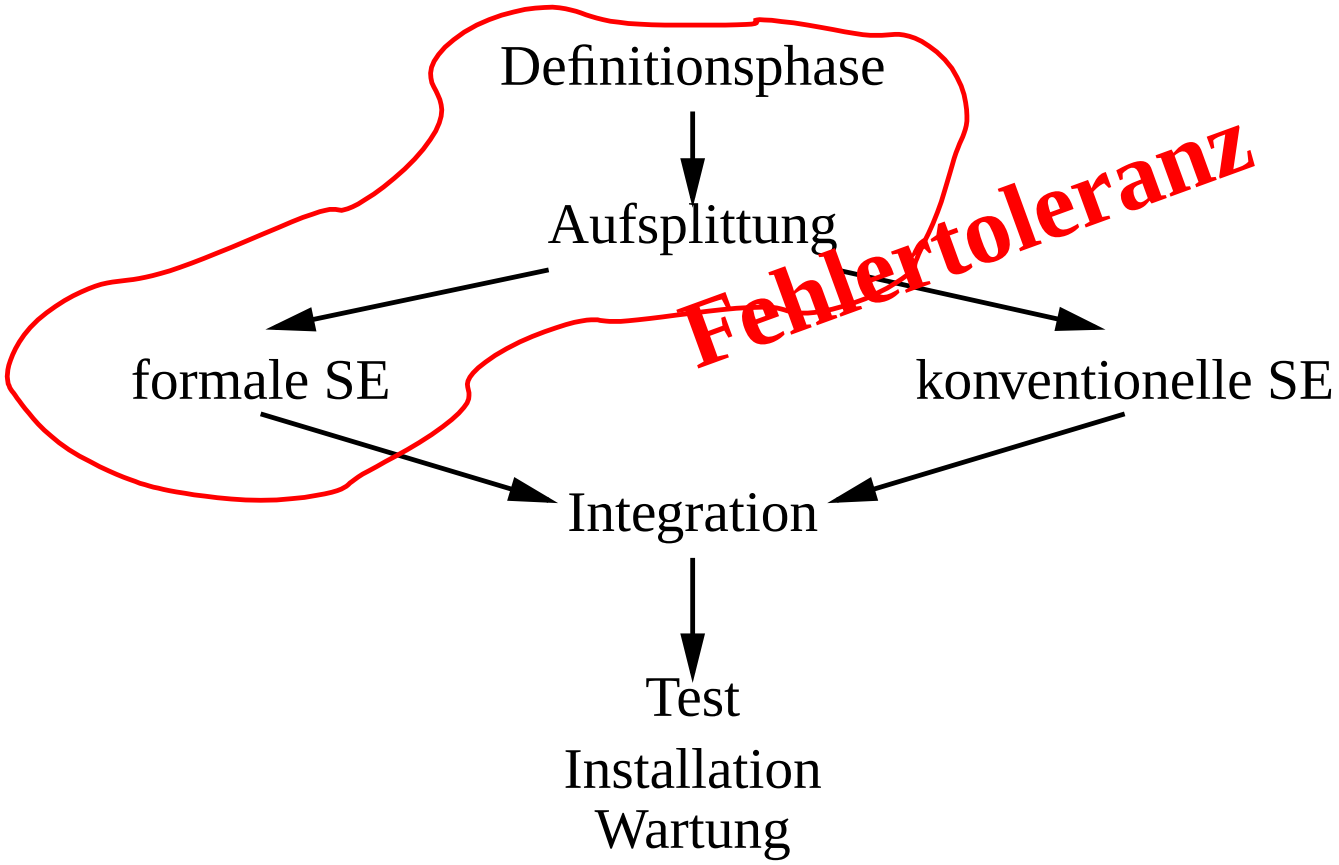
Formale Grundlagen der Fehlertoleranz in verteilten Systemen

