12 Einigungsprotokolle

12.1 Erläuterung der Fragestellung

12.2 Unlösbarkeitsaussagen

Fischer, M. J.; Lynch, N. A.; Merritt, M.: Easy impossibility proofs for distributed consensus problems. Distributed Computing, Vol. 1, 1986, pp. 26-39.

Fischer, M. J.; Lynch, N. A.; Paterson, M. S.: Impossibility of Distributed Consensus with One Faulty Process. Journal of the ACM, Vol. 32, No. 2, 1985, pp. 374-382.

12.3 Byzantinische Verständigung

Lamport, L.; Shostak, R.; Pease, Marshall: The Byzantine Generals Problem. ACM Transactions on Pogramming Languages and Systems, Vol. 4, No. 3, July 1982, pp. 382-401.

Pease, M.; Shostak, R.; Lamport, L.: Reaching Agreement in the Presence of Faults. Journal of the ACM, Vol. 27, No. 2, April 1980, pp. 228-234.

Dolev, D.; Fischer, M. J.; Fowler, R.; Lynch, N.; Strong, H. R.: An Efficient Algorithm for Byzantine Agreement without Authentication. Information and Control, vol. 52 (1982), pp. 257-274.

Burns, J. E.; Neiger, G.: Fast and simple distributed consensus. Distributed Computing, Vol. 8 (1994), pp. 59-64.

12.4 Paxos

Lamport, L.: The Part-Time Parliament. Research Report 49, Digital Equipment Corporation Systems Research Center, Palo Alto, CA, September 1989.11

De Prisco, R.: Revisiting the Paxos Algorithm. M. S. Thesis, Massachusets Institute of Technology, Laboratory for Computer Science, Cambridge, MA, June 1997. Technical Report MIT-LCS-TR-717, Lab. for Computer Science, MIT Cambridge, MA, USA, June 1997.

http://theory.lcs.mit.edu/tds/paxos.html

12.1 Erläuterung der Fragestellung

Systemmodell

- n Prozessoren, davon maximal m fehlerhaft.
- Voll vermaschtes Nachrichtensystem, nicht beeinträchtigt durch fehlerhafte Prozessoren.
- Zu jeder empfangenen Nachricht ist dem Empfänger der Absender bekannt.
- Das Kommunikationssystem ist zuverlässig (d. h. lediglich Prozessoren können fehlerhaft sein).
- Einigung soll zwischen zwei Möglichkeiten (0 und 1) erzielt werden.
- Abarbeitung in synchronisierten Runden oder asynchron.

Prozessorfehler

- Fail Stop
- Unterlassung von Nachrichtenversand
- Boshaftes Fehlverhalten (Byzantinisches Fehlverhalten)

Beglaubigte versus unbeglaubigte Nachrichten (signed versus oral)

- Beglaubigt meint, daß Nachrichten unterschrieben werden können und
  - die Unterschriften fehlerfreier Prozessoren nicht fälschbar sind,
  - der Inhalt einer Nachricht, die von einem fehlerfreien Prozessor unterzeichnet ist, nicht verfälscht werden kann und
  - jeder die Authentizität bestimmter Unterschriften feststellen kann.

Problemklassifikation

Einigung aller fehlerfreien Prozessoren auf einen Wert, der von einem Prozessor vorgegeben wird (Verständigungsproblem, 'agreement problem').

- Einigungsziel: Alle fehlerfreien Prozessoren einigen sich auf den gleichen Wert.
- Richtigkeit: Falls der vorgebende Prozessor fehlerfrei ist, erfolgt die Einigung auf seinen Wert.

Einigung auf einen Wert, der eine Funktion der Anfangswerte aller Prozessoren ist (Einigungsproblem, 'consensus problem').

- Einigungsziel: Alle fehlerfreien Prozessoren einigen sich auf den gleichen Wert.
- Richtigkeit: Wenn alle fehlerfreien Prozessoren den gleichen Wert vorgeben, einigen sich alle auf diesen Wert.

Einigung auf die Menge der Anfangswerte der Prozessoren (Konsistenzproblem, 'consistency problem').

- Einigungsziel: Alle fehlerfreien Prozessoren einigen sich auf den gleichen Wertevektor.
- Richtigkeit: Die Vorgabewerte fehlerfreier Prozessoren werden in dem Wertevektor richtig wiedergegeben.

Verwandtschaft der Probleme

Lösung des Verständigunsproblems --> Lösung des Konsistenzproblems

Jeder Knoten startet als Anführer eine Durchführung des Verständigungs-algorithmus, wodurch jeder Knoten einen Wertevektor mit den verlangten Eigenschaften ermittelt.

Lösung des Konsistenzproblems --> Lösung des Einigungsproblems

Das Ergebnis wird als mehrheitlicher Wert der erlangten Wertevektoren bestimmt. Enthalten die Wertevektoren beide Werte gleich oft, wird als Ergebnis der Wert 1 genommen.

Lösung des Einigungsproblems --> Lösung des Verständigunsproblems

1. Runde: Der Anführer sendet seinen Wert an alle einschließlich sich selbst.
2. Runde: Alle Knoten einschließlich Anführer führen den Einigungsalgorithmus durch, wobei sie als Anfangswert den in der 1. Runde erhaltenen Wert nehmen.

12.2 Unlösbarkeitsaussagen

Unlösbarkeit bei asynchronen Nachrichten

Modell

- Das System besteht aus n Prozessoren.
- Jeder Prozessor besitzt einen (allen anderen unbekannten) Anfangswert 0 oder 1.
- Jeder Prozessor besitzt unbeschränkten lokalen Speicher.
- Kommunikation erfolgt über ein zuverlässiges Nachrichtensystem.

  Die Operation send(p, m) übergibt die Nachricht (p, m) an das Nachrichtensystem. Dabei bezeichnet p den Empfänger und m den eigentlichen Nachrichteninhalt.

  Die Operation receive(p) entnimmt ein Paar (p, m) aus dem Nachrichtensystem und liefert m zurück oder sie liefert als Ergebnis Φ und läßt das Nachrichtensystem unverändert (asynchrones Empfangen).

S12.1 Satz

Es existiert kein Protokoll, das unter Verwendung asynchroner Nachrichten das Einigungsproblem lösen kann, wenn auch nur ein Prozessor fehlerhaft ist.

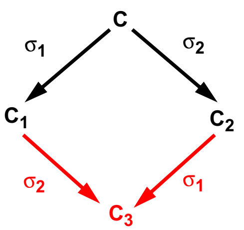
Beweis:

- **Bezeichnungen**
  - **Konfiguration: Umfaßt die Zustände sämtlicher Prozessoren und des Nachrichtensystems.**
  - **Schritt: Überführt eine Konfiguration in die nächste durch eine Zustandsänderung, die ein einzelner Prozessor bewirkt durch Ausführung von receive oder durch lokalen Übergang mit eventuellem Aussenden von Nachrichten.**
  - **Ereignis: Die Prozessoren sind als deterministisch vorausgesetzt, so daß ein Übergang vollständig durch ein Ereignis $e = (p, m)$ bestimmt ist. Eine Konfiguration C geht durch ein Ereignis e in die Konfiguration $e(C)$ über.**
  - **Lokales Ergebnis: Wert für den sich der Prozessor unwideruflich entschieden hat.**
  - **Ablaufplan: (Endliche oder unendliche) Folge σ von Ereignissen, die auf C angewandt werden kann.**
  - **Lauf: Schrittfolge, die bei Abwicklung eines Ablaufplans entsteht.**

- **Bezeichnungen (Forts.)**
  - **Erreichbar: Wenn σ endlich ist, heißt die am Ende des Ablaufs erreichte Konfiguration erreichbar.**
  - **Zulässiger Lauf: Lauf mit höchstens einem fehlerhaften Prozessor, in dem alle von fehlerfreien Prozessoren ausgesandten Nachrichten schließlich ankommen.**

L12.1 **Lemma**

Ausgehend von der Konfiguration C führe σ_1 bzw. σ_2 zu C_1 bzw. C_2. Falls die Prozesse, die in σ_1 einen Schritt ausführen, verschieden sind von all denen, die in σ_2 einen Schritt ausführen, dann kann σ_2 auf C_1 angewandt werden und σ_1 auf C_2 und beides führt zum gleichen Zustand C_3.



Beweis: Unmittelbare Folgerung der Definitionen.

Beweis des Satzes

- **Indirekt: Angenommen es gäbe ein Protokoll, das auch bei einem fehlerhaften Prozessor das Einigungsproblem löst.**

  Definition:
  Sei C eine Konfiguration und V die Menge lokaler Ergebnisse von C aus erreichbarer Konfigurationen.
  - **C heißt bivalent, wenn $|V| = 2$ ist.**
  - **C heißt i-valent, wenn $|V| = 1$ ist und die lokalen Ergebnisse, den Wert i besitzen.**

L12.2 **Lemma**

Es gibt eine bivalente Anfangskonfiguration.

Beweis:

- **Angenommen es gibt nur 0- und 1-valente Anfangskonfigurationen.**
- **Anfangskonfigurationen differieren nur in den Eingaberegistern. Also müssen zwei Anfangskonfigurationen C_0 und C_1 existieren, die sich nur im Wert des Eingaberegisters eines Prozessors P_p unterscheiden und von denen eine 0-valent und die andere 1-valent ist. (Man betrachte die Anordnung der Anfangskonfigurationen, in der die Vektoren aus den Eingaberegisterinhalten nach Gray-Code angeordnet sind.)**
- **σ sei ein zulässiger Lauf, der ohne Mitwirken von P_p von C_0 ausgehend zur Entscheidung führt (das muß gehen, da P_p der ausgefallene Prozessor sein könnte).**
- **σ kann auch auf C_1 angewandt werden.**
- **Die resultierenden Konfigurationen unterscheiden sich nur im lokalen Zustand von P_p.**
- **Gemäß Voraussetzung ermitteln $\sigma(C_0)$ und $\sigma(C_1)$ die gleichen Entscheidungswerte im Widerspruch dazu, daß C_0 0-valent ist und C_1 1-valent.**

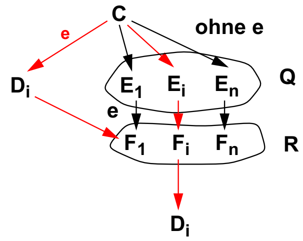
L12.3 **Lemma**

Sei C eine bivalente Konfiguration und $e = (p, m)$ ein in dieser Konfiguration anwendbares Ereignis. Weiter sei Q die Menge der Konfigurationen, die von C aus ohne Mitwirkung von P_p erreicht werden können, und es sei $R = e(Q)$.

Dann enthält R eine bivalente Konfiguration.

Beweis:

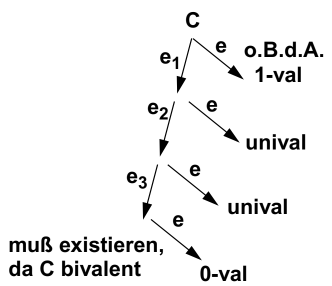
Angenommen R enthalte keine bivalente Konfiguration.



