Liveness und Fairness

- Liveness – Lebendigkeit
  - im Gegensatz zu Safety- bzw. Sicherheitseigenschaften
  - beschreibt **unausweichbares** Verhalten
  - macht wiederum nur Sinn bei Abstraktion vom konkretem Zeitverhalten:

    Konzentration auf mögliche Ereigisfolgen ohne “Timing”

- **Deadlock** ist noch eine Sicherheitseigenschaft:
  - “keine Zustand ohne Nachfolger ist erreichbar”

- **Livelock** als prototypische Lebendigkeitseigenschaft:
  - “System verfängt sich nie in einer Endlosschleife ohne *echten* Fortschritt”

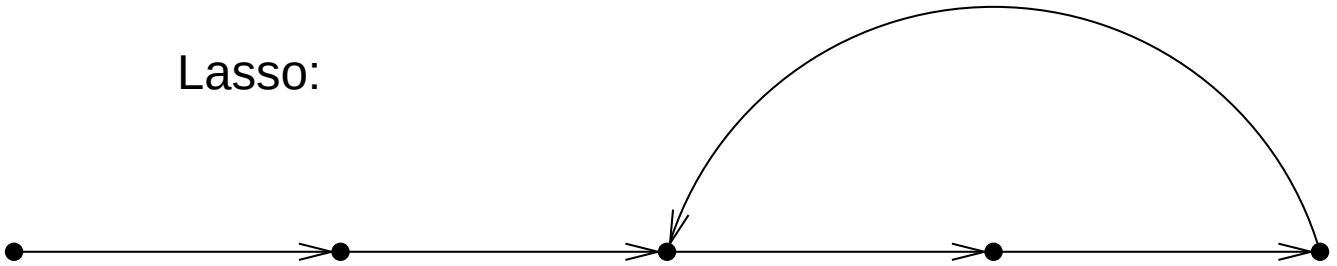
Weitere Beispiele zu Liveness

- Terminierung von Programmen/Prozessen/Protokollen:
  - z.B. Quicksort terminiert
  - z.B. IEEE Firewire: Initialisierungsphase terminiert mit einer korrekten Topologie
- Erwartete Ereignisse treten auch tatsächlich ein:
  - z.B. Befehle in superskalaren Prozessoren werden “Retired”
  - z.B. der Aufzug kommt wenn er gerufen wird
- in allen Beispielen werden **keine konkreten** Zeitschranken angegeben

Gegenbeispiele zu Lebendigkeitseigenschaften

- Endliche Systeme:

Lasso:



Kein Fortschritt

- Unendliche Systeme:
  - “Divergenz” möglich: Gegenbeispiel muss kein Lasso sein
  - z.B. Zähler über natürlichen Zahlen hochzählend erreicht nie wieder Anfangswert

Fairness

- Abstraktion vom konkreten “Scheduler”:
  - Reihenfolge der Ausführung von Prozessen ist im Modell beliebig
  - z.B. bei Interleaving oder voll asynchroner Parallelkomposition
- dadurch Artifakte bzw. unrealistische Gegenbeispiele zu Lebendigkeitseigenschaften:
  - Livelock durch permanentes Ignorieren eines Prozesses
- **Fairness** im Prinzip:
  - Scheduling **vernachlässigt** keinen (ausführbaren) Prozess für immer ...
  - ... ohne einen konkreten Scheduler anzugeben

    (letzeres oftmals nicht möglich, da relative Geschwindigkeit nicht bekannt)

Fairness Konkret

Definition ein *faires LTS* $A = (S, I, \Sigma, T, F)$ ist ein gewöhnliches LTS (S, I, Σ, T), mit $F \subseteq T$ die Menge der *fairen Übergänge* von A.