Felix Gärtner

TU Darmstadt

felix@informatik.tu-darmstadt.de

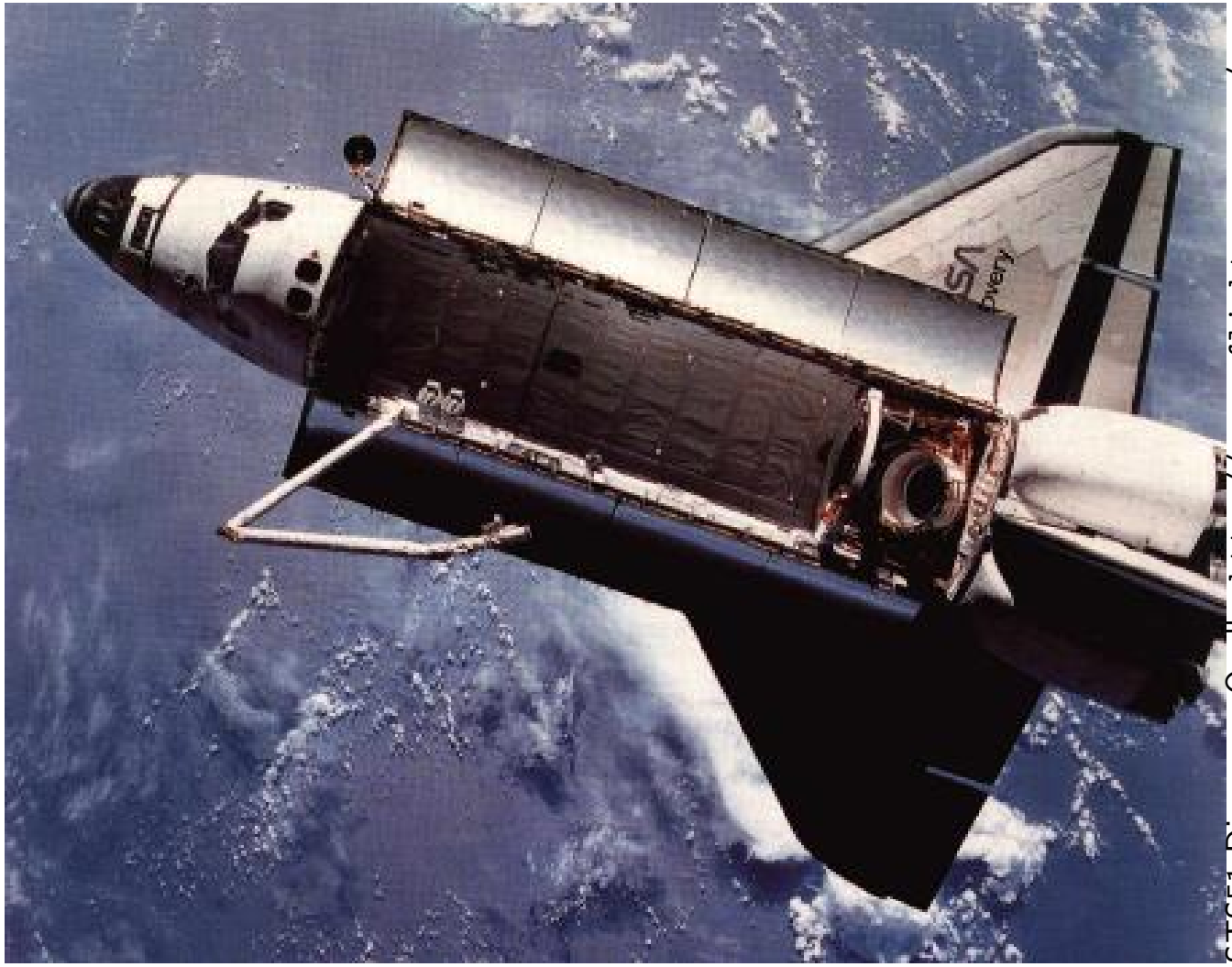
Bezug zur formalen Softwareentwicklung



Entwicklung fehlertoleranter Systeme

- Benötigt wissenschaftlich gesicherte Methodik.
- Bestandteile (von mir untersucht):
  - Modellbildung und -bewertung [Gärtner 1999a].
  - Spezielle Methoden der Verifikation [Gärtner 1999b; Mantel and Gärtner 2000].
  - Verständnis für die grundsätzliche Wirkungsweise von Fehlertoleranzverfahren. [Gärtner and Völzer 2000]
  - Algorithmische Grundbausteine [Gärtner and Kloppenburg 2000].

