11 Zustandssicherung, Rücksetzen

Koo R.; Toueg, S.: Checkpointing and Rollback-Recovery for Distributed Systems. IEEE Trans. on Software Engineering, vol. SE-13, No. 1, Jan. 1987, pp. 23-31.

11.1 Begriffsbildungen

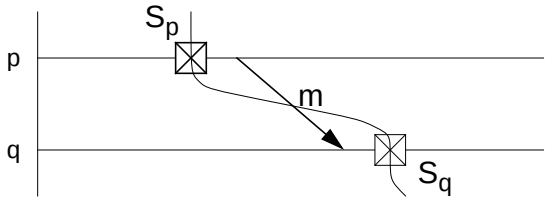
11.2 Problem der Erzeugung von Sicherungspunkten

11.3 Algorithmus von Koo und Toueg

11.1 Begriffsbildungen

Konsistenter globaler Zustand: Globaler Zustand, den das System nach bisheriger Beobachtung eingenommen haben kann.

Beispiel eines inkonsistenten Zustands

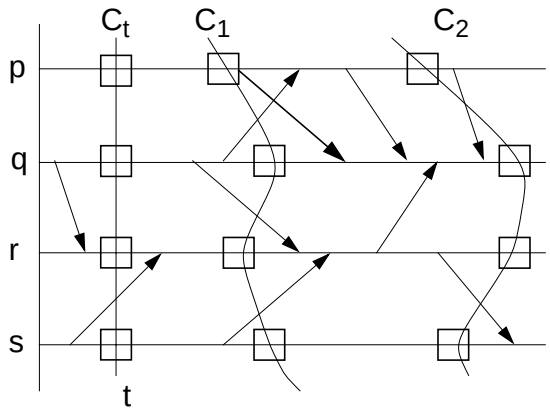


S_p: Zustand von p (das Versenden von m ist nicht aufgezeichnet)

S_q: Zustand von q (der Empfang von m ist aufgezeichnet)

Der globale Zustand (S_p, S_q) ist nicht konsistent.

Kanalzustände



C_t: Realzeit-Schnitt (Zustand des Systems zum Zeitpunkt t)

C_1: Schnitt, der zu einem global konsistenten Zustand gehört

C_2: Inkonsistenter Schnitt

Zustand des Kommunikationssystems bezüglich C_1

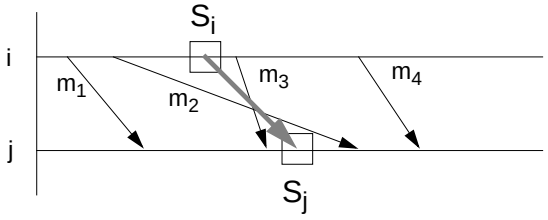
Je eine Nachricht in den Kanälen <q, p>, <q, r> und <s, r>

Informale Definition

Ein globaler Zustand S = (S_1, S_2, ..., S_i, ..., S_j, ... S_n) heißt konsistent, wenn für alle i und j gilt, daß eine von i an j gesendete Nachricht, die "vor S_j" empfangen wurde, nicht "nach S_i" gesendet worden sein kann.

Zustand eines Kanals <i, j> bezüglich eines globalen Zustandes S:

Alle Nachrichten die "vor S_i" gesendet wurden, aber erst "nach S_j" empfangen werden.



11.2 Erzeugung von Sicherungspunkten

Methode 1:

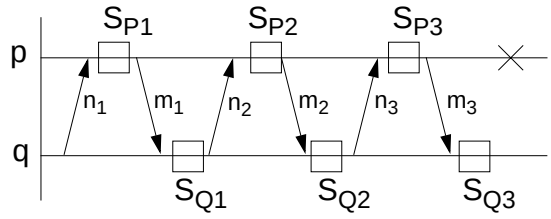
Prozesse sichern unabhängig voneinander lokale Zustände in einem stabilen Speicher. Nach einem Fehler wird eine konsistente Menge lokaler Zustandssicherungen bestimmt.

Methode 2:

Die Prozesse koordinieren die Zeitpunkte für Zustandssicherungen derart, daß jeder Prozeß nur jeweils die jüngste Sicherung aufbewahrt. Durch die Koordinierung wird dafür gesorgt, daß die Gesamtheit der jüngsten Sicherungen einen konsistenten globalen Zustand darstellt.