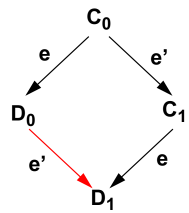
- **D_i sei eine von C aus erreichbare i-valente Konfiguration (die nach Vorraussetzung für $i = 0$ und $i = 1$ existieren muß).**
- **Dann gilt entweder $D_i \in Q \wedge F_i = e(D_i) \in R$ oder e wurde angewandt, um D_i zu erreichen, und $D_i \in R$ oder es existiert $F_i \in R$, das von E_i aus erreichbar ist.**
- **R muß also auf jeden Fall 0- und 1-valente Konfigurationen enthalten.**

Beweis (Forts.)

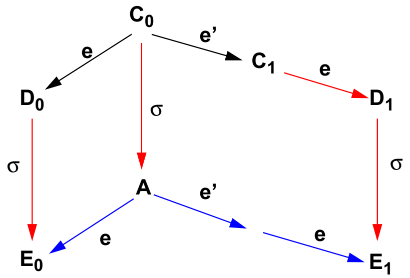
- **Zwei Konfigurationen werden als Nachbarn bezeichnet, wenn die eine in einem einzigen Schritt in die andere überführt werden kann.**
- **Es existieren Nachbarn $C_0, C_1 \in Q$ derart, daß $D_i = e(C_i)$ i-valent ist. Die Indizierung der Fi sei so gewählt, daß daß die 0-valenten kleiner Indizes haben als die 1-valenten.**



- **Sei $C_1 = e'(C_0)$ mit $e' = (p', m')$.**
  - **Ist $p' \neq p$, dann gilt $D_1 = e'(D_0)$, was unmöglich ist, da jeder Nachfolger einer 0-valenten Konfiguration wieder 0-valent ist.**



- Wenn $p = p'$ ist, betrachte man irgendeinen Entscheidungslauf von C_0 aus, an dem sich der Prozessor P_p nicht beteiligt.
  - Sei σ ein entsprechender Ablaufplan und $A = \sigma(C_0)$.
  - σ ist auch auf D_i anwendbar und führt zu einer i-valenten Konfiguration $E_i = \sigma(D_i)$.
  - Wegen $e(A) = E_0$ und $e(e'(A)) = E_1$ muß A bivalent sein im Widerspruch zur Annahme, daß A ein bis zur Entscheidung geführter Lauf ist.



Also führen beide Fälle zum Widerspruch und somit ist die Annahme falsch.

Beweis des Satzes (Fortsetzung)

Durch Konstruktion eines nicht-entscheidenden, beliebig langen Laufs

- Prozesse werden in einer Warteschlange geführt.
- Nachrichten werden in einem Nachrichtenpuffer gesammelt.
- Ein Abschnitt besteht aus einem oder mehreren Schritten. Er endet, wenn der erste Prozeß der Warteschlange einen Schritt ausgeführt hat, in dem die älteste, an ihn gerichtete und noch nicht bearbeitete Nachricht angenommen wurde (vorausgesetzt wird, daß es eine solche Nachricht gibt.)
- Der Prozeß wird dann wieder an das Ende der Warteschlange verwiesen.
- In einer endlosen Folge von Abschnitten führt jeder Prozeß unendlich viele Schritte aus und empfängt jede an ihn gerichtete Nachricht.

- Sei C_0 eine bivalente Anfangskonfiguration. Dann existieren Abläufe derart, daß jeder Abschnitt wieder mit einer bivalenten Konfiguration endet.
  - C sei eine bivalente Konfiguration und P_p der erste Prozeß in der Warteschlange. m oder ϕ sei die älteste Nachricht an P_p in C.
  - Sei $e = (p, m)$. Nach Lemma 3 gibt es eine bivalente Konfiguration C', die von C aus mit einem Ablaufplan erreichbar ist, in dem die Anwendung von e der letzte Schritt ist. Diese Folge definiert einen Abschnitt.
  - Da alle Abschnitte mit einer bivalenten Konfiguration enden, wird in dem so konstruierten Ablaufplan nie eine Entscheidung erreicht.

Unlösbarkeit bei zu vielen fehlerhaften Prozessoren

Modell eines verteilten Systems

- Kommunikationsgraph:

  Gerichteter Graph G mit Knotenmenge $k(G)$ und Verbindungsmenge $v(G)$

  Verbindungen treten nur paarweise auf, d. h. $v(k_1, k_2) \in v(G) \Leftrightarrow v(k_2, k_1) \in v(G)$

  (k_1, k_2) ist Ausgang von k_1 und Eingang von k_2.

  Ein Untergraph G_U von G ist die Menge aller Knoten aus U zusammen mit allen Verbindungen von $v(G)$, die Eingang oder Ausgang wenigstens eines Knotens aus U sind.

  Die Eingangsschnittstelle eines Untergraphen U ist die Menge aller Eingänge von Knoten aus U, die Ausgänge von Knoten aus $G - U$ sind.

- **System:**

  Ein System S ist ein Kommunikationsgraph, bei dem jedem Knoten eine *Einrichtung*, die ein Verhalten des Knotens erzeugt, und eine *Eingabe* zugeordnet ist.

  Ein System S hat ein *Systemverhalten* E, das sich als Tupel des *Verhaltens* aller Knoten und Verbindungen ergibt.

  Die Restriktion des Systemverhaltens E auf Knoten und Verbindungen eines Untergraphen G_U bildet das Szenario E_U von G_U.

- **Lokalitätsaxiom:**

  Seien S bzw. S' Systeme mit Verhalten E bzw. E' und isomorphen Untersystemen U bzw. U'. Wenn das Verhalten korrespondierender Eingänge von U und U' in den Szenarien E und E' gleich ist, dann sind auch die Szenarien E_U und $E'_{U'}$ gleich.

  (D. h. insbesondere, daß das Verhalten eines Teilsystems durch das Verhalten seiner Knoten und inneren Verbindungen sowie dem seiner Eingänge bestimmt ist.)

- **Fehleraxiom:**

  Sei A eine Einrichtung. Weiter seien $(E_1, \ldots, E_d)$ Verbindungsverhalten derart, daß E_i in einem Systemverhalten E^i das Verhalten des i-ten Ausgangs eines Knotens mit zugeordneter Einrichtung A ist. Außerdem sei u ein Knoten mit d Ausgängen $(u, v_1), \ldots, (u, v_d)$.

  Dann existiert eine Einrichtung F derart, daß in jedem System, in dem u die Einrichtung F zugeordnet ist, das Verhalten der Ausgangskanten (u, v_i) gleich E_i ist.

  Um diesen Sachverhalt deutlich zu machen, wird für F in diesem Fall $F_A(E_1, \ldots E_d)$ geschrieben.

- **Graphentheoretische Begriffe:**

  Ein Graph S überlagert einen Graphen G, wenn eine Abbildung der Knoten von S auf die von G existiert mit folgender Eigenschaft:

  Wenn Knoten u aus S mit d Nachbarknoten $v_1, \ldots, v_d$ verbunden ist und auf w aus G abgebildet wird, dann hat w ebenfalls d Nachbarn $x_1, \ldots, x_d$, wobei für $1 \le i \le d$ Knoten v_i auf Knoten x_i abgebildet wird.

  (D. h. S sieht lokal wie G aus.)

S12.2 Satz

In einem System aus n Prozessoren, von denen maximal m fehlerhaft sind, können keine Protokolle zur Lösung des Problems der Byzantinischen Einigung (Byzantine consensus) existieren, wenn $n \le 3m$ ist.

Beweis:

1. **Sei n = 3 und m = 1**

Angenommen die Einrichtungen A, B und C könnten bei der gezeichneten Zuordnung in G immer Einigung erzielen.

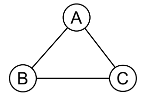
In der Überlagerung S ordne man den Knoten a und a' die Einrichtung A, den Knoten b und b' die Einrichtung B und den Knoten c und c' die Einrichtung C zu.

Nach Fehleraxiom existiert eine Einrichtung $F_A(E_{(a, b)}, E_{(a', c)})$ die für $E_{a, b}$ und $E_{a', c}$ das gleiche Eingangsverhalten erzeugt, wie das System S.

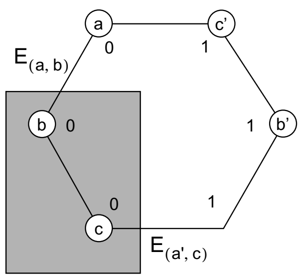
Ersetzt man in G die Einrichtung A durch $F_A(E_{(a, b)}, E_{(a', c)})$, so ist nach Lokalitätsaxiom $E_{b, c} = E_{B, C}$.
Auf Grund der Gültigkeitsforderung müssen sich B und C, und damit auch b und c auf den Wert 0 einigen.

Eine analoge Betrachtung für $E_{c, a'}$ führt zusammen mit der Tatsache, daß c zum Wert 0 kommen muß dazu, daß auch a' zum Ergebnis 0 kommen muß.

Graph G



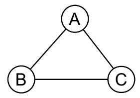
Überlagerung S



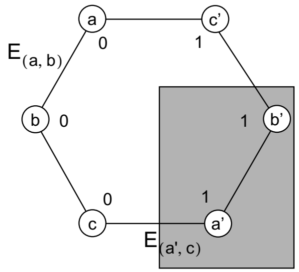
Wiederholung dieser Betrachtung für a' und b' zeigt, daß auch b' zum Ergebnis 0 kommen muß.
Andererseits schließt man analog zur anfänglichen Überlegung, daß sich die Knoten a' und b' auf den Wert 1 einigen müßten.

Dieser Widerspruch zeigt, daß die Annahme falsch ist.

Graph G



Überlagerung S



2. Allgemeiner Fall $n \le 3m$

Es werde wiederum angenommen, das Einigungsproblem sei lösbar.

Man partitioniere die Menge der Knoten in Teilmengen a, b und c derart, daß jede Teilmenge wenigstens ein und höchstens m Elemente enthält.

Die Vereinigung von je zwei Teilmengen enthält mindestens n-m Elemente.

A (B, C) sei die Menge der Einrichtungen von Knoten in a (b, c), die das Einigungsproblem lösen.