Beispiel: Space Shuttle



STS51 Discovery, Quelle: http://spaceflight.nasa.gov/

Fehlertolerante Arbeitsweise [Spector and Gifford 1984]

- Fünf redundante Computer (IBM).
- Vier davon fahren Steuerungssoftware.
- *I vote, you vote.*
- *Fail-operational, fail-safe.*
- Fünfter Computer fährt Backup-Software (von anderem Hersteller).

Weitere Methoden

- *triple modular redundancy* (TMR).
- *checkpointing/recovery*.
- *error detecting/error correcting codes*.
- *recovery blocks*.
- Replikation und *atomic broadcast*.
- . . .

Ausgangsfrage

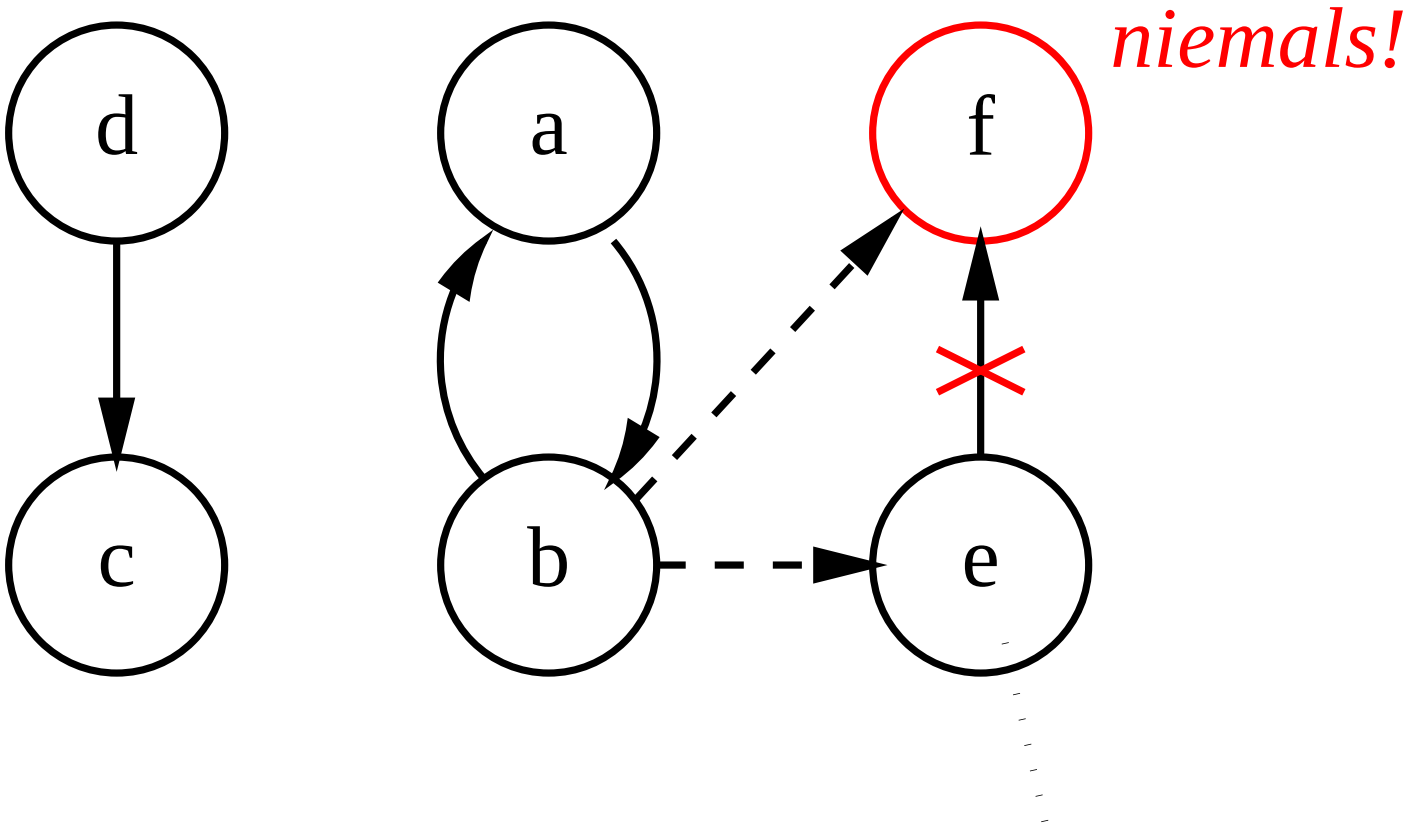
Was sind die grundlegenden Mechanismen, die in Fehlertoleranzverfahren eine Rolle spielen?