Probleme der Methode 1:

- **Großer Speicheraufwand**
- **Dominoeffekt**



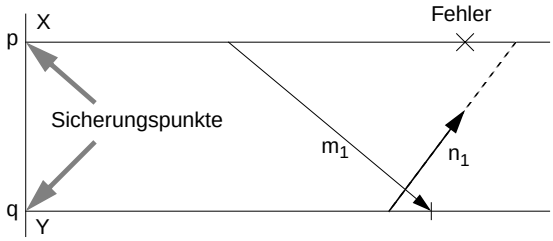
Deshalb weitere Untersuchung konzentriert auf Methode 2

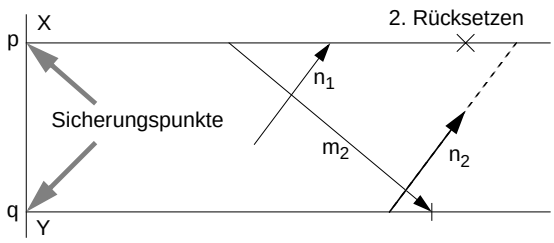
Rücksetzverfahren

Naive Vorgehensweise:

Jeder Prozeß setzt zu seinem zuletzt gesicherten Zustand zurück und nimmt dann die Arbeit wieder auf.

Problem: Livelock als Folge des nicht vollkommen zeitsynchronen Wiederanlaufs der Prozesse





Chandy-Lamport Algorithmus zur Bestimmung globaler Zustände

Annahme:

- **Nur zuverlässige FIFO-Kanäle**
- **Keine Ausfälle während des Sicherns, Rücksetzens und Wiederanlaufs**

Bemerkung:

- **Der Algorithmus kann nebenläufig zur eigentlichen Anwendung abgewickelt werden!**

Zustandssicherung: Erzeugung von Sicherungspunkten

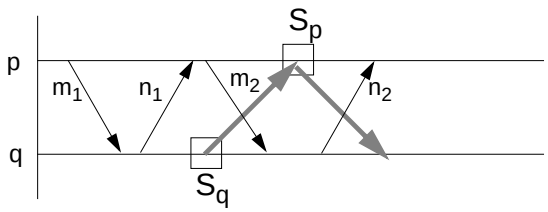
Ablauf

(a) Beim ersten Empfang einer Kontrollnachricht von q durch p und vor dem Senden weiterer Nachrichten:

- **p sichert seinen Zustand S_p**
- **p notiert den Kanal <q, p> als leer**
- **p schickt eine Kontrollnachricht an alle seine Nachbarn (auch an den, von dem die betrachtete Kontrollnachricht kam)**

(b) Bei nachfolgendem Erhalt einer Kontrollnachricht von r:

- **p notiert als Kanalzustand <r, p> die Sequenz der Nachrichten, die von r zwischen dem Festhalten von S_p und der eben erhaltenen Kontrollnachricht ankamen.**



Zustandssicherung: Erzeugung von Sicherungspunkten

Im weiteren:

- **Nur FIFO-Kanäle**
- **Ausfälle während des Rücksetzens und Wiederanlaufs sind möglich**

Ein Algorithmus zur Bildung konsistenter Sicherungspunkte heißt unverwüstlich (resilient), wenn er auch bei Prozessorausfällen funktioniert.

Eine (lokale) Zustandssicherung heißt dauerhaft (permanent), wenn sie nie rückgängig gemacht wird und (lokale) Berechnungen, die zur Erreichung des Sicherungspunktes notwendig waren, niemals wiederholt werden müssen.

Eine (lokale) Zustandssicherung heißt versuchsweise (tentative), wenn sie rückgängig oder dauerhaft gemacht werden kann.

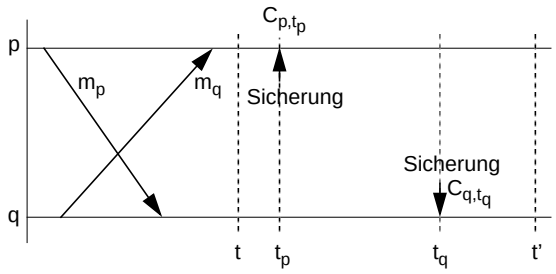
S11.1 Satz

Unverwüstliche Algorithmen zur Bildung konsistenter Sicherungspunkte können nicht nur dauerhafte lokale Zustandssicherungen vornehmen.

