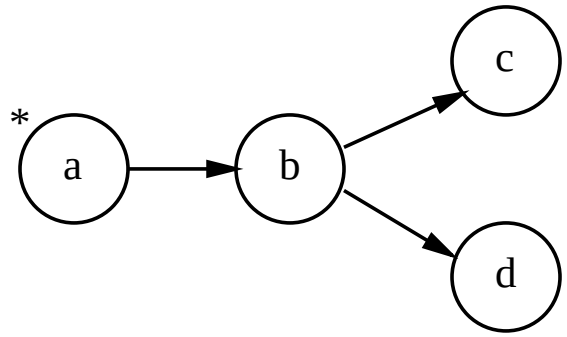
Fehlermodellierung ohne explizite Fehlerzustände

Felix Gärtner

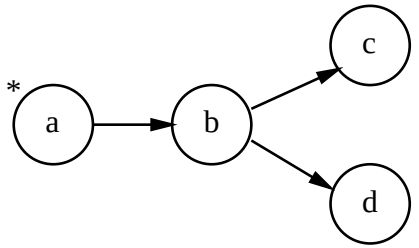
TU Darmstadt

Das Formalste zuerst . . .



- Programm: Automat mit Zustandsmenge C, Menge initialer Zustände $I \subseteq C$ und Zustandsübergangsrelation $T \subseteq C \times C$.
- Formal: Automat $= (C, I, T)$
- Automat erzeugt "Abläufe", z.B. abc, abd

Safety vs. Liveness



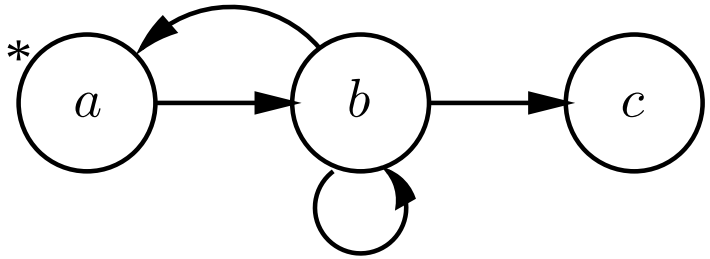
- (C, I, T) beschreibt nur die $safety$ des Programmes, d.h. das, was möglich ist.
- Kommt das System irgendwann in Zustand c oder d?
- Wir haben oft implizite *Fortschrittsannahmen*: Sie beschreiben die $liveness$ des Programmes, d.h., das, was nötig ist.
- Beispiel *Maximalität*: "Wann immer noch ein Zustandsübergang möglich ist, wird nach endlicher Zeit auch irgendein Zustandsübergang genommen."

Programm genauer

- $\Sigma = (C, I, T, A)$
- A ist eine Lebendigkeitsannahme: z.B. Maximalität = Menge aller Abläufe, die in c oder d enden.
- Forderung: Ablauf muß durch (C, I, T) erlaubt sein und in A liegen.
- Technische Forderung: A beschränkt (C, I, T) nicht (A ist *maschinenabgeschlossen* bzgl. (C, I, T)).

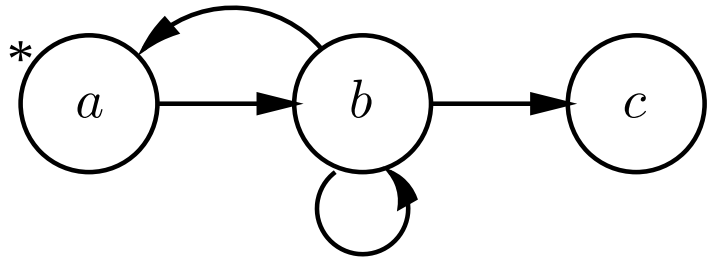
Lebendigkeitsannahmen

$\epsilon \supset$ Maximalität $\supset$ schwache Fairness $\supset$ starke Fairness



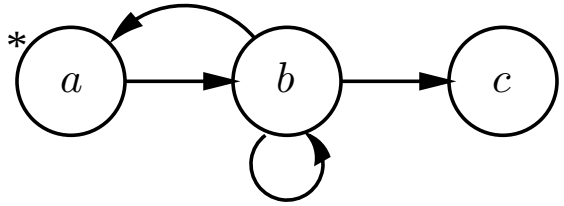
- ϵ: auch Ablaufpräfixe erlaubt, z.B. a, ab
- Maximalität: z.B. $abbbbb\ldots$
- schwache Fairness: z.B. $ababab\ldots$ (unendliche Schleife in einem Zustand nicht erlaubt)
- starke Fairness: z.B. $abbbabbbabc$ (unendliches Verwenden einer bestimmten Transition aus einem Zustand heraus nicht erlaubt)

Trick: transitionsgebundene Lebendigkeitsannahmen



- Beispiel: alle Transitionen stark fair, außer: (b, c) ist maximal.
- Ablauf: $abbabbabb\ldots$ jetzt erlaubt
- Bei globaler starker Fairness darf (c, d) nicht unendlich vernachlässigt werden.