Erste Antwort:

Redundanz!

Aber wie und warum?

Modellierung als Automaten



Speicherredundanz

Speicherredundanz

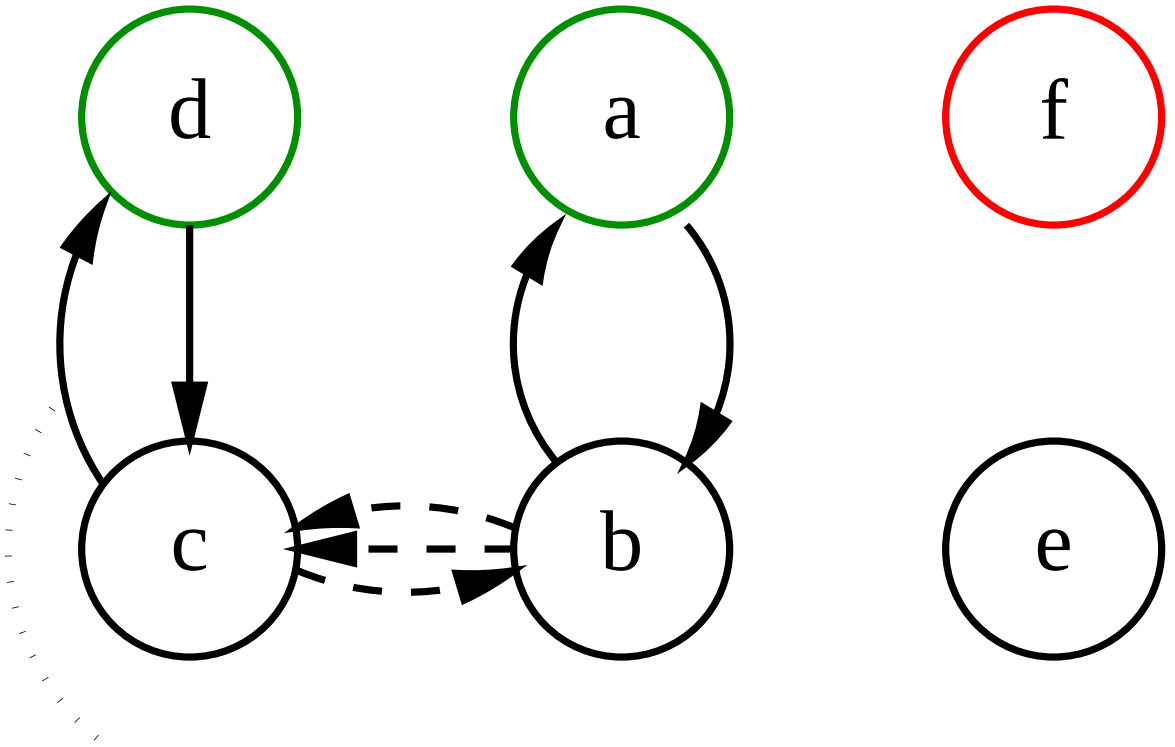
redundante Zustände = Speicherredundanz
Zustände, die nicht erreicht werden,
wenn keine Fehler auftreten.