Beweis:

- **Angenommen es existiere ein solcher Algorithmus** A.
- **Zum Zeitpunkt** $t > 0$ **habe** p **bereits die Nachricht** m_q **von** q **empfangen und** q **die Nachricht** m_p **von** p.
- **Prozeß** p **rufe zum Zeitpunkt** t **den Algorithmus** A **auf und er werde zum Zeitpunkt** t' **beendet.**
- **Der Prozeß** p **bilde zum Zeitpunkt** t_p **mit** $t < t_p \leq t'$ **die lokale Sicherung** C_{p,t_p}.
- **A unverwüstlich und** C_{p,t_p} **dauerhaft**

  ⇒ **Prozeß** q **besitzt dauerhafte Sicherung** C_{q,t_q} **mit** $t < t_q \leq t'$.

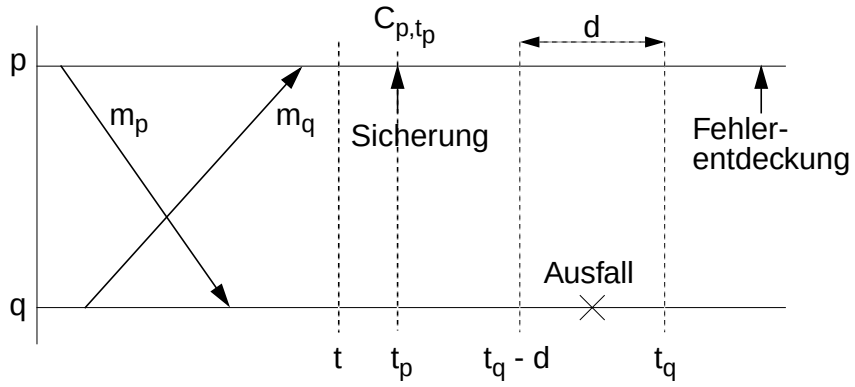


Annahme: Einer der Prozesse fällt aus

Die minimale Zeit zur Entdeckung eines Fehlers sei d.

(a) $t_p \leq t_q$

Man betrachte einen Ausfall von q in $(max(t, t_q - d), t_q)$



⇒ C_{p,t_p} **angelegt, aber nicht** C_{q,t_q} **, da q vorher ausgefallen ist.**

⇒ **Dieser Fall kann nicht auftreten.**

(b) $t_p > t_q$

Man betrachte einen Ausfall von p in $(max(t, t_p - d), t_p)$ **. Eine zu a) analoge Überlegung führt auch in diesem Fall zu einem Widerspruch.**

11.3

Algorithmus von Toueg

Eigenschaften des Algorithmus:

Unverwüstlich, d.h. Ausfälle bei Zustandssicherung oder Rücksetzen werden toleriert

Minimale Zahl von Prozessen ist involviert

Grundmuster des Algorithmus:

Analog 2-Phasen-Commit

1. **Phase: Veranlassung versuchsweiser Sicherungspunkte**
2. **Phase: Verbreitung und Durchsetzung der Entscheidung**

Spezielle Zustandsgröße:

willing_to_ckpt$_p$ **(** $\in$ **{"yes", "no"})**

Beschränkung in der Versendung von Basisnachrichten:

Zwischen dem Anlegen einer lokalen Sicherungskopie und dem Empfang der Entscheidung des Initiators werden keine Basisnachrichten versandt.