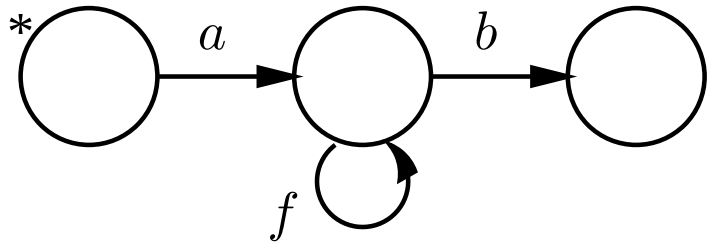
Möglichkeiten



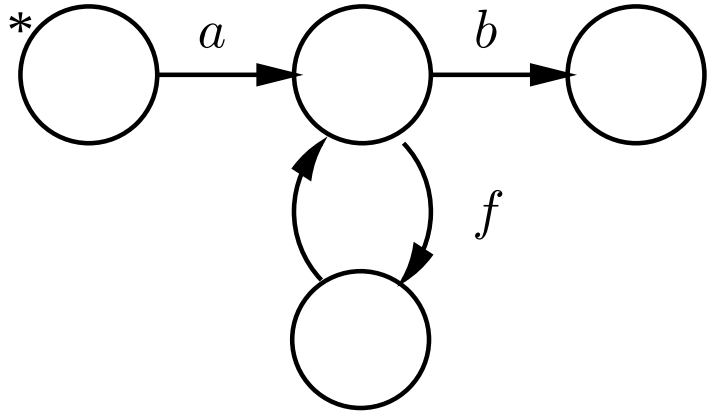
stark	schwach fair	$\Diamond c$?
$(b,a),(b,c),(b,b)$		ja
	$(b,a),(b,c),(b,b)$	nein
$(b,a),(b,c)$	(b,b)	ja
(b,a)	$(b,c),(b,b)$	nein
(b,b)	$(b,a),(b,c)$	ja

- Für alle nicht genannten Transitionen gelte mindestens Maximalität.
- Kann Terminierung auf verschiedene Arten “verletzen”.

Transition f soll nur endlich oft auftreten

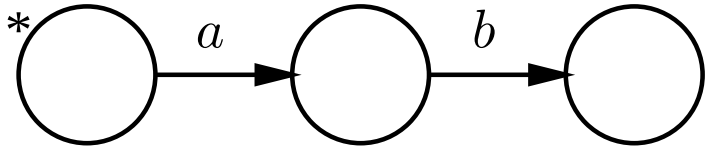


- b muß mindestens schwache Fairness besitzen (Maximalität erlaubt unendliche Schleife).



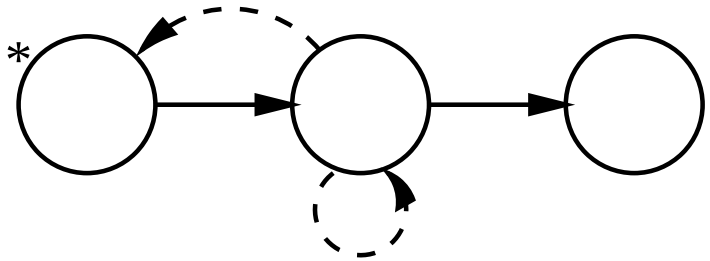
- b muß mindestens starke Fairness besitzen (schwache Fairness erlaubt unendliche Schleife).

“Anhalten” in mittlerem Zustand erlauben



- Transition b darf höchstens ϵ besitzen (schwächer als Maximalität).

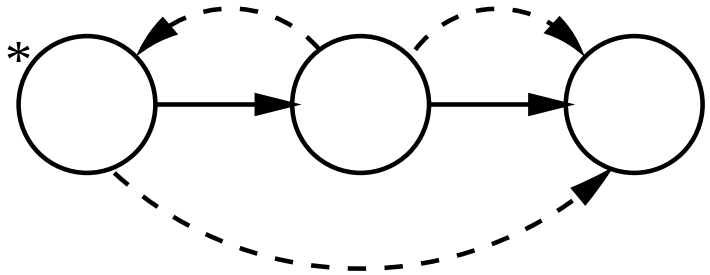
“Programmieransatz” zur Fehlermodellierung



- Ein Fehlermodell. . .
  - fügt Transitionen hinzu: Verletzungen der $safety$.
  - schwächt die Lebendigkeitsannahme für Transitionen ab: Verletzung der $liveness$ (z.B. Terminierung).
- Letzteres üblicherweise durch “Absturzzustand” realisiert.