- Bekannt aus der Kodierungstheorie
  [Rao and Fujiwara 1989].

- "Puffer" zur Fehlererkennung.

Problem: Lebendigkeit

immer wieder!



Zeitredundanz

Zeitredundanz

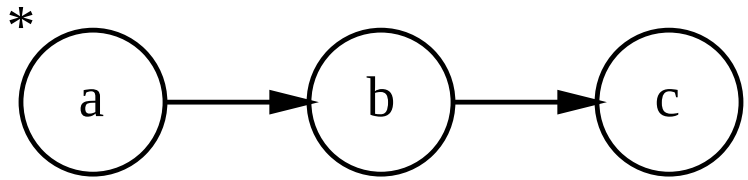
redundante Transitionen = Zeitredundanz
Transitionen, die nie ausgeführt werden,
wenn keine Fehler auftreten.

- Ergebnis ist neu!
- Ein Kapitel der Dissertation:
  Untersuchung der genauen Voraussetzungen, unter denen Speicher- und Zeitredundanz notwendig sind.

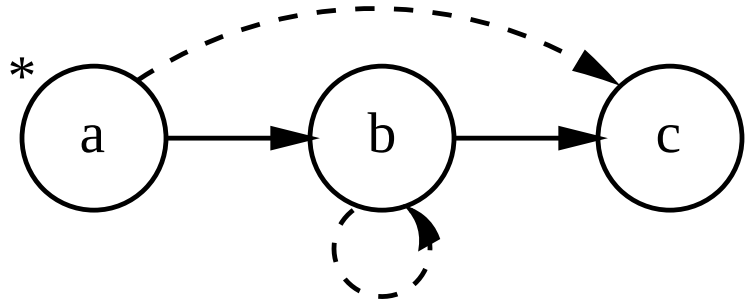
skip

Definition: Fehlermodell

- Fehlermodell $=$ Transformation F von Automaten, die Zustandsübergänge einbaut.



Σ



$F(\Sigma)$

Eigenschaften

- Automaten sind Generatoren von Abläufen.
- Ablauf $=$ Folge $s_1, s_2, \ldots$ von Zuständen.
- Eigenschaft $=$ Menge $\{\sigma_1, \sigma_2, \ldots\}$ von Abläufen.
- Ein Automat besitzt Eigenschaft E wenn alle seine Abläufe in E liegen.

Sicherheit und Lebendigkeit

- Sicherheitseigenschaft (*safety*): Eigenschaft, die immer im Endlichen verletzt wird.
- Beispiel: wechselseitiger Ausschluß.
- Lebendigkeitseigenschaft (*liveness*): Eigenschaft, die nur im Unendlichen verletzt werden kann.
- Beispiel: Aushungerungsfreiheit.
- Sicherheit und Lebendigkeit sind fundamental [Alpern and Schneider 1985].

Fehlertolerante Versionen

- Designprozeß:
  - Man hat ein System Σ_1, welches Sicherheitseigenschaft S verletzt, wenn Fehler aus F auftreten.
  - Möchte Σ_1 gerne in Σ_2 umwandeln, so daß Σ_2 S erfüllt, auch wenn Fehler aus F auftreten.
  - Σ_2 soll aber im fehlerfreien Fall dasselbe Verhalten haben wie Σ_1.

Σ_2 ist die *fehlertolerante Version* von Σ_1.