Zustandssicherung

Benutzernachrichten m enthalten eine vom Sender aufsteigend vergebene Sequenznummer m.l mit MIN < m.l < MAX

$R_{q,p}$:= Menge aller Nachrichten, die q seit der letzten dauerhaften Sicherung seines lokalen Zustandes von p erhalten hat.

$$\textbf{last\_rmsg}_q(p) := \begin{cases} \max(\{m.l \mid m \in R_{q,p}\}) & \text{falls } \{m.l \mid m \in R_{q,p}\} \neq \emptyset \\ MIN & \text{falls } \{m.l \mid m \in R_{q,p}\} = \emptyset \end{cases}$$

$S_{q,p}$:= Menge aller Nachrichten, die q seit der letzten versuchsweisen oder dauerhaften Sicherung seines lokalen Zustandes an p gesendet hat.

$$\textbf{first\_smsg}_q(p) := \begin{cases} \min\{m.l \mid m \in S_{q,p}\}) & \text{falls } \{m.l \mid m \in S_{q,p}\} \neq \emptyset \\ MIN & \text{falls } \{m.l \mid m \in S_{q,p}\} = \emptyset \end{cases}$$

$p \in \textbf{ckpt\_cohort}_q$:= q hat eine versuchsweise lokale Sicherungskopie angelegt und $\textbf{last\_rmsg}_q(p) > MIN$

Kontrollnachrichten:

- **"take a tentative checkpoint and $\textbf{last\_rmsg}_q(p)$"**
- **"make tentative checkpoint permanent"**
- **"undo tentative checkpoint"**

Daemon process

```
send(initiator, "take a tentative checkpoint and MAX");
await(initiator, willing_to_ckpt_initiator);
if (willing_to_ckpt_initiator == "yes")
    send(initiator, "make tentative checkpoint permanent");
else
    send(initiator, "undo tentative checkpoint");
```

INITIAL STATE

first_smsg$_p$(daemon) = MAX;

willing_to_ckpt$_p$ = "yes", falls p gewillt ist eine Sicherungskopie anzulegen, und
"no" sonst;

All processes p

```
UPON RECEIPT OF "take a tentative checkpoint and last_rmsg_q(p)" from q DO {
    if (willing_to_ckpt_p and last_rmsg_q(p) ≥ first_smsg_p(q) > MIN) {
        take_a_tentative_checkpoint;
        for (all r ∈ ckpt_cohort_p)
            send(r, "take a tentative checkpoint and last_rmsg_p(r)");
        for (all r ∈ ckpt_cohort_p) await(r, willing_to_ckpt_r);
        if (∃ r ∈ ckpt_cohort_p (willing_to_ckpt_r == "no"))
            willing_to_ckpt_p = "no";
    }
    send(q, willing_to_ckpt_p);
}

UPON RECEIPT OF m == "make tentative checkpoint permanent"
            or m == "undo tentative checkpoint" DO {
    if (m == "make tentative checkpoint permanent")
        make_tentative_checkpoint_permanent;
    else
        undo_tentative_checkpoint;
    for (all r ∈ ckpt_cohort_p) send(r, m);
}
```

p **erbt eine Sicherungsaufforderung von** q**, wenn er von** q **eine Nachricht "take a tentative checkpoint and last_rmsg$_q$(p)" erhält und das Prädikat willing_to_ckpt$_p$ and last_rmsg$_q$(p) ≥ first_smsg$_p$(q) > MIN erfüllt ist.**

L11.1 **Jeder Prozeß akzeptiert höchstens eine Aufforderung zum Anlegen einer versuchsweisen Sicherungskopie.**

Beweis:

Prozesse versenden vom Zeitpunkt des Anlegens einer Sicherungskopie bis zum Empfang der vom Initiator ausgehenden Entscheidung keine Benutzernachrichten, so daß in dieser Zeit first_smsg$_p$(q) = MIN ist. Also werden weitere Aufforderungen zum Anlegen einer Sicherungskopie nicht akzeptiert.

L11.2 **Jede lokale Ausführung des Algorithmus terminiert.**

Beweis:

(a) Falls die Aufforderung zum Anlegen einer Sicherungskopie nicht akzeptabel ist, terminiert der Algorithmus

(b) Falls p den Auftrag akzeptiert, legt es genau eine versuchsweise Sicherungskopie an. Es ist also nur zu zeigen, daß p nach Anlegen der Sicherungskopie terminiert.