Beispiele (1/3)

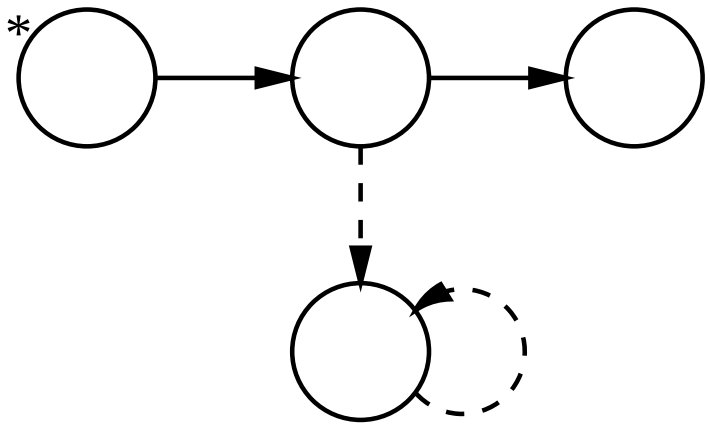
- *memory perturbation*



- Durch bloßes Hinzufügen von Transitionen modellierbar.

Beispiele (2/3)

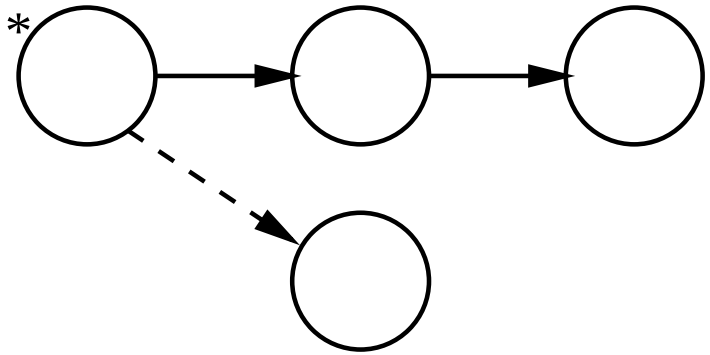
- *livelock* in mittlerem Zustand



- Zusätzlicher Zustand durch Abschwächung der Lebendigkeit vermeidbar.

Beispiele (3/3)

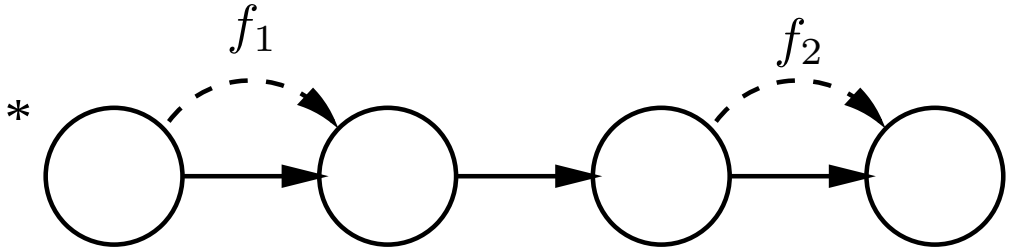
- *failstop*: Absturz im mittleren Zustand.



- Zusätzlicher Zustand durch Abschwächung der Lebendigkeit vermeidbar.

Wann kommt man ohne explizite Fehlerzustände aus?

- Beschränkung auf *lokale* Fehlerannahme, d.h. Auswirkungen von Fehlern in *einem* Fehlerbereich (z.B. ein Prozeß).
- Beschränkung auf "zeitfreies" Fehlverhalten: keine Echtzeitbedingungen.
- Gedächtnisloses Fehlverhalten (technischer Begriff: *fusionsabgeschlossen*); Beispiel: f_2 nur falls vorher f_1



Wofür?

- Aussagen über das Verhältnis zwischen zwei Systemen [GV01]:
  1. fehlerintolerantes Programm Σ_1 und
  2. fehlertolerante Version Σ_2 von Σ_1
     - Σ_1 erfüllt Spezifikation S wenn keine Fehler auftreten
     - Σ_1 verletzt S wenn Fehler auftreten
     - Σ_2 hat selbes Verhalten wie Σ_1 wenn keine Fehler auftreten
     - Σ_2 erfüllt S wenn Fehler auftreten

- Aussage: Σ_2 hat "mehr Zustände" als Σ_1

- Fehlerzustände verkomplizieren die Betrachtung.

Ausblick

- Interessante Analogie zu klassischen Fehlerdomänen [Lap92, Pow92]:
  - Wertefehler ($value$): zusätzliche Transitionen
  - Zeitfehler ($time$): Abschwächung der Lebendigkeit
- Fehlerzustände hinter einer Abstraktionsschicht (Lebendigkeitsannahme) verborgen (faktisch in den $scheduler$ verschoben).
- . . . kleine Erkenntnis, die das Leben leichter machen kann.

Literatur

[GV01] Felix C. Gärtner and Hagen Völzer. Defining redundancy in fault-tolerant computing. In *Brief Announcement at the 15th International Symposium on DIStributed Computing (DISC 2001)*, Lisbon, Portugal, October 2001.

[Lap92] Jean-Claude Laprie, editor. *Dependability: Basic concepts and Terminology*, volume 5 of *Dependable Computing and Fault-Tolerant Systems*. Springer-Verlag, 1992.

[Pow92] David Powell. Failure mode assumptions and assumption coverage. In Dhiraj K. Pradhan, editor, *Proceedings of the 22nd Annual International Symposium on Fault-Tolerant Computing (FTCS '92)*, pages 386–395, Boston, MA, July 1992. IEEE Computer Society Press.