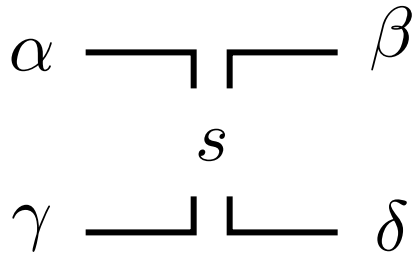
Fehlertoleranz und Sicherheit

- "Nie x" ist eine Sicherheitseigenschaft.
- Wann kann man fehlertolerante Versionen bezüglich einem Fehlermodell F und einer Sicherheitseigenschaft S bauen?
- Unmöglich, falls S direkt mittels Transitionen aus F verletzt werden kann.
- Oft möglich durch Löschen redundanter Transitionen.
- Resultierendes System Σ_2 hat Speicherredundanz.

Sicherheit und Speicherredundanz

- Speicherredundanz notwendig, um fehlertolerant bezüglich einer Sicherheitseigenschaft zu werden.
- Voraussetzung: F muß in Σ_1 und Σ_2 dieselben Fehler einbauen.
- Annahme: Sicherheitseigenschaft ist fusionsabgeschlossen.

Fusionsabgeschlossenheit



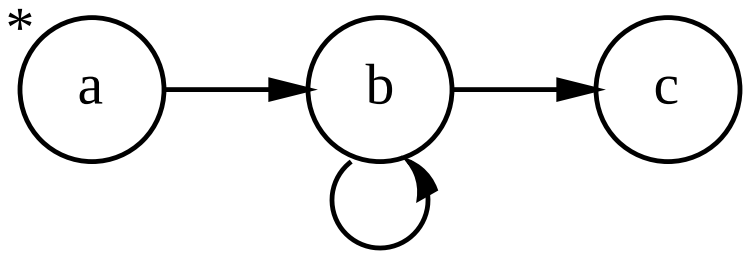
- Sicherheit allgemein: Ausschluß einer Menge endlicher Präfixe.
- Fusionsabgeschlossene Sicherheit: Ausschluß einer Menge von Transitionen.
- "Man kann an der Transition erkennen, ob sie ausgeführt werden darf oder nicht."

Fehlertoleranz und Lebendigkeit

- "Immer wieder x" ist eine Lebendigkeitseigenschaft.
- Was muß man tun, um fehlertolerante Versionen bezüglich einer Lebendigkeitseigenschaft zu konstruieren?

Lebendigkeitsannahmen

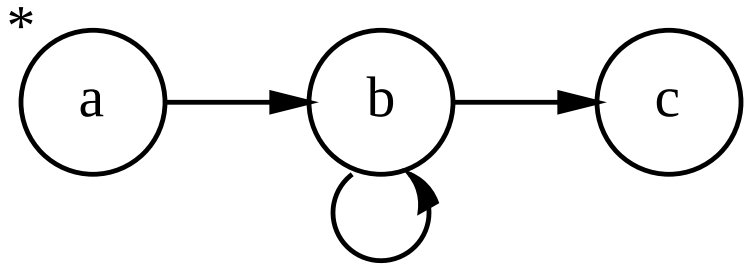
- Automat $\Sigma = (C, I, T, A)$



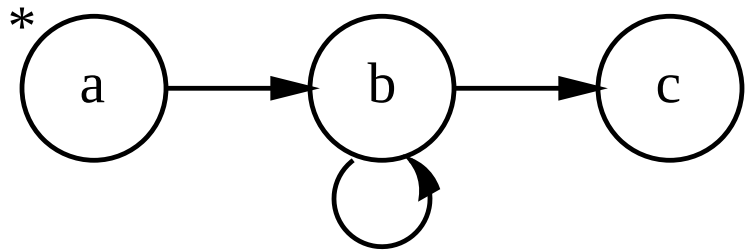
- Maximalität
- Schwache Fairness

Fehlermodelle und Lebendigkeit

- Fehlermodelle können auch die Lebendigkeitsannahme abschwächen.
- Beispiel: Abschwächung von schwacher Fairness zu Maximalität.