- **q ∈ ckpt_cohort$_p$ ⇒ auf jeden Fall send(p, willing_to_ckpt), bevor q auf die Entscheidung des Initiators wartet. Es kann daher keinen Wartezyklus geben, in dem p auf die Antwort von q wartet und q auf die Entscheidung des Initiators.**
- **Wenn q von p den Auftrag zum Anlegen einer Sicherungskopie akzeptiert, so hat vorher p akzeptiert. Da wegen L11.1 die Akzeptiert-von-Relation nicht zirkulär sein kann, ist eine zyklische Wartesituation nicht möglich.**

(c) Durchsetzen der Entscheidung ist trivial, da die Kanäle und Prozesse als zuverlässig vorausgesetzt sind.

L11.3

Wenn die dauerhaften Sicherungskopien vor Beginn des Algorithmus konsistent waren, so gilt dies auch nach Beendigung des Algorithmus.

Beweis:

(a) Falls keine neuen dauerhaften Sicherungskopien entstanden, ist die Aussage trivial.

(b) Annahme: Es wurden neue dauerhafte Sicherungskopien gebildet und der Zustand ist inkonsistent.

Dann existieren p und q derart, daß p nach Bildung seiner letzten dauerhaften Sicherungskopie eine Benutzernachricht m an q gesendet hat, die q vor Bildung seiner letzten dauerhaften Sicherungskopie erhalten hat. Da die alten Sicherungskopien konsistent waren, muß q seine neueste Sicherungskopie während des Ablaufs des Algorithmus gebildet haben. Vor der versuchsweisen Bildung muß demnach $last\_rmsg_q(p) \geq m.l$ gewesen sein.

Also galt $p \in ckpt\_cohort_q$ und p wurde von q aufgefordert eine versuchsweise Kopie anzulegen. Als p diese Aufforderung erhielt, muß $willing\_to\_ckpt_p$ = “yes” gewesen sein, da sonst q seinerseits seine versuchsweise Sicherungskopie nicht dauerhaft gemacht hätte.

Weiter muß p eine versuchsweise Kopie nach dem Versenden von m angelegt haben oder es muß $last\_rmsg_q(p) \geq first\_smsg_p(q) > MIN$ gewesen sein. In beiden Fällen muß p seine Sicherungskopie nach dem Versenden von m gebildet haben. Diese Sicherungskopie wird dauerhaft gemacht, wenn die von q dauerhaft gemacht wird. Also machte p nach dem Versenden von m seine versuchsweise Kopie dauerhaft im Widerspruch zur Annahme.

Es sei $P = \{p_0, p_1, ..., p_k\}$ **die Menge der Prozesse, die während der Durchführung des Algorithmus ihren lokalen Zustand sichern, wobei** p_0 **Initiator ist.**

$C(P) = \{c(p_0), c(p_1), ..., c(p_k)\}$ **sei die Menge der neuen zugehörigen Sicherungen.**

$C'(P) = \{c'(p_0), c'(p_1), ..., c'(p_k)\}$ **besitze die Eigenschaft, daß** $c(p_0) = c'(p_0)$ **ist und daß für** $1 \le i \le k$ **entweder** $c'(p_i)$ **gleich** $c(p_i)$ **oder** $c'(p_i)$ **gleich der letzten Sicherung vor Ausführung des Algorithmus ist.**

S11.2 $C'(P)$ **ist genau dann konsistent, wenn** $C'(P) = C(P)$ **ist.**

Beweis:

$\Leftarrow$: **L11.3**

$\Rightarrow$: **Die Relation *erbt eine Sicherungsaufforderung* ist nach L11.1 hierarchisch und, da es nur einen Initiator gibt, ein Baum** T**.**

Auf Grund der Konstruktion enthält T alle Prozesse.

Annahme: $C'(P) \neq C(P)$ **und** $C'(P)$ **konsistent. Sei** $r \in P$ **und** $c'(r) \neq c(r)$ **.**

Dann ist $(r \neq p_0)$ **und es existiert in T ein Pfad von** r **nach** p_0**. Auf diesem Pfad müssen** p **und** q **existieren derart, daß gilt:**

p ***erbt eine Sicherungsaufforderung*** **von** q**,** $c'(p) \neq c(p)$ **und** $c'(q) = c(q)$ **.**

Da p **erbt, muß gegolten haben**
willing_to_ckpt$_p$ and last_rmsg$_q$(p) $\ge$ first_smsg$_p$(q) > MIN.