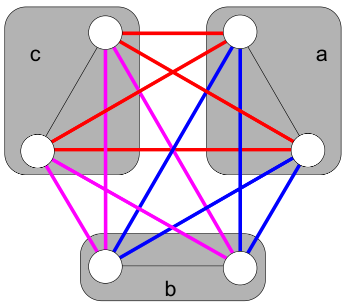
Man konstruiere eine Überlagerung durch Verdopplung des Graphen (Bezeichnungen des Doppels entstehen durch Anfügen von Apostroph) und Kantenersetzungen nach folgender Vorschrift

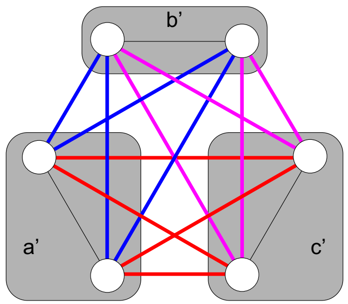
$x \in a \wedge y \in c \wedge (x, y) \in v(G) \Rightarrow (x, y') \in v(S)$

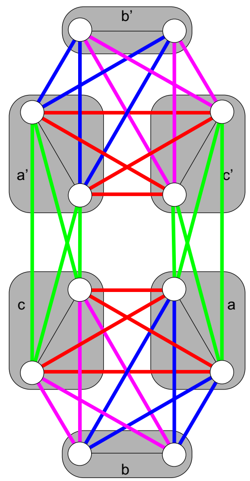
$x \in a \wedge y \in c \wedge (y, x) \in v(G) \Rightarrow (y', x) \in v(S)$

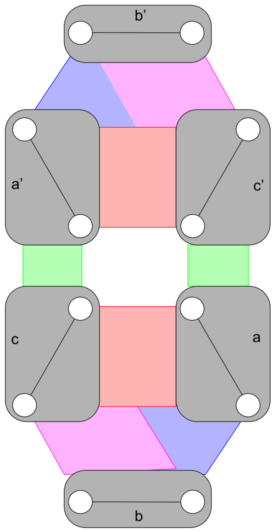
$x' \in a' \wedge y' \in c' \wedge (x', y') \in v(G) \Rightarrow (x', y) \in v(S)$

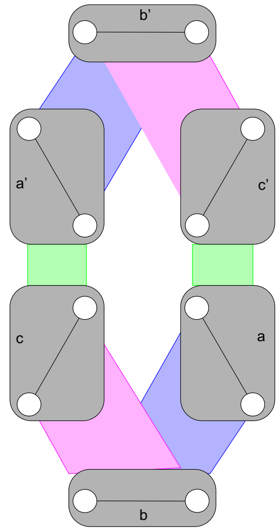
$x' \in a' \wedge y' \in c' \wedge (y', x') \in v(G) \Rightarrow (y, x') \in v(S)$





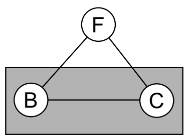






Ersetzt man in der Argumentation von Fall 1, die dortigen Knoten durch die entsprechenden Knotenmengen des jetzt betrachteten Falls, so kann wieder A durch eine Einrichtung F mit Fehlverhalten ersetzt werden.

Da B und C zusammen wenigstens n-m fehlerfreie Knoten enthalten, muß G korrektes Verhalten aufweisen, d. h. alle korrekten Knoten in b und c müssen sich für den Wert 0 entscheiden.



Damit läßt sich die Argumentation analog dem Fall 1 fortsetzen und führt in gleicher Weise zum Widerspruch.

12.3 Byzantinische Verständigung

12.3.1 Byzantinische Verständigung (Lamport-Shostak-Pease, 1982) mit unbeglaubigten Nachrichten

S12.3 Satz: Lamport-Shostak-Pease, 1982

In Systemen mit synchronisierbaren Runden löst nachfolgender rekursive Algorithmus das Verständigungsproblem auf der Basis unbeglaubigter Nachrichten, wenn $n > 3m$ **ist.**

Algorithmus:

Mehrheit(v_1, v_2, ..., v_{n-1}) = v, wenn wenigstens die Hälfte der v_i den Wert v hat.

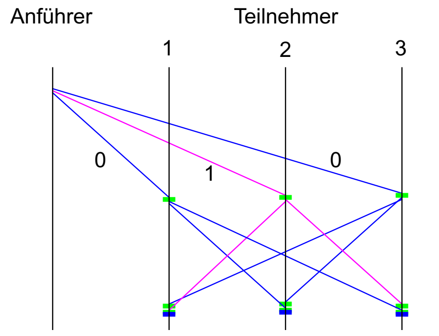
Algorithmus OM(0)

1. **Der Anführer sendet seinen Wert an alle anderen.**
2. **Jeder, außer dem Anführer, nimmt als Ergebnis den Wert, den er erhalten hat.**

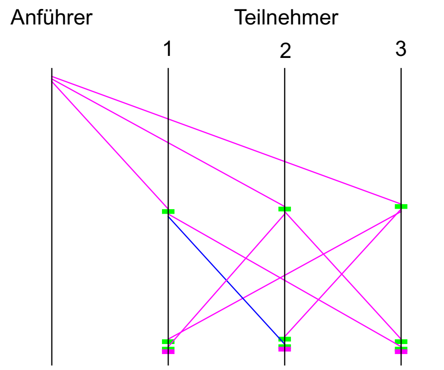
Algorithmus OM(m), m>0

1. **Der Anführer sendet seinen Wert an alle anderen.**
2. **Für jedes i sei v_i der Wert den Prozessor i vom Anführer erhalten hat, bzw. 1 wenn er keine Nachricht bekommt. Prozessor i handelt als Anführer im Algorithmus OM(m-1) mit den übrigen n-2 Prozessoren als Teilnehmern.**
3. **Für jedes i und jedes j $\neq$ i sei v_j der Wert den Prozessor i im Schritt 2 von Prozessor j (unter Verwendung von OM(m-1)) erhalten hat oder 1, wenn er einen solchen nicht erlangt hat. Prozessor i benutzt den Wert Mehrheit(v_1, v_2, ..., v_{n-1}).**

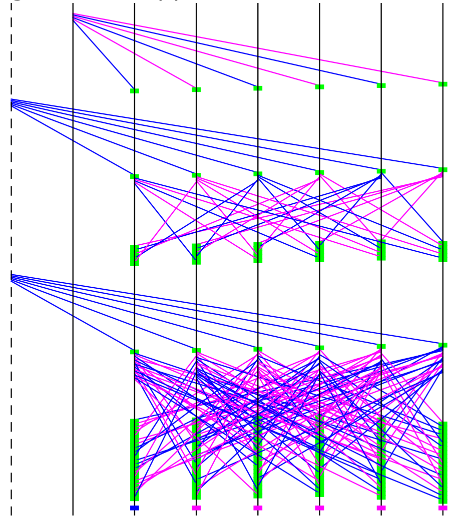
Beispiel1: Der Algorithmus OM(1) bei fehlerhaftem Anführer



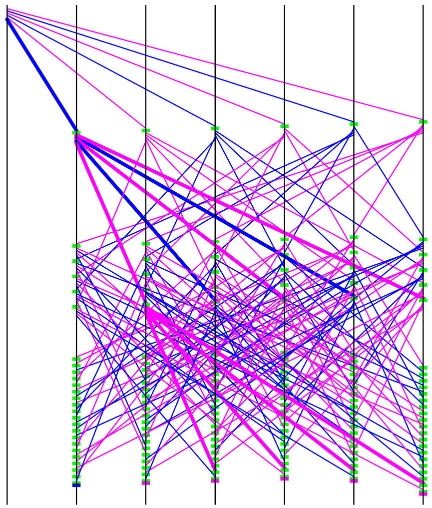
Beispiel 2: Der Algorithmus OM(1) bei fehlerhaftem Teilnehmer 1



Beispiel 3: Der Algorithmus OM(2), Teilnehmer 0 und 1 fehlerhaft



Beispiel 4: Der Algorithmus OM(2) von Lamport, Teilnehmer 0 und 1 fehlerhaft, Synchronisation durch Timeouts



Beweis

Zu erfüllende Bedingungen

- **IC1:** Alle fehlerfreien Prozessoren ermitteln den gleichen Wert.
- **IC2:** Wenn der Anführer fehlerfrei ist, ermitteln alle fehlerfreien Prozessoren seinen Wert.

L12.4 **Lemma**

Für jedes m und k, erfüllt OM(m) die Bedingung IC2, wenn n > 2k + m ist und höchstens k Prozessoren fehlerhaft sind.

Beweis:

Induktion über m

OM(0): trivial

Angenommen: Richtig für m - 1 mit m > 0

Im 1. Schritt v an alle anderen

Im 2. Schritt benutzen alle OM(m-1) für n-1 Prozessoren.

n > 2k + m ---> n - 1 > 2k + (m - 1), d. h. Ind.-Vor. ist erfüllt.

Also v_j = v für jeden fehlerfreien Prozessor j.

Höchstens k fehlerhafte und n - 1 > 2k + (m - 1) ≥ 2k
--->Mehrheit von n-1 Prozessoren fehlerfrei
--->Mehrheit ermittelt unter Verwendung von OM(m-1) den Wert v
--->Mehrheit(v_1, v_2, ..., v_{n-1}) = v

Beweis des Hauptsatzes durch Induktion über m

OM(0) trivial

Annahme OM(m-1) erfüllt Satz.

1. Anführer fehlerfrei
---> mit k = m folgt die Behauptung aus Lemma 1.

2. Anführer fehlerhaft
---> m fehlerhafte, darunter Anführer
n > 3m ---> n-1 > 3m-1 > 3(m-1)
---> Voraussetzung für OM(m-1) im 2. Schritt erfüllt
---> für jeden Prozessor j ermitteln alle fehlerfreien den gleichen Wert und damit den gleichen Wertevektor
---> alle fehlerfreien ermitteln im 3. Schritt den gleichen Wert v

12.3.2 **Byzantinische Verständigung (Lamport-Shostak-Pease) mit beglaubigten Nachrichten.**

S12.4 **Satz: Lamport-Shostak-Pease, 1982**

Der nachfolgende Algorithmus löst in jedem Fall das Verständigungsproblem auf der Basis beglaubigter Nachrichten.

Es sei V eine Menge von Werten. Weiter sei choice(V) eine Funktion mit folgenden Eigenschaften:

1. **Wenn** $V = \{v\}$ **ist, dann ist** $\text{choice}(V) = v$.
2. $\text{choice}(\emptyset) = 0$.

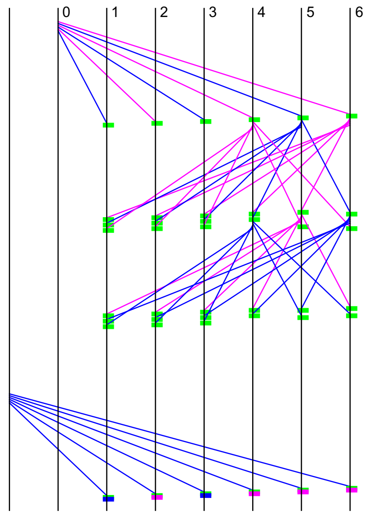
Im weiteren bezeichne $x{:}i$ **den Wert** x **beglaubigt von Prozessor i.**
Es bezeichnet also $v{:}j{:}i$ **den Wert** v **beglaubigt von Prozessor j und dann** $v{:}j$ **beglaubigt von Prozessor i.**

Dabei wird unterstellt, daß jeder Prozessor eine verlangte Nachricht entweder sendet oder dem Empfänger bekannt ist, daß sie nicht gesendet wird.