Σ



$F(\Sigma)$

Fehlertoleranz und Lebendigkeit

- Wann kann man fehlertolerante Versionen bezüglich einem Fehlermodell F und einer Lebendigkeitseigenschaft L bauen?
- Unmöglich, falls aus direktem Programmfluß ein *livelock* ausschließlich in F-Transitionen passieren kann.
- Oft möglich durch Hinzufügung von redundanten Transitionen.
- Resultierendes System Σ_2 hat Zeitredundanz.

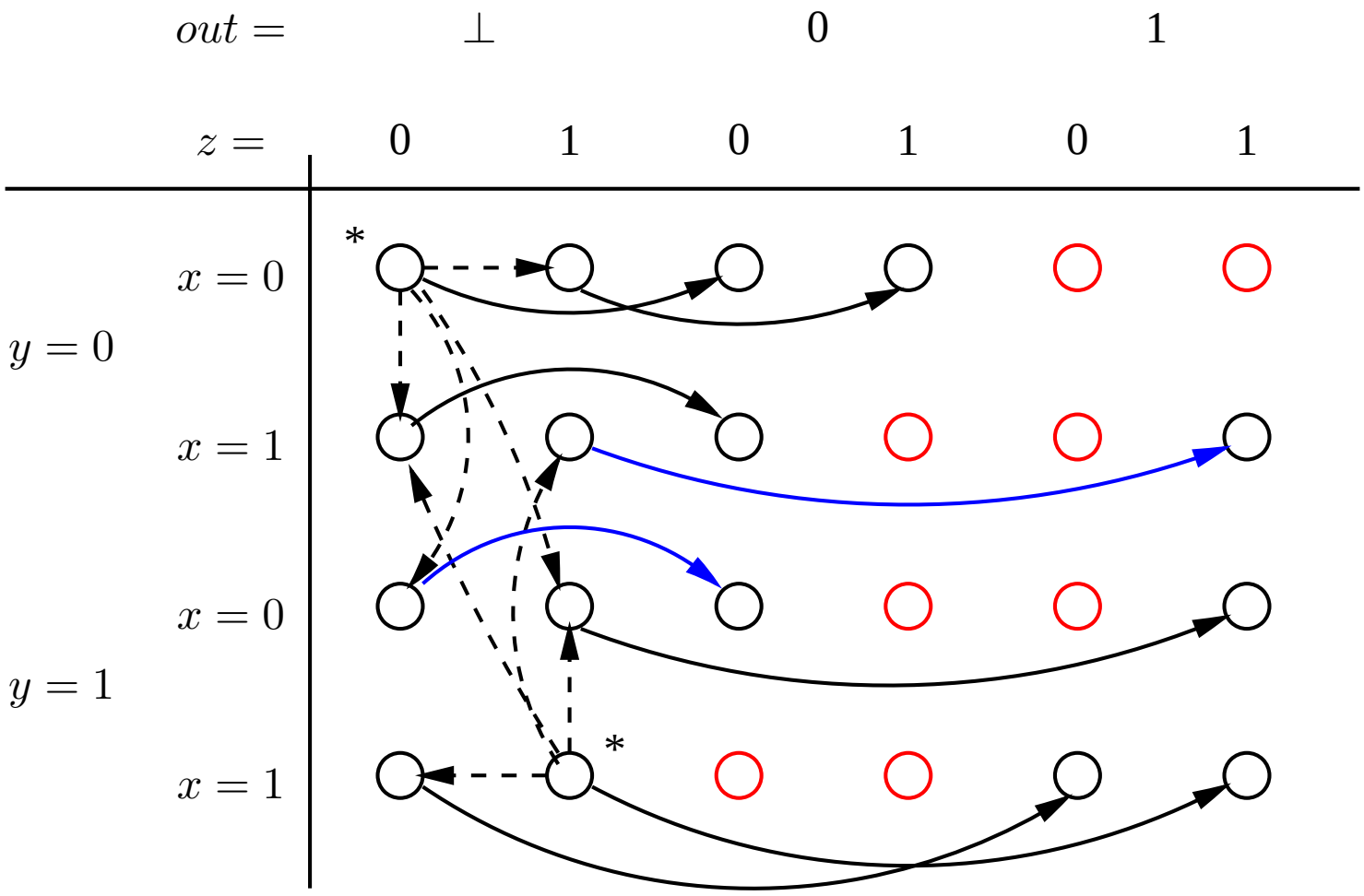
Lebendigkeit und Zeitredundanz

- Zeitredundanz notwendig, um fehlertolerant bezüglich einer Lebendigkeitsspezifikation zu werden.
- Voraussetzung: F schwächt bei beiden Programmen die Lebendigkeitsannahme in derselben Art ab.

Beispiel: TMR

- Komponente x berechnet einen Wert aus $\{0, 1\}$.
- Spezifikation: Nur korrekter Wert soll auf Ausgang out geschrieben werden (Sicherheit) und schließlich soll out von $\perp$ auf 0 oder 1 wechseln (Lebendigkeit).
- Zusätzliche Komponenten y und z stehen zur Verfügung.
- Fehlermodell: maximal eine Komponente berechner fehlerhaften Wert.

Speicher- und Zeitredundanz in TMR



Zusammenfassung

back

F-tolerant bzgl.	notwendig
Sicherheit	Speicherredundanz
Lebendigkeit	Zeitredundanz + Speicherredundanz

- Speicherredundanz bekannt aus der Kodierungstheorie.
- Relation zu *safety* neu!
- Zeitredundanz ist ein neuer Begriff!
- Relation zu *liveness* neu!

Nutzen

- "Keine Fehlertoleranz ohne Redundanz."
- Redundanz messbar (Zählen von redundanten Zuständen/Transitionen).
- Systeme bezüglich Redundanz vergleichbar:
  - TMR ist redundanzoptimal bzgl. Sicherheit und Lebendigkeit.
  - TMR ist nicht redundanzoptimal bzgl. Sicherheit.
- Methodologische Basis für effiziente Fehlertoleranzverfahren [Kulkarni and Arora 2000].

Danksagungen