Sei m **die Nachricht, die** q **von** p **erhalten hat und für die last_rmsg$_q$(p) = m.l war.**

Wegen $m.l \ge first\_smsg_p(q)$ **ist das Senden dieser Nachricht in** $c'(p)$ **nicht aufgezeichnet, aber ihr Empfang in** $c'(q)$**. Also ist** $C'(P)$ **nicht konsistent.**

Berücksichtigung von Prozessor-Ausfällen

Daemon process

send(initiator, "**take a tentative checkpoint and MAX**");
await(initiator, willing_to_ckpt$_{initiator}$);
if (willing_to_ckpt$_{initiator}$ == "**yes**")
    send(initiator, "**make tentative checkpoint permanent**");
else
    send(initiator, "**undo tentative checkpoint**");

INITIAL STATE

first_smsg$_p$(daemon) = MAX;
willing_to_ckpt$_p$ = "yes", falls p gewillt ist eine Sicherungskopie anzulegen, und
"no" sonst;

All processes p

UPON RECEIPT OF "**take a tentative checkpoint and** last_rmsg$_q$(p)" **from** q DO {
    if (willing_to_ckpt$_p$ **and** last_rmsg$_q$(p) ≥ first_smsg$_p$(q) > MIN) {
        take_a_tentative_checkpoint;
        for (**all** r ∈ ckpt_cohort$_p$)
            send(r, "**take a tentative checkpoint and** last_rmsg$_p$(r)");
        for (**all** r ∈ ckpt_cohort$_p$) **await**(r, willing_to_ckpt$_r$);
        if (∃ r ∈ ckpt_cohort$_p$ ((willing_to_ckpt$_r$ == "**no**") **or** has_failed(r)))
            willing_to_ckpt$_p$ = "**no**";
    }
    send(q, willing_to_ckpt$_p$);
}
UPON RECEIPT OF "**make tentative checkpoint permanent**"
        or m == "**undo tentative checkpoint**" DO {
    if (m == "**make tentative checkpoint permanent**")
        make_tentative_checkpoint_permanent;
    else
        undo_tentative_checkpoint;
    for (**all** r ∈ ckpt_cohort$_p$) **send**(r, m);
}

Rücksetzen und Wiederanlauf

$S_{q,p}$:= Menge aller Nachrichten, die q vor der letzten dauerhaften Sicherung seines lokalen Zustandes an p gesendet hat.

$$\textbf{last\_smsg}_q\textbf{(p)} := \begin{cases} \max(\{m.l \mid m \in S_{q,p}\}) & \text{falls } \{m.l \mid m \in S_{q,p}\} \neq \emptyset \\ MAX & \text{falls } \{m.l \mid m \in S_{q,p}\} = \emptyset \end{cases}$$

$p \in$ roll_cohort$_q$:= q kann an p Nachrichten senden.

Spezielle Zustandsgröße:

willing_to_roll$_p$ ($\in$ {"yes", "no"})

Kontrollnachrichten:

- **"prepare to roll back and last_smsg$_q$(p)"**
- **"roll back"**
- **"do not roll back"**

Daemon process

send(initiator, "**prepare to roll back and MIN**");
await(initiator, willing_to_roll$_{initiator}$);
if (willing_to_roll$_{initiator}$ == "**yes**")
    send(initiator, "**roll back**");
else
    send(initiator, "**do not roll back**");

INITIAL STATE

ready_to_roll$_p$ = "**yes**";
last_rmsg$_p$(daemon) = MAX;
willing_to_roll$_p$ = "yes", falls p gewillt ist zurückzusetzen, und
"no" sonst;

All processes p

```
UPON RECEIPT OF "prepare to roll back and last_smsg_q(p)" from q DO {
    if (willing_to_roll_p and last_rmsg_p(q) > last_smsg_q(p) and ready_to_roll_p) {
        ready_to_roll_p = "no";
        for (all r ∈ roll_cohort_p)
            send(r, "prepare to roll back and last_smsg_p(r)");
        for (all r ∈ roll_cohort_p) await(r, willing_to_roll_r);
        if (∃ r ∈ roll_cohort_p (willing_to_roll_r == "no" or has_failed(r)))
            willing_to_roll_p = "no";
    }
    send(q, willing_to_roll_p);
}

UPON RECEIPT OF m == "roll back"  or m == "do not roll back" and ready_to_roll_p == "no" DO {
    if (m == "roll back")
        restart_from_p's_permanent_checkpoint;
    else
        resume_execution;
    for (all r ∈ roll_cohort_p) send(r, m);
}
```