Algorithmus SM(m), $m \geq 0$

1. **Der Anführer sendet seinen von ihm selbst beglaubigten Anfangswert an alle anderen.**
2. **Für jedes i:**
   - **a Erhält ein Teilnehmer eine Nachricht der Form** $v{:}0$ **vom Anführer und hat bislang noch keine Nachricht erhalten, so führt er folgende Tätigkeiten durch:**
     - **i Er setzt** $V_i = \{v\}$.
     - **ii Er sendet die Nachricht** $v{:}0{:}i$ **an alle anderen Teilnehmer.**
   - **b Erhält ein Teilnehmer eine Nachricht der Form** $v{:}0{:}j_1{:}\ldots{:}j_k$ **und es ist** $v \notin V_i$**, dann führt er folgende Tätigkeiten durch:**
     - **i Er nimmt** v **in** V_i **auf.**
     - **ii Wenn** $k < m$ **ist, sendet er die Nachricht** $v{:}0{:}j_1{:}\ldots{:}j_k{:}i$ **an alle anderen Teilnehmer mit Ausnahme der Prozessoren** j_1 **bis** j_k.
3. **Für jedes i:**
   Wenn Prozessor i keine weiteren Nachrichten erhält, entscheidet er sich für den Wert $\text{choice}(V_i)$.

Beispiel 5: Lamport SM(5), Teilnehmer 0 bis 3 fehlerhaft (senden keine Nachrichten)



Beweis

Zu erfüllende Bedingungen

IC1: Alle fehlerfreien Prozessoren ermitteln den gleichen Wert.

IC2: Wenn der Anführer fehlerfrei ist, ermitteln alle fehlerfreien Prozessoren seinen Wert.

Es wird zuerst IC2 betrachtet.

Wenn der Anführer fehlerfrei ist, erhält jeder Teilnehmer in Schritt 2 den Wert v des Anführers. Da kein Teilnehmer die Unterschrift des Anführers fälschen kann, nimmt kein fehlerfreier Prozessor jemals einen anderen Wert in V **auf, so daß für alle fehlerfreien Prozessoren i gilt:** $V_i = \{v\}$ **. Sie entscheiden sich also im Schritt 3 für** v.

Da aus IC2 bei fehlerfreiem Anführer IC1 folgt, ist nur noch der Fall eines fehlerhaften Anführers zu betrachten.

Es genügt zu zeigen, daß $V_i = V_j$ ist, wenn die Prozessoren i und j fehlerfrei sind.

Prozessor i nehme in Schritt 2 v in V auf.

Macht er dies in Schritt 2A, dann sendet er ihn auch an j.

Macht er dies in Schritt 2B, dann muß er eine Nachricht $v{:}0{:}j_1{:}\ldots{:}j_k$ erhalten haben.

Es sind zwei Unterfälle zu betrachten:

1. $k < m$: **In diesem Fall sendet er die Nachricht $v{:}0{:}j_1{:}\ldots{:}j_k{:}i$ an j, so daß auch Prozessor j diesen Wert erhält.**
2. $k = m$: **Anführer fehlerhaft ---> höchstens $m-1$ der Prozessoren $j_1, \ldots, j_m$ fehlerhaft. Also muß wenigstens einer fehlerfrei sein und den Wert auch an Prozessor j gesendet haben.**

12.3.3 Byzantinische Verständigung (Dolev)

Reduktion der Nachrichtenzahl durch Erhöhung der maximalen Rundenzahl

Es sei $n = 3m + 1$:

Definitionen:

- **Runde**

  Algorithmus arbeitet in Runden

  $q_r = (q_r^1, \ldots, q_r^n)$ **Zustand am Anfang der r-ten Runde**

  q_r^i **enthält u. a. alle bis zu Beginn der r-ten Runde empfangenen Nachrichten**
- **Abarbeitung einer Runde in 2 Schritten:**
  1. **Aussenden von Nachrichten in Abhängigkeit vom Zustand am Anfang der Runde**
  2. **Empfang aller in der Runde gesandten Nachrichten und Ermittlung des Anfangszustandes für die nächste Runde**

- **Nachrichtenarten**

  ***-Nachricht, die als Wert des Anführers 1 zusichert.**

  k-Nachricht, die mitteilt, daß k überzeugt ist, eine zu recht bestehende *-Nachricht erhalten zu haben.
- **Lokale Zustände $q^j \in Q$ mit**

  $Q = \wp((\{*\} \cup \{1, \ldots, n\}) \times \{1, \ldots, n\}) \times IN$

  Paare (Nachrichtenkennung, Sender), Runde
- $LOW = m + 1$ **und** $HIGH = 2m + 1$

  Wenn eine bestimmte Aussage von wenigstens LOW Prozessen erhalten wurde, so muß sie von wenigstens einem korrekten gemacht worden sein.

  Wenn eine bestimmte Aussage von wenigstens HIGH Prozessen erhalten wurde, so muß sie von wenigstens LOW korrekten Prozessen gemacht worden sein.

- $W_x^j(q) = \{i \mid i \in \{1, \ldots, n\} \wedge (x, i) \in \pi_1(q^j)\}$

  Menge aller Prozessoren, die an j eine x-Nachricht gesandt haben ($x \in \{*, 1, \ldots, n\}$).
- ***j unterstützt i direkt*, wenn j von i eine *-Nachricht erhielt (d. h. $i \in W_*^j(q)$).**
- ***j unterstützt i indirekt*, wenn $|W_i^j(q)| \geq LOW$ ist, d. h. j weiß infolge der Mitteilung wenigstens eines korrekten Prozessors, daß i eine *-Nachricht erhalten hat.**
- **j *ist überzeugt*, daß i eine *-Nachricht ausgesandt hat, wenn $|W_i^j| \geq HIGH$ ist.**
- $C^j(q) = \{k \mid k \in \{1, \ldots, n\} \wedge k \neq s \wedge |W_k^j(q)| \geq HIGH\}$ **ist die Menge der *von j bestätigten Prozessoren* (ohne den Anführer s),**

  d. h. j ist zu Recht überzeugt, daß eine Mehrheit korrekter Prozessoren eine *-Nachricht versandt hat.

- Ein Prozessor j ist im Zustand q_r *aktiviert*, d. h. er sendet in der r-ten Runde an alle einschließlich sich selbst eine *-Nachricht,

  wenn er schon in einer früheren Runde aktiviert war,
  d. h. $j \in W_*^j(q_r)$ ist, **(I1)**

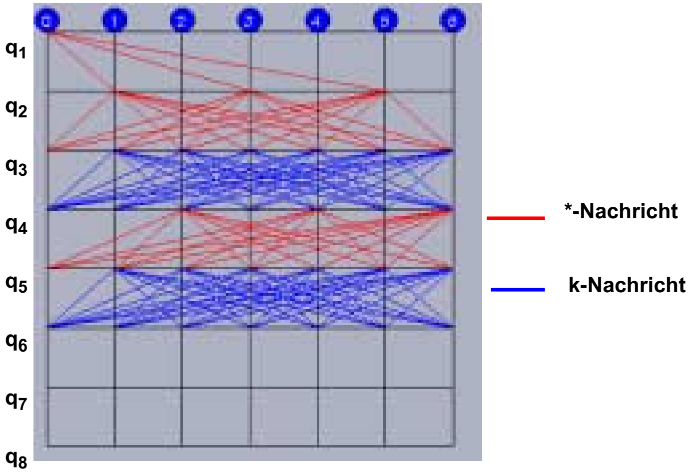
  oder wenn $|C^j(q_r)| \geq LOW + \max(0, \lceil r/2 \rceil - 2)$ ist **(I2)**

  oder wenn $r = 2$ ist und er in der ersten Runde vom
  Anführer eine *-Nachricht erhielt, d. h. $s \in W_*^j(q_2)$ ist. **(I3)**

Algorithmus

1. In der ersten Runde sendet ein korrekter Anführer eine *-Nachricht an alle einschließlich sich selbst, falls sein Wert 1 ist.
2. In der k-ten Runde (k > 1) senden die Prozessoren an alle anderen (einschließlich sich selbst)
   - eine *-Nachricht, wenn sie aktiviert sind, **(M1)**
   - eine j-Nachricht, wenn sie j direkt unterstützen(d. h. $j \in W_*^i(q)$),
     (weitergeleitete Aussage: j hat * mitgeteilt)
   - eine j-Nachricht, wenn sie j indirekt unterstützen (d. h. $|W_j^i(q)| \geq LOW$).**(M3)**
     (weitergeleitete Aussage: wenigstens ein Korrekter hat * von j erhalten)
3. Prozessoren, die von wenigstens HIGH Prozessoren überzeugt sind, d. h. $|\{i \mid |W_i^j| \geq HIGH\}| \geq HIGH$, entscheiden sich für den Wert 1.
4. Wenn ein Prozessor sich nach der $(2m+3)$-ten Runde für 1 entschieden hat, so nimmt er den Wert 1 als Ergebnis, sonst den Wert 0.

Beispiel 6: Fehlerhafter Anführer ist P_0,
Teilnehmer P_2, P_4 und P_6 erhalten anfänglich keine Nachricht



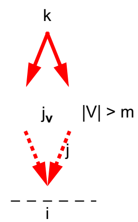
Veranschaulichung

$i \rightarrow j \dashrightarrow k$: P_i hat an P_j eine *-Nachricht gesandt und P_j hat dies an P_k weitergemeldet

$k \in W_j^i$: $j \rightarrow k \dashrightarrow i$

$k \in W_*^i$: $k \rightarrow i$

$k \in C^i$: Es gibt mindestens m+1 korrekte vom Anführer verschiedene Prozessoren j_v mit $k \rightarrow j_v \dashrightarrow i$.
(Eine Mehrheit Korrrekter hat eine *-Nachricht bekommen)
Prozessor k wird von i bestätigt, d. h. er verhält sich so, als ob er von k eine *-Nachricht bekommen hätte.



i zeugt für j: $j \rightarrow i$

i zeugt indirekt für j: Für mindestens einen korrekten Prozessor k gilt $j \rightarrow k \dashrightarrow i$

Einigungsprotokolle: Algorithmus von Dolev

Aktivierung von i

(I1) **$i \rightarrow i$ in vorangehender Runde**

(I2) $k_u \rightarrow j_{u,\, v_u} \dashrightarrow i$ **mit** $u \in U \wedge |U| \geq m + 1 + f(r)$ **und** $v_u \in V_u \wedge |V_u| \geq 2m + 1$

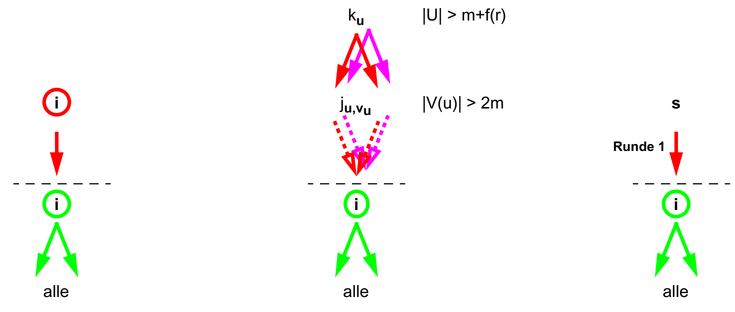
(also $k_u \in C^i$**)**

(I3) **$s \rightarrow i$ in Runde 1**

Einigungsprotokolle: Algorithmus von Dolev

Senden:

(M1) $(I1) \vee (I2) \vee (I3)$ **---> $i \rightarrow$ alle**

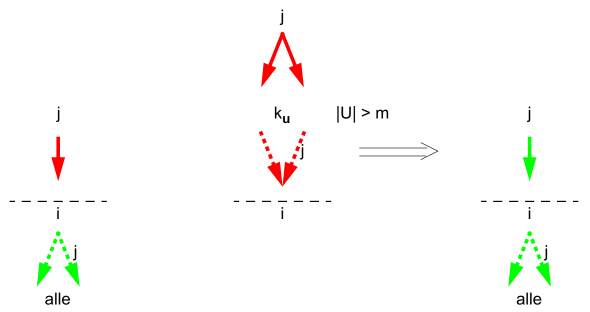


(I1)+(M1) (I2)+(M1) (I3)+(M1)

Einigungsprotokolle: Algorithmus von Dolev

(M2) **$j \rightarrow i$ ---> $j \rightarrow i \dashrightarrow$ alle**

(M3) **Für alle** $u \in U$ **mit** $|U| \geq m + 1$**:** **$j \rightarrow k_u \dashrightarrow i$ ---> $j \rightarrow i \dashrightarrow$ alle**

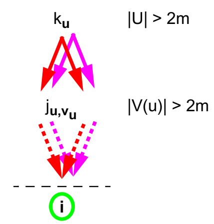


(M2) (M3) = Transformation + (M2)

Einigungsprotokolle: Algorithmus von Dolev

Entscheidung von i für Wert 1:

$k_u \rightarrow j_{u,\, v_u} \dashrightarrow i$ **mit** $u \in U \wedge |U| \geq 2m + 1$ **und** $v_u \in V_u \wedge |V_u| \geq 2m + 1$



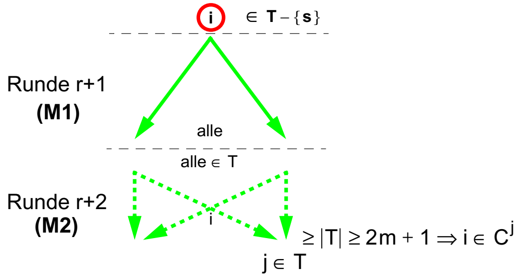
Beweis für das geforderte Verhalten

L12.5 **Für korrekte Prozessoren sind die Mengen W und C mit der Rundenzahl monoton wachsend.**

Beweis: Definitionen

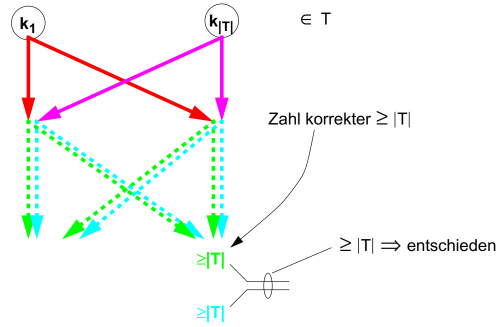
L12.6 $i, j \in T, i \neq s$: **i aktiviert in** q_r **---> i von j bestätigt in** q_{r+2}**, d. h.** $i \in C^j(q_{r+2})$

Beweis:



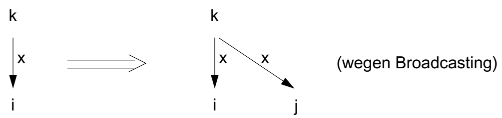
L12.7 **Alle** $i \in T$ **aktiviert in** q_r **---> alle** $i \in T$ **entschieden in** q_{r+2}

Beweis



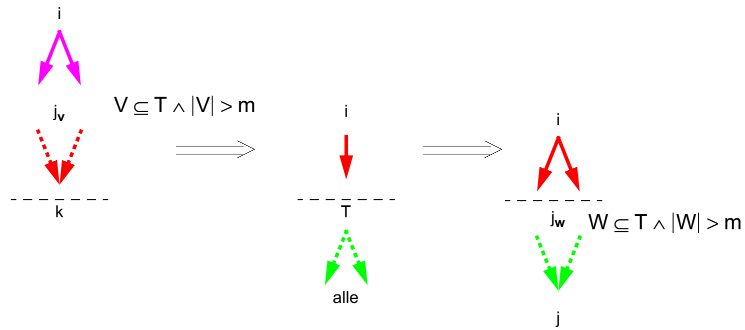
L12.8 $i, j \in T$: $W_x^i(q_r) \cap T = W_x^j(q_r) \cap T$

Beweis



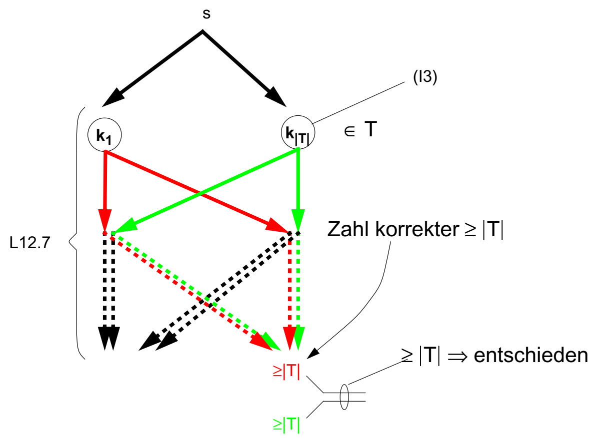
L12.9 $i, j \in T$**: ex.** $k \in T$ **mit** $i \in C^k(q_r)$ **---> für alle** $j \in T$ **ist** $i \in C^j(q_{r+1})$

Beweis



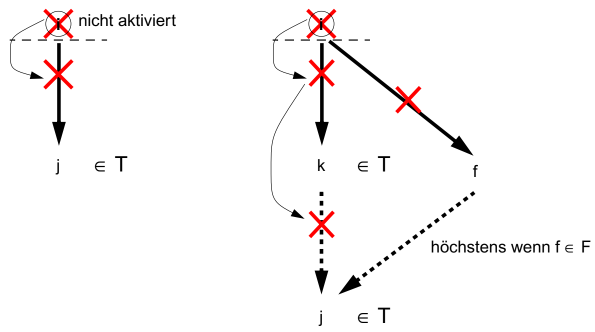
L12.10 $s \in T \wedge s_0 = 1$ **---> alle** $j \in T$ **entschieden in** q_4

Beweis



L12.11 $i, j \in T$**: i nicht aktiviert in** q_r **--->** $i \notin W_*^j(q_{r+1})$ **und** $|W_i^j(q_{r+2})| < LOW$

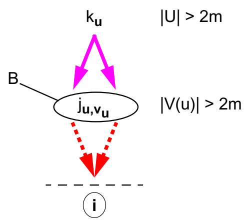
Beweis



Wegen $|F| \leq m$ folgt die Beh.

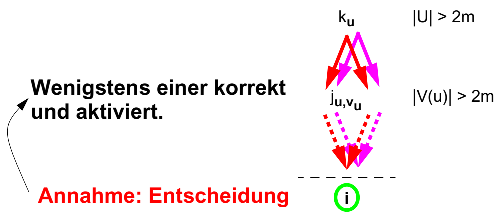
L12.12 $i \in T$**: i entschieden in** q_r **---> ex.** $B \subseteq T$ **mit** $|B| = LOW$ **und alle** $j \in B$ **aktiviert in** q_{r-1}

Beweis



L12.13 $s \in T \wedge s_0 = 0$ **---> kein** $i \in T$ **(aktiviert oder entschieden in** q_{d+1}**)**

Beweis

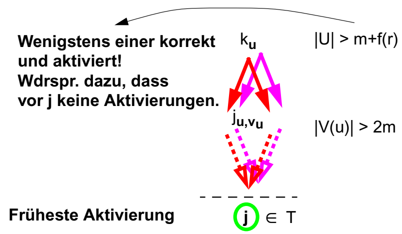


Also aktivierte Prozesse in Runde d.

Früheste Aktivierung erst in einer Runde >= 2, aktivierter Prozessor sei j.

Grund kann weder (I1) noch (I3) sein!

Also Aktivierung durch (I2).



L12.14 $A \subseteq T - \{s\} \wedge |A| = LOW$ **und alle** $i \in A$ **aktiviert in** q_r

---> **alle** $j \in T$ **entschieden in** q_{r+4}

Beweis

a) $s \in T$**: Wegen L12.13 muß** $s_0 = 1$ **sein. Dann folgt Behauptung aus (I3) und L12.7.**

b) $s \notin T$**: Es seien alle** $i \in A$ **aktiviert in** $q_{r'}$ **und r' minimal.**

Nach L12.6 alle $i \in A$ **von allen** $j \in T$ **bestätigt in** $q_{r'+2}$

b1) r' = 1: Kann nach Definition nicht sein.

b2) r' = 2: Wegen L12.6 alle $i \in A$ **von allen** $j \in T$ **bestätigt in** q_4**. Da f(4) = 0 also auch entschieden.**

b3) r' > 2: Es muß k existieren, das in $q_{r'}$ **erstmalig aktiviert wird. Grund kann nur I2 sein.**

$$\Rightarrow |C^k(q_{r'})| \geq LOW + \max(0, \lceil r'/2 \rceil - 2)$$

r' minimal $\Rightarrow k \notin C^k(q_{r'})$, **denn** $k \in C^k(q_{r'}) \Rightarrow |W_k^k(q_{r'})| \geq HIGH$ **nach Def.**

$\Rightarrow$ k aktiviert in $q_{r'-2}$ **wegen L12.11, Wdrspr.**

$\forall j(j \in T \Rightarrow C^j(q_{r'+2}) \supseteq C^k(q_{r'}))$ **wegen L12.5 und L12.9.**

$\forall j(j \in T \Rightarrow k \in C^j(q_{r'+2}))$ **wegen L12.6, da k in** q_r **aktiviert ist.**

Wegen der letzten beiden Aussagen zusammen mit $k \notin C^k(q_{r'})$

$$\Rightarrow |C^j(q_{r'+2})| \geq LOW + \max\left(0, \left\lceil \frac{r'}{2} \right\rceil - 2\right) + 1 = LOW + \max\left(0, \left\lceil \frac{r'+2}{2} \right\rceil - 2\right)$$

also j aktiviert in $q_{r'+2}$ **wegen (I2) und wegen L12.6 entschieden in** $q_{r'+4}$.

