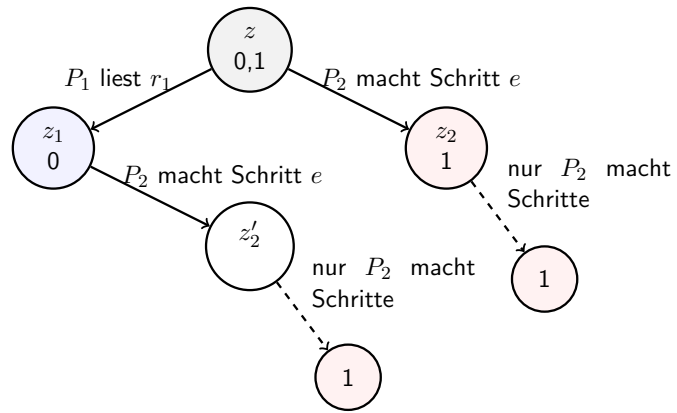
Beweis:

- Angenommen das gilt nicht.
- Starte mit einem bivalenten Anfangszustand.
- Solange es Schritte gibt, die zu einem bivalenten Zustand führen, führe diese Schritte aus.
- Wenn das unendlich lange möglich ist, dann ist der Algorithmus nicht wait-frei. Widerspruch. □

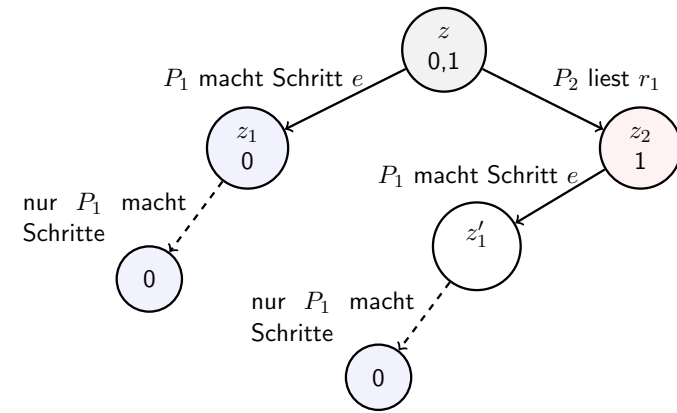
Beweisskizze (6)

- Sei nun ein Konsensusalgorithmus für 2 Prozesse gegeben.
- Gehe zu einem kritischem bivalenten Zustand z
- O.b.d.A. linkes Kind ist 0-valent, rechtes Kind ist 1-valent
- Untersuche alle Fälle.

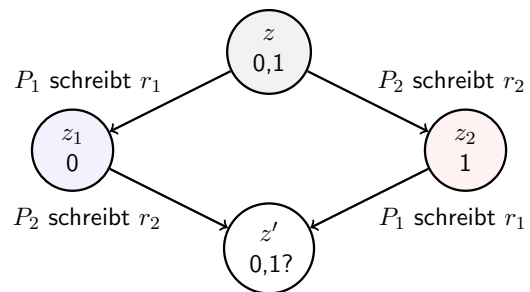
Beweisskizze (7)



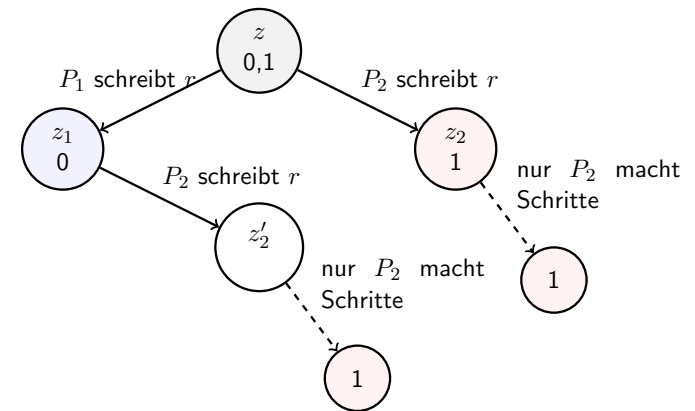
Beweisskizze (8)



Beweisskizze (9)



Beweisskizze (10)



Die Konsensus-Zahl

Definition

Für ein nebenläufiges Objekt vom Typ o ist die **Konsensus-Zahl** $\mathbb{CN}(o)$ die größte Zahl an Prozessen n für die man das Konsensus-Problem für n Prozesse lösen kann, indem man beliebig viele Objekte vom Typ o und beliebig viele atomare Register (mit *read* und *write*) verwendet.

Ist die Anzahl unbeschränkt, so sei $\mathbb{CN}(o) = \infty$.

Konsensus-Hierarchie

$\mathbb{CN}(o)$	Objekt o
1	atomares Register mit <i>read</i> und <i>write</i>
2	test-and-set-Objekt, fetch-and-increment-Objekt, fetch-and-add-Objekt, swap-Objekt, read-modify-write-Bit
$\Theta(\sqrt{m})$	swap ^{m} -Objekt
$2m - 2$	m -Register mit m -facher Zuweisung ($m > 1$)
∞	(drei-wertiges) RMW-Objekt, Compare-and-swap-Objekt, Sticky-Bit

Eigenschaften der Konsensus-Hierarchie

Theorem

Wenn o_1 und o_2 zwei Objekte sind mit $\mathbb{CN}(o_1) < \mathbb{CN}(o_2)$, dann gilt für ein System mit $\mathbb{CN}(o_2)$ Prozessen:

- Es gibt keine wait-freie Implementierung von Objekt o_2 ausschließlich mit Objekten vom Typ o_1 und atomaren Registern.
- Es gibt eine wait-freie Implementierung von Objekt o_1 ausschließlich mit Objekten vom Typ o_2 und atomaren Registern.
- Für Teil 1: Gäbe es eine Implementierung von o_2 -Objekten mit o_1 -Objekten, könnte man damit auch das Konsensusproblem für $\mathbb{CN}(o_2)$ Prozesse lösen. Dann gilt $\mathbb{CN}(o_1) \geq \mathbb{CN}(o_2)$. Widerspruch!
- Für Teil 2: Die Aussage folgt aus dem nächsten Theorem (nächste Folie), welches auch besagt: Mit o_2 -Objekten und atomaren Registern kann man jedes Objekt mit sequentieller Beschreibung für $\mathbb{CN}(o_2)$ Prozesse wait-frei implementieren.

Universalität

Theorem

Ein Objekt o mit $\mathbb{CN}(o) \geq n$ ist **universell** in einem System mit n Prozessen. D.h. jedes wait-freie Objekt (mit einer sequentiellen Beschreibung) kann durch Objekte vom Typ o und atomaren Registern in einem System mit n Prozessen implementiert werden.

Bemerkungen:

- Wir verzichten auf den genauen Beweis (dieser ist in der Literatur zu finden).
- Wesentliche Ideen des Beweises:
 - Jeder Prozess speichert die Zugriffe auf die Objekte und rechnet die Nachfolgestände der Objekte solange aus bis sein Zugriff stattfindet.
 - Ausrechnen ist aufgrund der sequentiellen Beschreibung kein echtes Problem.
 - Die Hürden sind:
 - Wann führt welcher Prozess eine Operation aus?
Für diese Entscheidungen wird der Konsensusalgorithmus verwendet!
 - Sicherstellen, dass die Operationen wait-frei sind.

Universalität (2)

Konsequenzen des Theorems:

- Universelle Objekte können viele andere Objekte implementieren (unabhängig vom Konsensus-Problem)
- Objekte mit Konsensus-Zahl $\geq n$ "reichen aus".