L11.4 **Jeder Prozeß akzeptiert höchstens eine Aufforderung zur Rücksetzvorbereitung ("prepare to roll back").**

Beweis:

Prozesse setzen nach Empfang einer Aufforderung zur Rücksetzvorbereitung ready_to_roll = false und ändern es bis zum Abschluß des Algorithmus nicht mehr. Also werden weitere Aufforderungen zum Anlegen einer Sicherungskopie nicht akzeptiert.

L11.5 **Jede lokale Ausführung des Algorithmus R terminiert.**

Beweis:

(a) **Falls die Aufforderung zum Rücksetzen ("prepare to roll back") nicht akzeptabel ist, terminiert der Algorithmus.**

(b) **Es ist also nur zu zeigen, daß p nach Annahme einer Rücksetzaufforderung terminiert (d. h. keine Verklemmungen auftreten).**

- **$q \in roll\_cohort_p$ wartet auf die Entscheidung des Initiators**

  $\Rightarrow$ q hat willing_to_roll bereits gesendet. Es kann daher keinen Wartezyklus geben, in dem p auf die Antwort von q wartet und q auf die Entscheidung des Initiators.

- **Wenn q von p den Auftrag zum Rücksetzen akzeptiert, so hat vorher p akzeptiert. Da wegen L11.4 die Akzeptiert-von-Relation nicht zirkulär sein kann, ist eine zyklische Wartesituation nicht möglich.**

(c) **Durchsetzen der Entscheidung ist trivial, da die Kanäle und Prozesse als zuverlässig vorausgesetzt sind.**

L11.6 **Wenn der Zustand des Systems vor Ausführung von R konsistent ist, so befindet es sich auch nach Ausführung von R in einem konsistenten Zustand.**

Beweis:

Annahme: **Entgegen der Behauptung sei nach Ablauf von R der Zustand inkonsistent.**

Dann müssen Prozesse p und q und eine Basisnachricht m von q an p existieren derart, daß während der Durchführung von R das Senden von m rückgängig gemacht wurde, aber nicht das Empfangen.

q hat nach Empfang der Rücksetzaufforderung keine Basisnachrichten mehr verschickt. Also hat q an p einen Rücksetzauftrag erst nach dem Versenden von m verschickt.

Da q das Senden von m rückgängig machte, muß
$m.l > last\_smsg_q(p)$ sein.

Andererseits kann p keine dauerhafte Sicherungskopie angelegt haben nach Empfang von m und vor q's Rücksetzaufforderung: Die Erzeugung dieser Sicherungskopie zusammen mit der Tatsache, daß q keine neue dauerhafte Sicherungskopie angelegt hat, widersprächen nämlich L10.3.

Als q's Rücksetzufforderung bei p ankam, mußte also p schon eine solche Aufforderung erhalten haben oder es akzeptierte die von q.

Da p und q die gleiche Aufforderung bekamen, muß p zurückgesetzt haben.

Fall 1: m erreichte p nach der als erste erhaltenen Rücksetzaufforderung .

Da es sich um FIFO-Kanäle handelt, kam m bei p vor der Rücksetzaufforderung von q an. Der Initiator traf seine Entscheidung nachdem p auf q's Anfrage geantwortet hatte. Also setzte p nach Erhalt von m zurück. Widerspruch!

Fall 2: m erreichte p vor der als erste erhaltenen Rücksetzaufforderung.

Wie schon gezeigt, legte p keine dauerhafte Sicherungskopie nach Erhalt von m an. Also wird beim Rücksetzen von p der Empfang von m rückgängig gemacht.

S11.3 Es sei E eine Ausführung von R, in der Initiator p_0 und eine Menge P weiterer Prozesse zurücksetzen. Sei E' eine Ausführung, die mit E übereinstimmt bis auf die Tatsache, daß nur eine echte Teilmenge von P zurücksetzt. Dann hinterläßt E' das System in einem inkonsistenten Zustand.