L12.15 $i \in T$**: i entschieden in** q_{d+1} **---> alle** $j \in T$ **entschieden in** q_{d+1}

Beweis

a) $(m = 0) \vee s \in T$: **L12.10 und L12.13**

b) $m > 0 \wedge s \notin T$: $\Rightarrow d > 4$

$i \in T$ **habe sich in** q_{d+1} **entschieden**

$\Rightarrow \exists A(A \subseteq T \wedge (|A| = LOW) \wedge \forall j(j \in A \Rightarrow j \text{ aktiviert\_in } q_d))$ **wegen L12.12.**

r sei erste Runde in der ein derartiges A existiert.

b1) $r \leq d - 3 \Rightarrow$ **Beh. wegen L12.14.**

b2) $r \geq d - 2 = 2m + 1 > 2$

$\Rightarrow \exists k(k \in A \wedge k \text{ aktiviert\_in } q_r \wedge \neg(k \text{ aktiviert\_in } q_{r-1}))$ **nach Def. von r und LOW**

Wegen (I2) $\Rightarrow |C^k(q_r)| \geq LOW + \max\left(0, \left\lceil \frac{r}{2} \right\rceil - 2\right) \geq LOW + m - 1 = HIGH - 1$.

Es ist $s \notin T \wedge s \notin C^k(q_r)$ **nach Vorraussetzung und Def. von** $C^k(q_r)$,

also enthält $C^k(q_r)$ **höchstens m-1 fehlerhafte Prozessoren.**

$$\Rightarrow \left|\overbrace{\{j \mid j \in N - \{s\} \wedge |W_j^k(q_r)| \geq HIGH\}}^{C^k(q_r) =}\right| \geq HIGH - 1$$

$$\Rightarrow |\{j \mid j \in T \wedge |W_j^k(q_r)| \geq HIGH\}| \geq LOW$$

$$\Rightarrow \exists A'(A' \subseteq T \cap C^k(q_r) \wedge (|A'| = LOW) \wedge \forall j(j \in A' \Rightarrow j \text{ aktiviert\_in } q_{r-1}))$$ **wegen L12.11 und L12.5 .**

Widerspruch zur Wahl von r, also ist dieser Fall nicht möglich.

L12.16 $s \in T$: $s_0 = 1$ **---> alle** $j \in T$ **entschieden in** q_{d+1}

$s_0 = 0$ **---> kein** $j \in T$ **entschieden in** q_{d+1}

Beweis

1. Zeile folgt unmittelbar aus L12.7.

2. Zeile bewiesen als L12.13.

12.3.4 Algorithmus von Burns und Neiger (Einigungs-Algorithmus, 1994)

Bezeichnungen

- **n** **Zahl der Prozessoren**
- **m** **Maximalzahl fehlerhafter Prozessoren**
- **f** **Zahl tatsächlich fehlerhafter Prozessoren**

Bedingungen

$f \leq m$ **und** $n \geq m^2 + 4m + 1$

Partitionierung von $m^2 + 4m + 1$ **Prozessoren in** $m + 1$ **disjunkte Gruppen** G_i

G_1 **enthält** $2m + 1$ **Prozessoren.**

Für $2 \leq i \leq m + 1$ **besteht** G_i **aus** $2(m + 2 - i) + 1$ **Prozessoren.**

d. h. es enthält

G_1	$2m + 1$	**Prozessoren,**
G_2	$2m + 1$	**Prozessoren,**
G_3	$2m - 1$	**Prozessoren,**
.		
.		
.		
G_{m+1}	3	**Prozessoren.**

Algorithmus

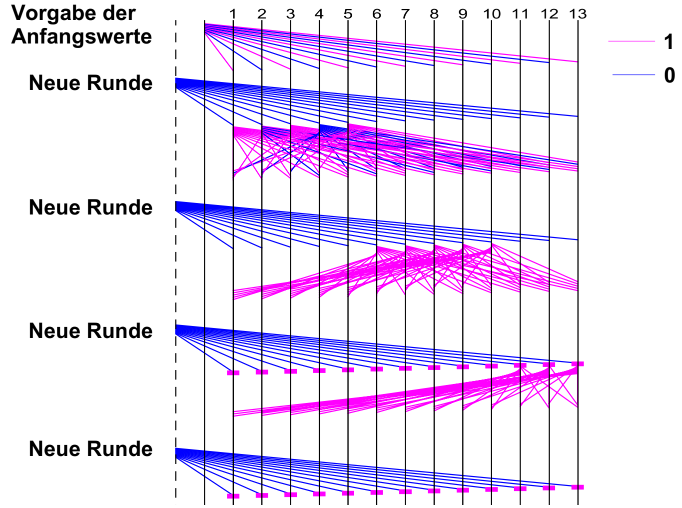
Der Algorithmus läuft in Runden.

- **Jeder Prozessor besitzt einen Wert, der anfänglich den Anfangswert darstellt.**
- **In jeder Runde r senden die Prozessoren der Gruppe G_r ihren Wert an alle anderen.**
- **Am Rundenende ermittelt jeder Prozessor der Runde seinen Wert durch Mehrheitsbildung. Für ausgebliebene Nachrichten wird der eigene Wert benutzt.**
- **Wenn ein Prozessor von allen Prozessoren der Gruppe r den gleichen Wert erhält, entscheidet er sich für diesen Wert.**
- **Prozessoren, die sich bis zum Ende der (m+1)-ten Runde noch nicht entschieden haben, entscheiden sich für ihren Wert.**

Jeder Prozessor p führt folgenden Algorithmus (in synchronisierten Runden) aus:

```
for (i = 1; i <= m + 1; i++) {
// Ein Schleifendurchlauf stellt jeweils eine Runde dar.
  if (G[i].contains(p))
    for (j = 1; j <= n; j++) send(j, p, v);
  tmp = 0;
  for (pix = G[i].first(); pix != 0; G[i].next(pix)) {
    receive(pix, &m); tmp += m;
  }
  if (!decided) {
    if (tmp == 0 || tmp == G[i].length()) {
      v = (tmp ? 1 : 0); decided = TRUE; break;
    } else {
      v = 2 * tmp > G[i].length() ? 1 : 0;
    }
  }
}
```


Beispiel: Einigungsprotokoll von Burns und Neiger (f = m = 2)



Grundgedanke

- **Da es** $m + 1$ **Gruppen gibt, muß wenigstens eine fehlerfreie existieren (d. h. alle ihre Mitglieder sind fehlerfrei).**

  Wenn eine fehlerfreie Gruppe sendet, erhalten alle die gleiche Multimenge von Werten.

  Wenn einmal alle fehlerfreien Prozessoren den gleichen Wert besitzen, so behalten sie ihn bei.

- **Falls** $f < m$ **müssen unter** $G_1, G_2, \ldots, G_{f+2}$ **wenigstens zwei fehlerfrei sein.**

- **Die Mitglieder der zweiten fehlerfreien Gruppe senden alle den gleichen Wert, also haben sich spätestens nach der Runde** $f + 2$ **alle fehlerfreien entschieden.**

(a) Wenn unter $G_1, \ldots, G_i$ **wenigstens eine fehlerfreie Gruppe ist, beenden alle fehlerfreien Prozessoren die i-te Runde mit dem gleichen Wert.**

(b) Wenn unter $G_1, \ldots, G_i$ **wenigstens zwei fehlerfreie Gruppen sind, haben sich nach der i-ten Runde alle fehlerfreien Prozessoren entschieden.**

(c) Wenn alle fehlerfreien Prozessoren, die i-te Runde mit dem gleichen Wert beginnen, beenden sie die Runde mit eben diesem Wert.

(d) Wenn sich ein fehlerfreier Prozessor am Ende der i-ten Runde entschieden hat, dann beenden alle fehlerfreien Prozessoren die Runde mit dem gleichen Wert.

Beweis durch vollständige Induktion

Induktionsanfang: i = 1

(a) G_1 fehlerfrei ---> alle erhalten gleiche Wertemenge
---> alle fehlerfreien entscheiden sich für gleichen Wert

(b) entfällt

(c) Folgt aus a) und Gewinnung des Wertes durch Mehrheitsbestimmung

(d) Entschieden ---> alle erhaltenen Werte gleich
---> von den fehlerfreien gleichen Wert erhalten
---> bilden die Mehrheit, da G_1 mindestens 2m+1 Prozessoren enthält
---> bei allen liefert die Mehrheit den gleichen Wert

Induktionsannahme: Aussage gültig bis einschließlich i

Induktionsschluß: Fall i+1

(a) 1. $G_1, \ldots, G_i$ fehlerhaft
---> G_{i+1} fehlerfrei
---> alle erhalten gleiche Wertemenge
---> alle fehlerfreien ermitteln gleichen Wert

2. Wenigstens eine Gruppe G_j fehlerfrei, j sei maximal
---> alle fehlerfreien haben am Ende der j-ten Runden den gleichen Wert
---> alle fehlerfreien haben am Ende der folgenden Runden den gleichen Wert (Eigenschaft c))

(b) 1. $G_1, \ldots, G_i$ enthalten nur eine fehlerfreie Gruppe
---> G_{i+1} fehlerfrei
---> alle erhalten in der (i+1)-ten Runde lauter gleiche Werte
---> alle fehlerfreien entscheiden sich für diesen Wert

2. Wenigstens zwei Gruppen aus $G_1, \ldots, G_i$ sind fehlerfrei
---> alle fehlerfreien haben sich schon zu Beginn der (i+1)-ten Runde entschieden

(c) 1. $G_1, \ldots, G_i$ enthalten wenigstens eine fehlerfreie
---> Behauptung wegen a) und Algorithmus

2. $G_1, \ldots, G_i$ fehlerhaft
---> G_{i+1} enthält höchstens $m-i < 2(m+2-(i+1))+1$ fehlerhafte
---> die fehlerfreien sind in der Mehrzahl
---> alle fehlerfreien entscheiden sich für den gemeinsamen Anfangswert

(d) G_p entscheide sich am Ende der (i+1)ten Runde
---> G_p hat von allen aus G_{i+1} den gleichen Wert bekommen und unter $G_1, \ldots, G_i$ ist höchstens eine fehlerfreie Gruppe
---> G_{i+1} enthält höchstens $m-i+1$ fehlerhafte Prozessoren

Da G_{i+1} insgesamt $2(m+2-(i+1))+1 > 2(m+1-i)$ Prozessoren enthält,ist die Mehrheit fehlerfrei
---> alle fehlerfreien haben am Ende der Runde den gleichen Wert

S12.5 **Satz: Der Algorithmus von Burns und Neiger löst das Einigungsproblem. Spätestens nach** $\min(f + 2, m + 1)$ **Runden haben sich alle Prozessoren entschieden.**

Beweis:

Terminierung: trivial

Richtigkeit: Unmittelbare Folgerung aus c)

Einigung: Da es $m + 1$ **Gruppen gibt, aber höchstens m fehlerhafte Prozessoren, muß wenigstens eine Gruppe fehlerfrei sein. Damit folgt aus a) die Einigung spätestens in der (m+1)-ten Runde.**

Ist $f < m$**, so enthält** $G_1, \ldots, G_{f+2}$ **wenigstens zwei fehlerfreie Gruppen, woraus nach b) die Behauptung folgt.**