- Diese Folien wurden hergestellt unter Verwendung von pdfLaTeX und Klaus Guntermanns PPower4.

References

Alpern, B. and Schneider, F. B. 1985. Defining liveness. *Information Processing Letters 21*, 181–185.

Gärtner, F. C. 1999a. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys 31*, 1 (March), 1–26.

Gärtner, F. C. 1999b. Transformational approaches to the specification and verification of fault-tolerant systems: Formal background and classification. *Journal of Universal Computer Science (J.UCS) 5*, 10 (Oct.), 668–692. Special Issue on Dependability Evaluation and Assessment.

Gärtner, F. C. and Kloppenburg, S. 2000. Consistent detection of global predicates under a weak fault assumption. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS2000)* (Nürnberg, Germany, Oct. 2000), pp. 94–103. IEEE Computer Society Press.

Gärtner, F. C. and Völzer, H. 2000. Redundancy in space in fault-tolerant systems. Technical Report TUD-BS-2000-06 (July), Department of Computer Science, Darmstadt University of Technology, Darmstadt, Germany.

Kulkarni, S. S. and Arora, A. 2000. Automating the addition of fault-tolerance. In M. Joseph Ed., *Formal Techniques in Real-Time and Fault-Tolerant Systems, 6th International Symposium (FTRTFT 2000) Proceedings*, Number 1926 in Lecture Notes in Computer Science (Pune, India, Sept. 2000), pp. 82–93. Springer-Verlag.

Mantel, H. and Gärtner, F. C. 2000. A case study in the mechanical verification of fault tolerance. *Journal of Experimental & Theoretical Artificial Intelligence (JETAI) 12*, 4 (Oct.), 473–488.

Rao, T. R. N. and Fujiwara, E. 1989. *Error-control coding for computer systems*. Prentice-Hall.

Spector, A. and Gifford, D. 1984. The space shuttle primary computer system. *Communications of the ACM 27*, 9, 874–900.