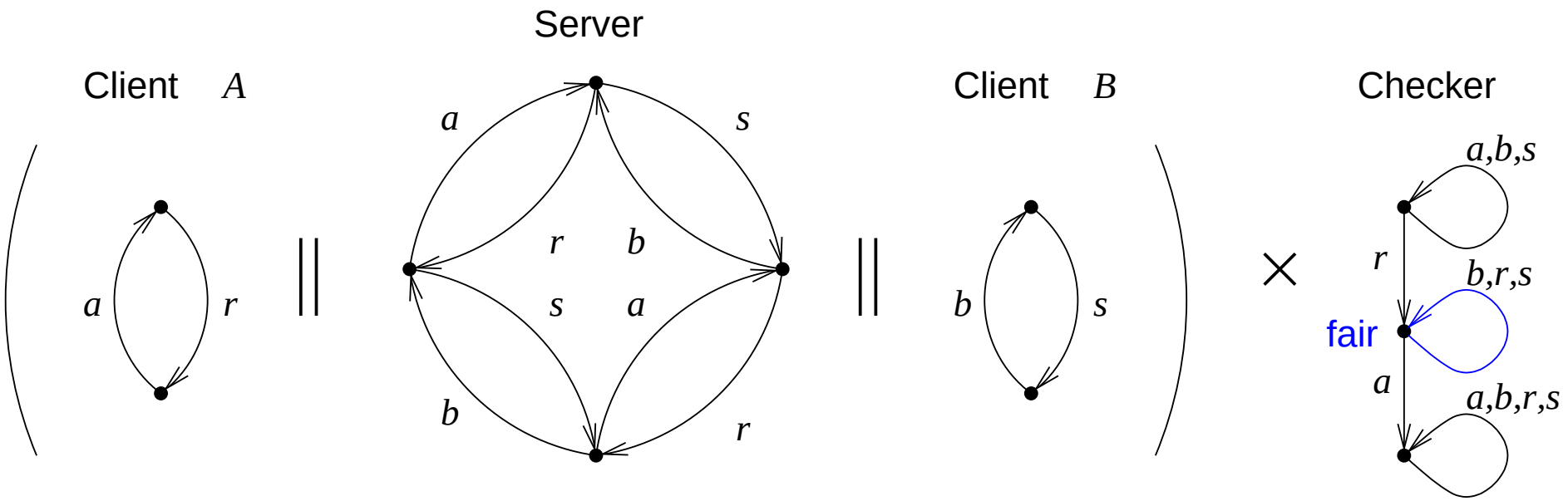
(ein Übergang ist einfach eine Trippel (s, a, s'))

Definition ein unendlicher Pfad $\pi = s_0 \xrightarrow{a_0} s_1 \rightarrow \ldots$ in einem fairen LTS ist *fair* gdw.

π unendlich viele faire Übergänge enthält: $\quad |\{i \mid (s_i, a_i, s_{i+1}) \in F\}| = \infty$.

Beispiel wähle für F die Menge der Übergänge in denen eine deterministische Komponente A_j einen lokalen oder globalen Übergang macht, oder aber "disabled" ist ($s \not\xrightarrow{a}$ für alle $a \in \Sigma_j$). Ein dadurch gegebener fairer Pfad in $A_1 \mid\mid \cdots \mid\mid A_n$ muss A_j immer wieder "ausführen".

Fairness im Checker



F = alle $\{b, r, s\}$-Übergänge, bei denen der Checker im unteren Zustand ist

(Gegenbeispiel dazu, dass ein a nach einem r irgendwann erfolgen muss)

es gibt einen fairen Pfad mit dem Trace $r(bs)^{\omega} = rbsbsbs\ldots$

(bei dem A genau einmal ausgeführt wird)