12.4 PAXOS

12.4.1 Aufgabenstellung

Modellannahmen

- **Prozessoren**
  - **Mögliches Fehlverhalten: Fail Stop**
    d. h. Prozessoren sind entweder *lebendig* oder *ausgefallen*
  - **Wiederanlauf erlaubt**
- **Kommunikationssystem**
  - **vollständig vermascht**
  - **jeder Prozessor kann auch an sich selbst senden**
  - **zuverlässige FIFO-Verbindungen**

Aufgabenstellung

- **Einigung**
  - **Wertevorrat für Anfangswerte beliebig**
  - **Einigung auf den Anfangswert eines der Prozessoren**

Synchronisation

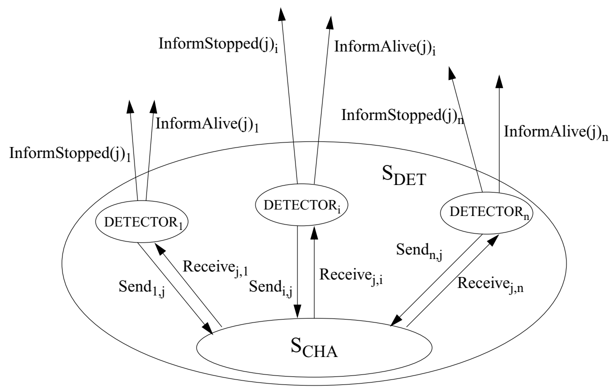
- **Die Ausführung von Zustandsübergängen erfolgt atomar**
- **L maximale Ausführungsdauer der (atomaren) Durchführung eines Zustandsübergangs**
- **D maximale Übertragungsdauer von Nachrichten**

Architektur

Multi-Paxos (Folge von Einigungsvorgängen)
Basic Paxos (ein einzelner Einigungsvorgang)
Wahl eines Anführers
Ausfallerkennung
Nachrichtensystem
Hardware

12.4.2 Ausfallerkennung

Systemstruktur beschrieben mit synchron gekoppelten E/A-Automaten



Formale Beschreibung der Komponenten

DETECTOR(z, c)$_i$

Signature:

Input: Receive(m)$_{j,i}$, Stop$_i$, Recover$_i$
Internal: Check(j)$_i$
Output: InformStopped(j)$_i$, InformAlive(j)$_i$, Send(m)$_{i,j}$

State:

Clock $\in$ R	initially arbitrary
Status $\in$ {alive, stopped}	initially alive
Alive $\in 2^I$	initially I
for all j $\in$ I	
Prevrec(j) $\in R^{\geq 0}$	initially arbitrary
Lastinform(j) $\in R^{\geq 0}$	initially *Clock*
Lastsend(j) $\in R^{\geq 0}$	initially *Clock*
Lastcheck(j) $\in R^{\geq 0}$	initially *Clock*

Actions:

input Stop$_i$
Eff: Status = stopped;

input Recover$_i$
Eff: Status = alive;

input Receive("Alive")$_{j,i}$
```
Eff: if (Status == alive) {
  Prevrec(j) = Clock;
  if (j ∉ Alive) {
    Alive = Alive ∪ {j};
    Lastcheck(j) = Clock + c;
  }
}
```


internal Check(j)$_i$
Pre: Status == alive && j $\in$ Alive
Eff: Lastcheck(j) = Clock + c;
if (Clock > Prevrec(j) + z + D) { Alive = Allive - {j}; }

output Send("Alive")$_{i,j}$
Pre: Status == alive
Eff: Lastsend(j) = Clock + z;

output InformStopped(j)$_i$
Pre: Status == alive && j $\notin$ Alive
Eff: Lastinform(j) = Clock + L;

output InformAlive(j)$_i$
Pre: Status == alive && j $\in$ Alive
Eff: Lastinform(j) = Clock + L;

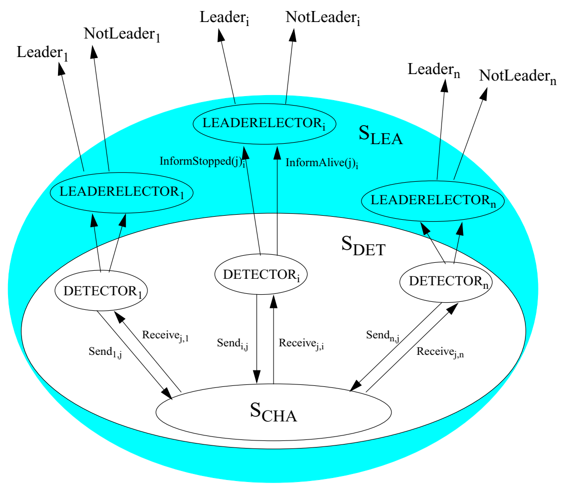
time_passage ν(t)
```
Pre: Status == alive
Eff: Let t' be such that {
        ∀j, Clock + t' ≤ Lastinform(j);
        ∀j, Clock + t' ≤ Lastsend(j);
        ∀j, Clock + t' ≤ Lastcheck(j);
     }
     Clock = Clock + t'
```

12.4.3 Wahl eines Anführers

Anführer fungiert als Leiter von Einigungsrunden

- **Zur Vereinfachung wird ein Verfahren ausgewählt, das eventuell zu mehreren Anführern führt**
- **Durch Zuteilung von logischen Zeitstempeln kann gewährleistet werden, daß sich schließlich einer durchsetzt (ähnlich Wahl bei allgemeinen Kommunikationsgraphen)**

Systemstruktur



Formale Beschreibung

LEADERELECTOR$_i$

Signature:

Input:	InformStopped(j)$_i$, InformAlive$_i$, Stop$_i$, Recover$_i$
Output:	Leader$_i$, NotLeader$_i$

State:

Status $\in$ {alive, stopped}	initially alive
Pool $\in 2^I$	initially {i}
Leader $\in$ I	initially i

Actions:

input Stop$_i$

```
Eff: Status = stopped;
```

input Recover$_i$

```
Eff: Status = alive;
```

input InformStopped(j)$_i$

```
Eff: if (Status == alive) {
        Pool = Pool - {j};
        Leader = max(Pool);
     }
```

input InformAlive(j)$_i$

```
Eff: if (Status == alive) {
        Pool = Pool ∪ {j};
        Leader = max(Pool);
     }
```

output Leader$_i$

Pre: Status == alive && i == Leader

Eff: skip;

output NotLeader$_i$

Pre: Status == alive && i != Leader

Eff: none;

12.4.4 Basic Paxos

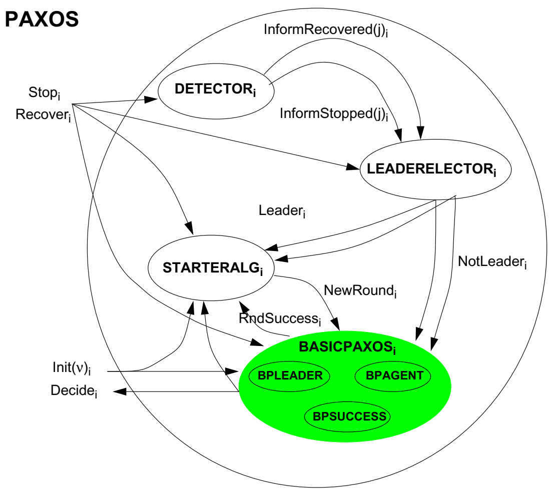
Modularisierung

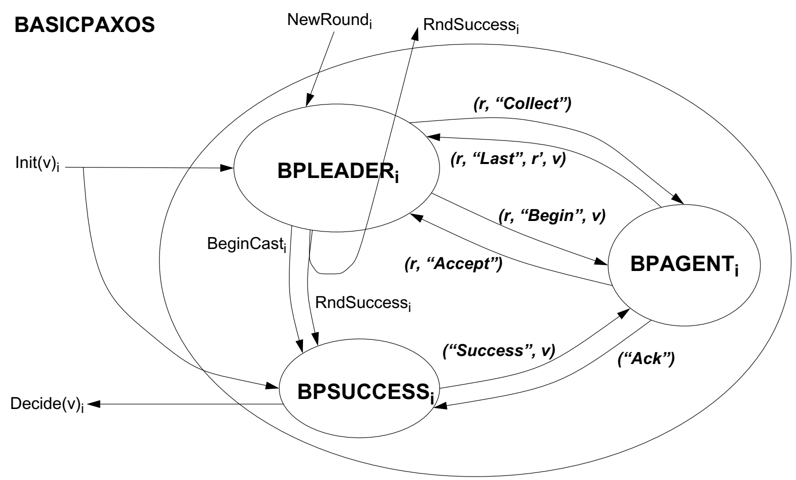
BASICPAXOS versucht eine Entscheidungsrunde durchzuführen, an deren Ende sich alle Teilnehmer der Runde entschieden haben.

Runden können möglicherweise wegen Teilausfällen nicht beendet werden. Entscheidungsrunde wird deshalb von STARTERALG überwacht. Diese Komponente startet nach einer gewissen Zeit eine neue Runde mit dem gleichen Anführer.

BPSUCCESS verbreitet das Ergebnis einer erfolgreich beendeten Entscheidungsrunde.

Systemstruktur





12.4.5 BASICPAXOS

Informelle Beschreibung

"Collect"-Nachrichten (r, "Collect"):
Werden vom Rundenanführer als Zeichen des Beginns der Runde r an alle versandt.

"Last"-Nachrichten (r, "Last", r', v'):
Werden bei Empfang einer "Collect"-Nachricht der Runde r an den Rundenanführer gesandt. Dabei ist r' die letzte Runde, in der der Sender einen Wert, nämlich v', als zustimmungsfähig akzeptiert hat. Existiert keine solche Runde wird r' = (0,i) gesetzt und als Wert der Anfangswert zurückgegeben bzw. nil.

"Begin"-Nachricht (r, "Begin", v):
Wird vom Rundenanführer als Vorschlag für den Einigungswert v versandt, der anhand der "Last"-Nachrichten ermittelt wird.

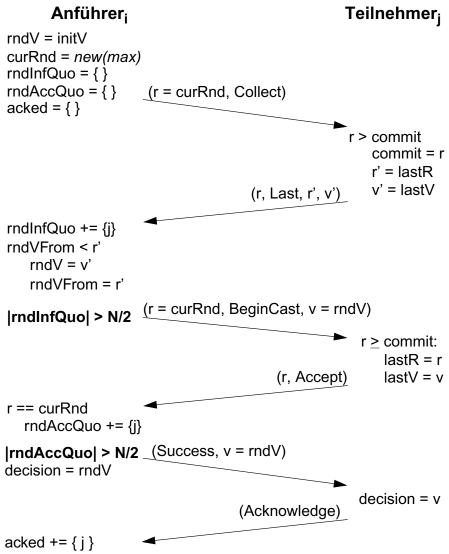
"Accept"-Nachricht (r, "Accept"):
Wird als Antwort auf "Begin"-Nachricht zurückgesandt zum Zeichen, daß der vorgeschlagene Wert akzeptiert wird.

"OldRound"-Nachricht (r, "OldRound", r'):
Wird als Antwort auf eine "Collect"- oder "Begin"-Nachricht gesandt zum Zeichen, daß der Sender in der späteren Runde r' einen Einigungswert akzeptiert hat.

"Success"-Nachricht ("Success", v):
Wird vom Rundenanführer am Ende einer erfolgreichen Runde als Entscheidungswert gesandt.

"Ack"-Nachricht ("Ack"):
Wird an den Rundenanführer gesandt, als Zeichen des Empfangs einer "Success"-Nachricht, d. h. der Kenntnis des Entscheidungswertes.

rndV	value of this round
rndVFrom	round from which rndV is taken
curRnd	number of current round
rndInfQuo	set of processes for which “Last” has been received
rndAccQuo	set of processes for which “Accept” has been received
acked	set of processes for which “Acknowledge” has been received
commit	number of committed round
lastR	number of latest round
lastV	value for round lastR
decision	value for which processor decided



Abstrakte Darstellung

Signature of BPLEADER

- **Input:** Receive(m)$_{j,i}$, m ∈ {"Last", "Accept", "OldRound"}
- **Internal:** Collect$_i$, GatherLast$_i$, GatherAccept$_i$, GatherOldRound$_i$
- **Output:** Send(m)$_{i,j}$, m ∈ {"Collect", "Begin"}, BeginCast$_i$, RndSuccess(v)$_i$

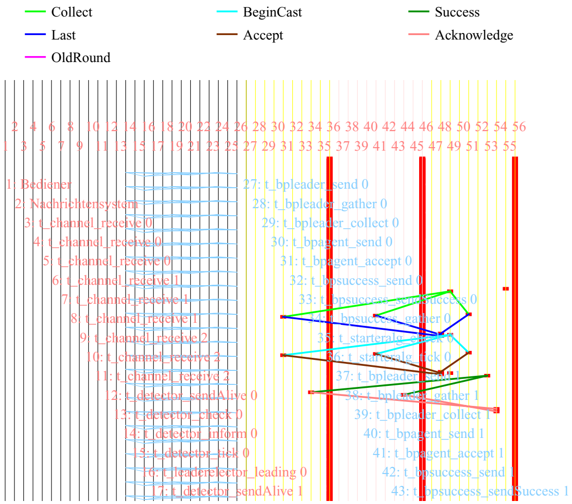
Signature of BPAGENT

- **Input:** Receive(m)$_{j,i}$, m ∈ {"Collect", "Begin"}
- **Internal:** LastAccept$_i$, Accept$_i$
- **Output:** Send(m)$_{i,j}$, m ∈ {"Last", "Accept", "OldRound"}

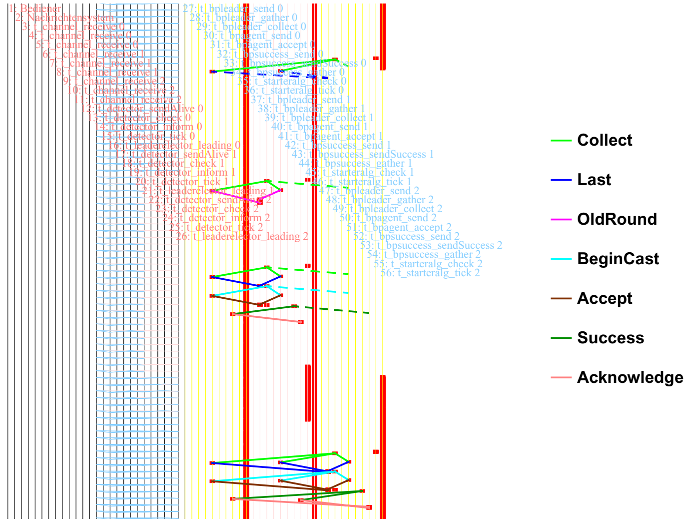
Signature of BPSUCCESS

- **Input:** Receive(m)$_{j,i}$, m ∈ {"Ack", "Success"}, RndSuccess(v)$_i$
- **Internal:** SendSuccess, GatherSuccess$_i$, GatherAck$_i$
- **Output:** Send(m)$_{i,j}$, m ∈ {"Success"}, Decide(v)$_i$

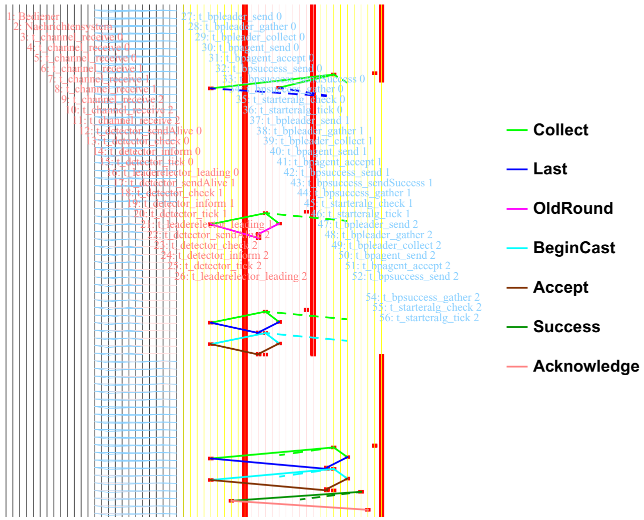
Ungestörter Ablauf



Zeitweiser Ausfall von Prozessor 2



Zeitweiser Ausfall der Prozessoren 1 und 2



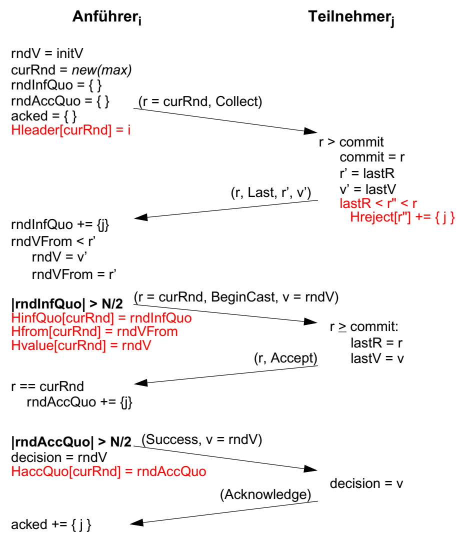
Gültigkeit

Anreicherung mit "Historien"-Variablen

Invariante 1: In jedem Zustand der Ausführung gilt für jede Runde r mit Hleader[r] = nil:

- **Hvalue[r]=nil**
- **Hfrom[r]= nil**
- **HinfQuo[r]={ }**
- **HaccQuo[r]={ }**

D11.1 **Eine Runde r heißt tot, wenn |Hreject[r]| ≥ n/2 ist.**



D4.2 R bezeichne den Wertevorrat für Rundennummern.

R_S bezeichne die Menge { $r \in R$ | $Hleader_i[r] \neq nil$ }.

R_V bezeichne die Menge { $r \in R$ | $Hvalue_i[r] \neq nil$ }.

Invariante 2: In jedem Systemzustand ist $R_V \subseteq R_S$.

D4.3 $r \in R_V$ heißt verankert, wenn für jede Runde $r' \in R_V$ mit $r' < r$ r' tot oder Hvalue[r'] = Hvalue[r] ist.

Invariante 3: In jedem Zustand, in dem Prozessor j an Prozessor i die Nachricht (r, "Last", r", v) gesendet hat, besitzen alle r' mit r" < r' < r die Eigenschaft $j \in$ Hreject(r').

Invariante 4: In jedem Zustand, in dem Prozessor $j \in rndInfQuo_i$ und $curRnd_i = r$ ist, gilt für alle Runden r' mit $rndVFrom_i < r' < r$ die Beziehung $j \in$ Hreject(r').

Invariante 5: In jedem Zustand, in dem Prozessor $j \in$ HinfQuo(r) ist, gilt für alle Runden r' mit Hfrom(r) < r' < r die Beziehung $j \in$ Hreject(r').

Invariante 6: In jedem Zustand ist jede lebendige (nicht tote) Runde verankert.

Invariante 7: In jedem Zustand besitzen alle Variablen $decision_i \neq nil$ den gleichen Wert. Dieser Wert ist Anfangswert mindestens eines Prozessors.

Terminierung

D4.4 Unter einem *Ausführungsfragment* wird ein Ausschnitt aus einer Ausführung des Algorithmus verstanden, der während eines (gewählten) Zeitintervalls stattfand.

Ein Ausführungsfragment heißt *stabil*, wenn während des Zeitintervalls, in dem es ausgeführt wurde, Prozessoren weder ausgefallen noch wiederangelaufen sind.

Ein Ausführungsfragment heißt *gut*, wenn es stabil ist und eine Mehrheit von Prozessoren während seiner Ausführung lebendig ist.

L12.17 In einer Runde, die in einem stabilen Ausführungsfragment enthalten ist, werden maximal 6n Nachrichten versandt.

L12.18 Ein Ausführungsfragment a, in dem noch kein Prozessor sich für einen Wert entschieden hat, gelte folgendes:

1. a ist stabil,
2. In a existiert genau ein Anführer i,
3. a dauert mindestens 4 * L + 2 * n * L + 2 * D Zeiteinheiten,
4. Prozeß i ist in a Anführer der (geeignet gewählten) Runde r und
5. Runde r ist erfolgreich.

Dann wird rndSuccess spätestens zum Zeitpunkt 7 * L + 4 * n *L + 4 * D ab Beginn des Ausführungsfragmentes ausgeführt.

L12.19 Wenn ein Ausführungsfragment stabil ist, wenigstens 3 * L + 2 * n * L + 2 * D Zeiteinheiten dauert und genau ein Anführer sich bereits vor Beginn des Fragmentes entschieden hat, dann weiß er, daß jeder lebendige Prozessor sich entschieden hat und es wurden höchstens 2n Nachrichten versandt.

L12.20 **Ein Ausführungsfragment a besitze folgende Eigenschaften:**

1. **a ist gut,**
2. **in a gibt es genau einen Anführer i und**
3. **a dauert mindestens 16 * L + 8 * n * L + 9 * D Zeiteinheiten.**

Dann hat der Anführer zum Zeitpunkt 16 * L + 8 * n * L + 9 * D eine Entscheidung getroffen.

L12.21 **Es sei a ein gutes Ausführungsfragment, das mindestens 24 * L + 10 * n * L + 13 * D Zeiteinheiten dauert. Dann gilt:**

1. **Der Anführer entscheidet sich nach höchstens 21 * L + 8 * n * L + 11 * D Zeiteinheiten und es werden höchstens 8n Nachrichten versandt.**
2. **Spätestens nach 24 * L + 10 * n * L + 13 * D Zeiteinheiten ab Beginn des Ausführungsfragments entscheidet sich jeder lebendige Prozessor und es wurden maximal 2n weitere Nachrichten versandt.**