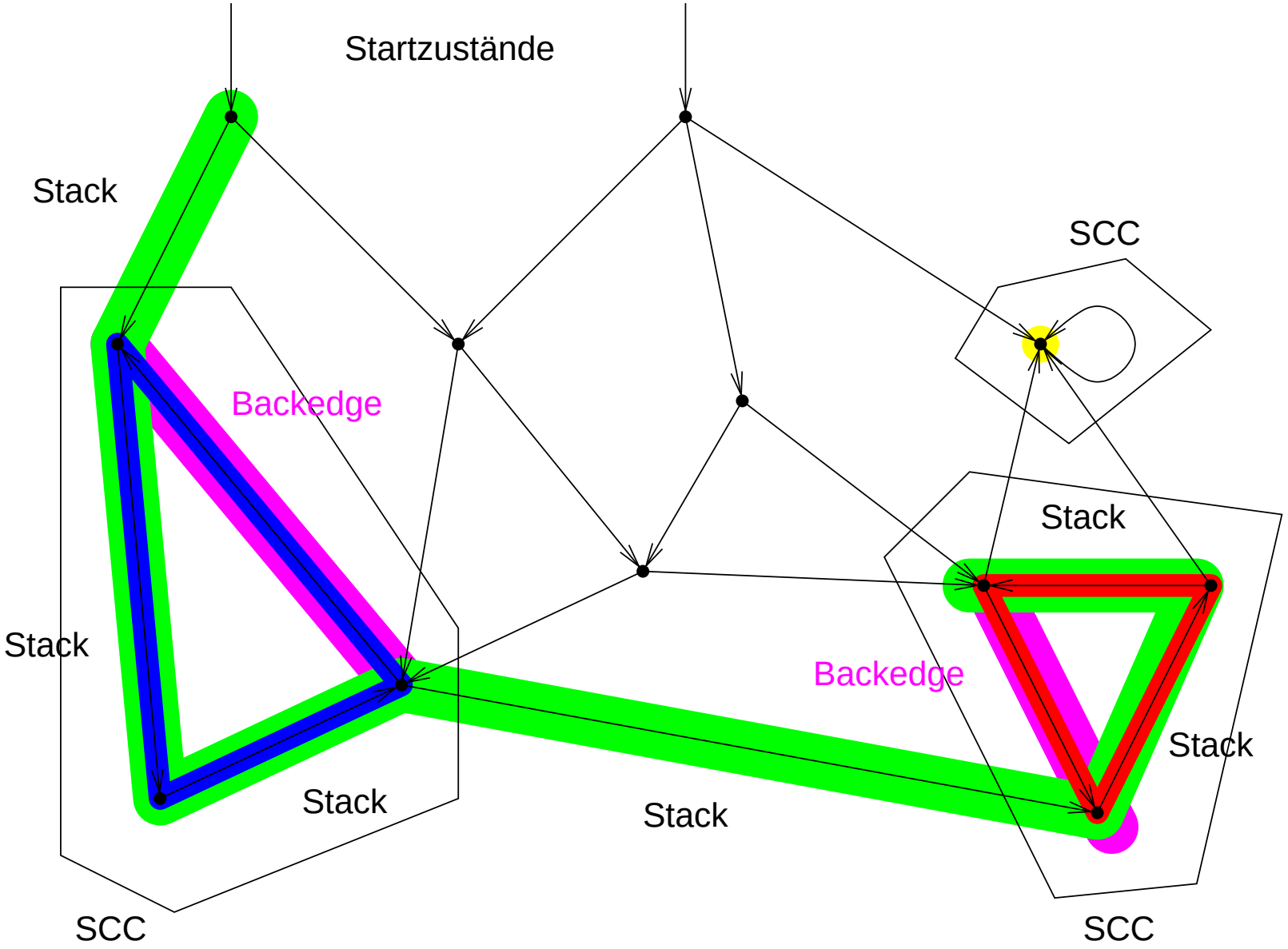
Suche nach Fairen Pfaden

- Im Prinzip (führt aber zu quadratischem Aufwand):
  - Backedge: Kante von `current` zu Knoten auf dem Stack der rekursiven DFS
  - Zyklus geschlossen durch Backedge ist fair gdw. er einen fairen Übergang hat
- SCC = strongly connected component
  - maximale Menge von Unterknoten eines Graphen (= Zustandraum),

    in der jeder Knoten von jedem anderen erreichbar ist
- jeder Zyklus (also auch der Zyklus eines Lasso) ist vollständig in einer SCC enthalten
  - SCC's lassen sich leicht in DFS finden:

    linearer Algorithmus von Tarjan zur Zerlegung eines Graphen in seine SCC

SCC


Startzustände
Stack
SCC
Backedge
Stack
Backedge
Stack
Stack
Stack
Stack
SCC
SCC

SCC Zerlegung nach Tarjan

- für jeden Knoten/Zustand merke
  1. *Depth First Search Index* (DFSI): ordnet Knoten wie sie gefunden werden
  2. minimal zu erreichender DFSI durch Backedge (MRDFSI)
     (auch über noch nicht besuchte Nachfolger)
- bestimme DFSI in der Prefix-Phase
  (bevor Nachfolger besucht werden)
- minimiere MRDFSI rekursiv über MRDFSI der Nachfolger
  (in der Suffix-Phase)
- `pop` in rekursiver DFS erst dann wenn und solange MRDFSI = DFSI
- all die Knoten mit demselben MRDFSI bilden eine SCC

Verallgemeinerte Fairness

Problem im Beispiel ist der Scheduler nicht *fair*

Definition ein *verallgemeinertes faires LTS* $A = (S, I, \Sigma, T, F_1, \ldots, F_n)$ ist ein LTS (S, I, Σ, T), mit $F_i \subseteq T$ eine Familie von Fairness Constraints.

Definition Übergang (s, a, s') heißt *fair* für das Fairness Constraint F_i gdw. $(s, a, s') \in F_i$.

Definition ein unendlicher Pfad $\pi = s_0 \stackrel{a_0}{\rightarrow} s_1 \rightarrow \ldots$ in einem verallgemeinerten fairen LTS ist *fair* gdw. π für jedes Fairness Constraint F_j unendlich viele faire Übergänge enthält: $|\{i \mid (s_i, a_i, s_{i+1}) \in F_j\}| = \infty$.

Voriges Beispiel wähle als zweites Fairness Constraint alle Übergänge, bei denen A beteiligt ist. Dann gibt es keinen fairen Pfad, also auch kein Gegenbeispiel, dem “Request” r folgt also immer ein “Acknowledge” a.

Faire Pfade bei Verallgemeinerter Fairness

- analog zum Algorithmus mit nur einem Fairness Constraint:
  - Teste beim Schließen eines Zyklus *alle* Fairness Constraints
  - oder überprüfe ob gefundene SCC alle Fairness Constraints “erlaubt”
- Alternative Konstruktion:
  - reduziere mehrere Fairness Constraints auf eines
  - zunächst lineare Ordnung der Fairness Constraints, z.B. $F_1 < \ldots < F_n$
  - Kreuzprodukt mit modulo n Zähler welcher von i nach $(i+1) \bmod n$ wechselt gdw. der Übergang im LTS das Fairness Constraint F_{i+1} erfüllt
  - einziges neues Fairness Constraint beim Zählerübergang von $n-1$ nach 0